

Image Processing: The Fundamentals

Image Processing: The Fundamentals

Maria Petrou

Costas Petrou



A John Wiley and Sons, Ltd., Publication

This edition first published 2010
© 2010 John Wiley & Sons Ltd

Registered office
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Library of Congress Cataloging-in-Publication Data
Petrou, Maria.

Image processing : the fundamentals / Maria Petrou, Costas Petrou. – 2nd ed.
p. cm.

Includes bibliographical references and index.

ISBN 978-0-470-74586-1 (cloth)

1. Image processing—Digital techniques.

TA1637.P48 2010

621.36'7—dc22

2009053150

ISBN 978-0-470-74586-1

A catalogue record for this book is available from the British Library.

Set in 10/12 Computer Modern by Laserwords Private Ltd, Chennai, India.
Printed in Singapore by Markono

*This book is dedicated to our mother and grandmother
Dionisia, for all her love and sacrifices.*

Contents

Preface	xxiii
1 Introduction	1
Why do we process images?	1
What is an image?	1
What is a digital image?	1
What is a spectral band?	2
Why do most image processing algorithms refer to grey images, while most images we come across are colour images?	2
How is a digital image formed?	3
If a sensor corresponds to a patch in the physical world, how come we can have more than one sensor type corresponding to the same patch of the scene?	3
What is the physical meaning of the brightness of an image at a pixel position?	3
Why are images often quoted as being 512×512 , 256×256 , 128×128 etc?	6
How many bits do we need to store an image?	6
What determines the quality of an image?	7
What makes an image blurred?	7
What is meant by image resolution?	7
What does “good contrast” mean?	10
What is the purpose of image processing?	11
How do we do image processing?	11
Do we use nonlinear operators in image processing?	12
What is a linear operator?	12
How are linear operators defined?	12
What is the relationship between the point spread function of an imaging device and that of a linear operator?	12
How does a linear operator transform an image?	12
What is the meaning of the point spread function?	13
Box 1.1. The formal definition of a point source in the continuous domain	14
How can we express in practice the effect of a linear operator on an image?	18
Can we apply more than one linear operators to an image?	22
Does the order by which we apply the linear operators make any difference to the result?	22
Box 1.2. Since matrix multiplication is not commutative, how come we can change the order by which we apply shift invariant linear operators?	22

Box 1.3. What is the stacking operator?	29
What is the implication of the separability assumption on the structure of matrix H ?	38
How can a separable transform be written in matrix form?	39
What is the meaning of the separability assumption?	40
Box 1.4. The formal derivation of the separable matrix equation	41
What is the “take home” message of this chapter?	43
What is the significance of equation (1.108) in linear image processing?	43
What is this book about?	44
2 Image Transformations	47
What is this chapter about?	47
How can we define an elementary image?	47
What is the outer product of two vectors?	47
How can we expand an image in terms of vector outer products?	47
How do we choose matrices h_c and h_r ?	49
What is a unitary matrix?	50
What is the inverse of a unitary transform?	50
How can we construct a unitary matrix?	50
How should we choose matrices U and V so that g can be represented by fewer bits than f ?	50
What is matrix diagonalisation?	50
Can we diagonalise any matrix?	50
2.1 Singular value decomposition	51
How can we diagonalise an image?	51
Box 2.1. Can we expand in vector outer products any image?	54
How can we compute matrices U , V and $\Lambda^{\frac{1}{2}}$ needed for image diagonalisation?	56
Box 2.2. What happens if the eigenvalues of matrix gg^T are negative?	56
What is the singular value decomposition of an image?	60
Can we analyse an eigenimage into eigenimages?	61
How can we approximate an image using SVD?	62
Box 2.3. What is the intuitive explanation of SVD?	62
What is the error of the approximation of an image by SVD?	63
How can we minimise the error of the reconstruction?	65
Are there any sets of elementary images in terms of which <i>any</i> image may be expanded?	72
What is a complete and orthonormal set of functions?	72
Are there any complete sets of orthonormal discrete valued functions?	73
2.2 Haar, Walsh and Hadamard transforms	74
How are the Haar functions defined?	74
How are the Walsh functions defined?	74
Box 2.4. Definition of Walsh functions in terms of the Rademacher functions	74
How can we use the Haar or Walsh functions to create image bases?	75
How can we create the image transformation matrices from the Haar and Walsh functions in practice?	76
What do the elementary images of the Haar transform look like?	80
Can we define an orthogonal matrix with entries only +1 or -1?	85
Box 2.5. Ways of ordering the Walsh functions	86
What do the basis images of the Hadamard/Walsh transform look like?	88

What are the advantages and disadvantages of the Walsh and the Haar transforms?	92
What is the Haar wavelet?	93
2.3 Discrete Fourier transform	94
What is the discrete version of the Fourier transform (DFT)?	94
Box 2.6. What is the inverse discrete Fourier transform?	95
How can we write the discrete Fourier transform in a matrix form?	96
Is matrix U used for DFT unitary?	99
Which are the elementary images in terms of which DFT expands an image?	101
Why is the discrete Fourier transform more commonly used than the other transforms?	105
What does the convolution theorem state?	105
Box 2.7. If a function is the convolution of two other functions, what is the rela- tionship of its DFT with the DFTs of the two functions?	105
How can we display the discrete Fourier transform of an image?	112
What happens to the discrete Fourier transform of an image if the image is rotated?	113
What happens to the discrete Fourier transform of an image if the image is shifted?	114
What is the relationship between the average value of the image and its DFT?	118
What happens to the DFT of an image if the image is scaled?	119
Box 2.8. What is the Fast Fourier Transform?	124
What are the advantages and disadvantages of DFT?	126
Can we have a real valued DFT?	126
Can we have a purely imaginary DFT?	130
Can an image have a purely real or a purely imaginary valued DFT?	137
2.4 The even symmetric discrete cosine transform (EDCT)	138
What is the even symmetric discrete cosine transform?	138
Box 2.9. Derivation of the inverse 1D even discrete cosine transform	143
What is the inverse 2D even cosine transform?	145
What are the basis images in terms of which the even cosine transform expands an image?	146
2.5 The odd symmetric discrete cosine transform (ODCT)	149
What is the odd symmetric discrete cosine transform?	149
Box 2.10. Derivation of the inverse 1D odd discrete cosine transform	152
What is the inverse 2D odd discrete cosine transform?	154
What are the basis images in terms of which the odd discrete cosine transform expands an image?	154
2.6 The even antisymmetric discrete sine transform (EDST)	157
What is the even antisymmetric discrete sine transform?	157
Box 2.11. Derivation of the inverse 1D even discrete sine transform	160
What is the inverse 2D even sine transform?	162
What are the basis images in terms of which the even sine transform expands an image?	163
What happens if we do not remove the mean of the image before we compute its EDST?	166
2.7 The odd antisymmetric discrete sine transform (ODST)	167
What is the odd antisymmetric discrete sine transform?	167

Box 2.12. Derivation of the inverse 1D odd discrete sine transform	171
What is the inverse 2D odd sine transform?	172
What are the basis images in terms of which the odd sine transform expands an image?	173
What is the “take home” message of this chapter?	176
3 Statistical Description of Images	177
What is this chapter about?	177
Why do we need the statistical description of images?	177
3.1 Random fields	178
What is a random field?	178
What is a random variable?	178
What is a random experiment?	178
How do we perform a random experiment with computers?	178
How do we describe random variables?	178
What is the probability of an event?	179
What is the distribution function of a random variable?	180
What is the probability of a random variable taking a specific value?	181
What is the probability density function of a random variable?	181
How do we describe many random variables?	184
What relationships may n random variables have with each other?	184
How do we define a random field?	189
How can we relate two random variables that appear in the same random field?	190
How can we relate two random variables that belong to two different random fields?	193
If we have just one image from an ensemble of images, can we calculate expectation values?	195
When is a random field homogeneous with respect to the mean?	195
When is a random field homogeneous with respect to the autocorrelation function?	195
How can we calculate the spatial statistics of a random field?	196
How do we compute the spatial autocorrelation function of an image in practice?	196
When is a random field ergodic with respect to the mean?	197
When is a random field ergodic with respect to the autocorrelation function?	197
What is the implication of ergodicity?	199
Box 3.1. Ergodicity, fuzzy logic and probability theory	200
How can we construct a basis of elementary images appropriate for expressing in an optimal way a whole set of images?	200
3.2 Karhunen-Loeve transform	201
What is the Karhunen-Loeve transform?	201
Why does diagonalisation of the autocovariance matrix of a set of images define a desirable basis for expressing the images in the set?	201
How can we transform an image so its autocovariance matrix becomes diagonal?	204
What is the form of the ensemble autocorrelation matrix of a set of images, if the ensemble is stationary with respect to the autocorrelation?	210
How do we go from the 1D autocorrelation function of the vector representation of an image to its 2D autocorrelation matrix?	211
How can we transform the image so that its autocorrelation matrix is diagonal?	213

How do we compute the K-L transform of an image in practice?	214
How do we compute the Karhunen-Loeve (K-L) transform of an ensemble of images?	215
Is the assumption of ergodicity realistic?	215
Box 3.2. How can we calculate the spatial autocorrelation matrix of an image, when it is represented by a vector?	215
Is the mean of the transformed image expected to be really 0?	220
How can we approximate an image using its K-L transform?	220
What is the error with which we approximate an image when we truncate its K-L expansion?	220
What are the basis images in terms of which the Karhunen-Loeve transform expands an image?	221
Box 3.3. What is the error of the approximation of an image using the Karhunen-Loeve transform?	226
3.3 Independent component analysis	234
What is Independent Component Analysis (ICA)?	234
What is the cocktail party problem?	234
How do we solve the cocktail party problem?	235
What does the central limit theorem say?	235
What do we mean by saying that “the samples of $x_1(t)$ are more Gaussianly distributed than either $s_1(t)$ or $s_2(t)$ ” in relation to the cocktail party problem? Are we talking about the <i>temporal</i> samples of $x_1(t)$, or are we talking about all possible versions of $x_1(t)$ at a given time?	235
How do we measure non-Gaussianity?	239
How are the moments of a random variable computed?	239
How is the kurtosis defined?	240
How is negentropy defined?	243
How is entropy defined?	243
Box 3.4. From all probability density functions with the same variance, the Gaussian has the maximum entropy	246
How is negentropy computed?	246
Box 3.5. Derivation of the approximation of negentropy in terms of moments	252
Box 3.6. Approximating the negentropy with nonquadratic functions	254
Box 3.7. Selecting the nonquadratic functions with which to approximate the negentropy	257
How do we apply the central limit theorem to solve the cocktail party problem?	264
How may ICA be used in image processing?	264
How do we search for the independent components?	264
How can we whiten the data?	266
How can we select the independent components from whitened data?	267
Box 3.8. How does the method of Lagrange multipliers work?	268
Box 3.9. How can we choose a direction that maximises the negentropy?	269
How do we perform ICA in image processing in practice?	274
How do we apply ICA to signal processing?	283
What are the major characteristics of independent component analysis?	289
What is the difference between ICA as applied in image and in signal processing? .	290
What is the “take home” message of this chapter?	292

4 Image Enhancement	293
What is image enhancement?	293
How can we enhance an image?	293
What is linear filtering?	293
4.1 Elements of linear filter theory	294
How do we define a 2D filter?	294
How are the frequency response function and the unit sample response of the filter related?	294
Why are we interested in the filter function in the real domain?	294
Are there any conditions which $h(k, l)$ must fulfil so that it can be used as a convolution filter?	294
Box 4.1. What is the unit sample response of the 2D ideal low pass filter?	296
What is the relationship between the 1D and the 2D ideal lowpass filters?	300
How can we implement in the real domain a filter that is infinite in extent?	301
Box 4.2. z -transforms	301
Can we define a filter directly in the real domain for convenience?	309
Can we define a filter in the real domain, without side lobes in the frequency domain?	309
4.2 Reducing high frequency noise	311
What are the types of noise present in an image?	311
What is impulse noise?	311
What is Gaussian noise?	311
What is additive noise?	311
What is multiplicative noise?	311
What is homogeneous noise?	311
What is zero-mean noise?	312
What is biased noise?	312
What is independent noise?	312
What is uncorrelated noise?	312
What is white noise?	313
What is the relationship between zero-mean uncorrelated and white noise?	313
What is iid noise?	313
Is it possible to have white noise that is not iid?	315
Box 4.3. The probability density function of a function of a random variable	320
Why is noise usually associated with high frequencies?	324
How do we deal with multiplicative noise?	325
Box 4.4. The Fourier transform of the delta function	325
Box 4.5. Wiener-Khinchine theorem	325
Is the assumption of Gaussian noise in an image justified?	326
How do we remove shot noise?	326
What is a rank order filter?	326
What is median filtering?	326
What is mode filtering?	328
How do we reduce Gaussian noise?	328
Can we have weighted median and mode filters like we have weighted mean filters?	333
Can we filter an image by using the linear methods we learnt in Chapter 2?	335
How do we deal with mixed noise in images?	337

Can we avoid blurring the image when we are smoothing it?	337
What is the edge adaptive smoothing?	337
Box 4.6. Efficient computation of the local variance	339
How does the mean shift algorithm work?	339
What is anisotropic diffusion?	342
Box 4.7. Scale space and the heat equation	342
Box 4.8. Gradient, Divergence and Laplacian	345
Box 4.9. Differentiation of an integral with respect to a parameter	348
Box 4.10. From the heat equation to the anisotropic diffusion algorithm	348
How do we perform anisotropic diffusion in practice?	349
4.3 Reducing low frequency interference	351
When does low frequency interference arise?	351
Can variable illumination manifest itself in high frequencies?	351
In which other cases may we be interested in reducing low frequencies?	351
What is the ideal high pass filter?	351
How can we enhance small image details using nonlinear filters?	357
What is unsharp masking?	357
How can we apply the unsharp masking algorithm locally?	357
How does the locally adaptive unsharp masking work?	358
How does the retinex algorithm work?	360
Box 4.11. Which are the grey values that are stretched most by the retinex algorithm?	360
How can we improve an image which suffers from variable illumination?	364
What is homomorphic filtering?	364
What is photometric stereo?	366
What does flatfielding mean?	366
How is flatfielding performed?	366
4.4 Histogram manipulation	367
What is the histogram of an image?	367
When is it necessary to modify the histogram of an image?	367
How can we modify the histogram of an image?	367
What is histogram manipulation?	368
What affects the semantic information content of an image?	368
How can we perform histogram manipulation and at the same time preserve the information content of the image?	368
What is histogram equalisation?	370
Why do histogram equalisation programs usually not produce images with flat histograms?	370
How do we perform histogram equalisation in practice?	370
Can we obtain an image with a perfectly flat histogram?	372
What if we do not wish to have an image with a flat histogram?	373
How do we do histogram hyperbolisation in practice?	373
How do we do histogram hyperbolisation with random additions?	374
Why should one wish to perform something other than histogram equalisation?	374
What if the image has inhomogeneous contrast?	375
Can we avoid damaging flat surfaces while increasing the contrast of genuine transitions in brightness?	377

How can we enhance an image by stretching only the grey values that appear in genuine brightness transitions?	377
How do we perform pairwise image enhancement in practice?	378
4.5 Generic deblurring algorithms	383
How does mode filtering help deblur an image?	383
Can we use an edge adaptive window to apply the mode filter?	385
How can mean shift be used as a generic deblurring algorithm?	385
What is toboggan contrast enhancement?	387
How do we do toboggan contrast enhancement in practice?	387
What is the “take home” message of this chapter?	393
5 Image Restoration	395
What is image restoration?	395
Why may an image require restoration?	395
What is image registration?	395
How is image restoration performed?	395
What is the difference between image enhancement and image restoration?	395
5.1 Homogeneous linear image restoration: inverse filtering	396
How do we model homogeneous linear image degradation?	396
How may the problem of image restoration be solved?	396
How may we obtain information on the frequency response function $\hat{H}(u, v)$ of the degradation process?	396
If we know the frequency response function of the degradation process, isn't the solution to the problem of image restoration trivial?	407
What happens at frequencies where the frequency response function is zero?	408
Will the zeros of the frequency response function and the image always coincide?	408
How can we avoid the amplification of noise?	408
How do we apply inverse filtering in practice?	410
Can we define a filter that will automatically take into consideration the noise in the blurred image?	417
5.2 Homogeneous linear image restoration: Wiener filtering	419
How can we express the problem of image restoration as a least square error estimation problem?	419
Can we find a linear least squares error solution to the problem of image restoration?	419
What is the linear least mean square error solution of the image restoration problem?	420
Box 5.1. The least squares error solution	420
Box 5.2. From the Fourier transform of the correlation functions of images to their spectral densities	427
Box 5.3. Derivation of the Wiener filter	428
What is the relationship between Wiener filtering and inverse filtering?	430
How can we determine the spectral density of the noise field?	430
How can we possibly use Wiener filtering, if we know nothing about the statistical properties of the unknown image?	430
How do we apply Wiener filtering in practice?	431

5.3 Homogeneous linear image restoration: Constrained matrix inversion	436
If the degradation process is assumed linear, why don't we solve a system of linear equations to reverse its effect instead of invoking the convolution theorem?	436
Equation (5.146) seems pretty straightforward, why bother with any other approach?	436
Is there any way by which matrix H can be inverted?	437
When is a matrix block circulant?	437
When is a matrix circulant?	438
Why can block circulant matrices be inverted easily?	438
Which are the eigenvalues and eigenvectors of a circulant matrix?	438
How does the knowledge of the eigenvalues and the eigenvectors of a matrix help in inverting the matrix?	439
How do we know that matrix H that expresses the linear degradation process is block circulant?	444
How can we diagonalise a block circulant matrix?	445
Box 5.4. Proof of equation (5.189)	446
Box 5.5. What is the transpose of matrix H ?	448
How can we overcome the extreme sensitivity of matrix inversion to noise?	455
How can we incorporate the constraint in the inversion of the matrix?	456
Box 5.6. Derivation of the constrained matrix inversion filter	459
What is the relationship between the Wiener filter and the constrained matrix inversion filter?	462
How do we apply constrained matrix inversion in practice?	464
5.4 Inhomogeneous linear image restoration: the whirl transform	468
How do we model the degradation of an image if it is linear but inhomogeneous?	468
How may we use constrained matrix inversion when the distortion matrix is not circulant?	477
What happens if matrix H is really very big and we cannot take its inverse?	481
Box 5.7. Jacobi's method for inverting large systems of linear equations	482
Box 5.8. Gauss-Seidel method for inverting large systems of linear equations	485
Does matrix H as constructed in examples 5.41, 5.43, 5.44 and 5.45 fulfil the conditions for using the Gauss-Seidel or the Jacobi method?	485
What happens if matrix H does not satisfy the conditions for the Gauss-Seidel method?	486
How do we apply the gradient descent algorithm in practice?	487
What happens if we do not know matrix H ?	489
5.5 Nonlinear image restoration: MAP estimation	490
What does MAP estimation mean?	490
How do we formulate the problem of image restoration as a MAP estimation?	490
How do we select the most probable configuration of restored pixel values, given the degradation model and the degraded image?	490
Box 5.9. Probabilities: prior, a priori, posterior, a posteriori, conditional	491
Is the minimum of the cost function unique?	491
How can we select then one solution from all possible solutions that minimise the cost function?	493
Can we combine the posterior and the prior probabilities for a configuration x ?	493
Box 5.10. Parseval's theorem	496

How do we model in general the cost function we have to minimise in order to restore an image?	499
What is the reason we use a temperature parameter when we model the joint probability density function, since its does not change the configuration for which the probability takes its maximum?	501
How does the temperature parameter allow us to focus or defocus in the solution space?	501
How do we model the prior probabilities of configurations?	501
What happens if the image has genuine discontinuities?	502
How do we minimise the cost function?	503
How do we create a possible new solution from the previous one?	503
How do we know when to stop the iterations?	505
How do we reduce the temperature in simulated annealing?	506
How do we perform simulated annealing with the Metropolis sampler in practice? .	506
How do we perform simulated annealing with the Gibbs sampler in practice? . .	507
Box 5.11. How can we draw random numbers according to a given probability density function?	508
Why is simulated annealing slow?	511
How can we accelerate simulated annealing?	511
How can we coarsen the configuration space?	512
5.6 Geometric image restoration	513
How may geometric distortion arise?	513
Why do lenses cause distortions?	513
How can a geometrically distorted image be restored?	513
How do we perform the spatial transformation?	513
How may we model the lens distortions?	514
How can we model the inhomogeneous distortion?	515
How can we specify the parameters of the spatial transformation model?	516
Why is grey level interpolation needed?	516
Box 5.12. The Hough transform for line detection	520
What is the “take home” message of this chapter?	526
6 Image Segmentation and Edge Detection	527
What is this chapter about?	527
What exactly is the purpose of image segmentation and edge detection?	527
6.1 Image segmentation	528
How can we divide an image into uniform regions?	528
What do we mean by “labelling” an image?	528
What can we do if the valley in the histogram is not very sharply defined?	528
How can we minimise the number of misclassified pixels?	529
How can we choose the minimum error threshold?	530
What is the minimum error threshold when object and background pixels are normally distributed?	534
What is the meaning of the two solutions of the minimum error threshold equation?	535
How can we estimate the parameters of the Gaussian probability density functions that represent the object and the background?	537

What are the drawbacks of the minimum error threshold method?	541
Is there any method that does not depend on the availability of models for the distributions of the object and the background pixels?	541
Box 6.1. Derivation of Otsu's threshold	542
Are there any drawbacks in Otsu's method?	545
How can we threshold images obtained under variable illumination?	545
If we threshold the image according to the histogram of $\ln f(x, y)$, are we thresholding it according to the reflectance properties of the imaged surfaces?	545
Box 6.2. The probability density function of the sum of two random variables	546
Since straightforward thresholding methods break down under variable illumination, how can we cope with it?	548
What do we do if the histogram has only one peak?	549
Are there any shortcomings of the grey value thresholding methods?	550
How can we cope with images that contain regions that are not uniform but they are <i>perceived</i> as uniform?	551
Can we improve histogramming methods by taking into consideration the spatial proximity of pixels?	553
Are there any segmentation methods that take into consideration the spatial proximity of pixels?	553
How can one choose the seed pixels?	554
How does the split and merge method work?	554
What is morphological image reconstruction?	554
How does morphological image reconstruction allow us to identify the seeds needed for the watershed algorithm?	557
How do we compute the gradient magnitude image?	557
What is the role of the number we subtract from f to create mask g in the morphological reconstruction of f by g ?	558
What is the role of the shape and size of the structuring element in the morphological reconstruction of f by g ?	560
How does the use of the gradient magnitude image help segment the image by the watershed algorithm?	566
Are there any drawbacks in the watershed algorithm which works with the gradient magnitude image?	568
Is it possible to segment an image by filtering?	574
How can we use the mean shift algorithm to segment an image?	574
What is a graph?	576
How can we use a graph to represent an image?	576
How can we use the graph representation of an image to segment it?	576
What is the normalised cuts algorithm?	576
Box 6.3. The normalised cuts algorithm as an eigenvalue problem	576
Box 6.4. How do we minimise the Rayleigh quotient?	585
How do we apply the normalised graph cuts algorithm in practice?	589
Is it possible to segment an image by considering the <i>dissimilarities</i> between regions, as opposed to considering the similarities between pixels?	589
6.2 Edge detection	591
How do we measure the dissimilarity between neighbouring pixels?	591

What is the smallest possible window we can choose?	592
What happens when the image has noise?	593
Box 6.5. How can we choose the weights of a 3×3 mask for edge detection?	595
What is the best value of parameter K ?	596
Box 6.6. Derivation of the Sobel filters	596
In the general case, how do we decide whether a pixel is an edge pixel or not?	601
How do we perform linear edge detection in practice?	602
Are Sobel masks appropriate for all images?	605
How can we choose the weights of the mask if we need a larger mask owing to the presence of significant noise in the image?	606
Can we use the optimal filters for edges to detect lines in an image in an optimal way?	609
What is the fundamental difference between step edges and lines?	609
Box 6.7. Convolving a random noise signal with a filter	615
Box 6.8. Calculation of the signal to noise ratio after convolution of a noisy edge signal with a filter	616
Box 6.9. Derivation of the good locality measure	617
Box 6.10. Derivation of the count of false maxima	619
Can edge detection lead to image segmentation?	620
What is hysteresis edge linking?	621
Does hysteresis edge linking lead to closed edge contours?	621
What is the Laplacian of Gaussian edge detection method?	623
Is it possible to detect edges and lines simultaneously?	623
6.3 Phase congruency and the monogenic signal	625
What is phase congruency?	625
What is phase congruency for a 1D digital signal?	625
How does phase congruency allow us to detect lines and edges?	626
Why does phase congruency coincide with the maximum of the local energy of the signal?	626
How can we measure phase congruency?	627
Couldn't we measure phase congruency by simply averaging the phases of the harmonic components?	627
How do we measure phase congruency in practice?	630
How do we measure the local energy of the signal?	630
Why should we perform convolution with the two basis signals in order to get the projection of the local signal on the basis signals?	632
Box 6.11. Some properties of the continuous Fourier transform	637
If all we need to compute is the local energy of the signal, why don't we use Parseval's theorem to compute it in the real domain inside a local window?	647
How do we decide which filters to use for the calculation of the local energy?	648
How do we compute the local energy of a 1D signal in practice?	651
How can we tell whether the maximum of the local energy corresponds to a symmetric or an antisymmetric feature?	652
How can we compute phase congruency and local energy in 2D?	659
What is the analytic signal?	659
How can we generalise the Hilbert transform to 2D?	660
How do we compute the Riesz transform of an image?	660

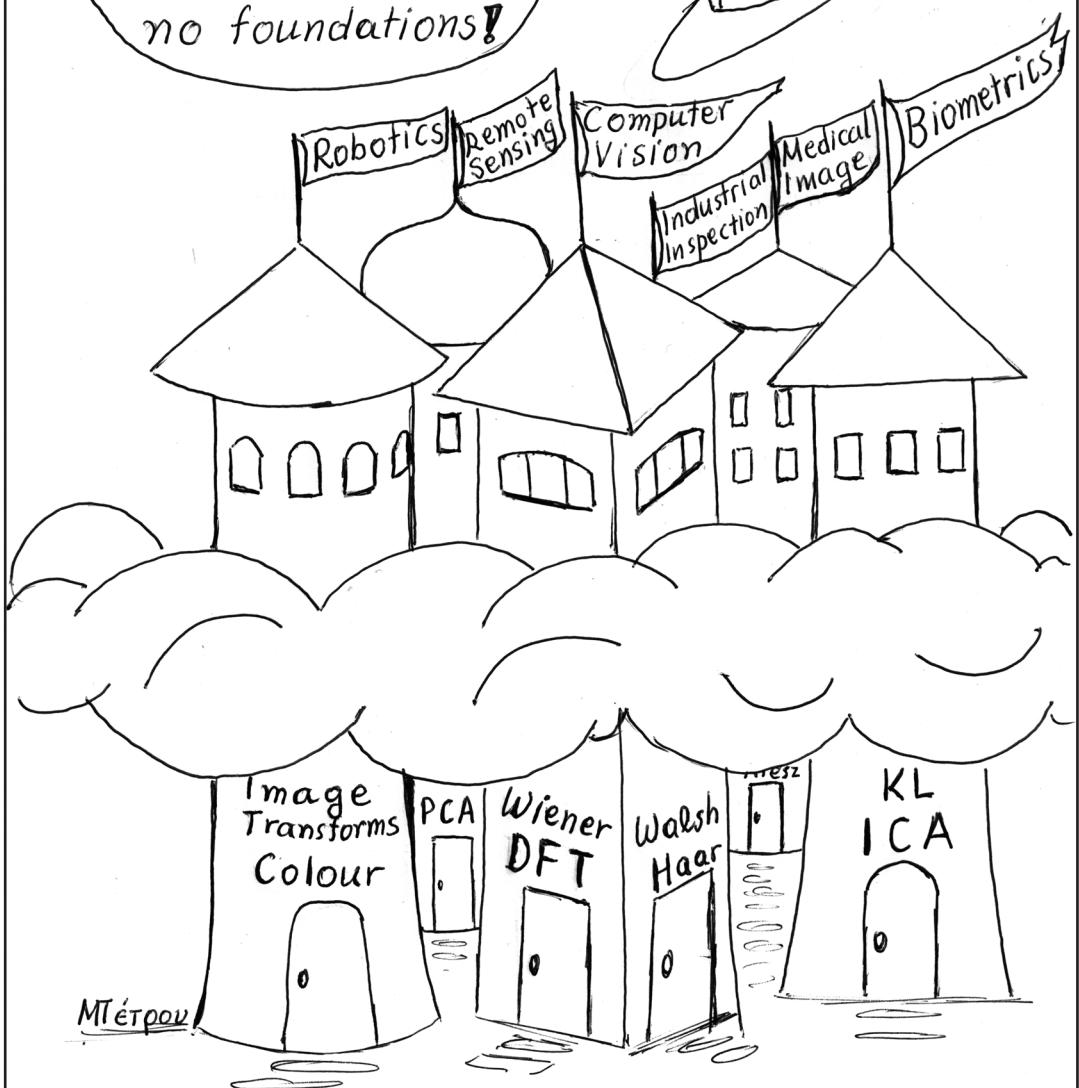
How can the monogenic signal be used?	660
How do we select the even filter we use?	661
What is the “take home” message of this chapter?	668
7 Image Processing for Multispectral Images	669
What is a multispectral image?	669
What are the problems that are special to multispectral images?	669
What is this chapter about?	670
7.1 Image preprocessing for multispectral images	671
Why may one wish to replace the bands of a multispectral image with other bands?	671
How do we usually construct a grey image from a multispectral image?	671
How can we construct a single band from a multispectral image that contains the maximum amount of image information?	671
What is principal component analysis?	672
Box 7.1. How do we measure information?	673
How do we perform principal component analysis in practice?	674
What are the advantages of using the principal components of an image, instead of the original bands?	675
What are the disadvantages of using the principal components of an image instead of the original bands?	675
Is it possible to work out only the first principal component of a multispectral image if we are not interested in the other components?	682
Box 7.2. The power method for estimating the largest eigenvalue of a matrix	682
What is the problem of spectral constancy?	684
What influences the spectral signature of a pixel?	684
What is the reflectance function?	684
Does the imaging geometry influence the spectral signature of a pixel?	684
How does the imaging geometry influence the light energy a pixel receives?	685
How do we model the process of image formation for Lambertian surfaces?	685
How can we eliminate the dependence of the spectrum of a pixel on the imaging geometry?	686
How can we eliminate the dependence of the spectrum of a pixel on the spectrum of the illuminating source?	686
What happens if we have more than one illuminating sources?	687
How can we remove the dependence of the spectral signature of a pixel on the imaging geometry and on the spectrum of the illuminant?	687
What do we have to do if the imaged surface is not made up from the same material?	688
What is the spectral unmixing problem?	688
How do we solve the linear spectral unmixing problem?	689
Can we use library spectra for the pure materials?	689
How do we solve the linear spectral unmixing problem when we know the spectra of the pure components?	690
Is it possible that the inverse of matrix Q cannot be computed?	693
What happens if the library spectra have been sampled at different wavelengths from the mixed spectrum?	693

What happens if we do not know which pure substances might be present in the mixed substance?	694
How do we solve the linear spectral unmixing problem if we do not know the spectra of the pure materials?	695
7.2 The physics and psychophysics of colour vision	700
What is colour?	700
What is the interest in colour from the engineering point of view?	700
What influences the colour we perceive for a dark object?	700
What causes the variations of the daylight?	701
How can we model the variations of the daylight?	702
Box 7.3. Standard illuminants	704
What is the observed variation in the natural materials?	706
What happens to the light once it reaches the sensors?	711
Is it possible for different materials to produce the same recording by a sensor?	713
How does the human visual system achieve colour constancy?	714
What does the trichromatic theory of colour vision say?	715
What defines a colour system?	715
How are the tristimulus values specified?	715
Can all monochromatic reference stimuli be matched by simply adjusting the intensities of the primary lights?	715
Do all people require the same intensities of the primary lights to match the same monochromatic reference stimulus?	717
Who are the people with normal colour vision?	717
What are the most commonly used colour systems?	717
What is the <i>CIE RGB</i> colour system?	717
What is the <i>XYZ</i> colour system?	718
How do we represent colours in 3D?	718
How do we represent colours in 2D?	718
What is the chromaticity diagram?	719
Box 7.4. Some useful theorems from 3D geometry	721
What is the chromaticity diagram for the <i>CIE RGB</i> colour system?	724
How does the human brain perceive colour brightness?	725
How is the alychne defined in the <i>CIE RGB</i> colour system?	726
How is the <i>XYZ</i> colour system defined?	726
What is the chromaticity diagram of the <i>XYZ</i> colour system?	728
How is it possible to create a colour system with imaginary primaries, in practice?	729
What if we wish to model the way a particular individual sees colours?	729
If different viewers require different intensities of the primary lights to see white, how do we calibrate colours between different viewers?	730
How do we make use of the reference white?	730
How is the <i>sRGB</i> colour system defined?	732
Does a colour change if we double all its tristimulus values?	733
How does the description of a colour, in terms of a colour system, relate to the way we describe colours in everyday language?	733
How do we compare colours?	733
What is a metric?	733
Can we use the Euclidean metric to measure the difference of two colours?	734

Which are the perceptually uniform colour spaces?	734
How is the <i>Luv</i> colour space defined?	734
How is the <i>Lab</i> colour space defined?	735
How do we choose values for (X_n, Y_n, Z_n) ?	735
How can we compute the <i>RGB</i> values from the <i>Luv</i> values?	735
How can we compute the <i>RGB</i> values from the <i>Lab</i> values?	736
How do we measure perceived saturation?	737
How do we measure perceived differences in saturation?	737
How do we measure perceived hue?	737
How is the perceived hue angle defined?	738
How do we measure perceived differences in hue?	738
What affects the way we perceive colour?	740
What is meant by temporal context of colour?	740
What is meant by spatial context of colour?	740
Why distance matters when we talk about spatial frequency?	741
How do we explain the spatial dependence of colour perception?	741
7.3 Colour image processing in practice	742
How does the study of the human colour vision affect the way we do image processing?	742
How perceptually uniform are the perceptually uniform colour spaces in practice?	742
How should we convert the image <i>RGB</i> values to the <i>Luv</i> or the <i>Lab</i> colour spaces?	742
How do we measure hue and saturation in image processing applications?	747
How can we emulate the spatial dependence of colour perception in image processing?	752
What is the relevance of the phenomenon of metamerism to image processing?	756
How do we cope with the problem of metamerism in an industrial inspection application?	756
What is a Monte-Carlo method?	757
How do we remove noise from multispectral images?	759
How do we rank vectors?	760
How do we deal with mixed noise in multispectral images?	760
How do we enhance a colour image?	761
How do we restore multispectral images?	767
How do we compress colour images?	767
How do we segment multispectral images?	767
How do we apply k -means clustering in practice?	767
How do we extract the edges of multispectral images?	769
What is the “take home” message of this chapter?	770
Bibliographical notes	775
References	777
Index	781

Look!

The Earthlings have
found a way to
build towers with
no foundations!



The little green men from outer space!

Preface

Since the first edition of this book in 1999, the field of Image Processing has seen many developments. First of all, the proliferation of colour sensors caused an explosion of research in colour vision and colour image processing. Second, application of image processing to biomedicine has really taken off, with medical image processing nowadays being almost a field of its own. Third, image processing has become more sophisticated, having reached out even further afield, into other areas of research, as diverse as graph theory and psychophysics, to borrow methodologies and approaches.

This new edition of the book attempts to capture these new insights, without, however, forgetting the well known and established methods of image processing of the past. The book may be treated as three books interlaced: the advanced proofs and peripheral material are presented in grey boxes; they may be omitted in a first reading or for an undergraduate course. The back bone of the book is the text given in the form of questions and answers. We believe that the order of the questions is that of coming naturally to the reader when they encounter a new concept. There are 255 figures and 384 fully worked out examples aimed at clarifying these concepts. **Examples with a number prefixed with a “B” refer to the boxed material and again they may be omitted in a first reading or an undergraduate course.** The book is accompanied by a CD with all the MatLab programs that produced the examples and the figures. There is also a collection of slide presentations in pdf format, available from the accompanying web page of the book, that may help the lecturer who wishes to use this material for teaching.

We have made a great effort to make the book easy to read and we hope that learning about the “nuts and bolts” behind the image processing algorithms will make the subject even more exciting and a pleasure to delve into.

Over the years of writing this book, we were helped by various people. We would particularly like to thank Mike Brookes, Nikos Mitianoudis, Antonis Katartzis, Mohammad Jahangiri, Tania Stathaki and Vladimir Jeliazkov, for useful discussions, Mohammad Jahangiri, Leila Favaedi and Olga Duran for help with some figures, and Pedro Garcia-Sevilla for help with typesetting the book.

Maria Petrou and Costas Petrou

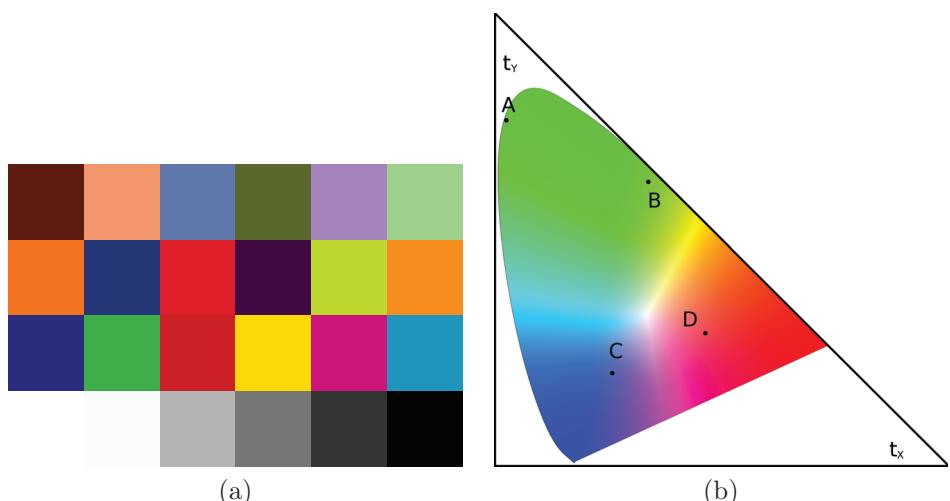


Plate I: (a) The colours of the Macbeth colour chart. (b) The chromaticity diagram of the XYZ colour system. Points A and B represent colours which, although further apart than points C and D , are perceived as more similar than the colours represented by C and D .

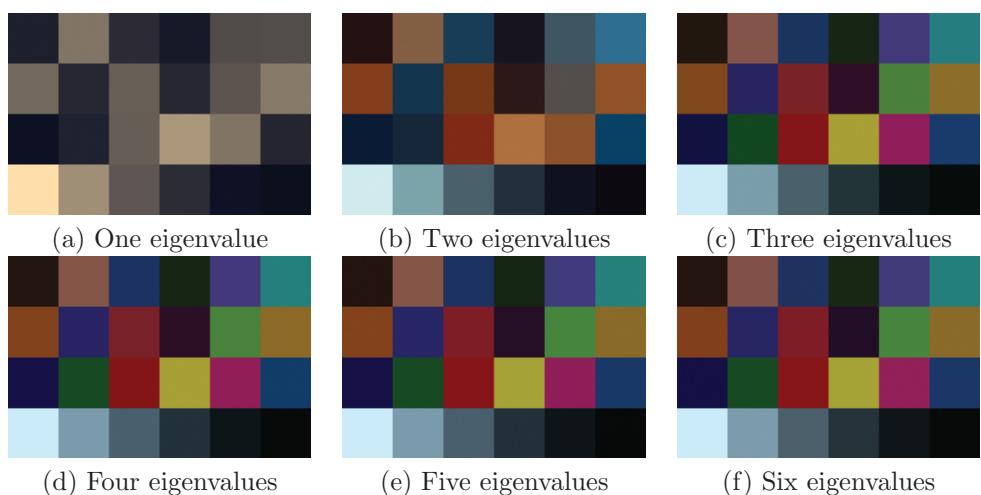


Plate II: The inclusion of extra eigenvalues beyond the third one changes the colour appearance very little (see example 7.12, on page 713).

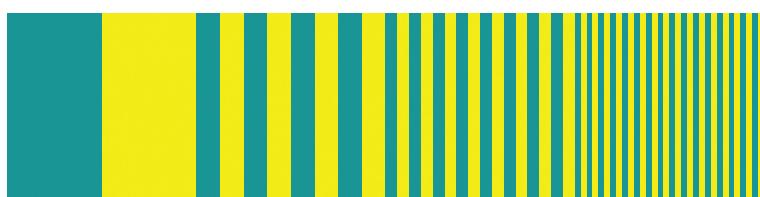




Plate IV: Colour perception depends on colour context (see page 740).



(a) 5% impulse noise



(b) 5% impulse + Gaussian ($\sigma = 15$)



(c) Vector median filtering



(d) α -trimmed vector median filtering

Plate V: At the top, images affected by impulse noise and mixed noise, and at the bottom their restored versions, using vector median filtering, with window size 3×3 , and α -trimmed vector median filtering, with $\alpha = 0.2$ and window size 5×5 (example 7.32, page 761).



Plate VI: (a) “A Street in Shanghai” (344×512). As seen from (b) 2m, (c) 4m and (d) 10m distance. In (b) a border of 10 pixels around should be ignored, in (c) the stripe affected by border effects is 22 pixels wide, while in (d) is 34 pixels wide (example 7.28, page 754).



Plate VII: Enhancing colours by increasing their saturation to its maximum, while retaining their hue. Threshold= 0.04 and $\gamma = 1/\sqrt{6}$ were used for the saturation (see page 761).



(a) "Vina del Mar-Valparaiso"

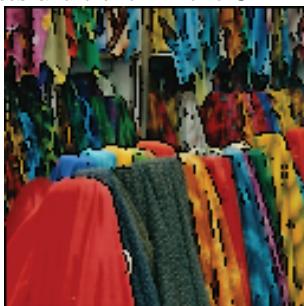


(b) After colour enhancement

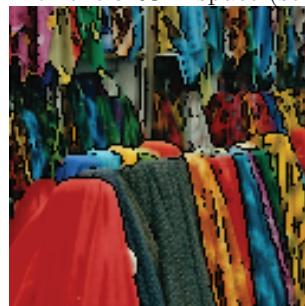
Plate VIII: Enhancing colours by increasing their saturation to the maximum, while retaining their hue. Threshold= 0.01 and $\gamma = 1/\sqrt{6}$ were used for the saturation (see page 761).

(a) Original (184×256)(b) 10-means ($sRGB$)(c) Mean shift ($sRGB$)(d) Mean shift ($CIE RGB$)

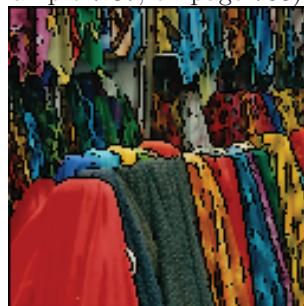
Plate IX: "The Merchant in Al-Ain" segmented in Luv space, assuming that the original values are either in the $CIE RGB$ or the $sRGB$ space (see example 7.37, on page 768).



(a) From the average band



(b) From the 1st PC



(c) From all bands

Plate X: The edges superimposed on the original image (see example 7.38, on page 769).

Chapter 1

Introduction

Why do we process images?

Image Processing has been developed in response to three major problems concerned with pictures:

- picture digitisation and coding to facilitate transmission, printing and storage of pictures;
- picture enhancement and restoration in order, for example, to interpret more easily pictures of the surface of other planets taken by various probes;
- picture segmentation and description as an early stage to Machine Vision.

Image Processing nowadays refers mainly to the processing of digital images.

What is an image?

A **panchromatic image** is a 2D light intensity function, $f(x, y)$, where x and y are spatial coordinates and the value of f at (x, y) is proportional to the brightness of the scene at that point. If we have a **multispectral image**, $f(x, y)$ is a vector, each component of which indicates the brightness of the scene at point (x, y) at the corresponding spectral **band**.

What is a digital image?

A **digital image** is an image $f(x, y)$ that has been discretised both in spatial coordinates and in brightness. It is represented by a 2D integer array, or a series of 2D arrays, one for each colour band. The digitised brightness value is called **grey level**.

Each element of the array is called **pixel** or **pel**, derived from the term “picture element”. Usually, the size of such an array is a few hundred pixels by a few hundred pixels and there are several dozens of possible different grey levels. Thus, a digital image looks like this

$$f(x, y) = \begin{bmatrix} f(1, 1) & f(1, 2) & \dots & f(1, N) \\ f(2, 1) & f(2, 2) & \dots & f(2, N) \\ \vdots & \vdots & & \vdots \\ f(N, 1) & f(N, 2) & \dots & f(N, N) \end{bmatrix} \quad (1.1)$$

with $0 \leq f(x, y) \leq G - 1$, where usually N and G are expressed as positive integer powers of 2 ($N = 2^n$, $G = 2^m$).

What is a spectral band?

A colour band is a range of wavelengths of the electromagnetic spectrum, over which the sensors we use to capture an image have nonzero sensitivity. Typical colour images consist of three colour bands. This means that they have been captured by three different sets of sensors, each set made to have a different sensitivity function. Figure 1.1 shows typical sensitivity curves of a multispectral camera.

All the methods presented in this book, apart from those in Chapter 7, will refer to single band images.

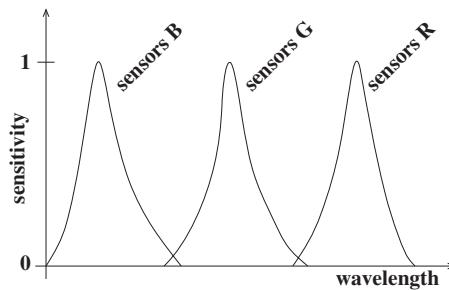


Figure 1.1: The spectrum of the light which reaches a sensor is multiplied with the sensitivity function of the sensor and recorded by the sensor. This recorded value is the brightness of the image in the location of the sensor and in the band of the sensor. This figure shows the sensitivity curves of three different sensor types.

Why do most image processing algorithms refer to grey images, while most images we come across are colour images?

For various reasons.

1. A lot of the processes we apply to a grey image can be easily extended to a colour image by applying them to each band separately.
2. A lot of the information conveyed by an image is expressed in its grey form and so colour is not necessary for its extraction. That is the reason black and white television receivers had been perfectly acceptable to the public for many years and black and white photography is still popular with many photographers.
3. For many years colour digital cameras were expensive and not widely available. A lot of image processing techniques were developed around the type of image that was available. These techniques have been well established in image processing.

Nevertheless, colour is an important property of the natural world, and so we shall examine its role in image processing in a separate chapter in this book.

How is a digital image formed?

Each pixel of an image corresponds to a part of a physical object in the 3D world. This physical object is illuminated by some light which is partly reflected and partly absorbed by it. Part of the reflected light reaches the array of sensors used to image the scene and is responsible for the values recorded by these sensors. One of these sensors makes up a pixel and its field of view corresponds to a small patch in the imaged scene. The recorded value by each sensor depends on its sensitivity curve. When a photon of a certain wavelength reaches the sensor, its energy is multiplied with the value of the sensitivity curve of the sensor at that wavelength and is accumulated. The total energy collected by the sensor (during the exposure time) is eventually used to compute the grey value of the pixel that corresponds to this sensor.

If a sensor corresponds to a patch in the physical world, how come we can have more than one sensor type corresponding to the same patch of the scene?

Indeed, it is not possible to have three different sensors with three different sensitivity curves corresponding exactly to the same patch of the physical world. That is why digital cameras have the three different types of sensor slightly displaced from each other, as shown in figure 1.2, with the sensors that are sensitive to the green wavelengths being twice as many as those sensitive to the blue and the red wavelengths. The recordings of the three sensor types are interpolated and superimposed to create the colour image. Recently, however, cameras have been constructed where the three types of sensor are combined so they exactly exist on top of each other and so they view exactly the same patch in the real world. These cameras produce much sharper colour images than the ordinary cameras.

R	G	R	G	R	G
G	B	G	B	G	B
R	G	R	G	R	G
G	B	G	B	G	B
R	G	R	G	R	G
G	B	G	B	G	B

Figure 1.2: The *RGB* sensors as they are arranged in a typical digital camera.

What is the physical meaning of the brightness of an image at a pixel position?

The brightness values of different pixels have been created by using the energies recorded by the corresponding sensors. They have significance only *relative* to each other and they are meaningless in absolute terms. So, pixel values between different images should only be compared if either care has been taken for the physical processes used to form the two images to be identical, or the brightness values of the two images have somehow been normalised so that the effects of the different physical processes have been removed. In that case, we say that the sensors are **calibrated**.

Example 1.1

You are given a triple array of 3×3 sensors, arranged as shown in figure 1.3, with their sampled sensitivity curves given in table 1.1. The last column of this table gives in some arbitrary units the energy, E_{λ_i} , carried by photons with wavelength λ_i .

Wavelength	Sensors B	Sensors G	Sensors R	Energy
λ_0	0.2	0.0	0.0	1.00
λ_1	0.4	0.2	0.1	0.95
λ_2	0.8	0.3	0.2	0.90
λ_3	1.0	0.4	0.2	0.88
λ_4	0.7	0.6	0.3	0.85
λ_5	0.2	1.0	0.5	0.81
λ_6	0.1	0.8	0.6	0.78
λ_7	0.0	0.6	0.8	0.70
λ_8	0.0	0.3	1.0	0.60
λ_9	0.0	0.0	0.6	0.50

Table 1.1: The sensitivity curves of three types of sensor for wavelengths in the range $[\lambda_0, \lambda_9]$, and the corresponding energy of photons of these particular wavelengths in some arbitrary units.

The shutter of the camera is open long enough for 10 photons to reach the locations of the sensors.

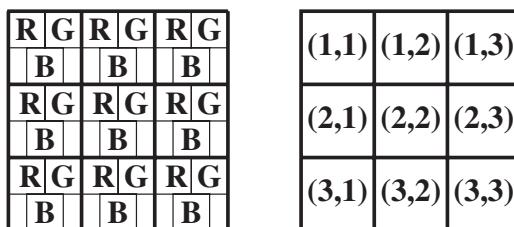


Figure 1.3: Three 3×3 sensor arrays interlaced. On the right, the locations of the pixels they make up. Although the three types of sensor are slightly misplaced with respect to each other, we assume that each triplet is coincident and forms a single pixel.

For simplicity, we consider that exactly the same types of photon reach all sensors that correspond to the same location.

The wavelengths of the photons that reach the pixel locations of each triple sensor, as identified in figure 1.3, are:

- Location (1, 1):** $\lambda_0, \lambda_9, \lambda_9, \lambda_8, \lambda_7, \lambda_8, \lambda_1, \lambda_0, \lambda_1, \lambda_1$
- Location (1, 2):** $\lambda_1, \lambda_3, \lambda_3, \lambda_4, \lambda_4, \lambda_5, \lambda_2, \lambda_6, \lambda_4, \lambda_5$
- Location (1, 3):** $\lambda_6, \lambda_7, \lambda_7, \lambda_0, \lambda_5, \lambda_6, \lambda_6, \lambda_1, \lambda_5, \lambda_9$
- Location (2, 1):** $\lambda_0, \lambda_1, \lambda_0, \lambda_2, \lambda_1, \lambda_1, \lambda_4, \lambda_3, \lambda_3, \lambda_1$
- Location (2, 2):** $\lambda_3, \lambda_3, \lambda_4, \lambda_3, \lambda_4, \lambda_4, \lambda_5, \lambda_2, \lambda_9, \lambda_4$
- Location (2, 3):** $\lambda_7, \lambda_7, \lambda_6, \lambda_7, \lambda_6, \lambda_1, \lambda_5, \lambda_9, \lambda_8, \lambda_7$
- Location (3, 1):** $\lambda_6, \lambda_6, \lambda_1, \lambda_8, \lambda_7, \lambda_8, \lambda_9, \lambda_9, \lambda_8, \lambda_7$
- Location (3, 2):** $\lambda_0, \lambda_4, \lambda_3, \lambda_4, \lambda_1, \lambda_5, \lambda_4, \lambda_0, \lambda_2, \lambda_1$
- Location (3, 3):** $\lambda_3, \lambda_4, \lambda_1, \lambda_0, \lambda_0, \lambda_4, \lambda_2, \lambda_5, \lambda_2, \lambda_4$

Calculate the values that each sensor array will record and thus produce the three *photon energy bands* recorded.

We denote by $g_X(i, j)$ the value that will be recorded by sensor type X at location (i, j) . For sensors R , G and B in location $(1, 1)$, the recorded values will be:

$$\begin{aligned} g_R(1, 1) &= 2E_{\lambda_0} \times 0.0 + 2E_{\lambda_9} \times 0.6 + 2E_{\lambda_8} \times 1.0 + 1E_{\lambda_7} \times 0.8 + 3E_{\lambda_1} \times 0.1 \\ &= 1.0 \times 0.6 + 1.2 \times 1.0 + 0.7 \times 0.8 + 2.85 \times 0.1 \\ &= 2.645 \end{aligned} \tag{1.2}$$

$$\begin{aligned} g_G(1, 1) &= 2E_{\lambda_0} \times 0.0 + 2E_{\lambda_9} \times 0.0 + 2E_{\lambda_8} \times 0.3 + 1E_{\lambda_7} \times 0.6 + 3E_{\lambda_1} \times 0.2 \\ &= 1.2 \times 0.3 + 0.7 \times 0.6 + 2.85 \times 0.2 \\ &= 1.35 \end{aligned} \tag{1.3}$$

$$\begin{aligned} g_B(1, 1) &= 2E_{\lambda_0} \times 0.2 + 2E_{\lambda_9} \times 0.0 + 2E_{\lambda_8} \times 0.0 + 1E_{\lambda_7} \times 0.0 + 3E_{\lambda_1} \times 0.4 \\ &= 2.0 \times 0.2 + 2.85 \times 0.4 \\ &= 1.54 \end{aligned} \tag{1.4}$$

Working in a similar way, we deduce that the energies recorded by the three sensor arrays are:

$$\begin{aligned} E_R &= \begin{pmatrix} 2.645 & 2.670 & 3.729 \\ 1.167 & 4.053 & 4.576 \\ 4.551 & 1.716 & 1.801 \end{pmatrix} \\ E_G &= \begin{pmatrix} 1.350 & 4.938 & 4.522 \\ 2.244 & 4.176 & 4.108 \\ 2.818 & 2.532 & 2.612 \end{pmatrix} \\ E_B &= \begin{pmatrix} 1.540 & 5.047 & 1.138 \\ 4.995 & 5.902 & 0.698 \\ 0.536 & 4.707 & 5.047 \end{pmatrix} \end{aligned} \tag{1.5}$$

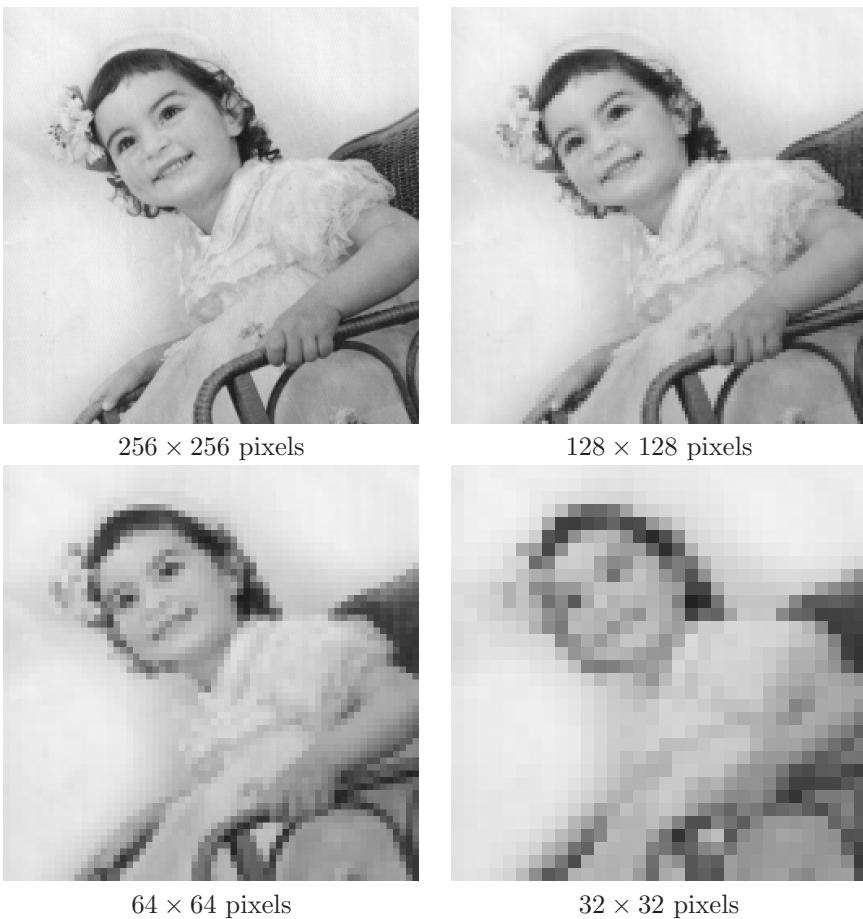


Figure 1.4: Keeping the number of grey levels constant and decreasing the number of pixels with which we digitise the same field of view produces the checkerboard effect.

Why are images often quoted as being 512×512 , 256×256 , 128×128 etc?

Many calculations with images are simplified when the size of the image is a power of 2. We shall see some examples in Chapter 2.

How many bits do we need to store an image?

The number of bits, b , we need to store an image of size $N \times N$, with 2^m grey levels, is:

$$b = N \times N \times m \quad (1.6)$$

So, for a typical 512×512 image with 256 grey levels ($m = 8$) we need 2,097,152 bits or 262,144 8-bit bytes. That is why we often try to reduce m and N , without significant loss of image quality.

What determines the quality of an image?

The quality of an image is a complicated concept, largely subjective and very much application dependent. Basically, an image is of good quality if it is not noisy and

- (1) it is not blurred;
- (2) it has high resolution;
- (3) it has good contrast.

What makes an image blurred?

Image blurring is caused by incorrect image capturing conditions. For example, out of focus camera, or relative motion of the camera and the imaged object. The amount of image blurring is expressed by the so called **point spread function** of the imaging system.

What is meant by image resolution?

The resolution of an image expresses how much detail we can see in it and clearly depends on the number of pixels we use to represent a scene (parameter N in equation (1.6)) and the number of grey levels used to quantise the brightness values (parameter m in equation (1.6)).

Keeping m constant and decreasing N results in the **checkerboard effect** (figure 1.4). Keeping N constant and reducing m results in **false contouring** (figure 1.5). Experiments have shown that the more detailed a picture is, the less it improves by keeping N constant and increasing m . So, for a detailed picture, like a picture of crowds (figure 1.6), the number of grey levels we use does not matter much.

Example 1.2

Assume that the range of values recorded by the sensors of example 1.1 is from 0 to 10. From the values of the three bands captured by the three sets of sensors, create digital bands with 3 bits each ($m = 3$).

For 3-bit images, the pixels take values in the range $[0, 2^3 - 1]$, ie in the range $[0, 7]$. Therefore, we have to divide the expected range of values to 8 equal intervals: $10/8 = 1.25$. So, we use the following conversion table:

All pixels with recorded value in the range $[0.0, 1.25)$	get grey value 0
All pixels with recorded value in the range $[1.25, 2.5)$	get grey value 1
All pixels with recorded value in the range $[2.5, 3.75)$	get grey value 2
All pixels with recorded value in the range $[3.75, 5.0)$	get grey value 3
All pixels with recorded value in the range $[5.0, 6.25)$	get grey value 4
All pixels with recorded value in the range $[6.25, 7.5)$	get grey value 5
All pixels with recorded value in the range $[7.5, 8.75)$	get grey value 6
All pixels with recorded value in the range $[8.75, 10.0]$	get grey value 7

This mapping leads to the following bands of the recorded image.

$$R = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 3 & 3 \\ 3 & 1 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 1 & 3 & 3 \\ 1 & 3 & 3 \\ 2 & 2 & 2 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 4 & 0 \\ 3 & 4 & 0 \\ 0 & 3 & 4 \end{pmatrix} \quad (1.7)$$



Figure 1.5: Keeping the size of the image constant (249×199) and reducing the number of grey levels ($= 2^m$) produces false contouring. To display the images, we always map the different grey values to the range $[0, 255]$.

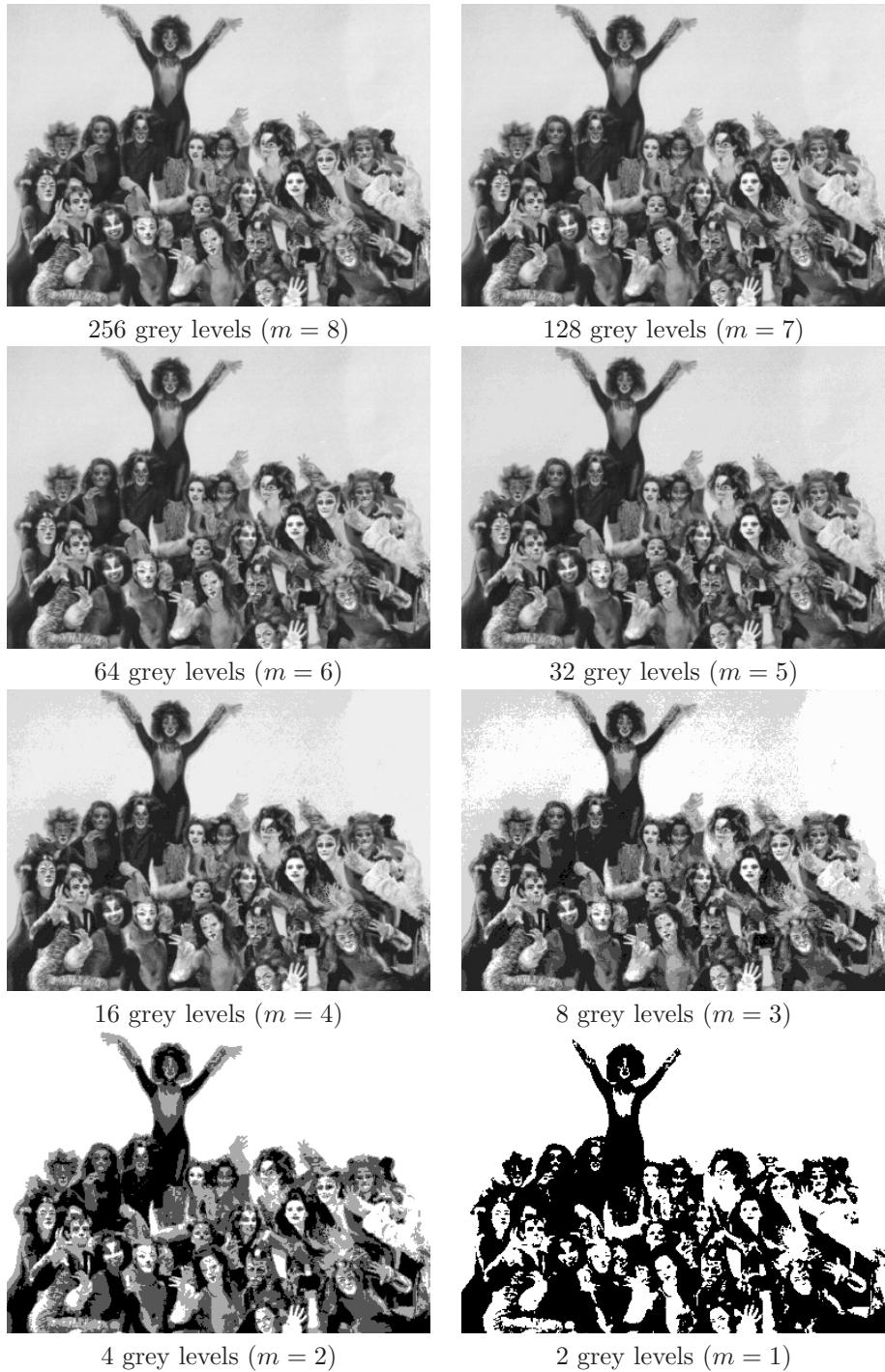


Figure 1.6: Keeping the number of pixels constant and reducing the number of grey levels does not affect much the appearance of an image that contains a lot of details.

What does “good contrast” mean?

Good contrast means that the grey values present in the image range from black to white, making use of the full range of brightness to which the human vision system is sensitive.

Example 1.3

Consider each band created in example 1.2 as a separate grey image. Do these images have good contrast? If not, propose some way by which bands with good contrast could be created from the recorded sensor values.

The images created in example 1.2 do not have good contrast because none of them contains the value 7 (which corresponds to the maximum brightness and which would be displayed as white by an image displaying device).

The reason for this is the way the quantisation was performed: the look up table created to convert the real recorded values to digital values took into consideration the full range of possible values a sensor may record (ie from 0 to 10). To utilise the full range of grey values for each image, we should have considered the minimum and the maximum value of its pixels, and map that range to the 8 distinct grey levels. For example, for the image that corresponds to band R, the values are in the range [1.167, 4.576]. If we divide this range in 8 equal sub-ranges, we shall have:

All pixels with recorded value in the range [0.536, 1.20675)	get grey value 0
All pixels with recorded value in the range [1.20675, 1.8775)	get grey value 1
All pixels with recorded value in the range [1.8775, 2.54825)	get grey value 2
All pixels with recorded value in the range [2.54825, 3.219)	get grey value 3
All pixels with recorded value in the range [3.219, 3.88975)	get grey value 4
All pixels with recorded value in the range [3.88975, 4.5605)	get grey value 5
All pixels with recorded value in the range [4.5605, 5.23125)	get grey value 6
All pixels with recorded value in the range [5.23125, 5.902)	get grey value 7

We must create one such look up table for each band. The grey images we create this way are:

$$R = \begin{pmatrix} 3 & 3 & 6 \\ 0 & 6 & 7 \\ 7 & 1 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 0 & 7 & 7 \\ 1 & 6 & 6 \\ 3 & 2 & 2 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 6 & 0 \\ 6 & 7 & 0 \\ 0 & 6 & 6 \end{pmatrix} \quad (1.8)$$

Example 1.4

Repeat example 1.3, now treating the three bands as parts of the same colour image.

If we treat all three bands as a single colour image (as they are meant to be), we must

find the minimum and maximum value over all three recorded bands, and create a look up table appropriate for all bands. In this case, the range of the recorded values is [0.536, 5.902]. The look up table we create by mapping this range to the range [0, 7] is

All pixels with recorded value in the range [0.536, 1.20675)	get grey value 0
All pixels with recorded value in the range [1.20675, 1.8775)	get grey value 1
All pixels with recorded value in the range [1.8775, 2.54825)	get grey value 2
All pixels with recorded value in the range [2.54825, 3.219)	get grey value 3
All pixels with recorded value in the range [3.219, 3.88975)	get grey value 4
All pixels with recorded value in the range [3.88975, 4.5605)	get grey value 5
All pixels with recorded value in the range [4.5605, 5.23125)	get grey value 6
All pixels with recorded value in the range [5.23125, 5.902]	get grey value 7

The three image bands we create this way are:

$$R = \begin{pmatrix} 3 & 3 & 4 \\ 1 & 5 & 6 \\ 5 & 1 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 1 & 6 & 5 \\ 2 & 5 & 5 \\ 3 & 2 & 3 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 6 & 0 \\ 6 & 7 & 0 \\ 0 & 6 & 6 \end{pmatrix} \quad (1.9)$$

Note that each of these bands if displayed as a separate grey image may not display the full range of grey values, but it will have grey values that are consistent with those of the other three bands, and so they can be directly compared. For example, by looking at the three digital bands we can say that pixel (1,1) has the same brightness (within the limits of the digitisation error) in bands G or B. Such a statement was not possible in example 1.3 because the three digital bands were not calibrated.

What is the purpose of image processing?

Image processing has multiple purposes.

- To improve the quality of an image in a subjective way, usually by increasing its contrast. This is called **image enhancement**.
- To use as few bits as possible to represent the image, with minimum deterioration in its quality. This is called **image compression**.
- To improve an image in an objective way, for example by reducing its blurring. This is called **image restoration**.
- To make explicit certain characteristics of the image which can be used to identify the contents of the image. This is called **feature extraction**.

How do we do image processing?

We perform image processing by using image transformations. Image transformations are performed using **operators**. An operator takes as input an image and produces another image. In this book we shall put emphasis on a particular class of operators, called **linear operators**.

Do we use nonlinear operators in image processing?

Yes. We shall see several examples of them in this book. However, nonlinear operators cannot be collectively characterised. They are usually problem- and application-specific, and they are studied as individual processes used for specific tasks. On the contrary, linear operators can be studied collectively, because they share important common characteristics, irrespective of the task they are expected to perform.

What is a linear operator?

Consider \mathcal{O} to be an operator which takes images into images. If f is an image, $\mathcal{O}(f)$ is the result of applying \mathcal{O} to f . \mathcal{O} is linear if

$$\mathcal{O}[af + bg] = a\mathcal{O}[f] + b\mathcal{O}[g] \quad (1.10)$$

for all images f and g and all scalars a and b .

How are linear operators defined?

Linear operators are defined in terms of their **point spread functions**. The point spread function of an operator is what we get out if we apply the operator on a point source:

$$\mathcal{O}[\text{point source}] \equiv \text{point spread function} \quad (1.11)$$

Or

$$\mathcal{O}[\delta(\alpha - x, \beta - y)] \equiv h(x, \alpha, y, \beta) \quad (1.12)$$

where $\delta(\alpha - x, \beta - y)$ is a point source of brightness 1 centred at point (x, y) .

What is the relationship between the point spread function of an imaging device and that of a linear operator?

They both express the effect of either the imaging device or the operator on a point source. In the real world a star is the nearest to a point source. Assume that we capture the image of a star by using a camera. The star will appear in the image like a blob: the camera received the light of the point source and spread it into a blob. The bigger the blob, the more blurred the image of the star will look. So, the point spread function of the camera measures the amount of blurring present in the images captured by this camera. The camera, therefore, acts like a linear operator which accepts as input the ideal brightness function of the continuous real world and produces the recorded digital image. That is why we use the term “point spread function” to characterise both cameras and linear operators.

How does a linear operator transform an image?

If the operator is *linear*, when the point source is a times brighter, the result will be a times higher:

$$\mathcal{O}[a\delta(\alpha - x, \beta - y)] = ah(x, \alpha, y, \beta) \quad (1.13)$$

An image is a collection of point sources (the *pixels*) each with its own brightness value. For example, assuming that an image f is 3×3 in size, we may write:

$$f = \begin{pmatrix} f(1,1) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & f(1,2) & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \cdots + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & f(3,3) \end{pmatrix} \quad (1.14)$$

We may say that an image is the sum of these point sources. Then the effect of an operator characterised by point spread function $h(x, \alpha, y, \beta)$ on an image $f(x, y)$ can be written as:

$$g(\alpha, \beta) = \sum_{x=1}^N \sum_{y=1}^N f(x, y) h(x, \alpha, y, \beta) \quad (1.15)$$

where $g(\alpha, \beta)$ is the output “image”, $f(x, y)$ is the input image and the size of the image is $N \times N$. Here we treat $f(x, y)$ as the brightness of a point source located at position (x, y) . Applying an operator on it produces the point spread function of the operator times the strength of the source, ie times the grey value $f(x, y)$ at that location. Then, as the operator is linear, we sum over all such point sources, ie we sum over all pixels.

What is the meaning of the point spread function?

The point spread function $h(x, \alpha, y, \beta)$ expresses how much the input value at position (x, y) influences the output value at position (α, β) . If the influence expressed by the point spread function is independent of the actual positions but depends only on the *relative* position of the influencing and the influenced pixels, we have a **shift invariant** point spread function:

$$h(x, \alpha, y, \beta) = h(\alpha - x, \beta - y) \quad (1.16)$$

Then equation (1.15) is a *convolution*:

$$g(\alpha, \beta) = \sum_{x=1}^N \sum_{y=1}^N f(x, y) h(\alpha - x, \beta - y) \quad (1.17)$$

If the columns are influenced independently from the rows of the image, then the point spread function is **separable**:

$$h(x, \alpha, y, \beta) \equiv h_c(x, \alpha) h_r(y, \beta) \quad (1.18)$$

The above expression serves also as the definition of functions $h_c(x, \alpha)$ and $h_r(y, \beta)$. Then equation (1.15) may be written as a cascade of two 1D transformations:

$$g(\alpha, \beta) = \sum_{x=1}^N h_c(x, \alpha) \sum_{y=1}^N f(x, y) h_r(y, \beta) \quad (1.19)$$

If the point spread function is both shift invariant and separable, then equation (1.15) may be written as a cascade of two 1D convolutions:

$$g(\alpha, \beta) = \sum_{x=1}^N h_c(\alpha - x) \sum_{y=1}^N f(x, y) h_r(\beta - y) \quad (1.20)$$

Box 1.1. The formal definition of a point source in the continuous domain

Let us define an extended source of constant brightness

$$\delta_n(x, y) \equiv n^2 \text{rect}(nx, ny) \quad (1.21)$$

where n is a positive constant and

$$\text{rect}(nx, ny) \equiv \begin{cases} 1 & \text{inside a rectangle } |nx| \leq \frac{1}{2}, |ny| \leq \frac{1}{2} \\ 0 & \text{elsewhere} \end{cases} \quad (1.22)$$

The total brightness of this source is given by

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \delta_n(x, y) dx dy = n^2 \underbrace{\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \text{rect}(nx, ny) dx dy}_{\text{area of rectangle}} = 1 \quad (1.23)$$

and is independent of n .

As $n \rightarrow +\infty$, we create a sequence, δ_n , of extended square sources which gradually shrink with their brightness remaining constant. At the limit, δ_n becomes Dirac's delta function

$$\delta(x, y) \begin{cases} \neq 0 & \text{for } x = y = 0 \\ = 0 & \text{elsewhere} \end{cases} \quad (1.24)$$

with the property:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \delta(x, y) dx dy = 1 \quad (1.25)$$

Integral

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \delta_n(x, y) g(x, y) dx dy \quad (1.26)$$

is the average of image $g(x, y)$ over a square with sides $\frac{1}{n}$ centred at $(0, 0)$. At the limit, we have

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \delta(x, y) g(x, y) dx dy = g(0, 0) \quad (1.27)$$

which is the value of the image at the origin. Similarly,

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(x, y) \delta_n(x - a, y - b) dx dy \quad (1.28)$$

is the average value of g over a square $\frac{1}{n} \times \frac{1}{n}$ centred at $x = a, y = b$, since:

$$\begin{aligned}\delta_n(x - a, y - b) &= n^2 \text{rect}[n(x - a), n(y - b)] \\ &= \begin{cases} n^2 & |n(x - a)| \leq \frac{1}{2} \quad |n(y - b)| \leq \frac{1}{2} \\ 0 & \text{elsewhere} \end{cases} \quad (1.29)\end{aligned}$$

We can see that this is a square source centred at (a, b) by considering that $|n(x - a)| \leq \frac{1}{2}$ means $-\frac{1}{2} \leq n(x - a) \leq \frac{1}{2}$, ie $-\frac{1}{2n} \leq x - a \leq \frac{1}{2n}$, or $a - \frac{1}{2n} \leq x \leq a + \frac{1}{2n}$. Thus, we have $\delta_n(x - a, y - b) = n^2$ in the region $a - \frac{1}{2n} \leq x \leq a + \frac{1}{2n}$, $b - \frac{1}{2n} \leq y \leq b + \frac{1}{2n}$. At the limit of $n \rightarrow +\infty$, integral (1.28) is the value of the image g at $x = a, y = b$, ie:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(x, y) \delta_n(x - a, y - b) dx dy = g(a, b) \quad (1.30)$$

This equation is called the **shifting property** of the delta function. This equation also shows that *any* image $g(a, b)$ can be expressed as a superposition of point sources.

Example 1.5

The following 3×3 image

$$f = \begin{pmatrix} 0 & 2 & 6 \\ 1 & 4 & 7 \\ 3 & 5 & 7 \end{pmatrix} \quad (1.31)$$

is processed by a linear operator \mathcal{O} which has a point spread function $h(x, \alpha, y, \beta)$ defined as:

$$\begin{array}{cccc} h(1, 1, 1, 1) = 1.0 & h(1, 1, 1, 2) = 0.5 & h(1, 1, 1, 3) = 0.0 & h(1, 2, 1, 1) = 0.5 \\ h(1, 2, 1, 2) = 0.0 & h(1, 2, 1, 3) = 0.4 & h(1, 3, 1, 1) = 0.5 & h(1, 3, 1, 2) = 1.0 \\ h(1, 3, 1, 3) = 0.6 & h(2, 1, 1, 1) = 0.8 & h(2, 1, 1, 2) = 0.7 & h(2, 1, 1, 3) = 0.4 \\ h(2, 2, 1, 1) = 0.6 & h(2, 2, 1, 2) = 0.5 & h(2, 2, 1, 3) = 0.4 & h(2, 3, 1, 1) = 0.4 \\ h(2, 3, 1, 2) = 0.8 & h(2, 3, 1, 3) = 1.0 & h(3, 1, 1, 1) = 0.9 & h(3, 1, 1, 2) = 0.5 \\ h(3, 1, 1, 3) = 0.5 & h(3, 2, 1, 1) = 0.6 & h(3, 2, 1, 2) = 0.5 & h(3, 2, 1, 3) = 0.3 \\ h(3, 3, 1, 1) = 0.5 & h(3, 3, 1, 2) = 0.9 & h(3, 3, 1, 3) = 1.0 & h(1, 1, 2, 1) = 1.0 \\ h(1, 1, 2, 2) = 0.6 & h(1, 1, 2, 3) = 0.2 & h(1, 2, 2, 1) = 0.0 & h(1, 2, 2, 2) = 0.2 \\ h(1, 2, 2, 3) = 0.4 & h(1, 3, 2, 1) = 0.4 & h(1, 3, 2, 2) = 1.0 & h(1, 3, 2, 3) = 0.6 \\ h(2, 1, 2, 1) = 0.8 & h(2, 1, 2, 2) = 0.7 & h(2, 1, 2, 3) = 0.6 & h(2, 2, 2, 1) = 0.6 \\ h(2, 2, 2, 2) = 0.5 & h(2, 2, 2, 3) = 0.5 & h(2, 3, 2, 1) = 0.5 & h(2, 3, 2, 2) = 1.0 \\ h(2, 3, 2, 3) = 1.0 & h(3, 1, 2, 1) = 0.7 & h(3, 1, 2, 2) = 0.5 & h(3, 1, 2, 3) = 0.5 \\ h(3, 2, 2, 1) = 0.6 & h(3, 2, 2, 2) = 0.5 & h(3, 2, 2, 3) = 0.5 & h(3, 3, 2, 1) = 0.5 \\ h(3, 3, 2, 2) = 1.0 & h(3, 3, 2, 3) = 1.0 & h(1, 1, 3, 1) = 1.0 & h(1, 1, 3, 2) = 0.6 \\ h(1, 1, 3, 3) = 1.0 & h(1, 2, 3, 1) = 0.5 & h(1, 2, 3, 2) = 0.1 & h(1, 2, 3, 3) = 0.6 \\ h(1, 3, 3, 1) = 0.5 & h(1, 3, 3, 2) = 1.0 & h(1, 3, 3, 3) = 0.6 & h(2, 1, 3, 1) = 0.5 \\ h(2, 1, 3, 2) = 0.7 & h(2, 1, 3, 3) = 0.5 & h(2, 2, 3, 1) = 0.6 & h(2, 2, 3, 2) = 0.5 \\ h(2, 2, 3, 3) = 0.5 & h(2, 3, 3, 1) = 0.4 & h(2, 3, 3, 2) = 0.9 & h(2, 3, 3, 3) = 1.0 \end{array}$$

$$\begin{aligned}
 h(3, 1, 3, 1) &= 0.8 & h(3, 1, 3, 2) &= 0.5 & h(3, 1, 3, 3) &= 0.5 & h(3, 2, 3, 1) &= 0.5 \\
 h(3, 2, 3, 2) &= 0.5 & h(3, 2, 3, 3) &= 0.8 & h(3, 3, 3, 1) &= 0.4 & h(3, 3, 3, 2) &= 0.4 \\
 h(3, 3, 3, 3) &= 1.0
 \end{aligned}$$

Work out the output image.

The point spread function of the operator $h(x, \alpha, y, \beta)$ gives the weight with which the pixel value at input position (x, y) contributes to output pixel position (α, β) . Let us call the output image $g(\alpha, \beta)$. We show next how to calculate $g(1, 1)$. For $g(1, 1)$, we need to use the values of $h(x, 1, y, 1)$ to weigh the values of pixels (x, y) of the input image:

$$\begin{aligned}
 g(1, 1) &= \sum_{x=1}^3 \sum_{y=1}^3 f(x, y)h(x, 1, y, 1) \\
 &= f(1, 1)h(1, 1, 1, 1) + f(1, 2)h(1, 1, 2, 1) + f(1, 3)h(1, 1, 3, 1) \\
 &\quad + f(2, 1)h(2, 1, 1, 1) + f(2, 2)h(2, 1, 2, 1) + f(2, 3)h(2, 1, 3, 1) \\
 &\quad + f(3, 1)h(3, 1, 1, 1) + f(3, 2)h(3, 1, 2, 1) + f(3, 3)h(3, 1, 3, 1) \\
 &= 2 \times 1.0 + 6 \times 1.0 + 1 \times 0.8 + 4 \times 0.8 + 7 \times 0.5 + 3 \times 0.9 \\
 &\quad + 5 \times 0.7 + 7 \times 0.8 = 27.3
 \end{aligned} \tag{1.32}$$

For $g(1, 2)$, we need to use the values of $h(x, 1, y, 2)$:

$$g(1, 2) = \sum_{x=1}^3 \sum_{y=1}^3 f(x, y)h(x, 1, y, 2) = 20.1 \tag{1.33}$$

The other values are computed in a similar way. Finally, the output image is:

$$g = \begin{pmatrix} 27.3 & 20.1 & 18.9 \\ 19.4 & 16.0 & 18.4 \\ 16.0 & 29.3 & 33.4 \end{pmatrix} \tag{1.34}$$

Example 1.6

Is the operator of example 1.5 shift invariant?

No, it is not. For example, pixel $(2, 2)$ influences the value of pixel $(1, 2)$ with weight $h(2, 1, 2, 2) = 0.7$. These two pixels are at distance $2 - 1 = 1$ along the x axis and at distance $2 - 2 = 0$ along the y axis. At the same relative distance are also pixels $(3, 3)$ and $(2, 3)$. The value of $h(3, 2, 3, 3)$, however, is $0.8 \neq 0.7$.

Example 1.7

The point spread function of an operator that operates on images of size 3×3 is

$$\begin{array}{cccc}
 h(1, 1, 1, 1) = 1.0 & h(1, 1, 1, 2) = 0.5 & h(1, 1, 1, 3) = 0.0 & h(1, 2, 1, 1) = 0.5 \\
 h(1, 2, 1, 2) = 0.0 & h(1, 2, 1, 3) = 0.4 & h(1, 3, 1, 1) = 0.5 & h(1, 3, 1, 2) = 1.0 \\
 h(1, 3, 1, 3) = 0.6 & h(2, 1, 1, 1) = 0.8 & h(2, 1, 1, 2) = 0.7 & h(2, 1, 1, 3) = 0.4 \\
 h(2, 2, 1, 1) = 1.0 & h(2, 2, 1, 2) = 0.5 & h(2, 2, 1, 3) = 0.0 & h(2, 3, 1, 1) = 0.5 \\
 h(2, 3, 1, 2) = 0.0 & h(2, 3, 1, 3) = 0.4 & h(3, 1, 1, 1) = 0.9 & h(3, 1, 1, 2) = 0.5 \\
 h(3, 1, 1, 3) = 0.5 & h(3, 2, 1, 1) = 0.8 & h(3, 2, 1, 2) = 0.7 & h(3, 2, 1, 3) = 0.4 \\
 h(3, 3, 1, 1) = 1.0 & h(3, 3, 1, 2) = 0.5 & h(3, 3, 1, 3) = 0.0 & h(1, 1, 2, 1) = 1.0 \\
 h(1, 1, 2, 2) = 1.0 & h(1, 1, 2, 3) = 0.5 & h(1, 2, 2, 1) = 0.0 & h(1, 2, 2, 2) = 0.5 \\
 h(1, 2, 2, 3) = 0.0 & h(1, 3, 2, 1) = 0.4 & h(1, 3, 2, 2) = 0.5 & h(1, 3, 2, 3) = 1.0 \\
 h(2, 1, 2, 1) = 0.8 & h(2, 1, 2, 2) = 0.8 & h(2, 1, 2, 3) = 0.7 & h(2, 2, 2, 1) = 1.0 \\
 h(2, 2, 2, 2) = 1.0 & h(2, 2, 2, 3) = 0.5 & h(2, 3, 2, 1) = 0.0 & h(2, 3, 2, 2) = 0.5 \\
 h(2, 3, 2, 3) = 0.0 & h(3, 1, 2, 1) = 0.7 & h(3, 1, 2, 2) = 0.9 & h(3, 1, 2, 3) = 0.5 \\
 h(3, 2, 2, 1) = 0.8 & h(3, 2, 2, 2) = 0.8 & h(3, 2, 2, 3) = 0.7 & h(3, 3, 2, 1) = 1.0 \\
 h(3, 3, 2, 2) = 1.0 & h(3, 3, 2, 3) = 0.5 & h(1, 1, 3, 1) = 1.0 & h(1, 1, 3, 2) = 1.0 \\
 h(1, 1, 3, 3) = 1.0 & h(1, 2, 3, 1) = 0.5 & h(1, 2, 3, 2) = 0.0 & h(1, 2, 3, 3) = 0.5 \\
 h(1, 3, 3, 1) = 0.5 & h(1, 3, 3, 2) = 0.4 & h(1, 3, 3, 3) = 0.5 & h(2, 1, 3, 1) = 0.5 \\
 h(2, 1, 3, 2) = 0.8 & h(2, 1, 3, 3) = 0.8 & h(2, 2, 3, 1) = 1.0 & h(2, 2, 3, 2) = 1.0 \\
 h(2, 2, 3, 3) = 1.0 & h(2, 3, 3, 1) = 0.5 & h(2, 3, 3, 2) = 0.0 & h(2, 3, 3, 3) = 0.5 \\
 h(3, 1, 3, 1) = 0.8 & h(3, 1, 3, 2) = 0.7 & h(3, 1, 3, 3) = 0.9 & h(3, 2, 3, 1) = 0.5 \\
 h(3, 2, 3, 2) = 0.8 & h(3, 2, 3, 3) = 0.8 & h(3, 3, 3, 1) = 1.0 & h(3, 3, 3, 2) = 1.0 \\
 h(3, 3, 3, 3) = 1.0 & & &
 \end{array}$$

Is it shift variant or shift invariant?

In order to show that the function is shift variant, it is enough to show that for at least two pairs of pixels, that correspond to input-output pixels in the same relative position, the values of the function are different. This is what we did in example 1.6. If we cannot find such an example, we must then check for shift invariance. The function must have the same value for all pairs of input-output pixels that are in the same relative position in order to be shift invariant. As the range of values each of the arguments of this function takes is $[1, 3]$, the relative coordinates of pairs of input-output pixels take values in the range $[-2, 2]$. We observe the following.

For $x - \alpha = -2$ and $y - \beta = -2$ we have: $h(1, 3, 1, 3) = 0.6$

For $x - \alpha = -2$ and $y - \beta = -1$ we have: $h(1, 3, 1, 2) = h(1, 3, 2, 3) = 1.0$

For $x - \alpha = -2$ and $y - \beta = 0$ we have: $h(1, 3, 1, 1) = h(1, 3, 2, 2) = h(1, 3, 3, 3) = 0.5$

For $x - \alpha = -2$ and $y - \beta = 1$ we have: $h(1, 3, 2, 1) = h(1, 3, 3, 2) = 0.4$

For $x - \alpha = -2$ and $y - \beta = 2$ we have: $h(1, 3, 3, 1) = 0.5$

For $x - \alpha = -1$ and $y - \beta = -2$ we have: $h(1, 2, 1, 3) = h(2, 3, 1, 3) = 0.4$

For $x - \alpha = -1$ and $y - \beta = -1$ we have:

$h(1, 2, 1, 2) = h(2, 3, 2, 3) = h(1, 2, 2, 3) = h(2, 3, 1, 2) = 0.0$

For $x - \alpha = -1$ and $y - \beta = 0$ we have:

$h(1, 2, 1, 1) = h(1, 2, 2, 2) = h(1, 2, 3, 3) = h(2, 3, 1, 1) = h(2, 3, 2, 2) = h(2, 3, 3, 3) = 0.5$
 For $x - \alpha = -1$ and $y - \beta = 1$ we have:
 $h(1, 2, 2, 1) = h(1, 2, 3, 2) = h(2, 3, 2, 1) = h(2, 3, 3, 2) = 0.0$
 For $x - \alpha = -1$ and $y - \beta = 2$ we have: $h(1, 2, 3, 1) = 0.5$
 For $x - \alpha = 0$ and $y - \beta = -2$ we have: $h(1, 1, 1, 3) = h(2, 2, 1, 3) = h(3, 3, 1, 3) = 0.0$
 For $x - \alpha = 0$ and $y - \beta = -1$ we have:
 $h(1, 1, 1, 2) = h(2, 2, 1, 2) = h(3, 3, 1, 2) = h(1, 1, 2, 3) = h(2, 2, 2, 3) = h(3, 3, 2, 3) = 0.5$
 For $x - \alpha = 0$ and $y - \beta = 0$ we have:
 $h(1, 1, 1, 1) = h(2, 2, 1, 1) = h(3, 3, 1, 1) = h(1, 1, 2, 2) = h(2, 2, 2, 2) =$
 $h(3, 3, 2, 2) = h(1, 1, 3, 3) = h(2, 2, 3, 3) = h(3, 3, 3, 3) = 1.0$
 For $x - \alpha = 0$ and $y - \beta = 1$ we have:
 $h(1, 1, 2, 1) = h(2, 2, 2, 1) = h(3, 3, 2, 1) = h(1, 1, 3, 2) = h(2, 2, 3, 2) = h(3, 3, 3, 2) = 1.0$
 For $x - \alpha = 0$ and $y - \beta = 2$ we have: $h(1, 1, 3, 1) = h(2, 2, 3, 1) = h(3, 3, 3, 1) = 1.0$
 For $x - \alpha = 1$ and $y - \beta = -2$ we have: $h(2, 1, 1, 3) = h(3, 2, 1, 3) = 0.4$
 For $x - \alpha = 1$ and $y - \beta = -1$ we have:
 $h(2, 1, 1, 2) = h(2, 1, 2, 3) = h(3, 2, 1, 2) = h(3, 2, 2, 3) = 0.7$
 For $x - \alpha = 1$ and $y - \beta = 0$ we have:
 $h(2, 1, 1, 1) = h(2, 1, 2, 2) = h(2, 1, 3, 3) = h(3, 2, 1, 1) = h(3, 2, 2, 2) = h(3, 2, 3, 3) = 0.8$
 For $x - \alpha = 1$ and $y - \beta = 1$ we have:
 $h(2, 1, 2, 1) = h(2, 1, 3, 2) = h(3, 2, 2, 1) = h(3, 2, 3, 2) = 0.8$
 For $x - \alpha = 1$ and $y - \beta = 2$ we have: $h(2, 1, 3, 1) = h(3, 2, 3, 1) = 0.5$
 For $x - \alpha = 2$ and $y - \beta = -2$ we have: $h(3, 1, 1, 3) = 0.5$
 For $x - \alpha = 2$ and $y - \beta = -1$ we have: $h(3, 1, 1, 2) = h(3, 1, 2, 3) = 0.5$
 For $x - \alpha = 2$ and $y - \beta = 0$ we have: $h(3, 1, 1, 1) = h(3, 1, 2, 2) = h(3, 1, 3, 3) = 0.9$
 For $x - \alpha = 2$ and $y - \beta = 1$ we have: $h(3, 1, 2, 1) = h(3, 1, 3, 2) = 0.7$
 For $x - \alpha = 2$ and $y - \beta = 2$ we have: $h(3, 1, 3, 1) = 0.8$
 So, this is a shift invariant point spread function.

How can we express in practice the effect of a linear operator on an image?

This is done with the help of matrices. We can rewrite equation (1.15) as follows:

$$\begin{aligned}
 g(\alpha, \beta) = & \\
 & f(1, 1)h(1, \alpha, 1, \beta) + f(2, 1)h(2, \alpha, 1, \beta) + \dots + f(N, 1)h(N, \alpha, 1, \beta) \\
 & + f(1, 2)h(1, \alpha, 2, \beta) + f(2, 2)h(2, \alpha, 2, \beta) + \dots + f(N, 2)h(N, \alpha, 2, \beta) \\
 & + \dots + f(1, N)h(1, \alpha, N, \beta) + f(2, N)h(2, \alpha, N, \beta) + \dots \\
 & + f(N, N)h(N, \alpha, N, \beta)
 \end{aligned} \tag{1.35}$$

The right-hand side of this expression can be thought of as the dot product of vector

$$\begin{aligned}
 \mathbf{h}_{\alpha\beta}^T \equiv & \\
 & [h(1, \alpha, 1, \beta), h(2, \alpha, 1, \beta), \dots, h(N, \alpha, 1, \beta), h(1, \alpha, 2, \beta), h(2, \alpha, 2, \beta), \dots, \\
 & h(N, \alpha, 2, \beta), \dots, h(2, \alpha, N, \beta), \dots, h(N, \alpha, N, \beta)]
 \end{aligned} \tag{1.36}$$

with vector:

$$\begin{aligned} \mathbf{f}^T &\equiv [f(1,1), f(2,1), \dots, f(N,1), f(1,2), f(2,2), \dots, f(N,2), \\ &\quad \dots, f(1,N), f(2,N), \dots, f(N,N)] \end{aligned} \quad (1.37)$$

This last vector is actually the image $f(x, y)$ written as a vector by stacking its columns one under the other. If we imagine writing $g(\alpha, \beta)$ in the same way, then vectors $\mathbf{h}_{\alpha\beta}^T$ will arrange themselves as the rows of a matrix H , where for $\beta = 1$, α will run from 1 to N to give the first N rows of the matrix, then for $\beta = 2$, β will run again from 1 to N to give the second N rows of the matrix, and so on. Thus, equation (1.15) may be written in a more compact way as:

$$\mathbf{g} = H\mathbf{f} \quad (1.38)$$

This is the fundamental equation of linear image processing. H here is a square $N^2 \times N^2$ matrix that is made up of $N \times N$ submatrices of size $N \times N$ each, arranged in the following way:

$$H = \begin{pmatrix} \alpha \downarrow \left(\begin{array}{c} x \rightarrow \\ y=1 \\ \beta=1 \\ x \rightarrow \end{array} \right) & \alpha \downarrow \left(\begin{array}{c} x \rightarrow \\ y=2 \\ \beta=1 \\ x \rightarrow \end{array} \right) & \dots & \alpha \downarrow \left(\begin{array}{c} x \rightarrow \\ y=N \\ \beta=1 \\ x \rightarrow \end{array} \right) \\ \alpha \downarrow \left(\begin{array}{c} y=1 \\ \beta=2 \\ x \rightarrow \end{array} \right) & \alpha \downarrow \left(\begin{array}{c} y=2 \\ \beta=2 \\ x \rightarrow \end{array} \right) & \dots & \alpha \downarrow \left(\begin{array}{c} y=N \\ \beta=1 \\ x \rightarrow \end{array} \right) \\ \vdots & \vdots & & \vdots \\ \alpha \downarrow \left(\begin{array}{c} y=1 \\ \beta=N \\ x \rightarrow \end{array} \right) & \alpha \downarrow \left(\begin{array}{c} y=2 \\ \beta=N \\ x \rightarrow \end{array} \right) & \dots & \alpha \downarrow \left(\begin{array}{c} y=N \\ \beta=N \\ x \rightarrow \end{array} \right) \end{pmatrix} \quad (1.39)$$

In this representation each bracketed expression represents an $N \times N$ submatrix made up from function $h(x, \alpha, y, \beta)$ for fixed values of y and β and with variables x and α taking up all their possible values in the directions indicated by the arrows. This schematic structure of matrix H is said to correspond to a **partition** of this matrix into N^2 square submatrices.

Example 1.8

A linear operator is such that it replaces the value of each pixel by the average of its four nearest neighbours. Apply this operator to a 3×3 image g . Assume that the image is repeated ad infinitum in all directions, so that all its pixels have neighbours. Work out the 9×9 matrix H that corresponds to this operator by using equation (1.39).

If the image is repeated in all directions, the image and the neighbours of its border pixels will look like this:

$$\begin{array}{c|ccccc|cc} g_{33} & | & g_{31} & g_{32} & g_{33} & | & g_{11} \\ \hline - & - & - & - & - & - & - \\ g_{13} & | & g_{11} & g_{12} & g_{13} & | & g_{11} \\ g_{23} & | & g_{21} & g_{22} & g_{23} & | & g_{21} \\ g_{33} & | & g_{31} & g_{32} & g_{33} & | & g_{31} \\ \hline - & - & - & - & - & - & - \\ g_{13} & | & g_{11} & g_{12} & g_{13} & | & g_{11} \end{array} \quad (1.40)$$

The result then of replacing every pixel by the average of its four nearest neighbours is:

$$\begin{array}{ccc} \frac{g_{31}+g_{12}+g_{21}+g_{13}}{4} & \frac{g_{32}+g_{13}+g_{22}+g_{11}}{4} & \frac{g_{33}+g_{11}+g_{23}+g_{12}}{4} \\ \frac{g_{11}+g_{22}+g_{31}+g_{23}}{4} & \frac{g_{12}+g_{23}+g_{32}+g_{21}}{4} & \frac{g_{13}+g_{21}+g_{33}+g_{22}}{4} \\ \frac{g_{21}+g_{32}+g_{11}+g_{33}}{4} & \frac{g_{22}+g_{33}+g_{12}+g_{31}}{4} & \frac{g_{23}+g_{31}+g_{13}+g_{32}}{4} \end{array} \quad (1.41)$$

In order to construct matrix H we must deduce from the above result the weight by which a pixel at position (x, y) of the input image contributes to the value of a pixel at position (α, β) of the output image. Such value will be denoted by $h(x, \alpha, y, \beta)$. Matrix H will be made up from these values arranged as follows:

$$H = \left[\begin{array}{ccccc} h(1, 1, 1, 1) & h(2, 1, 1, 1) & h(3, 1, 1, 1) & h(1, 1, 2, 1) & h(2, 1, 2, 1) \\ h(1, 2, 1, 1) & h(2, 2, 1, 1) & h(3, 2, 1, 1) & h(1, 2, 2, 1) & h(2, 2, 2, 1) \\ h(1, 3, 1, 1) & h(2, 3, 1, 1) & h(3, 3, 1, 1) & h(1, 3, 2, 1) & h(2, 3, 2, 1) \\ h(1, 1, 1, 2) & h(2, 1, 1, 2) & h(3, 1, 1, 2) & h(1, 1, 2, 2) & h(2, 1, 2, 2) \\ h(1, 2, 1, 2) & h(2, 2, 1, 2) & h(3, 2, 1, 2) & h(1, 2, 2, 2) & h(2, 2, 2, 2) \\ h(1, 3, 1, 2) & h(2, 3, 1, 2) & h(3, 3, 1, 2) & h(1, 3, 2, 2) & h(2, 3, 2, 2) \\ h(1, 1, 1, 3) & h(2, 1, 1, 3) & h(3, 1, 1, 3) & h(1, 1, 2, 3) & h(2, 1, 2, 3) \\ h(1, 2, 1, 3) & h(2, 2, 1, 3) & h(3, 2, 1, 3) & h(1, 2, 2, 3) & h(2, 2, 2, 3) \\ h(1, 3, 1, 3) & h(2, 3, 1, 3) & h(3, 3, 1, 3) & h(1, 3, 2, 3) & h(2, 3, 2, 3) \\ \\ h(3, 1, 2, 1) & h(1, 1, 3, 1) & h(2, 1, 3, 1) & h(3, 1, 3, 1) & \\ h(3, 2, 2, 1) & h(1, 2, 3, 1) & h(2, 2, 3, 1) & h(3, 2, 3, 1) & \\ h(3, 3, 2, 1) & h(1, 3, 3, 1) & h(2, 3, 3, 1) & h(3, 3, 3, 1) & \\ h(3, 1, 2, 2) & h(1, 1, 3, 2) & h(2, 1, 3, 2) & h(3, 1, 3, 2) & \\ h(3, 2, 2, 2) & h(1, 2, 3, 2) & h(2, 2, 3, 2) & h(3, 2, 3, 2) & \\ h(3, 3, 2, 2) & h(1, 3, 3, 2) & h(2, 3, 3, 2) & h(3, 3, 3, 2) & \\ h(3, 1, 2, 3) & h(1, 1, 3, 3) & h(2, 1, 3, 3) & h(3, 1, 3, 3) & \\ h(3, 2, 2, 3) & h(1, 2, 3, 3) & h(2, 2, 3, 3) & h(3, 2, 3, 3) & \\ h(3, 3, 2, 3) & h(1, 3, 3, 3) & h(2, 3, 3, 3) & h(3, 3, 3, 3) & \end{array} \right] \quad (1.42)$$

By inspection we deduce that:

$$H = \left[\begin{array}{cccccccc} 0 & 1/4 & 1/4 & 1/4 & 0 & 0 & 1/4 & 0 & 0 \\ 1/4 & 0 & 1/4 & 0 & 1/4 & 0 & 0 & 1/4 & 0 \\ 1/4 & 1/4 & 0 & 0 & 0 & 1/4 & 0 & 0 & 1/4 \\ 1/4 & 0 & 0 & 0 & 1/4 & 1/4 & 1/4 & 0 & 0 \\ 0 & 1/4 & 0 & 1/4 & 0 & 1/4 & 0 & 1/4 & 0 \\ 0 & 0 & 1/4 & 1/4 & 1/4 & 0 & 0 & 0 & 1/4 \\ 1/4 & 0 & 0 & 1/4 & 0 & 0 & 0 & 1/4 & 1/4 \\ 0 & 1/4 & 0 & 0 & 1/4 & 0 & 1/4 & 0 & 1/4 \\ 0 & 0 & 1/4 & 0 & 0 & 1/4 & 1/4 & 1/4 & 0 \end{array} \right] \quad (1.43)$$

Example 1.9

The effect of a linear operator is to subtract from every pixel its right neighbour. This operator is applied to image (1.40). Work out the output image and the H matrix that corresponds to this operator.

The result of this operator will be:

$$\begin{array}{ccc} g_{11} - g_{12} & g_{12} - g_{13} & g_{13} - g_{11} \\ g_{21} - g_{22} & g_{22} - g_{23} & g_{23} - g_{21} \\ g_{31} - g_{32} & g_{32} - g_{33} & g_{33} - g_{31} \end{array} \quad (1.44)$$

By following the procedure we followed in example 1.8, we can work out matrix H , which here, for convenience, we call \hat{H} :

$$\hat{H} = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.45)$$

Example 1.10

The effect of a linear operator is to subtract from every pixel its bottom right neighbour. This operator is applied to image (1.40). Work out the output image and the H matrix that corresponds to this operator.

The result of this operator will be:

$$\begin{array}{ccc} g_{11} - g_{22} & g_{12} - g_{23} & g_{13} - g_{21} \\ g_{21} - g_{32} & g_{22} - g_{33} & g_{23} - g_{31} \\ g_{31} - g_{12} & g_{32} - g_{13} & g_{33} - g_{11} \end{array} \quad (1.46)$$

By following the procedure we followed in example 1.8, we can work out matrix H ,

which here, for convenience, we call \tilde{H} :

$$\tilde{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.47)$$

Can we apply more than one linear operators to an image?

We can apply as many operators as we like.

Does the order by which we apply the linear operators make any difference to the result?

No, if the operators are shift invariant. This is a very important and convenient property of the commonly used linear operators.

Box 1.2. Since matrix multiplication is not commutative, how come we can change the order by which we apply shift invariant linear operators?

In general, if A and B are two matrices, $AB \neq BA$. However, matrix H , which expresses the effect of a linear operator, has a particular structure. We can see that from equation (1.39): the $N^2 \times N^2$ matrix H can be divided into N^2 submatrices of size $N \times N$ each. There are at most N distinct such submatrices and they have a particular structure: the second column of their elements can be produced from the first column, by shifting all elements by one position down. The element that sticks out at the bottom is put at the empty position at the top. The third column is produced by applying the same procedure to the second column as so on (see example 1.11). A matrix that has this property is called **circulant matrix**. At the same time, these submatrices are also arranged in a circulant way: the second column of them is produced from the first column by shifting all of them by one position down. The submatrix that sticks out at the bottom is put at the empty position at the top. The third column of matrices is created from the second by applying the same procedure and so on (see example 1.12). A matrix that has this property is called **block circulant**. It is this particular structure that allows one to exchange the order by which two operators with matrices H and \tilde{H} , respectively, are applied to an image \mathbf{g} written as a vector:

$$H\tilde{H}\mathbf{g} = \tilde{H}H\mathbf{g} \quad (1.48)$$

Example B1.11

Write down the form you expect two 3×3 circulant matrices to have and show that their product commutes.

A 3×3 circulant matrix will contain at most 3 distinct elements. Let us call them α, β and γ for matrix A , and $\tilde{\alpha}, \tilde{\beta}$ and $\tilde{\gamma}$ for matrix \tilde{A} . Since these matrices are circulant, they must have the following structure:

$$A = \begin{pmatrix} \alpha & \gamma & \beta \\ \beta & \alpha & \gamma \\ \gamma & \beta & \alpha \end{pmatrix} \quad \tilde{A} = \begin{pmatrix} \tilde{\alpha} & \tilde{\gamma} & \tilde{\beta} \\ \tilde{\beta} & \tilde{\alpha} & \tilde{\gamma} \\ \tilde{\gamma} & \tilde{\beta} & \tilde{\alpha} \end{pmatrix} \quad (1.49)$$

We can work out the product $A\tilde{A}$:

$$A\tilde{A} = \begin{pmatrix} \alpha & \gamma & \beta \\ \beta & \alpha & \gamma \\ \gamma & \beta & \alpha \end{pmatrix} \begin{pmatrix} \tilde{\alpha} & \tilde{\gamma} & \tilde{\beta} \\ \tilde{\beta} & \tilde{\alpha} & \tilde{\gamma} \\ \tilde{\gamma} & \tilde{\beta} & \tilde{\alpha} \end{pmatrix} = \begin{pmatrix} \alpha\tilde{\alpha} + \gamma\tilde{\beta} + \beta\tilde{\gamma} & \alpha\tilde{\gamma} + \gamma\tilde{\alpha} + \beta\tilde{\beta} & \alpha\tilde{\beta} + \gamma\tilde{\gamma} + \beta\tilde{\alpha} \\ \beta\tilde{\alpha} + \alpha\tilde{\beta} + \gamma\tilde{\gamma} & \beta\tilde{\gamma} + \alpha\tilde{\alpha} + \gamma\tilde{\beta} & \beta\tilde{\beta} + \alpha\tilde{\gamma} + \gamma\tilde{\alpha} \\ \gamma\tilde{\alpha} + \beta\tilde{\beta} + \alpha\tilde{\gamma} & \gamma\tilde{\gamma} + \beta\tilde{\alpha} + \alpha\tilde{\beta} & \gamma\tilde{\beta} + \beta\tilde{\gamma} + \alpha\tilde{\alpha} \end{pmatrix} \quad (1.50)$$

We get the same result by working out $\tilde{A}A$:

$$\tilde{A}A = \begin{pmatrix} \tilde{\alpha} & \tilde{\gamma} & \tilde{\beta} \\ \tilde{\beta} & \tilde{\alpha} & \tilde{\gamma} \\ \tilde{\gamma} & \tilde{\beta} & \tilde{\alpha} \end{pmatrix} \begin{pmatrix} \alpha & \gamma & \beta \\ \beta & \alpha & \gamma \\ \gamma & \beta & \alpha \end{pmatrix} = \begin{pmatrix} \tilde{\alpha}\alpha + \tilde{\gamma}\beta + \tilde{\beta}\gamma & \tilde{\alpha}\gamma + \tilde{\gamma}\alpha + \tilde{\beta}\beta & \tilde{\alpha}\beta + \tilde{\gamma}\gamma + \tilde{\beta}\alpha \\ \tilde{\beta}\alpha + \tilde{\alpha}\beta + \tilde{\gamma}\gamma & \tilde{\beta}\gamma + \tilde{\alpha}\alpha + \tilde{\gamma}\beta & \tilde{\beta}\beta + \tilde{\alpha}\gamma + \tilde{\gamma}\alpha \\ \tilde{\gamma}\alpha + \tilde{\beta}\beta + \tilde{\alpha}\gamma & \tilde{\gamma}\gamma + \tilde{\beta}\alpha + \tilde{\alpha}\beta & \tilde{\gamma}\beta + \tilde{\beta}\gamma + \tilde{\alpha}\alpha \end{pmatrix} \quad (1.51)$$

We can see that $A\tilde{A} = \tilde{A}A$, ie that these two matrices commute.

Example B1.12

Identify the 3×3 submatrices from which the H matrices of examples 1.8, 1.9 and 1.10 are made.

We can easily identify that matrix H of example 1.8 is made up from submatrices:

$$H_{11} \equiv \begin{pmatrix} 0 & 1/4 & 1/4 \\ 1/4 & 0 & 1/4 \\ 1/4 & 1/4 & 0 \end{pmatrix} \quad H_{21} = H_{31} \equiv \begin{pmatrix} 1/4 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/4 \end{pmatrix} \quad (1.52)$$

Matrix \hat{H} of example 1.9 is made up from submatrices:

$$\hat{H}_{11} \equiv \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \hat{H}_{21} \equiv \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \hat{H}_{31} \equiv \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad (1.53)$$

Matrix \tilde{H} of example 1.10 is made up from submatrices:

$$\tilde{H}_{11} \equiv \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \tilde{H}_{21} \equiv \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \tilde{H}_{31} \equiv \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{pmatrix} \quad (1.54)$$

Thus, matrices H , \hat{H} and \tilde{H} of examples 1.8, 1.9 and 1.10, respectively, may be written as:

$$H = \begin{bmatrix} H_{11} & H_{31} & H_{21} \\ H_{21} & H_{11} & H_{31} \\ H_{31} & H_{21} & H_{11} \end{bmatrix} \quad \hat{H} = \begin{bmatrix} \hat{H}_{11} & \hat{H}_{31} & \hat{H}_{21} \\ \hat{H}_{21} & \hat{H}_{11} & \hat{H}_{31} \\ \hat{H}_{31} & \hat{H}_{21} & \hat{H}_{11} \end{bmatrix} \quad \tilde{H} = \begin{bmatrix} \tilde{H}_{11} & \tilde{H}_{31} & \tilde{H}_{21} \\ \tilde{H}_{21} & \tilde{H}_{11} & \tilde{H}_{31} \\ \tilde{H}_{31} & \tilde{H}_{21} & \tilde{H}_{11} \end{bmatrix} \quad (1.55)$$

Each one of the submatrices is circulant and they are arranged in a circulant manner, so matrices H , \hat{H} and \tilde{H} are block circulant.

Example 1.13

Apply the operator of example 1.9 to the output image (1.41) of example 1.8 by working directly on the output image. Then apply the operator of example 1.8 to the output image (1.44) of example 1.9. Compare the two answers.

The operator of example 1.9 subtracts from every pixel the value of its right neighbour. We remember that the image is assumed to be wrapped round so that all pixels have neighbours in all directions. We perform this operation on image (1.41) and obtain:

$$\left[\begin{array}{ccc} \frac{g_{31}+g_{12}+g_{21}-g_{32}-g_{22}-g_{11}}{4} & \frac{g_{32}+g_{13}+g_{22}-g_{33}-g_{23}-g_{12}}{4} & \frac{g_{33}+g_{11}+g_{23}-g_{31}-g_{21}-g_{13}}{4} \\ \frac{g_{11}+g_{22}+g_{31}-g_{12}-g_{32}-g_{21}}{4} & \frac{g_{12}+g_{23}+g_{32}-g_{13}-g_{33}-g_{22}}{4} & \frac{g_{13}+g_{21}+g_{33}-g_{11}-g_{31}-g_{23}}{4} \\ \frac{g_{21}+g_{32}+g_{11}-g_{22}-g_{12}-g_{31}}{4} & \frac{g_{22}+g_{33}+g_{12}-g_{23}-g_{13}-g_{32}}{4} & \frac{g_{23}+g_{31}+g_{13}-g_{21}-g_{11}-g_{33}}{4} \end{array} \right] \quad (1.56)$$

The operator of example 1.8 replaces every pixel with the average of its four neighbours.

We apply it to image (1.44) and obtain:

$$\left[\begin{array}{cc} \frac{g_{31}-g_{32}+g_{12}-g_{13}+g_{21}-g_{22}+g_{13}-g_{11}}{4} & \frac{g_{32}-g_{33}+g_{13}-g_{11}+g_{22}-g_{23}+g_{11}-g_{12}}{4} \\ \frac{g_{11}-g_{12}+g_{22}-g_{23}+g_{31}-g_{32}+g_{23}-g_{21}}{4} & \frac{g_{12}-g_{13}+g_{23}-g_{21}+g_{32}-g_{33}+g_{21}-g_{22}}{4} \\ \frac{g_{21}-g_{22}+g_{32}-g_{33}+g_{11}-g_{12}+g_{33}-g_{31}}{4} & \frac{g_{22}-g_{23}+g_{33}-g_{31}+g_{12}-g_{13}+g_{31}-g_{32}}{4} \\ & \frac{g_{33}-g_{31}+g_{11}-g_{12}+g_{23}-g_{21}+g_{12}-g_{13}}{4} \\ & \frac{g_{13}-g_{11}+g_{21}-g_{22}+g_{33}-g_{31}+g_{22}-g_{23}}{4} \\ & \frac{g_{23}-g_{21}+g_{31}-g_{32}+g_{13}-g_{11}+g_{32}-g_{33}}{4} \end{array} \right] \quad (1.57)$$

By comparing outputs (1.56) and (1.57) we see that we get the same answer whichever is the order by which we apply the operators.

Example 1.14

Use matrix multiplication to derive the matrix with which an image (written in vector form) has to be multiplied from the left so that the operators of examples 1.8 and 1.9 are applied in a cascaded way. Does the answer depend on the order by which the operators are applied?

If we apply first the operator of example 1.9 and then the operator of example 1.8, we must compute the product $H\hat{H}$ of matrices H and \hat{H} given by equations (1.43) and (1.45), respectively:

$$\begin{aligned} H\hat{H} &= \begin{bmatrix} 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -1/4 & 1/4 & 1/4 & 1/4 & -1/4 & -1/4 & 0 & 0 & 0 \\ 1/4 & -1/4 & 1/4 & -1/4 & 1/4 & -1/4 & 0 & 0 & 0 \\ 1/4 & 1/4 & -1/4 & -1/4 & -1/4 & 1/4 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1/4 & 1/4 & 1/4 & 1/4 & -1/4 & -1/4 \\ 0 & 0 & 0 & 1/4 & -1/4 & 1/4 & -1/4 & 1/4 & -1/4 \\ 0 & 0 & 0 & 1/4 & 1/4 & -1/4 & -1/4 & -1/4 & 1/4 \\ 1/4 & -1/4 & -1/4 & 0 & 0 & 0 & -1/4 & 1/4 & 1/4 \\ -1/4 & 1/4 & -1/4 & 0 & 0 & 0 & 1/4 & -1/4 & 1/4 \\ -1/4 & -1/4 & 1/4 & 0 & 0 & 0 & 1/4 & 1/4 & -1/4 \end{bmatrix} \quad (1.58) \end{aligned}$$

If we apply first the operator of example 1.8 and then the operator of example 1.9, we must compute the product $\hat{H}H$ of matrices H and \hat{H} given by equations (1.43) and (1.45), respectively:

$$\begin{aligned} \hat{H}H &= \\ \left[\begin{array}{ccccccccc} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{ccccccccc} 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 \end{array} \right] \\ &= \left[\begin{array}{cccccccc} -1/4 & 1/4 & 1/4 & 1/4 & -1/4 & -1/4 & 0 & 0 & 0 \\ 1/4 & -1/4 & 1/4 & -1/4 & 1/4 & -1/4 & 0 & 0 & 0 \\ 1/4 & 1/4 & -1/4 & -1/4 & -1/4 & 1/4 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1/4 & 1/4 & 1/4 & 1/4 & -1/4 & -1/4 \\ 0 & 0 & 0 & 1/4 & -1/4 & 1/4 & -1/4 & 1/4 & -1/4 \\ 0 & 0 & 0 & 1/4 & 1/4 & -1/4 & -1/4 & -1/4 & 1/4 \\ 1/4 & -1/4 & -1/4 & 0 & 0 & 0 & -1/4 & 1/4 & 1/4 \\ -1/4 & 1/4 & -1/4 & 0 & 0 & 0 & 1/4 & -1/4 & 1/4 \\ -1/4 & -1/4 & 1/4 & 0 & 0 & 0 & 1/4 & 1/4 & -1/4 \end{array} \right] \quad (1.59) \end{aligned}$$

By comparing results (1.58) and (1.59) we see that the order by which we multiply the matrices, and by extension the order by which we apply the operators, does not matter.

Example 1.15

Process image (1.40) with matrix (1.59) and compare your answer with the output images produced in example 1.13.

To process image (1.40) by matrix (1.59), we must write it in vector form, by stacking its columns one under the other:

$$\begin{bmatrix}
 -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 0 & 0 & 0 \\
 \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & 0 & 0 & 0 \\
 \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\
 0 & 0 & 0 & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \\
 0 & 0 & 0 & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \\
 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \\
 \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 0 & 0 & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \\
 -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -\frac{1}{4} & \\
 -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} &
 \end{bmatrix}
 \begin{bmatrix}
 g_{11} \\ g_{21} \\ g_{31} \\ g_{12} \\ g_{22} \\ g_{32} \\ g_{13} \\ g_{23} \\ g_{33}
 \end{bmatrix}
 =
 \begin{bmatrix}
 \frac{-g_{11}+g_{21}+g_{31}+g_{12}-g_{22}-g_{32}}{4} \\
 \frac{g_{11}-g_{21}+g_{31}-g_{12}+g_{22}-g_{32}}{4} \\
 \frac{g_{11}+g_{21}-g_{31}-g_{12}-g_{22}+g_{32}}{4} \\
 \frac{-g_{12}+g_{22}+g_{32}+g_{13}-g_{23}-g_{33}}{4} \\
 \frac{g_{12}-g_{22}+g_{32}-g_{13}+g_{23}-g_{33}}{4} \\
 \frac{g_{12}+g_{22}-g_{32}-g_{13}-g_{23}+g_{33}}{4} \\
 \frac{g_{11}-g_{21}-g_{31}-g_{13}+g_{23}+g_{33}}{4} \\
 \frac{-g_{11}+g_{21}-g_{31}+g_{13}-g_{23}+g_{33}}{4} \\
 \frac{-g_{11}-g_{21}+g_{31}+g_{13}+g_{23}-g_{33}}{4}
 \end{bmatrix}
 \quad (1.60)$$

To create an image out of the output vector (1.60), we have to use its first three elements as the first column of the image, the next three elements as the second column of the image, and so on. The image we obtain is:

$$\begin{bmatrix}
 \frac{-g_{11}+g_{21}+g_{31}+g_{12}-g_{22}-g_{32}}{4} & \frac{-g_{12}+g_{22}+g_{32}+g_{13}-g_{23}-g_{33}}{4} & \frac{g_{11}-g_{21}-g_{31}-g_{13}+g_{23}+g_{33}}{4} \\
 \frac{g_{11}-g_{21}+g_{31}-g_{12}+g_{22}-g_{32}}{4} & \frac{g_{12}-g_{22}+g_{32}-g_{13}+g_{23}-g_{33}}{4} & \frac{-g_{11}+g_{21}-g_{31}+g_{13}-g_{23}+g_{33}}{4} \\
 \frac{g_{11}+g_{21}-g_{31}-g_{12}-g_{22}+g_{32}}{4} & \frac{g_{12}+g_{22}-g_{32}-g_{13}-g_{23}+g_{33}}{4} & \frac{-g_{11}-g_{21}+g_{31}+g_{13}+g_{23}-g_{33}}{4}
 \end{bmatrix}
 \quad (1.61)$$

By comparing (1.61) and (1.56) we see that we obtain the same output image either we apply the operators locally, or we operate on the whole image by using the corresponding matrix.

Example 1.16

By examining matrices H , \hat{H} and \tilde{H} given by equations (1.43), (1.45) and (1.47), respectively, deduce the point spread function of the corresponding operators.

As these operators are shift invariant, by definition, in order to work out their point spread functions starting from their corresponding H matrix, we have to reason about the structure of the matrix as exemplified by equation (1.39). The point spread function will be the same for all pixels of the input image, so we might as well pick one of them; say we pick the pixel in the middle of the input image, with coordinates $x = 2$ and $y = 2$. Then, we have to read the values of the elements of the matrix that correspond to all possible combinations of α and β , which indicate the coordinates of the output image. According to equation (1.39), β takes all its possible values along a column of submatrices, while α takes all its possible values along a column of any one of the submatrices. Fixing the value of $x = 2$ means that we shall have to read only the middle column of the submatrix we use. Fixing the value of $y = 2$ means that we shall have to read only the middle column of submatrices. The middle column then of

matrix H , when wrapped to form a 3×3 image, will be the point spread function of the operator, ie it will represent the output image we shall get if the operator is applied to an input image that consists of only 0s except in the central pixel where it has value 1 (a single point source). We have to write the first three elements of the central column of the H matrix as the first column of the output image, the next three elements as the second column, and the last three elements as the last column of the output image. For the three operators that correspond to matrices H , \hat{H} and \tilde{H} , we obtain:

$$h = \begin{pmatrix} 0 & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 \end{pmatrix} \quad \hat{h} = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \tilde{h} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad (1.62)$$

Example 1.17

What will be the effect of the operator that corresponds to matrix (1.59) on a 3×3 input image that depicts a point source in the middle?

The output image will be the point spread function of the operator. Following the reasoning of example 1.16, we deduce that the point spread function of this composite operator is:

$$\begin{pmatrix} -1/4 & 1/4 & 0 \\ 1/4 & -1/4 & 0 \\ -1/4 & 1/4 & 0 \end{pmatrix} \quad (1.63)$$

Example 1.18

What will be the effect of the operator that corresponds to matrix (1.59) on a 3×3 input image that depicts a point source in position (1, 2)?

In this case we want to work out the output for $x = 1$ and $y = 2$. We select from matrix (1.59) the second column of submatrices, and the first column of them and wrap it to form a 3×3 image. We obtain:

$$\begin{pmatrix} 1/4 & -1/4 & 0 \\ -1/4 & 1/4 & 0 \\ -1/4 & 1/4 & 0 \end{pmatrix} \quad (1.64)$$

Alternatively, we multiply from the left input image

$$g^T = (0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0) \quad (1.65)$$

with matrix $H\hat{H}$ given by equation (1.58) and write the output vector as an image. We get exactly the same answer.

Box 1.3. What is the stacking operator?

The stacking operator allows us to write an $N \times N$ image array as an $N^2 \times 1$ vector, or an $N^2 \times 1$ vector as an $N \times N$ square array.

We define some vectors \mathbf{V}_n and some matrices N_n as:

$$\mathbf{V}_n \equiv \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \begin{array}{l} \left. \right\} \text{rows 1 to } n-1 \\ \left. \right\} \text{row } n \\ \left. \right\} \text{rows } n+1 \text{ to } N \end{array} \quad (1.66)$$

$$N_n \equiv \begin{bmatrix} \mathbf{0} \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 \\ \mathbf{0} \end{bmatrix} \quad \begin{array}{l} \left. \right\} n-1 \text{ square } N \times N \text{ matrices on top of each other with all their elements 0} \\ \left. \right\} \text{the } n^{\text{th}} \text{ matrix is the unit matrix} \\ \left. \right\} N-n \text{ square } N \times N \text{ matrices on the top of each other with all their elements 0} \end{array} \quad (1.67)$$

The dimensions of \mathbf{V}_n are $N \times 1$ and of N_n $N^2 \times N$. Then vector \mathbf{f} which corresponds to the $N \times N$ square matrix f is given by:

$$\mathbf{f} = \sum_{n=1}^N N_n f \mathbf{V}_n \quad (1.68)$$

It can be shown that if \mathbf{f} is an $N^2 \times 1$ vector, we can write it as an $N \times N$ matrix f , the first column of which is made up from the first N elements of \mathbf{f} , the second column

from the second N elements of \mathbf{f} , and so on, by using the following expression:

$$f = \sum_{n=1}^N N_n^T \mathbf{f} \mathbf{V}_n^T \quad (1.69)$$

Example B1.19

You are given a 3×3 image f and you are asked to use the stacking operator to write it in vector form.

Let us say that:

$$f = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \quad (1.70)$$

We define vectors \mathbf{V}_n and matrices N_n for $n = 1, 2, 3$:

$$\mathbf{V}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{V}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{V}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (1.71)$$

$$N_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad N_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad N_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (1.72)$$

According to equation (1.68):

$$\mathbf{f} = N_1 f \mathbf{V}_1 + N_2 f \mathbf{V}_2 + N_3 f \mathbf{V}_3 \quad (1.73)$$

We shall calculate each term separately:

$$\begin{aligned}
 N_1 f \mathbf{V}_1 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \\
 N_1 f \mathbf{V}_1 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_{11} \\ f_{21} \\ f_{31} \end{pmatrix} = \begin{pmatrix} f_{11} \\ f_{21} \\ f_{31} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{1.74}
 \end{aligned}$$

Similarly:

$$N_2 f \mathbf{V}_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ f_{12} \\ f_{22} \\ f_{32} \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad N_3 f \mathbf{V}_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ f_{13} \\ f_{23} \\ f_{33} \end{pmatrix} \tag{1.75}$$

Then by substituting into (1.73), we obtain vector \mathbf{f} .

Example B1.20

You are given a 9×1 vector \mathbf{f} . Use the stacking operator to write it as a 3×3 matrix.

Let us say that:

$$\mathbf{f} = \begin{pmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{pmatrix} \quad (1.76)$$

According to equation (1.69)

$$f = N_1^T \mathbf{f} \mathbf{V}_1^T + N_2^T \mathbf{f} \mathbf{V}_2^T + N_3^T \mathbf{f} \mathbf{V}_3^T \quad (1.77)$$

(where N_1 , N_2 , N_3 , \mathbf{V}_1 , \mathbf{V}_2 and \mathbf{V}_3 are defined in Box 1.3)

We shall calculate each term separately:

$$\begin{aligned} N_1^T \mathbf{f} \mathbf{V}_1^T &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{pmatrix} (1 \ 0 \ 0) \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_{11} & 0 & 0 \\ f_{21} & 0 & 0 \\ f_{31} & 0 & 0 \\ f_{12} & 0 & 0 \\ f_{22} & 0 & 0 \\ f_{32} & 0 & 0 \\ f_{13} & 0 & 0 \\ f_{23} & 0 & 0 \\ f_{33} & 0 & 0 \end{pmatrix} = \begin{pmatrix} f_{11} & 0 & 0 \\ f_{21} & 0 & 0 \\ f_{31} & 0 & 0 \end{pmatrix} \quad (1.78) \end{aligned}$$

Similarly:

$$N_2^T \mathbf{f} \mathbf{V}_2^T = \begin{pmatrix} 0 & f_{12} & 0 \\ 0 & f_{22} & 0 \\ 0 & f_{32} & 0 \end{pmatrix}, \quad N_3^T \mathbf{f} \mathbf{V}_3^T = \begin{pmatrix} 0 & 0 & f_{13} \\ 0 & 0 & f_{23} \\ 0 & 0 & f_{33} \end{pmatrix} \quad (1.79)$$

Then by substituting into (1.77), we obtain matrix f .

Example B1.21

Show that the stacking operator is linear.

To show that an operator is linear we must show that we shall get the same answer either we apply it to two images w and g and sum up the results with weights α and β , respectively, or we apply it to the weighted sum $\alpha w + \beta g$ directly. We start by applying the stacking operator to composite image $\alpha w + \beta g$, following equation (1.68):

$$\begin{aligned} \sum_{n=1}^N N_n(\alpha w + \beta g) \mathbf{V}_n &= \sum_{n=1}^N N_n(\alpha w \mathbf{V}_n + \beta g \mathbf{V}_n) \\ &= \sum_{n=1}^N (N_n \alpha w \mathbf{V}_n + N_n \beta g \mathbf{V}_n) \\ &= \sum_{n=1}^N N_n \alpha w \mathbf{V}_n + \sum_{n=1}^N N_n \beta g \mathbf{V}_n \end{aligned} \quad (1.80)$$

Since α and β do not depend on the summing index n , they may come out of the summands:

$$\sum_{n=1}^N N_n(\alpha w + \beta g) \mathbf{V}_n = \alpha \sum_{n=1}^N N_n w \mathbf{V}_n + \beta \sum_{n=1}^N N_n g \mathbf{V}_n \quad (1.81)$$

Then, we define vector \mathbf{w} to be the vector version of image w , given by $\sum_{n=1}^N N_n w \mathbf{V}_n$ and vector \mathbf{g} to be the vector version of image g , given by $\sum_{n=1}^N N_n g \mathbf{V}_n$, to obtain:

$$\sum_{n=1}^N N_n(\alpha w + \beta g) \mathbf{V}_n = \alpha \mathbf{w} + \beta \mathbf{g} \quad (1.82)$$

This proves that the stacking operator is linear, because if we apply it separately to images w and g we shall get vectors \mathbf{w} and \mathbf{g} , respectively, which, when added with weights α and β , will produce the result we obtained above by applying the operator to the composite image $\alpha w + \beta g$ directly.

Example B1.22

Consider a 9×9 matrix H that is partitioned into nine 3×3 submatrices. Show that if we multiply it from the left with matrix N_2^T , defined in Box 1.3, we shall extract the second row of its submatrices.

We apply definition (1.67) for $N = 3$ and $n = 2$ to define matrix N_2 and we write explicitly all the elements of matrix H before we perform the multiplication:

$$\begin{aligned}
 N_2^T H &= \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \times \\
 &\quad \left(\begin{array}{ccc|ccc|ccc} h_{11} & h_{12} & h_{13} & | & h_{14} & h_{15} & h_{16} & | & h_{17} & h_{18} & h_{19} \\ h_{21} & h_{22} & h_{23} & | & h_{24} & h_{25} & h_{26} & | & h_{27} & h_{28} & h_{29} \\ h_{31} & h_{32} & h_{33} & | & h_{34} & h_{35} & h_{36} & | & h_{37} & h_{38} & h_{39} \\ \hline - & - & - & | & - & - & - & | & - & - & - \\ h_{41} & h_{42} & h_{43} & | & h_{44} & h_{45} & h_{46} & | & h_{47} & h_{48} & h_{49} \\ h_{51} & h_{52} & h_{53} & | & h_{54} & h_{55} & h_{56} & | & h_{57} & h_{58} & h_{59} \\ h_{61} & h_{62} & h_{63} & | & h_{64} & h_{65} & h_{66} & | & h_{67} & h_{68} & h_{69} \\ \hline - & - & - & | & - & - & - & | & - & - & - \\ h_{71} & h_{72} & h_{73} & | & h_{74} & h_{75} & h_{76} & | & h_{77} & h_{78} & h_{79} \\ h_{81} & h_{82} & h_{83} & | & h_{84} & h_{85} & h_{86} & | & h_{87} & h_{88} & h_{89} \\ h_{91} & h_{92} & h_{93} & | & h_{94} & h_{95} & h_{96} & | & h_{97} & h_{98} & h_{99} \end{array} \right) \\
 &= \begin{pmatrix} h_{41} & h_{42} & h_{43} & | & h_{44} & h_{45} & h_{46} & | & h_{47} & h_{48} & h_{49} \\ h_{51} & h_{52} & h_{53} & | & h_{54} & h_{55} & h_{56} & | & h_{57} & h_{58} & h_{59} \\ h_{61} & h_{62} & h_{63} & | & h_{64} & h_{65} & h_{66} & | & h_{67} & h_{68} & h_{69} \end{pmatrix} \quad (1.83)
 \end{aligned}$$

We note that the result is a matrix made up from the middle row of partitions of the original matrix H .

Example B1.23

Consider a 9×9 matrix H that is partitioned into nine 3×3 submatrices. Show that if we multiply it from the right with matrix N_3 , defined in Box 1.3, we shall extract the third column of its submatrices.

We apply definition (1.67) for $N = 3$ and $n = 3$ to define matrix N_3 and we write explicitly all the elements of matrix H before we perform the multiplication:

$$\begin{aligned}
 & HN_3 = \\
 & \left(\begin{array}{ccc|ccc|ccc} h_{11} & h_{12} & h_{13} & | & h_{14} & h_{15} & h_{16} & | & h_{17} & h_{18} & h_{19} \\ h_{21} & h_{22} & h_{23} & | & h_{24} & h_{25} & h_{26} & | & h_{27} & h_{28} & h_{29} \\ h_{31} & h_{32} & h_{33} & | & h_{34} & h_{35} & h_{36} & | & h_{37} & h_{38} & h_{39} \\ \hline - & - & - & | & - & - & - & | & - & - & - \\ h_{41} & h_{42} & h_{43} & | & h_{44} & h_{45} & h_{46} & | & h_{47} & h_{48} & h_{49} \\ h_{51} & h_{52} & h_{53} & | & h_{54} & h_{55} & h_{56} & | & h_{57} & h_{58} & h_{59} \\ h_{61} & h_{62} & h_{63} & | & h_{64} & h_{65} & h_{66} & | & h_{67} & h_{68} & h_{69} \\ \hline - & - & - & | & - & - & - & | & - & - & - \\ h_{71} & h_{72} & h_{73} & | & h_{74} & h_{75} & h_{76} & | & h_{77} & h_{78} & h_{79} \\ h_{81} & h_{82} & h_{83} & | & h_{84} & h_{85} & h_{86} & | & h_{87} & h_{88} & h_{89} \\ h_{91} & h_{92} & h_{93} & | & h_{94} & h_{95} & h_{96} & | & h_{97} & h_{98} & h_{99} \end{array} \right) \left(\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right) \\
 & = \left(\begin{array}{ccc} h_{17} & h_{18} & h_{19} \\ h_{27} & h_{28} & h_{29} \\ h_{37} & h_{38} & h_{39} \\ \hline - & - & - \\ h_{47} & h_{48} & h_{49} \\ h_{57} & h_{58} & h_{59} \\ h_{67} & h_{68} & h_{69} \\ \hline - & - & - \\ h_{77} & h_{78} & h_{79} \\ h_{87} & h_{88} & h_{89} \\ h_{97} & h_{98} & h_{99} \end{array} \right) \quad (1.84)
 \end{aligned}$$

We observe that the result is a matrix made up from the last column of the partitions of the original matrix H .

Example B1.24

Multiply the 3×9 matrix produced in example 1.22 with the 9×3 matrix produced in example 1.23 and show that the resultant 3×3 matrix is the sum of the individual multiplications of the corresponding partitions.

$$\left(\begin{array}{ccc|ccc|ccc} h_{41} & h_{42} & h_{43} & h_{44} & h_{45} & h_{46} & h_{47} & h_{48} & h_{49} \\ h_{51} & h_{52} & h_{53} & h_{54} & h_{55} & h_{56} & h_{57} & h_{58} & h_{59} \\ h_{61} & h_{62} & h_{63} & h_{64} & h_{65} & h_{66} & h_{67} & h_{68} & h_{69} \end{array} \right) \left(\begin{array}{ccc} h_{17} & h_{18} & h_{19} \\ h_{27} & h_{28} & h_{29} \\ h_{37} & h_{38} & h_{39} \\ \hline - & - & - \\ h_{47} & h_{48} & h_{49} \\ h_{57} & h_{58} & h_{59} \\ h_{67} & h_{68} & h_{69} \\ \hline - & - & - \\ h_{77} & h_{78} & h_{79} \\ h_{87} & h_{88} & h_{89} \\ h_{97} & h_{98} & h_{99} \end{array} \right) =$$

$$\begin{aligned} & \left(\begin{array}{l} h_{41}h_{17} + h_{42}h_{27} + h_{43}h_{37} + h_{44}h_{47} + h_{45}h_{57} + h_{46}h_{67} + h_{47}h_{77} + h_{48}h_{87} + h_{49}h_{97} \\ h_{51}h_{17} + h_{52}h_{27} + h_{53}h_{37} + h_{54}h_{47} + h_{55}h_{57} + h_{56}h_{67} + h_{57}h_{77} + h_{58}h_{87} + h_{59}h_{97} \\ h_{61}h_{17} + h_{62}h_{27} + h_{63}h_{37} + h_{64}h_{47} + h_{65}h_{57} + h_{66}h_{67} + h_{67}h_{77} + h_{68}h_{87} + h_{69}h_{97} \end{array} \right. \\ & \quad \left. \begin{array}{l} h_{41}h_{18} + h_{42}h_{28} + h_{43}h_{38} + h_{44}h_{48} + h_{45}h_{58} + h_{46}h_{68} + h_{47}h_{78} + h_{48}h_{88} + h_{49}h_{98} \\ h_{51}h_{18} + h_{52}h_{28} + h_{53}h_{38} + h_{54}h_{48} + h_{55}h_{58} + h_{56}h_{68} + h_{57}h_{78} + h_{58}h_{88} + h_{59}h_{98} \\ h_{61}h_{18} + h_{62}h_{28} + h_{63}h_{38} + h_{64}h_{48} + h_{65}h_{58} + h_{66}h_{68} + h_{67}h_{78} + h_{68}h_{88} + h_{69}h_{98} \end{array} \right. \\ & \quad \left. \begin{array}{l} h_{41}h_{19} + h_{42}h_{29} + h_{43}h_{39} + h_{44}h_{49} + h_{45}h_{59} + h_{46}h_{69} + h_{47}h_{79} + h_{48}h_{89} + h_{49}h_{99} \\ h_{51}h_{19} + h_{52}h_{29} + h_{53}h_{39} + h_{54}h_{49} + h_{55}h_{59} + h_{56}h_{69} + h_{57}h_{79} + h_{58}h_{89} + h_{59}h_{99} \\ h_{61}h_{19} + h_{62}h_{29} + h_{63}h_{39} + h_{64}h_{49} + h_{65}h_{59} + h_{66}h_{69} + h_{67}h_{79} + h_{68}h_{89} + h_{69}h_{99} \end{array} \right) = \end{aligned}$$

$$\begin{aligned} & \left(\begin{array}{ccc} h_{41}h_{17} + h_{42}h_{27} + h_{43}h_{37} & h_{41}h_{18} + h_{42}h_{28} + h_{43}h_{38} & h_{41}h_{19} + h_{42}h_{29} + h_{43}h_{39} \\ h_{51}h_{17} + h_{52}h_{27} + h_{53}h_{37} & h_{51}h_{18} + h_{52}h_{28} + h_{53}h_{38} & h_{51}h_{19} + h_{52}h_{29} + h_{53}h_{39} \\ h_{61}h_{17} + h_{62}h_{27} + h_{63}h_{37} & h_{61}h_{18} + h_{62}h_{28} + h_{63}h_{38} & h_{61}h_{19} + h_{62}h_{29} + h_{63}h_{39} \end{array} \right) + \\ & \left(\begin{array}{ccc} h_{44}h_{47} + h_{45}h_{57} + h_{46}h_{67} & h_{44}h_{48} + h_{45}h_{58} + h_{46}h_{68} & h_{44}h_{49} + h_{45}h_{59} + h_{46}h_{69} \\ h_{54}h_{47} + h_{55}h_{57} + h_{56}h_{67} & h_{54}h_{48} + h_{55}h_{58} + h_{56}h_{68} & h_{54}h_{49} + h_{55}h_{59} + h_{56}h_{69} \\ h_{64}h_{47} + h_{65}h_{57} + h_{66}h_{67} & h_{64}h_{48} + h_{65}h_{58} + h_{66}h_{68} & h_{64}h_{49} + h_{65}h_{59} + h_{66}h_{69} \end{array} \right) + \\ & \left(\begin{array}{ccc} h_{47}h_{77} + h_{48}h_{87} + h_{49}h_{97} & h_{47}h_{78} + h_{48}h_{88} + h_{49}h_{98} & h_{47}h_{79} + h_{48}h_{89} + h_{49}h_{99} \\ h_{57}h_{77} + h_{58}h_{87} + h_{59}h_{97} & h_{57}h_{78} + h_{58}h_{88} + h_{59}h_{98} & h_{57}h_{79} + h_{58}h_{89} + h_{59}h_{99} \\ h_{67}h_{77} + h_{68}h_{87} + h_{69}h_{97} & h_{67}h_{78} + h_{68}h_{88} + h_{69}h_{98} & h_{67}h_{79} + h_{68}h_{89} + h_{69}h_{99} \end{array} \right) = \end{aligned}$$

$$\begin{aligned} & \left(\begin{array}{ccc} h_{41} & h_{42} & h_{43} \\ h_{51} & h_{52} & h_{53} \\ h_{61} & h_{62} & h_{63} \end{array} \right) \left(\begin{array}{ccc} h_{17} & h_{18} & h_{19} \\ h_{27} & h_{28} & h_{29} \\ h_{37} & h_{38} & h_{39} \end{array} \right) + \left(\begin{array}{ccc} h_{44} & h_{45} & h_{46} \\ h_{54} & h_{55} & h_{56} \\ h_{64} & h_{65} & h_{66} \end{array} \right) \left(\begin{array}{ccc} h_{47} & h_{48} & h_{49} \\ h_{57} & h_{58} & h_{59} \\ h_{67} & h_{68} & h_{69} \end{array} \right) \\ & + \left(\begin{array}{ccc} h_{47} & h_{48} & h_{49} \\ h_{57} & h_{58} & h_{59} \\ h_{67} & h_{68} & h_{69} \end{array} \right) \left(\begin{array}{ccc} h_{77} & h_{78} & h_{79} \\ h_{87} & h_{88} & h_{89} \\ h_{97} & h_{98} & h_{99} \end{array} \right) \end{aligned}$$

Example B1.25

Use the stacking operator to show that the order of two linear operators can be interchanged as long as the multiplication of two circulant matrices is commutative.

Consider an operator with matrix H applied to vector \mathbf{f} constructed from an image f , by using equation (1.68):

$$\tilde{\mathbf{f}} \equiv H\mathbf{f} = H \sum_{n=1}^N N_n f \mathbf{V}_n \quad (1.85)$$

The result is vector $\tilde{\mathbf{f}}$, to which we can apply matrix \hat{H} that corresponds to another linear operator:

$$\tilde{\tilde{\mathbf{f}}} \equiv \hat{H}\tilde{\mathbf{f}} = \hat{H}H \sum_{n=1}^N N_n f \mathbf{V}_n \quad (1.86)$$

From this output vector $\tilde{\tilde{\mathbf{f}}}$ we can construct the output image $\tilde{\tilde{f}}$ by applying equation (1.69):

$$\tilde{\tilde{f}} = \sum_{m=1}^N N_m^T \tilde{\mathbf{f}} \mathbf{V}_m^T \quad (1.87)$$

We may replace $\tilde{\mathbf{f}}$ from (1.86) to obtain:

$$\begin{aligned} \tilde{\tilde{f}} &= \sum_{m=1}^N N_m^T \hat{H}H \sum_{n=1}^N N_n f \mathbf{V}_n \mathbf{V}_m^T \\ &= \sum_{m=1}^N \sum_{n=1}^N (N_m^T \hat{H})(HN_n) f (\mathbf{V}_n \mathbf{V}_m^T) \end{aligned} \quad (1.88)$$

The various factors in (1.88) have been grouped together to facilitate interpretation.

Factor $(N_m^T \hat{H})$ extracts from matrix \hat{H} the m^{th} row of its partitions (see example 1.22), while factor (HN_n) extracts from matrix H the n^{th} column of its partitions (see example 1.23). The product of these two factors extracts the (m, n) partition of matrix $\hat{H}H$, which is a submatrix of size $N \times N$. This submatrix is equal to the sum of the products of the corresponding partitions of matrices $(N_m^T \hat{H})$ and (HN_n) (see example 1.24). Since these products are products of circulant matrices (see example 1.11), the order by which they are multiplied does not matter.

So, we shall get the same answer here either we have $(N_m^T \hat{H})(HN_n)$ or $(N_m^T H)(\hat{H}N_n)$, ie we shall get the same answer whichever way we apply the two linear operators.

What is the implication of the separability assumption on the structure of matrix H ?

According to the separability assumption, we can replace $h(x, \alpha, y, \beta)$ with the product of two functions, $h_c(x, \alpha)h_r(y, \beta)$. Then inside each partition of H in equation (1.39), $h_c(x, \alpha)$ remains constant and we may write for H :

$$\begin{pmatrix} h_{r11} \begin{pmatrix} h_{c11} & \dots & h_{cN1} \\ h_{c12} & \dots & h_{cN2} \\ \vdots & & \vdots \\ h_{c1N} & \dots & h_{cNN} \end{pmatrix} & \dots & h_{rN1} \begin{pmatrix} h_{c11} & \dots & h_{cN1} \\ h_{c12} & \dots & h_{cN2} \\ \vdots & & \vdots \\ h_{c1N} & \dots & h_{cNN} \end{pmatrix} \\ h_{r12} \begin{pmatrix} h_{c11} & \dots & h_{cN1} \\ h_{c12} & \dots & h_{cN2} \\ \vdots & & \vdots \\ h_{c1N} & \dots & h_{cNN} \end{pmatrix} & \dots & h_{rN2} \begin{pmatrix} h_{c11} & \dots & h_{cN1} \\ h_{c12} & \dots & h_{cN2} \\ \vdots & & \vdots \\ h_{c1N} & \dots & h_{cNN} \end{pmatrix} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ h_{r1N} \begin{pmatrix} h_{c11} & \dots & h_{cN1} \\ h_{c12} & \dots & h_{cN2} \\ \vdots & & \vdots \\ h_{c1N} & \dots & h_{cNN} \end{pmatrix} & \dots & h_{rNN} \begin{pmatrix} h_{c11} & \dots & h_{cN1} \\ h_{c12} & \dots & h_{cN2} \\ \vdots & & \vdots \\ h_{c1N} & \dots & h_{cNN} \end{pmatrix} \end{pmatrix} \quad (1.89)$$

Here the arguments of functions $h_c(x, \alpha)$ and $h_r(y, \beta)$ have been written as indices to save space. We say then that matrix H is the **Kronecker product** of matrices h_r^T and h_c^T and we write this as:

$$H = h_r^T \otimes h_c^T \quad (1.90)$$

Example 1.26

Calculate the Kronecker product $A \otimes B$ where

$$A \equiv \begin{pmatrix} 1 & 2 & 3 \\ 4 & 3 & 1 \\ 2 & 4 & 1 \end{pmatrix} \quad B \equiv \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \end{pmatrix} \quad (1.91)$$

$$\begin{aligned}
A \otimes B &= \begin{pmatrix} 1 \times B & 2 \times B & 3 \times B \\ 4 \times B & 3 \times B & 1 \times B \\ 2 \times B & 4 \times B & 1 \times B \end{pmatrix} = \\
&\left(\begin{array}{c} 1 \times \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 2 \times \begin{pmatrix} 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \end{pmatrix} & 2 \times \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \end{pmatrix} & 3 \times \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \end{pmatrix} \\ \end{array} \right) = \\
&\left(\begin{array}{c} 1 \times \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \end{pmatrix} & 2 \times \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \end{pmatrix} & 3 \times \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 3 \end{pmatrix} \\ \end{array} \right) = \\
&\left(\begin{array}{c} \begin{pmatrix} 2 & 0 & 1 & 4 & 0 & 2 & 6 & 0 & 3 \\ 0 & 1 & 3 & 0 & 2 & 6 & 0 & 3 & 9 \\ 2 & 1 & 0 & 4 & 2 & 0 & 6 & 3 & 0 \\ 8 & 0 & 4 & 6 & 0 & 3 & 2 & 0 & 1 \\ 0 & 4 & 12 & 0 & 3 & 9 & 0 & 1 & 3 \\ 8 & 4 & 0 & 6 & 3 & 0 & 2 & 1 & 0 \\ 4 & 0 & 2 & 8 & 0 & 4 & 2 & 0 & 1 \\ 0 & 2 & 6 & 0 & 4 & 12 & 0 & 1 & 3 \\ 4 & 2 & 0 & 8 & 4 & 0 & 2 & 1 & 0 \end{pmatrix} \end{array} \right) \quad (1.92)
\end{aligned}$$

How can a separable transform be written in matrix form?

Consider again equation (1.19) which expresses the separable linear transform of an image:

$$g(\alpha, \beta) = \sum_{x=1}^N h_c(x, \alpha) \sum_{y=1}^N f(x, y) h_r(y, \beta) \quad (1.93)$$

Notice that factor $\sum_{y=1}^N f(x, y) h_r(y, \beta)$ actually represents the product $f h_r$ of two $N \times N$ matrices, which must be another matrix $s \equiv f h_r$ of the same size. Let us define an element of s as:

$$s(x, \beta) \equiv \sum_{y=1}^N f(x, y) h_r(y, \beta) = f h_r \quad (1.94)$$

Then (1.93) may be written as:

$$g(\alpha, \beta) = \sum_{x=1}^N h_c(x, \alpha) s(x, \beta) \quad (1.95)$$

Thus, in matrix form:

$$g = h_c^T s = h_c^T f h_r \quad (1.96)$$

What is the meaning of the separability assumption?

Let us assume that operator \mathcal{O} has point spread function $h(x, \alpha, y, \beta)$, which is separable. The separability assumption implies that operator \mathcal{O} operates on the rows of the image matrix f independently from the way it operates on its columns. These independent operations are expressed by the two matrices h_r and h_c , respectively. That is why we chose subscripts r and c to denote these matrices ($r = \text{rows}$, $c = \text{columns}$). Matrix h_r is used to multiply the image from the right. Ordinary matrix multiplication then means that the *rows* of the image are multiplied with it. Matrix h_c is used to multiply the image from the left. Thus, the *columns* of the image are multiplied with it.

Example B1.27

Are the operators which correspond to matrices H , \hat{H} and \tilde{H} given by equations (1.43), (1.45) and (1.47), respectively, separable?

If an operator is separable, we must be able to write its H matrix as the Kronecker product of two matrices. In other words, we must check whether from every submatrix of H we can get out a common factor, such that the submatrix that remains is the same for all partitions. We can see that this is possible for matrix \hat{H} , but it is not possible for matrices H and \tilde{H} . For example, some of the partitions of matrices H and \tilde{H} are diagonal, while others are not. It is impossible to express them then as the product of a scalar with the same 3×3 matrix. On the other hand, all partitions of \hat{H} are diagonal (or zero), so we can see that the common factors we can get out from each partition form matrix

$$A \equiv \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{pmatrix} \quad (1.97)$$

while the common matrix that is multiplied in each partition by these coefficients is the 3×3 identity matrix:

$$I \equiv \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1.98)$$

So, we may write:

$$\hat{H} = A \otimes I \quad (1.99)$$

Box 1.4. The formal derivation of the separable matrix equation

We can use equations (1.68) and (1.69) with (1.38) as follows. First, express the output image g using (1.69) in terms of \mathbf{g} :

$$g = \sum_{m=1}^N N_m^T \mathbf{g} \mathbf{V}_m^T \quad (1.100)$$

Then express \mathbf{g} in terms of H and \mathbf{f} from (1.38) and replace \mathbf{f} in terms of f using (1.68):

$$\mathbf{g} = H \sum_{n=1}^N N_n f \mathbf{V}_n \quad (1.101)$$

Substitute (1.101) into (1.100) and group factors with the help of brackets to obtain:

$$g = \sum_{m=1}^N \sum_{n=1}^N (N_m^T H N_n) f(\mathbf{V}_n \mathbf{V}_m^T) \quad (1.102)$$

H is a $N^2 \times N^2$ matrix. We may think of it as partitioned in $N \times N$ submatrices stacked together. Then it can be shown that $N_m^T H N_n$ is the H_{mn} such submatrix (see example 1.28).

Under the separability assumption, matrix H is the Kronecker product of matrices h_c and h_r :

$$H = h_c^T \otimes h_r^T \quad (1.103)$$

Then partition H_{mn} is essentially $h_r^T(m, n)h_c^T$. If we substitute this in (1.102), we obtain:

$$\begin{aligned} g &= \sum_{m=1}^N \sum_{n=1}^N \underbrace{h_r^T(m, n) h_c^T}_{\text{a scalar}} f(\mathbf{V}_n \mathbf{V}_m^T) \\ \Rightarrow g &= h_c^T f \sum_{m=1}^N \sum_{n=1}^N h_r^T(m, n) \mathbf{V}_n \mathbf{V}_m^T \end{aligned} \quad (1.104)$$

Product $\mathbf{V}_n \mathbf{V}_m^T$ is the product between an $N \times 1$ matrix with the only nonzero element at position n , with a $1 \times N$ matrix, with the only nonzero element at position m . So, it is an $N \times N$ square matrix with the only nonzero element at position (n, m) . When multiplied with $h_r^T(m, n)$, it places the (m, n) element of the h_r^T matrix in position (n, m) and sets to zero all other elements. The sum over all m 's and n 's is matrix h_r . So, from (1.104) we have:

$$g = h_c^T f h_r \quad (1.105)$$

Example 1.28

You are given a 9×9 matrix H which is partitioned into nine 3×3 submatrices. Show that $N_2^T H N_3$, where N_2 and N_3 are matrices of the stacking operator, is partition H_{23} of matrix H .

$$H \equiv \begin{pmatrix} h_{11} & h_{12} & h_{13} & | & h_{14} & h_{15} & h_{16} & | & h_{17} & h_{18} & h_{19} \\ h_{21} & h_{22} & h_{23} & | & h_{24} & h_{25} & h_{26} & | & h_{27} & h_{28} & h_{29} \\ h_{31} & h_{32} & h_{33} & | & h_{34} & h_{35} & h_{36} & | & h_{37} & h_{38} & h_{39} \\ - & - & - & | & - & - & - & | & - & - & - \\ h_{41} & h_{42} & h_{43} & | & h_{44} & h_{45} & h_{46} & | & h_{47} & h_{48} & h_{49} \\ h_{51} & h_{52} & h_{53} & | & h_{54} & h_{55} & h_{56} & | & h_{57} & h_{58} & h_{59} \\ h_{61} & h_{62} & h_{63} & | & h_{64} & h_{65} & h_{66} & | & h_{67} & h_{68} & h_{69} \\ - & - & - & | & - & - & - & | & - & - & - \\ h_{71} & h_{72} & h_{73} & | & h_{74} & h_{75} & h_{76} & | & h_{77} & h_{78} & h_{79} \\ h_{81} & h_{82} & h_{83} & | & h_{84} & h_{85} & h_{86} & | & h_{87} & h_{88} & h_{89} \\ h_{91} & h_{92} & h_{93} & | & h_{94} & h_{95} & h_{96} & | & h_{97} & h_{98} & h_{99} \end{pmatrix} \quad (1.106)$$

$$\begin{aligned} N_2^T H N_3 &= \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} & h_{16} & h_{17} & h_{18} & h_{19} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} & h_{26} & h_{27} & h_{28} & h_{29} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} & h_{36} & h_{37} & h_{38} & h_{39} \\ h_{41} & h_{42} & h_{43} & h_{44} & h_{45} & h_{46} & h_{47} & h_{48} & h_{49} \\ h_{51} & h_{52} & h_{53} & h_{54} & h_{55} & h_{56} & h_{57} & h_{58} & h_{59} \\ h_{61} & h_{62} & h_{63} & h_{64} & h_{65} & h_{66} & h_{67} & h_{68} & h_{69} \\ h_{71} & h_{72} & h_{73} & h_{74} & h_{75} & h_{76} & h_{77} & h_{78} & h_{79} \\ h_{81} & h_{82} & h_{83} & h_{84} & h_{85} & h_{86} & h_{87} & h_{88} & h_{89} \\ h_{91} & h_{92} & h_{93} & h_{94} & h_{95} & h_{96} & h_{97} & h_{98} & h_{99} \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} h_{17} & h_{18} & h_{19} \\ h_{27} & h_{28} & h_{29} \\ h_{37} & h_{38} & h_{39} \\ h_{47} & h_{48} & h_{49} \\ h_{57} & h_{58} & h_{59} \\ h_{67} & h_{68} & h_{69} \\ h_{77} & h_{78} & h_{79} \\ h_{87} & h_{88} & h_{89} \\ h_{97} & h_{98} & h_{99} \end{pmatrix} \\ &= \begin{pmatrix} h_{47} & h_{48} & h_{49} \\ h_{57} & h_{58} & h_{59} \\ h_{67} & h_{68} & h_{69} \end{pmatrix} = H_{23} \end{aligned} \quad (1.107)$$

What is the “take home” message of this chapter?

Under the assumption that the operator with which we manipulate an image is *linear* and *separable*, this operation may be expressed by an equation of the form

$$g = h_c^T f h_r \quad (1.108)$$

where f and g are the input and output images, respectively, and h_c and h_r are matrices expressing the point spread function of the operator.

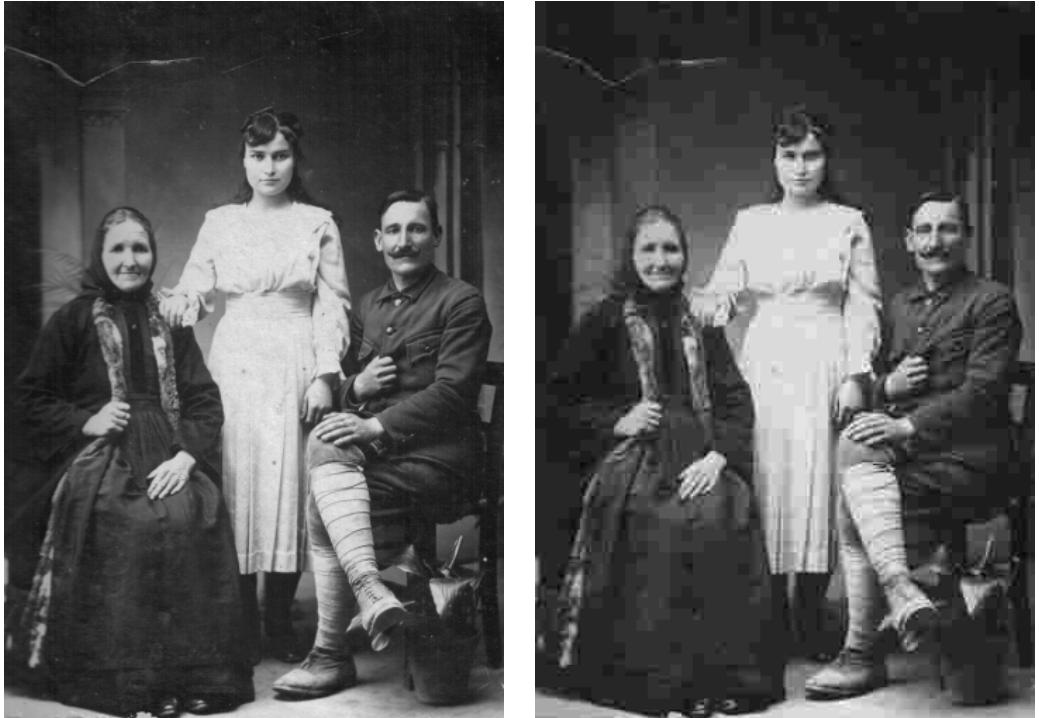


Figure 1.7: The original image “Ancestors” and a compressed version of it.

What is the significance of equation (1.108) in linear image processing?

In linear image processing we are trying to solve the following four problems in relation to equation (1.108).

- Given an image f , choose matrices h_c and h_r so that the output image g is “better” than f , according to some subjective criteria. This is the problem of **image enhancement**. Linear methods are not very successful here. Most of image enhancement is done with the help of nonlinear methods.

- Given an image f , choose matrices h_c and h_r so that g can be represented by fewer bits than f , without much loss of detail. This is the problem of **image compression**. Quite a few image compression methods rely on such an approach.
- Given an image g and an estimate of $h(x, \alpha, y, \beta)$, recover image f . This is the problem of **image restoration**. A lot of commonly used approaches to image restoration follow this path.
- Given an image f , choose matrices h_c and h_r so that output image g salientizes certain features of f . This is the problem of **feature extraction**. Algorithms that attempt to do that often include a linear step that can be expressed by equation (1.108), but most of the times they also include nonlinear components.

Figures 1.7–1.11 show examples of these processes.



Figure 1.8: The original image “Birthday” and its enhanced version.

What is this book about?

This book is about introducing the mathematical foundations of image processing in the context of specific applications in the four main themes of image processing as identified above. The themes of image enhancement, image restoration and feature extraction will be discussed in detail. The theme of image compression is only touched upon as this could be the topic of a whole book on its own. This book puts emphasis on linear methods, but several nonlinear techniques relevant to image enhancement, image restoration and feature extraction will also be presented.



Figure 1.9: The blurred original image “Hara” and its restored version.

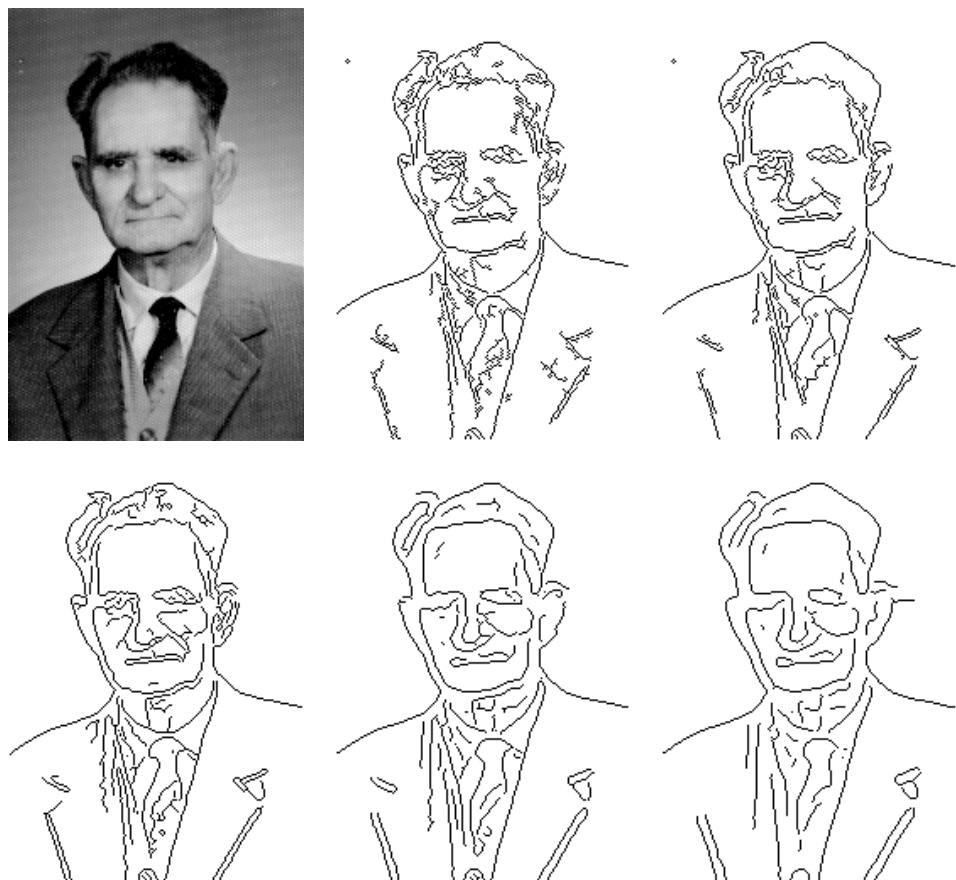


Figure 1.10: The original image “Mitsos” and its edge maps of decreasing detail (indicating locations where the brightness of the image changes abruptly).



(a) Image "Siblings"



(b) Thresholding and binarisation



(c) Gradient magnitude



(d) Region segmentation

Figure 1.11: There are various ways to reduce the information content of an image and salient aspects of interest for further analysis.

Chapter 2

Image Transformations

What is this chapter about?

This chapter is concerned with the development of some of the most important tools of linear image processing, namely the ways by which we express an image as the linear superposition of some elementary images.

How can we define an elementary image?

There are many ways to do that. We already saw one in Chapter 1: an elementary image has all its pixels black except one that has value 1. By shifting the position of the nonzero pixel to all possible positions, we may create N^2 different such elementary images in terms of which we may expand any $N \times N$ image. In this chapter, we shall use more sophisticated elementary images and define an elementary image as the **outer product** of two vectors.

What is the outer product of two vectors?

Consider two vectors $N \times 1$:

$$\begin{aligned}\mathbf{u}_i^T &= (u_{i1}, u_{i2}, \dots, u_{iN}) \\ \mathbf{v}_j^T &= (v_{j1}, v_{j2}, \dots, v_{jN})\end{aligned}\tag{2.1}$$

Their outer product is defined as:

$$\mathbf{u}_i \mathbf{v}_j^T = \begin{pmatrix} u_{i1} \\ u_{i2} \\ \vdots \\ u_{iN} \end{pmatrix} \begin{pmatrix} v_{j1} & v_{j2} & \dots & v_{jN} \end{pmatrix} = \begin{pmatrix} u_{i1}v_{j1} & u_{i1}v_{j2} & \dots & u_{i1}v_{jN} \\ u_{i2}v_{j1} & u_{i2}v_{j2} & \dots & u_{i2}v_{jN} \\ \vdots & \vdots & & \vdots \\ u_{iN}v_{j1} & u_{iN}v_{j2} & \dots & u_{iN}v_{jN} \end{pmatrix}\tag{2.2}$$

Therefore, the outer product of these two vectors is an $N \times N$ matrix which may be thought of as an image.

How can we expand an image in terms of vector outer products?

We saw in the previous chapter that a general separable linear transformation of an image matrix f may be written as

$$g = h_c^T f h_r \quad (2.3)$$

where g is the output image and h_c and h_r are the transforming matrices.

We may use the inverse matrices of h_c^T and h_r to solve this expression for f in terms of g , as follows: multiply both sides of the equation with $(h_c^T)^{-1}$ on the left and with h_r^{-1} on the right:

$$(h_c^T)^{-1} g h_r^{-1} = (h_c^T)^{-1} h_c^T f h_r h_r^{-1} = f \quad (2.4)$$

Thus we write:

$$f = (h_c^T)^{-1} g h_r^{-1} \quad (2.5)$$

Let us assume that we partition matrices $(h_c^T)^{-1}$ and h_r^{-1} in their column and row vectors, respectively:

$$(h_c^T)^{-1} \equiv (\mathbf{u}_1 | \mathbf{u}_2 | \dots | \mathbf{u}_N), \quad h_r^{-1} \equiv \begin{pmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_N^T \end{pmatrix} \quad (2.6)$$

Then:

$$f = (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_N) g \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_N^T \end{pmatrix} \quad (2.7)$$

We may also write matrix g as the sum of N^2 , $N \times N$ matrices, each one having only one nonzero element:

$$g = \begin{pmatrix} g_{11} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} + \begin{pmatrix} 0 & g_{12} & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} + \dots + \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & g_{NN} \end{pmatrix} \quad (2.8)$$

Then equation (2.7) may be written as:

$$f = \sum_{i=1}^N \sum_{j=1}^N g_{ij} \mathbf{u}_i \mathbf{v}_j^T \quad (2.9)$$

This is an expansion of image f in terms of vector outer products. The outer product $\mathbf{u}_i \mathbf{v}_j^T$ may be interpreted as an “image” so that the sum over all combinations of the outer products, appropriately weighted by the g_{ij} coefficients, represents the original image f .

Example 2.1

Derive the term with $i = 2$ and $j = 1$ on the right-hand side of equation (2.9).

Let us denote by $u_{i1}, u_{i2}, \dots, u_{iN}$ the elements of vector \mathbf{u}_i and by $v_{i1}, v_{i2}, \dots, v_{iN}$ the elements of vector \mathbf{v}_i .

If we substitute g from equation (2.8) into equation (2.7), the right-hand side of equation (2.7) will consist of N^2 terms of similar form. One such term is:

$$\begin{aligned}
 & (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_N) \begin{pmatrix} 0 & 0 & \dots & 0 \\ g_{21} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_N^T \end{pmatrix} \\
 &= (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_N) \begin{pmatrix} 0 & 0 & \dots & 0 \\ g_{21} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \begin{pmatrix} v_{11} & v_{12} & \dots & v_{1N} \\ v_{21} & v_{22} & \dots & v_{2N} \\ \vdots & \vdots & & \vdots \\ v_{N1} & v_{N2} & \dots & v_{NN} \end{pmatrix} \\
 &= \begin{pmatrix} u_{11} & u_{21} & \dots & u_{N1} \\ u_{12} & u_{22} & \dots & u_{N2} \\ \vdots & \vdots & & \vdots \\ u_{1N} & u_{2N} & \dots & u_{NN} \end{pmatrix} \begin{pmatrix} 0 & 0 & \dots & 0 \\ g_{21}v_{11} & g_{21}v_{12} & \dots & g_{21}v_{1N} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \\
 &= \begin{pmatrix} u_{21}g_{21}v_{11} & u_{21}g_{21}v_{12} & \dots & u_{21}g_{21}v_{1N} \\ u_{22}g_{21}v_{11} & u_{22}g_{21}v_{12} & \dots & u_{22}g_{21}v_{1N} \\ \dots & \dots & & \dots \\ u_{2N}g_{21}v_{11} & u_{2N}g_{21}v_{12} & \dots & u_{2N}g_{21}v_{1N} \end{pmatrix} \\
 &= g_{21} \begin{pmatrix} u_{21}v_{11} & u_{21}v_{12} & \dots & u_{21}v_{1N} \\ u_{22}v_{11} & u_{22}v_{12} & \dots & u_{22}v_{1N} \\ \dots & \dots & & \dots \\ u_{2N}v_{11} & u_{2N}v_{12} & \dots & u_{2N}v_{1N} \end{pmatrix} = g_{21}\mathbf{u}_2\mathbf{v}_1^T
 \end{aligned}$$

How do we choose matrices h_c and h_r ?

There are various options for the choice of matrices h_c and h_r , according to what we wish to achieve. For example, we may choose them so that the transformed image may be represented by fewer bits than the original one, or we may choose them so that truncation of the expansion of the original image smooths it by omitting its high frequency components, or optimally approximates it according to some predetermined criterion. It is often convenient to choose matrices h_c and h_r to be **unitary** so that the transform is easily invertible. If matrices h_c and h_r are chosen to be unitary, equation (2.3) represents a **unitary transform** of f , and g is termed the **unitary transform domain** of image f .

What is a unitary matrix?

A matrix U is called unitary if its inverse is the complex conjugate of its transpose, ie

$$UU^{T*} = I \quad (2.10)$$

where I is the unit matrix. We sometimes write superscript “ H ” instead of “ $T*$ ” and call $U^{T*} \equiv U^H$ the **Hermitian transpose** or **conjugate transpose** of matrix U .

If the elements of the matrix are real numbers, we use the term **orthogonal** instead of unitary.

What is the inverse of a unitary transform?

If matrices h_c and h_r in (2.3) are unitary, then the inverse of it is:

$$f = h_c^* g h_r^H \quad (2.11)$$

For simplicity, from now and on we shall write U instead of h_c and V instead of h_r , so that the expansion of an image f in terms of vector outer products may be written as:

$$f = U^* g V^H \quad (2.12)$$

How can we construct a unitary matrix?

If we consider equation (2.10), we see that for matrix U to be unitary, the requirement is that the dot product of any of its columns with the complex conjugate of any other column must be zero, while the magnitude of any of its column vectors must be 1. In other words, U is unitary if its columns form a set of **orthonormal** vectors.

How should we choose matrices U and V so that g can be represented by fewer bits than f ?

If we want to represent image f with fewer than N^2 number of elements, we may choose matrices U and V so that the transformed image g is a diagonal matrix. Then we could represent image f with the help of equation (2.9) using only the N nonzero elements of g . This can be achieved with a process called **matrix diagonalisation** and it is called **Singular Value Decomposition (SVD)** of the image.

What is matrix diagonalisation?

Diagonalisation of a matrix A is the process by which we identify two matrices A_u and A_v so that matrix $A_u A v \equiv J$ is diagonal.

Can we diagonalise any matrix?

In general no. For a start, a matrix has to be square in order to be diagonalisable. If a matrix is square and symmetric, then we can always diagonalise it.

2.1 Singular value decomposition

How can we diagonalise an image?

An image is not always square and almost never symmetric. We cannot, therefore, apply matrix diagonalisation directly. What we do is to create a symmetric matrix from it, which is then diagonalised. The symmetric matrix we create out of an image g is gg^T (see example 2.2). The matrices, which help us then express an image as the sum of vector outer products, are constructed from matrix gg^T , rather than from the image itself directly. This is the process of **Singular Value Decomposition (SVD)**. It can be shown then (see Box 2.1), that if gg^T is a matrix of rank r , matrix g can be written as

$$g = U\Lambda^{\frac{1}{2}}V^T \quad (2.13)$$

where U and V are orthogonal matrices of size $N \times r$ and $\Lambda^{\frac{1}{2}}$ is a diagonal $r \times r$ matrix.

Example 2.2

You are given an image which is represented by a matrix g . Show that matrix gg^T is symmetric.

A matrix is symmetric when it is equal to its transpose. Therefore, we must show that the transpose of gg^T is equal to gg^T . Consider the transpose of gg^T :

$$(gg^T)^T = (g^T)^T g^T = gg^T \quad (2.14)$$

Example B2.3

If Λ is a diagonal 2×2 matrix and Λ^m is defined by putting all nonzero elements of Λ to the power of m , show that:

$$\Lambda^{-\frac{1}{2}}\Lambda\Lambda^{-\frac{1}{2}} = I \quad \text{and} \quad \Lambda^{-\frac{1}{2}}\Lambda^{\frac{1}{2}} = I. \quad (2.15)$$

Indeed:

$$\begin{aligned} \Lambda^{-\frac{1}{2}}\Lambda\Lambda^{-\frac{1}{2}} &= \begin{pmatrix} \lambda_1^{-\frac{1}{2}} & 0 \\ 0 & \lambda_2^{-\frac{1}{2}} \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} \lambda_1^{-\frac{1}{2}} & 0 \\ 0 & \lambda_2^{-\frac{1}{2}} \end{pmatrix} \\ &= \begin{pmatrix} \lambda_1^{-\frac{1}{2}} & 0 \\ 0 & \lambda_2^{-\frac{1}{2}} \end{pmatrix} \begin{pmatrix} \lambda_1^{\frac{1}{2}} & 0 \\ 0 & \lambda_2^{\frac{1}{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned} \quad (2.16)$$

This also shows that $\Lambda^{-\frac{1}{2}}\Lambda^{\frac{1}{2}} = I$.

Example B2.4

Assume that H is a 3×3 matrix and partition it into a 2×3 submatrix H_1 and a 1×3 submatrix H_2 . Show that:

$$H^T H = H_1^T H_1 + H_2^T H_2 \quad (2.17)$$

Let us say that:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ \tilde{h}_{31} & \tilde{h}_{32} & \tilde{h}_{33} \end{pmatrix}, \quad H_1 \equiv \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \end{pmatrix}, \quad \text{and} \quad H_2 \equiv (\tilde{h}_{31} \quad \tilde{h}_{32} \quad \tilde{h}_{33}) \quad (2.18)$$

We start by computing the left-hand side of (2.17):

$$H^T H = \begin{pmatrix} h_{11} & h_{21} & \tilde{h}_{31} \\ h_{12} & h_{22} & \tilde{h}_{32} \\ h_{13} & h_{23} & \tilde{h}_{33} \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ \tilde{h}_{31} & \tilde{h}_{32} & \tilde{h}_{33} \end{pmatrix} =$$

$$\begin{pmatrix} h_{11}^2 + h_{21}^2 + \tilde{h}_{31}^2 & h_{11}h_{12} + h_{21}h_{22} + \tilde{h}_{31}\tilde{h}_{32} & h_{11}h_{13} + h_{21}h_{23} + \tilde{h}_{31}\tilde{h}_{33} \\ h_{12}h_{11} + h_{22}h_{21} + \tilde{h}_{32}\tilde{h}_{31} & h_{12}^2 + h_{22}^2 + \tilde{h}_{32}^2 & h_{12}h_{13} + h_{22}h_{23} + \tilde{h}_{32}\tilde{h}_{33} \\ h_{13}h_{11} + h_{23}h_{21} + \tilde{h}_{33}\tilde{h}_{31} & h_{13}h_{12} + h_{23}h_{22} + \tilde{h}_{33}\tilde{h}_{32} & h_{13}^2 + h_{23}^2 + \tilde{h}_{33}^2 \end{pmatrix} \quad (2.19)$$

Next, we compute the right-hand side of (2.17), by computing each term separately:

$$\begin{aligned} H_1^T H_1 &= \begin{pmatrix} h_{11} & h_{21} \\ h_{12} & h_{22} \\ h_{13} & h_{23} \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \end{pmatrix} \\ &= \begin{pmatrix} h_{11}^2 + h_{21}^2 & h_{11}h_{12} + h_{21}h_{22} & h_{11}h_{13} + h_{21}h_{23} \\ h_{12}h_{11} + h_{22}h_{21} & h_{12}^2 + h_{22}^2 & h_{12}h_{13} + h_{22}h_{23} \\ h_{13}h_{11} + h_{23}h_{21} & h_{13}h_{12} + h_{23}h_{22} & h_{13}^2 + h_{23}^2 \end{pmatrix} \quad (2.20) \end{aligned}$$

$$H_2^T H_2 = \begin{pmatrix} \tilde{h}_{31} \\ \tilde{h}_{32} \\ \tilde{h}_{33} \end{pmatrix} (\tilde{h}_{31} \quad \tilde{h}_{32} \quad \tilde{h}_{33}) = \begin{pmatrix} \tilde{h}_{31}^2 & \tilde{h}_{31}\tilde{h}_{32} & \tilde{h}_{31}\tilde{h}_{33} \\ \tilde{h}_{32}\tilde{h}_{31} & \tilde{h}_{32}^2 & \tilde{h}_{32}\tilde{h}_{33} \\ \tilde{h}_{33}\tilde{h}_{31} & \tilde{h}_{33}\tilde{h}_{32} & \tilde{h}_{33}^2 \end{pmatrix} \quad (2.21)$$

Adding $H_1^T H_1$ and $H_2^T H_2$ we obtain the same answer as the one we obtained by calculating the left-hand side of equation (2.17) directly.

Example B2.5

Show that if we partition an $N \times N$ matrix S into an $r \times N$ submatrix S_1 and an $(N - r) \times N$ submatrix S_2 , equation

$$SAS^T = \begin{pmatrix} S_1 AS_1^T & | & S_1 AS_2^T \\ \hline S_2 AS_1^T & | & S_2 AS_2^T \end{pmatrix} \quad (2.22)$$

is correct, with A being an $N \times N$ matrix.

Trivially:

$$SAS^T = \begin{pmatrix} S_1 \\ \hline S_2 \end{pmatrix} A \begin{pmatrix} S_1^T & | & S_2^T \end{pmatrix} \quad (2.23)$$

Consider the multiplication of A with $(S_1^T \mid S_2^T)$. The rows of A will be multiplied with the columns of $(S_1^T \mid S_2^T)$. Schematically:

$$\begin{pmatrix} \dots \\ \dots \\ \dots \\ \dots \end{pmatrix} \begin{pmatrix} \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \end{pmatrix} \quad (2.24)$$

Then it becomes clear that the result will be $(AS_1^T \mid AS_2^T)$. Next we consider the multiplication of $\begin{pmatrix} S_1 \\ \hline S_2 \end{pmatrix}$ with $(AS_1^T \mid AS_2^T)$. The rows of $\begin{pmatrix} S_1 \\ \hline S_2 \end{pmatrix}$ will multiply the columns of $(AS_1^T \mid AS_2^T)$. Schematically:

$$\begin{pmatrix} \dots \\ \dots \\ \dots \\ \dots \\ \hline \dots \\ \dots \\ \dots \\ \dots \\ \dots \end{pmatrix} \begin{pmatrix} \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & | & \vdots & \vdots & \vdots \end{pmatrix} \quad (2.25)$$

Then the result is obvious.

Example B2.6

Show that if $Agg^T A^T = 0$ then $Ag = 0$, where A and g are an $r \times N$ and an $N \times N$ real matrix, respectively.

We may write

$$Agg^T A^T = Ag(Ag)^T = 0 \quad (2.26)$$

Ag is an $r \times N$ matrix. Let us call it B . We have, therefore, $BB^T = 0$:

$$\begin{aligned} & \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1r} \\ b_{21} & b_{22} & \dots & b_{2r} \\ \dots & \dots & \dots & \dots \\ b_{r1} & b_{r2} & \dots & b_{rr} \end{pmatrix} \begin{pmatrix} b_{11} & b_{21} & \dots & b_{r1} \\ b_{12} & b_{22} & \dots & b_{r2} \\ \dots & \dots & \dots & \dots \\ b_{1r} & b_{2r} & \dots & b_{rr} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{pmatrix} \Rightarrow \\ & \begin{pmatrix} b_{11}^2 + b_{12}^2 + \dots + b_{1r}^2 & \dots & \dots & \dots \\ \dots & b_{21}^2 + b_{22}^2 + \dots + b_{2r}^2 & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & b_{r1}^2 + b_{r2}^2 + \dots + b_{rr}^2 \end{pmatrix} \\ & = \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{pmatrix} \end{aligned} \quad (2.27)$$

Equating the corresponding elements, we obtain, for example:

$$b_{11}^2 + b_{12}^2 + \dots + b_{1r}^2 = 0 \quad (2.28)$$

The only way that the sum of the squares of r real numbers can be 0 is if each one of them is 0. Similarly for all other diagonal elements of BB^T . This means that $B = 0$, ie that $Ag = 0$.

Box 2.1. Can we expand in vector outer products any image?

Yes. Consider an image g and its transpose g^T . Matrix gg^T is real and symmetric (see example 2.2) and let us say that it has r nonzero eigenvalues. Let λ_i be its i^{th} eigenvalue. Then it is known, from linear algebra, that there exists an orthogonal matrix S (made up from the eigenvectors of gg^T) such that:

$$\begin{aligned}
Sgg^T S^T &= \left(\begin{array}{cccc|cccc} \lambda_1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \lambda_r & 0 & 0 & \dots & 0 \\ \hline - & - & - & - & - & - & - & - \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \end{array} \right) \\
&= \begin{pmatrix} \Lambda & | & \mathbf{0} \\ \mathbf{0} & | & \mathbf{0} \end{pmatrix} \tag{2.29}
\end{aligned}$$

where Λ and $\mathbf{0}$ represent the partitions of the diagonal matrix above. Similarly, we can partition matrix S to an $r \times N$ matrix S_1 and an $(N - r) \times N$ matrix S_2 :

$$S = \begin{pmatrix} S_1 \\ - \\ S_2 \end{pmatrix} \tag{2.30}$$

Because S is orthogonal, and by using the result of example 2.4, we have:

$$\begin{aligned}
S^T S &= I \Rightarrow S_1^T S_1 + S_2^T S_2 = I \Rightarrow \\
S_1^T S_1 &= I - S_2^T S_2 \Rightarrow S_1^T S_1 g = g - S_2^T S_2 g
\end{aligned} \tag{2.31}$$

From (2.29) and examples 2.5 and 2.6 we clearly have:

$$S_1 g g^T S_1^T = \Lambda \tag{2.32}$$

$$S_2 g g^T S_2^T = 0 \Rightarrow S_2 g = \mathbf{0} \tag{2.33}$$

Using (2.33) into (2.31) we have:

$$S_1^T S_1 g = g \tag{2.34}$$

This means that $S_1^T S_1 = I$, ie S_1 is an orthogonal matrix. We multiply both sides of equation (2.32) from left and right with $\Lambda^{-\frac{1}{2}}$ to get:

$$\Lambda^{-\frac{1}{2}} S_1 g g^T S_1^T \Lambda^{-\frac{1}{2}} = \Lambda^{-\frac{1}{2}} \Lambda \Lambda^{-\frac{1}{2}} = I \tag{2.35}$$

Since $\Lambda^{-\frac{1}{2}}$ is diagonal, $\Lambda^{-\frac{1}{2}} = (\Lambda^{-\frac{1}{2}})^T$. So the above equation may be rewritten as:

$$\Lambda^{-\frac{1}{2}} S_1 g (\Lambda^{-\frac{1}{2}} S_1 g)^T = I \tag{2.36}$$

Therefore, there exists a matrix $q \equiv \Lambda^{-\frac{1}{2}}S_1g$, the inverse of which is its transpose (ie it is orthogonal). We may express matrix S_1g as $\Lambda^{\frac{1}{2}}q$ and substitute in (2.34) to obtain:

$$S_1^T \Lambda^{\frac{1}{2}} q = g \quad \text{or} \quad g = S_1^T \Lambda^{\frac{1}{2}} q \quad (2.37)$$

In other words, g is expressed as a diagonal matrix $\Lambda^{\frac{1}{2}}$ made up from the square roots of the nonzero eigenvalues of gg^T , multiplied from left and right with the two orthogonal matrices S_1 and q . This result expresses the diagonalisation of image g .

How can we compute matrices U , V and $\Lambda^{\frac{1}{2}}$ needed for image diagonalisation?

We know, from linear algebra, that matrix diagonalisation means that a real square matrix A may be written as $U\Lambda U^T$, where U is made up from the eigenvectors of A written as columns, and Λ is a diagonal matrix made up from the eigenvalues of A written along the diagonal in the order corresponding to the eigenvectors that make up the columns of U . We need this information in the proof that follows.

If we take the transpose of (2.13) we have:

$$g^T = V\Lambda^{\frac{1}{2}}U^T \quad (2.38)$$

Multiply (2.13) with (2.38) by parts, to obtain:

$$gg^T = U\Lambda^{\frac{1}{2}}V^T V\Lambda^{\frac{1}{2}}U^T = U\Lambda^{\frac{1}{2}}I\Lambda^{\frac{1}{2}}U^T = U\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}U^T = U\Lambda U^T \quad (2.39)$$

This shows that matrix Λ consists of the r nonzero eigenvalues of matrix gg^T while U is made up from the eigenvectors of the same matrix.

Similarly, if we multiply (2.38) with (2.13) by parts, we get:

$$g^T g = V\Lambda V^T \quad (2.40)$$

This shows that matrix V is made up from the eigenvectors of matrix $g^T g$.

Box 2.2. What happens if the eigenvalues of matrix gg^T are negative?

We shall show that the eigenvalues of gg^T are always non-negative numbers. Let us assume that λ is an eigenvalue of matrix gg^T and \mathbf{u} is the corresponding eigenvector. We have then:

$$gg^T \mathbf{u} = \lambda \mathbf{u} \quad (2.41)$$

Multiply both sides with \mathbf{u}^T from the left:

$$\mathbf{u}^T gg^T \mathbf{u} = \mathbf{u}^T \lambda \mathbf{u} \quad (2.42)$$

Since λ is a scalar, it can change position on the right-hand side of the equation. Also, because of the associativity of matrix multiplication, we may write:

$$(\mathbf{u}^T g)(g^T \mathbf{u}) = \lambda \mathbf{u}^T \mathbf{u} \quad (2.43)$$

Since \mathbf{u} is an eigenvector, $\mathbf{u}^T \mathbf{u} = 1$. Therefore:

$$(g^T \mathbf{u})^T (g^T \mathbf{u}) = \lambda \quad (2.44)$$

$g^T \mathbf{u}$ is some vector \mathbf{y} . Then we have: $\lambda = \mathbf{y}^T \mathbf{y}$ which means that λ is non-negative since $\mathbf{y}^T \mathbf{y}$ is the square magnitude of vector \mathbf{y} .

Example 2.7

If λ_i are the eigenvalues of gg^T and \mathbf{u}_i the corresponding eigenvectors, show that $g^T g$ has the same eigenvalues, with the corresponding eigenvectors given by $\mathbf{v}_i = g^T \mathbf{u}_i$.

By definition:

$$gg^T \mathbf{u}_i = \lambda_i \mathbf{u}_i \quad (2.45)$$

Multiply both sides from the left with g^T :

$$g^T gg^T \mathbf{u}_i = g^T \lambda_i \mathbf{u}_i \quad (2.46)$$

As λ_i is a scalar, it may change position with respect to the other factors on the right-hand side of (2.46). Also, by the associativity of matrix multiplication:

$$g^T g(g^T \mathbf{u}_i) = \lambda_i (g^T \mathbf{u}_i) \quad (2.47)$$

This identifies $g^T \mathbf{u}_i$ as an eigenvector of $g^T g$ with λ_i the corresponding eigenvalue.

Example 2.8

You are given an image: $g = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$. Compute the eigenvectors \mathbf{u}_i of gg^T and \mathbf{v}_i of $g^T g$.

The transpose of g is:

$$g^T = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \quad (2.48)$$

We start by computing first gg^T :

$$gg^T = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 6 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad (2.49)$$

The eigenvalues of gg^T will be computed from its characteristic equation:

$$\begin{vmatrix} 1 - \lambda & 2 & 0 \\ 2 & 6 - \lambda & 1 \\ 0 & 1 & 1 - \lambda \end{vmatrix} = 0 \Rightarrow (1 - \lambda)[(6 - \lambda)(1 - \lambda) - 1] - 2[2(1 - \lambda)] = 0 \\ \Rightarrow (1 - \lambda)[(6 - \lambda)(1 - \lambda) - 1 - 4] = 0 \quad (2.50)$$

One eigenvalue is $\lambda = 1$. The other two are the roots of:

$$6 - 6\lambda - \lambda + \lambda^2 - 5 = 0 \Rightarrow \lambda^2 - 7\lambda + 1 = 0 \Rightarrow \lambda = \frac{7 \pm \sqrt{49 - 4}}{2} = \frac{7 \pm 6.7}{2} \\ \Rightarrow \lambda = 6.854 \text{ or } \lambda = 0.146 \quad (2.51)$$

In descending order, the eigenvalues are:

$$\lambda_1 = 6.854, \lambda_2 = 1, \lambda_3 = 0.146 \quad (2.52)$$

Let $\mathbf{u}_i = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ be the eigenvector which corresponds to eigenvalue λ_i . Then:

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 6 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \lambda_i \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \Rightarrow \begin{array}{l} x_1 + 2x_2 = \lambda_i x_1 \\ 2x_1 + 6x_2 + x_3 = \lambda_i x_2 \\ x_2 + x_3 = \lambda_i x_3 \end{array} \quad (2.53)$$

For $\lambda_i = 6.854$

$$2x_2 - 5.854x_1 = 0 \quad (2.54)$$

$$2x_1 - 0.854x_2 + x_3 = 0 \quad (2.55)$$

$$x_2 - 5.854x_3 = 0 \quad (2.56)$$

Multiply (2.55) with 5.854 and add equation (2.56) to get:

$$11.7x_1 - 4x_2 = 0 \quad (2.57)$$

Equation (2.57) is the same as (2.54). So we have really only two independent equations for the three unknowns. We choose the value of x_1 to be 1. Then:

$$x_2 = 2.927 \text{ and from (2.55)} \quad x_3 = -2 + 0.85 \times 2.925 = -2 + 2.5 = 0.5 \quad (2.58)$$

Thus, the first eigenvector is

$$\begin{pmatrix} 1 \\ 2.927 \\ 0.5 \end{pmatrix} \quad (2.59)$$

and after normalisation, ie division with $\sqrt{1^2 + 2.927^2 + 0.5^2} = 3.133$, we obtain:

$$\mathbf{u}_1 = \begin{pmatrix} 0.319 \\ 0.934 \\ 0.160 \end{pmatrix} \quad (2.60)$$

For $\lambda_i = 1$, the system of linear equations we have to solve is:

$$\begin{aligned} x_1 + 2x_2 &= x_1 \Rightarrow x_2 = 0 \\ 2x_1 + x_3 &= 0 \Rightarrow x_3 = -2x_1 \end{aligned} \quad (2.61)$$

Choose $x_1 = 1$. Then $x_3 = -2$. Since $x_2 = 0$, we must divide all components with $\sqrt{1^2 + 2^2} = \sqrt{5}$ for the eigenvector to have unit length:

$$\mathbf{u}_2 = \begin{pmatrix} 0.447 \\ 0 \\ -0.894 \end{pmatrix} \quad (2.62)$$

For $\lambda_i = 0.146$, the system of linear equations we have to solve is:

$$\begin{aligned} 0.854x_1 + 2x_2 &= 0 \\ 2x_1 + 5.854x_2 + x_3 &= 0 \\ x_2 + 0.854x_3 &= 0 \end{aligned} \quad (2.63)$$

Choose $x_1 = 1$. Then $x_2 = -\frac{0.854}{2} = -0.427$ and $x_3 = -\frac{0.427}{0.854} = 0.5$. Therefore, the third eigenvector is:

$$\begin{pmatrix} 1 \\ -0.427 \\ 0.5 \end{pmatrix}, \quad (2.64)$$

and after division with $\sqrt{1 + 0.427^2 + 0.5^2} = 1.197$ we obtain:

$$\mathbf{u}_3 = \begin{pmatrix} 0.835 \\ -0.357 \\ 0.418 \end{pmatrix} \quad (2.65)$$

The corresponding eigenvectors of $g^T g$ are given by $g^T \mathbf{u}_i$; ie the first one is:

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0.319 \\ 0.934 \\ 0.160 \end{pmatrix} = \begin{pmatrix} 2.187 \\ 0.934 \\ 1.094 \end{pmatrix} \quad (2.66)$$

We normalise it by dividing with $\sqrt{2.187^2 + 0.934^2 + 1.094^2} = 2.618$, to obtain:

$$\mathbf{v}_1 = \begin{pmatrix} 0.835 \\ 0.357 \\ 0.418 \end{pmatrix} \quad (2.67)$$

Similarly

$$\mathbf{v}_2 = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0.447 \\ 0 \\ -0.894 \end{pmatrix} = \begin{pmatrix} 0.447 \\ 0 \\ -0.894 \end{pmatrix}, \quad (2.68)$$

while the third eigenvector is

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} -0.835 \\ -0.357 \\ 0.418 \end{pmatrix} = \begin{pmatrix} 0.121 \\ -0.357 \\ 0.061 \end{pmatrix} \quad (2.69)$$

which after normalisation becomes:

$$\mathbf{v}_3 = \begin{pmatrix} 0.319 \\ -0.934 \\ 0.160 \end{pmatrix} \quad (2.70)$$

What is the singular value decomposition of an image?

The Singular Value Decomposition (SVD) of an image f is its expansion in terms of vector outer products, where the vectors used are the eigenvectors of ff^T and f^Tf , and the coefficients of the expansion are the eigenvalues of these matrices. In that case, equation (2.9) may be written as

$$f = \sum_{i=1}^r \lambda_i^{\frac{1}{2}} \mathbf{u}_i \mathbf{v}_i^T \quad (2.71)$$

since the only nonzero terms are those with $i = j$. Elementary images $\mathbf{u}_i \mathbf{v}_i^T$ are known as the **eigenimages** of image f .

Can we analyse an eigenimage into eigenimages?

No. An eigenimage $N \times N$ may be written as the outer product of two vectors, say vectors \mathbf{u} and \mathbf{v} :

$$\mathbf{u}\mathbf{v}^T = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} (v_1 \ v_2 \ \dots \ v_N) = \begin{pmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_N \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_N \\ \dots & \dots & \dots & \dots \\ u_N v_1 & u_N v_2 & \dots & u_N v_N \end{pmatrix} \quad (2.72)$$

Any row of the outer product of two vectors may be written as the linear function of any other row. For example, we can see from (2.72) row number 1 is row number 2 times u_1/u_2 . So, an eigenimage is a matrix with rank 1, ie it has only one nonzero eigenvalue and only one eigenvector: it cannot be analysed any further.

Example 2.9

Consider a 2×2 image that can be written as the outer product of two vectors. Show that it has only one nonzero eigenvalue and that the corresponding eigenvector is parallel to the first of the two vectors, the outer product of which makes the image.

Let us say that the image can be written as the outer product of vectors $\mathbf{a}^T = (a_1, a_2)$ and $\mathbf{b}^T = (b_1, b_2)$:

$$\mathbf{ab}^T = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} (b_1 \ b_2) = \begin{pmatrix} a_1 b_1 & a_1 b_2 \\ a_2 b_1 & a_2 b_2 \end{pmatrix} \quad (2.73)$$

We solve the characteristic equation of this matrix to work out its eigenvalues:

$$\begin{aligned} & \left| \begin{array}{cc} a_1 b_1 - \lambda & a_1 b_2 \\ a_2 b_1 & a_2 b_2 - \lambda \end{array} \right| = 0 \\ & \Rightarrow (a_1 b_1 - \lambda)(a_2 b_2 - \lambda) - a_1 b_2 a_2 b_1 = 0 \\ & a_1 b_2 a_2 b_1 - \lambda a_2 b_2 - \lambda a_1 b_1 + \lambda^2 - a_1 b_2 a_2 b_1 = 0 \\ & \lambda(\lambda - a_2 b_2 - a_1 b_1) = 0 \\ & \lambda = a_2 b_2 + a_1 b_1 \quad \text{or} \quad \lambda = 0 \end{aligned} \quad (2.74)$$

So, only one eigenvalue is different from zero. The corresponding eigenvector $(x_1, x_2)^T$ is the solution of:

$$\begin{aligned} \begin{pmatrix} a_1 b_1 & a_1 b_2 \\ a_2 b_1 & a_2 b_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= (a_2 b_2 + a_1 b_1) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \Rightarrow \\ \begin{array}{rcl} a_1 b_1 x_1 + a_1 b_2 x_2 & = & (a_2 b_2 + a_1 b_1) x_1 \\ a_2 b_1 x_1 + a_2 b_2 x_2 & = & (a_2 b_2 + a_1 b_1) x_2 \end{array} & \Rightarrow \\ \begin{array}{rcl} a_1 x_2 & = & a_2 x_1 \\ a_2 x_1 & = & a_1 x_2 \end{array} & \Rightarrow \\ x_2 & = & \frac{a_2}{a_1} x_1 \end{aligned} \quad (2.75)$$

Choose $x_1 = a_1$. Then $x_2 = a_2$, and the eigenvector is $(a_1, a_2)^T$ times a constant that will make sure its length is normalised to 1. So, the eigenvector is parallel to vector \mathbf{a} since they only differ by a multiplicative constant.

How can we approximate an image using SVD?

If in equation (2.71) we decide to keep only $k < r$ terms, we shall reproduce an approximated version of the image:

$$f_k = \sum_{i=1}^k \lambda_i^{\frac{1}{2}} \mathbf{u}_i \mathbf{v}_i^T \quad (2.76)$$

Example 2.10

A 256×256 grey image with 256 grey levels is to be transmitted. How many terms can be kept in its SVD before the transmission of the transformed image becomes too inefficient in comparison with the transmission of the original image? (Assume that real numbers require 32 bits each.)

Assume that $\lambda_i^{\frac{1}{2}}$ is incorporated into one of the vectors \mathbf{u}_i or \mathbf{v}_i in equation (2.76). When we transmit term i of the SVD expansion of the image, we must transmit the two vectors \mathbf{u}_i and \mathbf{v}_i , that are made up from 256 elements each, which are real numbers. We must, therefore, transmit

$$2 \times 32 \times 256 \text{ bits per term.}$$

If we want to transmit the full image, we shall have to transmit $256 \times 256 \times 8$ bits (since each pixel requires 8 bits). Then the maximum number of terms transmitted before the SVD becomes uneconomical is:

$$k = \frac{256 \times 256 \times 8}{2 \times 32 \times 256} = \frac{256}{8} = 32 \quad (2.77)$$

Box 2.3. What is the intuitive explanation of SVD?

Let us consider a 2×3 matrix (an image) A . Matrix $A^T A$ is a 3×3 matrix. Let us consider its effect on a 3×1 vector \mathbf{u} : $A^T A \mathbf{u} = A^T(A\mathbf{u})$. When matrix A operates on vector \mathbf{u} , it produces 2×1 vector $\tilde{\mathbf{u}}$:

$$\begin{aligned} \tilde{\mathbf{u}} &\equiv A\mathbf{u} \Rightarrow \\ \begin{pmatrix} \tilde{u}_1 \\ \tilde{u}_2 \end{pmatrix} &= \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \end{aligned} \quad (2.78)$$

This is nothing else than a projection of vector \mathbf{u} from a 3D space to a 2D space. Next, let us consider the effect of A^T on this vector:

$$\begin{aligned} A^T(A\mathbf{u}) &= A^T\tilde{\mathbf{u}} \Rightarrow \\ \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{pmatrix} \begin{pmatrix} \tilde{u}_1 \\ \tilde{u}_2 \end{pmatrix} &\equiv \begin{pmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \end{pmatrix} \end{aligned} \quad (2.79)$$

This is nothing else than an upsampling and embedding of vector $\tilde{\mathbf{u}}$ from a 2D space into a 3D space. Now, if vector \mathbf{u} is an eigenvector of matrix A^TA , the result of this operation, namely projecting it on a lower dimensionality space and then embedding it back into the high dimensionality space we started from, will produce a vector that has the same orientation as the original vector \mathbf{u} , and magnitude λ times the original magnitude: $A^TA\mathbf{u} = \lambda\mathbf{u}$, where λ is the corresponding eigenvalue of matrix A^TA . When λ is large ($\lambda > 1$), this process, of projecting the vector in a low dimensionality space and upsampling it again back to its original space, will make the vector larger and “stronger”, while if λ is small ($\lambda < 1$), the vector will shrink because of this process. We may think of this operation as a “resonance”: eigenvectors with large eigenvalues gain energy from this process and emerge λ times stronger, as if they resonate with the matrix. So, when we compute the eigenimages of matrix A , as the outer products of the eigenvectors that resonate with matrix A^TA (or AA^T), and arrange them in order of decreasing corresponding eigenvectors, effectively we find the modes of the image: those components that contain the most energy and “resonate” best with the image when the image is seen as an operator that projects a vector to a lower dimensionality space and then embeds it back to its original space.

What is the error of the approximation of an image by SVD?

The difference between the original and the approximated image is:

$$D \equiv f - f_k = \sum_{i=k+1}^r \lambda_i^{\frac{1}{2}} \mathbf{u}_i \mathbf{v}_i^T \quad (2.80)$$

We may calculate how big this error is by calculating the norm of matrix D , ie the sum of the squares of its elements. From (2.80) it is obvious that, if u_{im} is the m^{th} element of vector \mathbf{u}_i and v_{in} is the n^{th} element of vector \mathbf{v}_i , the mn^{th} element of D is:

$$\begin{aligned}
d_{mn} &= \sum_{i=k+1}^r \lambda_i^{\frac{1}{2}} u_{im} v_{in} \Rightarrow \\
d_{mn}^2 &= \left(\sum_{i=k+1}^r \lambda_i^{\frac{1}{2}} u_{im} v_{in} \right)^2 \\
&= \sum_{i=k+1}^r \lambda_i u_{im}^2 v_{in}^2 + 2 \sum_{i=k+1}^r \sum_{j=k+1, j \neq i}^r \lambda_i^{\frac{1}{2}} \lambda_j^{\frac{1}{2}} u_{im} v_{in} u_{jm} v_{jn}
\end{aligned} \tag{2.81}$$

The norm of matrix D will be the sum of the squares of all its elements:

$$\begin{aligned}
\|D\| &= \sum_m \sum_n d_{mn}^2 \\
&= \sum_m \sum_n \sum_{i=k+1}^r \lambda_i u_{im}^2 v_{in}^2 + 2 \sum_m \sum_n \sum_{i=k+1}^r \sum_{j=k+1, j \neq i}^r \lambda_i^{\frac{1}{2}} \lambda_j^{\frac{1}{2}} u_{im} v_{in} u_{jm} v_{jn} \\
&= \sum_{i=k+1}^r \lambda_i \sum_m u_{im}^2 \sum_n v_{in}^2 + 2 \sum_{i=k+1}^r \sum_{j=k+1, j \neq i}^r \lambda_i^{\frac{1}{2}} \lambda_j^{\frac{1}{2}} \sum_m u_{im} u_{jm} \sum_n v_{in} v_{jn}
\end{aligned} \tag{2.82}$$

However, $\mathbf{u}_i, \mathbf{v}_i$ are eigenvectors and therefore they form an orthonormal set. So

$$\begin{aligned}
\sum_m u_{im}^2 &= 1, \quad \sum_n v_{in}^2 = 1, \\
\sum_n v_{in} v_{jn} &= 0 \quad \text{and} \quad \sum_m u_{im} u_{jm} = 0 \quad \text{for } i \neq j
\end{aligned} \tag{2.83}$$

since $\mathbf{u}_i \mathbf{u}_j^T = 0$ and $\mathbf{v}_i \mathbf{v}_j^T = 0$ for $i \neq j$. Then:

$$\|D\| = \sum_{i=k+1}^r \lambda_i \tag{2.84}$$

Therefore, the square error of the approximate reconstruction of the image using equation (2.76) is equal to the sum of the omitted eigenvalues.

Example 2.11

For a 3×3 matrix D show that its norm, defined as the trace of $D^T D$, is equal to the sum of the squares of its elements.

Let us assume that:

$$D \equiv \begin{pmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{pmatrix} \tag{2.85}$$

Then:

$$D^T D = \begin{pmatrix} d_{11} & d_{21} & d_{31} \\ d_{12} & d_{22} & d_{32} \\ d_{13} & d_{23} & d_{33} \end{pmatrix} \begin{pmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{pmatrix} =$$

$$\begin{pmatrix} d_{11}^2 + d_{21}^2 + d_{31}^2 & d_{11}d_{12} + d_{21}d_{22} + d_{31}d_{32} & d_{11}d_{13} + d_{21}d_{23} + d_{31}d_{33} \\ d_{12}d_{11} + d_{22}d_{21} + d_{32}d_{31} & d_{12}^2 + d_{22}^2 + d_{32}^2 & d_{12}d_{13} + d_{22}d_{23} + d_{32}d_{33} \\ d_{13}d_{11} + d_{23}d_{21} + d_{33}d_{31} & d_{13}d_{12} + d_{23}d_{22} + d_{33}d_{32} & d_{13}^2 + d_{23}^2 + d_{33}^2 \end{pmatrix} \quad (2.86)$$

Finally:

$$\begin{aligned} \text{trace}[D^T D] &= (d_{11}^2 + d_{21}^2 + d_{31}^2) + (d_{12}^2 + d_{22}^2 + d_{32}^2) + (d_{13}^2 + d_{23}^2 + d_{33}^2) \\ &= \text{sum of all elements of } D \text{ squared.} \end{aligned} \quad (2.87)$$

How can we minimise the error of the reconstruction?

If we arrange the eigenvalues λ_i of matrices $f^T f$ and ff^T in decreasing order and truncate the expansion at some integer $k < r$, where r is the rank of these matrices, we approximate the image f by f_k , which is the least square error approximation. This is because the sum of the squares of the elements of the difference matrix is minimal, since it is equal to the sum of the unused eigenvalues which have been chosen to be the smallest ones.

Notice that the singular value decomposition of an image is optimal in the *least square error* sense but the basis images (eigenimages), with respect to which we expanded the image, are determined by the image itself. (They are determined by the eigenvectors of $f^T f$ and ff^T .)

Example 2.12

In the singular value decomposition of the image of example 2.8, only the first term is kept while the others are set to zero. Verify that the square error of the reconstructed image is equal to the sum of the omitted eigenvalues.

If we keep only the first eigenvalue, the image is approximated by the first eigenimage only, weighted by the square root of the corresponding eigenvalue:

$$\begin{aligned}
g_1 &= \sqrt{\lambda_1} \mathbf{u}_1 \mathbf{v}_1^T = \sqrt{6.85} \begin{pmatrix} 0.319 \\ 0.934 \\ 0.160 \end{pmatrix} (0.835 \quad 0.357 \quad 0.418) \\
&= \begin{pmatrix} 0.835 \\ 2.444 \\ 0.419 \end{pmatrix} (0.835 \quad 0.357 \quad 0.418) = \begin{pmatrix} 0.697 & 0.298 & 0.349 \\ 2.041 & 0.873 & 1.022 \\ 0.350 & 0.150 & 0.175 \end{pmatrix} \quad (2.88)
\end{aligned}$$

The error of the reconstruction is given by the difference between g_1 and the original image:

$$g - g_1 = \begin{pmatrix} 0.303 & -0.298 & -0.349 \\ -0.041 & 0.127 & -0.022 \\ -0.350 & -0.150 & 0.825 \end{pmatrix} \quad (2.89)$$

The sum of the squares of the errors is:

$$\begin{aligned}
0.303^2 + 0.298^2 + 0.349^2 + 0.041^2 + 0.127^2 + 0.022^2 + 0.350^2 + 0.150^2 + 0.825^2 \\
= 1.146 \quad (2.90)
\end{aligned}$$

This is exactly equal to the sum of the two omitted eigenvalues λ_2 and λ_3 .

Example 2.13

Perform the singular value decomposition (SVD) of the following image:

$$g = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \quad (2.91)$$

Thus, identify the eigenimages of the above image.

We start by computing gg^T :

$$gg^T = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 2 \end{pmatrix} \quad (2.92)$$

The eigenvalues of gg^T are the solutions of:

$$\begin{vmatrix} 2 - \lambda & 0 & 2 \\ 0 & 1 - \lambda & 0 \\ 2 & 0 & 2 - \lambda \end{vmatrix} = 0 \Rightarrow (2 - \lambda)^2(1 - \lambda) - 4(1 - \lambda) = 0 \\
\Rightarrow (1 - \lambda)(\lambda - 4)\lambda = 0 \quad (2.93)$$

The eigenvalues are: $\lambda_1 = 4$, $\lambda_2 = 1$, $\lambda_3 = 0$. The first corresponding eigenvector is the solution of the system of equations:

$$\begin{pmatrix} 2 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 4 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \Rightarrow \begin{array}{l} 2x_1 + 2x_3 = 4x_1 \\ x_2 = 4x_2 \\ 2x_1 + 2x_3 = 4x_3 \end{array} \Rightarrow \begin{array}{l} x_1 = x_3 \\ x_2 = 0 \\ x_1 = x_3 \end{array} \quad (2.94)$$

We choose $x_1 = x_3 = \frac{1}{\sqrt{2}}$ so that the eigenvector has unit length. Thus,

$\mathbf{u}_1^T = \left(\frac{1}{\sqrt{2}} \ 0 \ \frac{1}{\sqrt{2}} \right)$. For the second eigenvalue, we have:

$$\begin{pmatrix} 2 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \Rightarrow \begin{array}{l} 2x_1 + 2x_3 = x_1 \\ x_2 = x_2 \\ 2x_1 + 2x_3 = x_3 \end{array} \Rightarrow \begin{array}{l} x_1 = -2x_3 \\ x_2 = x_2 \\ x_3 = -2x_1 \end{array} \quad (2.95)$$

The second of the above equations conveys no information, giving us the option to choose whatever value of x_2 we want. However, we cannot choose $x_2 = 0$, because the first and the third equations appear to be incompatible, unless $x_1 = x_3 = 0$. If x_2 were 0 too, we would have the trivial solution which does not represent an eigenvector.

So, the above three equations are satisfied if $x_1 = x_3 = 0$ and x_2 is anything apart from 0. Then x_2 is chosen to be 1 so that \mathbf{u}_2 has also unit length. Thus, $\mathbf{u}_2^T = (0 \ 1 \ 0)$. Because g is symmetric, $gg^T = g^Tg$ and the eigenvectors of gg^T are the same as the eigenvectors of g^Tg . Then the SVD of g is:

$$\begin{aligned} g &= \sqrt{\lambda_1} \mathbf{u}_1 \mathbf{u}_1^T + \sqrt{\lambda_2} \mathbf{u}_2 \mathbf{u}_2^T = 2 \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} (0 \ 1 \ 0) \\ &= \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad (2.96) \end{aligned}$$

These two matrices are the eigenimages of g .

Example 2.14

Perform the singular value decomposition of the following image and identify its eigenimages:

$$g = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad (2.97)$$

Start by computing gg^T :

$$gg^T = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix} \quad (2.98)$$

The eigenvalues of this matrix are the solutions of:

$$\begin{vmatrix} 1 - \lambda & 0 & 1 \\ 0 & 2 - \lambda & 0 \\ 1 & 0 & 1 - \lambda \end{vmatrix} = 0 \Rightarrow (1 - \lambda)^2(2 - \lambda) - (2 - \lambda) = 0 \\ \Rightarrow (2 - \lambda)[(1 - \lambda)^2 - 1] = 0 \Rightarrow (2 - \lambda)(1 - \lambda - 1)(1 - \lambda + 1) = 0 \quad (2.99)$$

So, $\lambda_1 = 2$, $\lambda_2 = 2$, $\lambda_3 = 0$.

The first eigenvector is:

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 2 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \Rightarrow \begin{array}{l} x_1 + x_3 = 2x_1 \\ 2x_2 = 2x_2 \\ x_1 + x_3 = 2x_3 \end{array} \Rightarrow \begin{array}{l} x_1 = x_3 \\ x_2 \text{ any value} \end{array} \quad (2.100)$$

Choose $x_1 = x_3 = \frac{1}{\sqrt{2}}$ and $x_2 = 0$, so $\mathbf{u}_1 = \left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right)^T$.

The second eigenvector must satisfy the same constraints and must be orthogonal to \mathbf{u}_1 . Therefore:

$$\mathbf{u}_2 = (0, 1, 0)^T \quad (2.101)$$

Because g is symmetric, $gg^T = g^Tg$ and the eigenvectors of gg^T are the same as the eigenvectors of g^Tg . Then the SVD of g is:

$$\begin{aligned} g &= \sqrt{\lambda_1} \mathbf{u}_1 \mathbf{v}_1^T + \sqrt{\lambda_2} \mathbf{u}_2 \mathbf{v}_2^T \\ &= \sqrt{2} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} (0 \ 1 \ 0) + \sqrt{2} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \\ &= \sqrt{2} \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 \end{pmatrix} + \sqrt{2} \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \end{aligned} \quad (2.102)$$

These two matrices are the eigenimages of g . Note that the answer would not have changed if we exchanged the definitions of \mathbf{u}_1 and \mathbf{u}_2 , the order of which is meaningless, since they both correspond to the same eigenvalue with multiplicity 2 (ie it is 2-fold degenerate).

Example 2.15

Show the different stages of the SVD of the following image:

$$g = \begin{pmatrix} 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 100 & 100 & 100 & 255 & 255 \\ 255 & 255 & 100 & 150 & 150 & 150 & 100 & 255 \\ 255 & 255 & 100 & 150 & 200 & 150 & 100 & 255 \\ 255 & 255 & 100 & 150 & 150 & 150 & 100 & 255 \\ 255 & 255 & 255 & 100 & 100 & 100 & 255 & 255 \\ 255 & 255 & 255 & 255 & 50 & 255 & 255 & 255 \\ 50 & 50 & 50 & 50 & 255 & 255 & 255 & 255 \end{pmatrix} \quad (2.103)$$

The gg^T matrix is:

$$gg^T = \begin{pmatrix} 520200 & 401625 & 360825 & 373575 & 360825 & 401625 & 467925 & 311100 \\ 401625 & 355125 & 291075 & 296075 & 291075 & 355125 & 381125 & 224300 \\ 360825 & 291075 & 282575 & 290075 & 282575 & 291075 & 330075 & 205025 \\ 373575 & 296075 & 290075 & 300075 & 290075 & 296075 & 332575 & 217775 \\ 360825 & 291075 & 282575 & 290075 & 282575 & 291075 & 330075 & 205025 \\ 401625 & 355125 & 291075 & 296075 & 291075 & 355125 & 381125 & 224300 \\ 467925 & 381125 & 330075 & 332575 & 330075 & 381125 & 457675 & 258825 \\ 311100 & 224300 & 205025 & 217775 & 205025 & 224300 & 258825 & 270100 \end{pmatrix} \quad (2.104)$$

Its eigenvalues sorted in decreasing order are:

$$\begin{array}{cccc} 2593416.500 & 111621.508 & 71738.313 & 34790.875 \\ 11882.712 & 0.009 & 0.001 & 0.000 \end{array}$$

The last three eigenvalues are practically 0, so we compute only the eigenvectors that correspond to the first five eigenvalues. These eigenvectors are the columns of the following matrix:

$$\begin{pmatrix} 0.441 & -0.167 & -0.080 & -0.388 & 0.764 \\ 0.359 & 0.252 & -0.328 & 0.446 & 0.040 \\ 0.321 & 0.086 & 0.440 & 0.034 & -0.201 \\ 0.329 & 0.003 & 0.503 & 0.093 & 0.107 \\ 0.321 & 0.086 & 0.440 & 0.035 & -0.202 \\ 0.359 & 0.252 & -0.328 & 0.446 & 0.040 \\ 0.407 & 0.173 & -0.341 & -0.630 & -0.504 \\ 0.261 & -0.895 & -0.150 & 0.209 & -0.256 \end{pmatrix} \quad (2.105)$$

The \mathbf{v}_i eigenvectors, computed as $g^T \mathbf{u}_i$, turn out to be the columns of the following

matrix:

$$\begin{pmatrix} 0.410 & 0.389 & 0.264 & 0.106 & -0.012 \\ 0.410 & 0.389 & 0.264 & 0.106 & -0.012 \\ 0.316 & 0.308 & -0.537 & -0.029 & 0.408 \\ 0.277 & 0.100 & 0.101 & -0.727 & 0.158 \\ 0.269 & -0.555 & 0.341 & 0.220 & 0.675 \\ 0.311 & -0.449 & -0.014 & -0.497 & -0.323 \\ 0.349 & -0.241 & -0.651 & 0.200 & -0.074 \\ 0.443 & -0.160 & 0.149 & 0.336 & -0.493 \end{pmatrix} \quad (2.106)$$

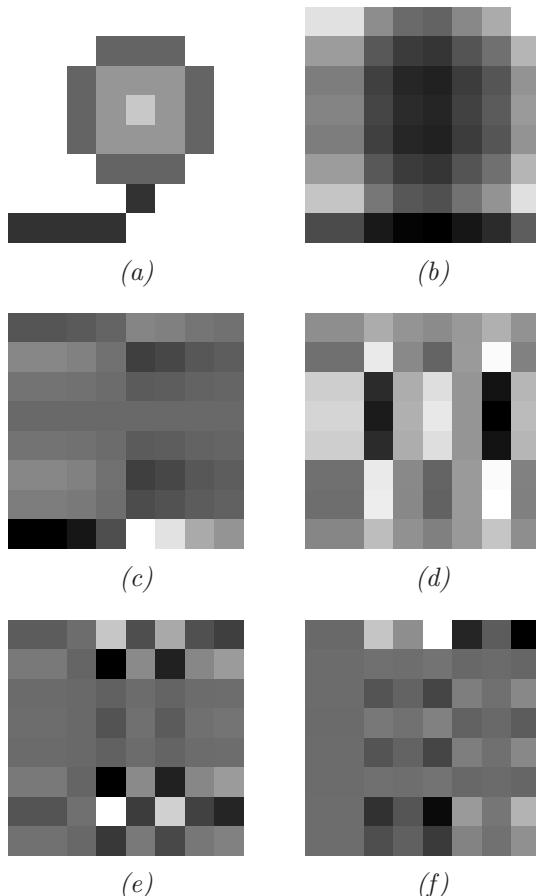


Figure 2.1: The original image and its five eigenimages, each scaled independently to have values from 0 to 255.

In figure 2.1 the original image and its five eigenimages are shown. Each eigenimage has been scaled so that its grey values vary between 0 and 255. These eigenimages have to be weighted by the square root of the appropriate eigenvalue and added to produce the original image. The five images shown in figure 2.2 are the reconstructed images when one, two, ..., five eigenvalues were used for the reconstruction.

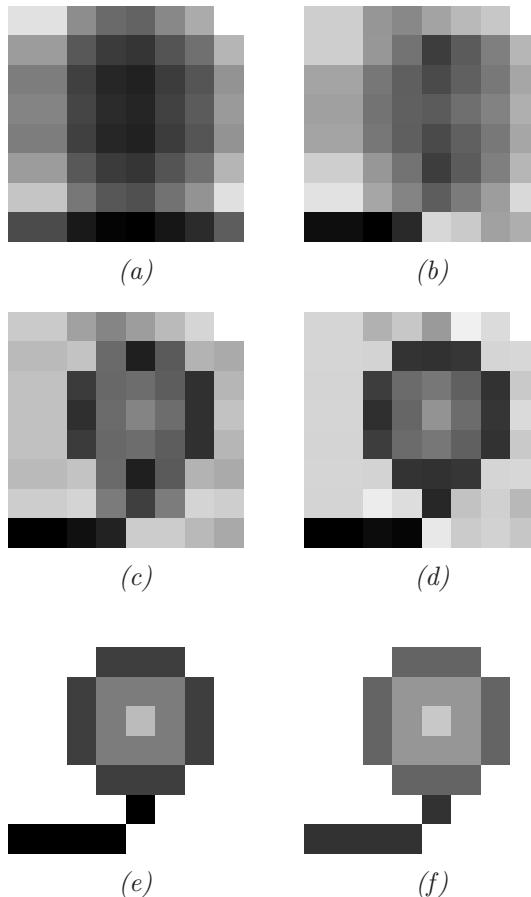


Figure 2.2: Image reconstruction using one, two, ..., five eigenimages from top right to bottom left sequentially, with the original image shown in (f).

Then we calculate the sum of the squared errors for each reconstructed image according to the formula:

$$\sum_{\text{all pixels}} (\text{reconstructed pixel} - \text{original pixel})^2 \quad (2.107)$$

We obtain:

Square error for image 2.2a: 230033.32 ($\lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 = 230033.41$)

Square error for image 2.2b: 118412.02 ($\lambda_3 + \lambda_4 + \lambda_5 = 118411.90$)

Square error for image 2.2c: 46673.53 ($\lambda_4 + \lambda_5 = 46673.59$)

Square error for image 2.2d: 11882.65 ($\lambda_5 = 11882.71$)

Square error for image 2.2e: 0

We see that the sum of the omitted eigenvalues agrees very well with the value of the square error for each reconstructed image.

Are there any sets of elementary images in terms of which *any* image may be expanded?

Yes. They are defined in terms of **complete** and **orthonormal** sets of discrete valued discrete functions.

What is a complete and orthonormal set of functions?

A set of functions $S_n(t)$, where n is an integer, is said to be **orthogonal** over an interval $[0, T]$ with weight function $w(t)$, if:

$$\int_0^T w(t)S_n(t)S_m(t)dt = \begin{cases} k \neq 0 & \text{if } n = m \\ 0 & \text{if } n \neq m \end{cases} \quad (2.108)$$

In other words, the set of functions $S_n(t)$, for n an integer index identifying the individual functions, is orthogonal when the integral of the product of any two of these functions over a certain interval, possibly weighted by a function $w(t)$, is zero, unless the two functions are the same function, in which case the result is equal to a nonzero constant k . The set is called **orthonormal**, if $k = 1$. Note that from an orthogonal set of functions we can easily create an orthonormal set by a simple scaling of the functions. The set is called **complete**, if we cannot find any other function which is orthogonal to the set and does not belong to the set. An example of a complete and orthogonal set is the set of functions $S_n(t) \equiv e^{jnt}$, which are used as the basis functions of the Fourier transform.

Example 2.16

Show that the columns of an orthogonal matrix form a set of orthonormal vectors.

Let us say that A is an $N \times N$ orthogonal matrix (ie $A^T = A^{-1}$), and let us consider its column vectors $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_N$. We obviously have:

$$A^{-1}A = I \Rightarrow A^T A = I \Rightarrow \begin{pmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_N^T \end{pmatrix} (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_N) = I \Rightarrow$$

$$\begin{pmatrix} \mathbf{u}_1^T \mathbf{u}_1 & \mathbf{u}_1^T \mathbf{u}_2 & \dots & \mathbf{u}_1^T \mathbf{u}_N \\ \mathbf{u}_2^T \mathbf{u}_1 & \mathbf{u}_2^T \mathbf{u}_2 & \dots & \mathbf{u}_2^T \mathbf{u}_N \\ \vdots & \vdots & & \vdots \\ \mathbf{u}_N^T \mathbf{u}_1 & \mathbf{u}_N^T \mathbf{u}_2 & \dots & \mathbf{u}_N^T \mathbf{u}_N \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} \quad (2.109)$$

This proves that the columns of A form an orthonormal set of vectors, since $\mathbf{u}_i^T \mathbf{u}_j = 0$ for $i \neq j$ and $\mathbf{u}_i^T \mathbf{u}_i = 1$ for every i .

Example 2.17

Show that the inverse of an orthogonal matrix is also orthogonal.

An orthogonal matrix is defined as:

$$A^T = A^{-1} \quad (2.110)$$

To prove that A^{-1} is also orthogonal, it is enough to prove that $(A^{-1})^T = (A^{-1})^{-1}$. This is equivalent to $(A^{-1})^T = A$, which is readily derived if we take the transpose of equation (2.110).

Example 2.18

Show that the rows of an orthogonal matrix also form a set of orthonormal vectors.

Since A is an orthogonal matrix, so is A^{-1} (see example 2.17). The columns of an orthogonal matrix form a set of orthonormal vectors (see example 2.16). Therefore, the columns of A^{-1} , which are the rows of A , form a set of orthonormal vectors.

Are there any complete sets of orthonormal discrete valued functions?

Yes. There is, for example, the set of **Haar functions**, which take values from the set of numbers $\{0, \pm 1, \pm \sqrt{2}^p, \text{ for } p = 1, 2, 3, \dots\}$ and the set of **Walsh functions**, which take values from the set of numbers $\{+1, -1\}$.

2.2 Haar, Walsh and Hadamard transforms

How are the Haar functions defined?

They are defined recursively by equations

$$\begin{aligned} H_0(t) &\equiv 1 \text{ for } 0 \leq t < 1 \\ H_1(t) &\equiv \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq t < 1 \end{cases} \\ H_{2^p+n}(t) &\equiv \begin{cases} \sqrt{2}^p & \text{for } \frac{n}{2^p} \leq t < \frac{n+0.5}{2^p} \\ -\sqrt{2}^p & \text{for } \frac{n+0.5}{2^p} \leq t < \frac{n+1}{2^p} \\ 0 & \text{elsewhere} \end{cases} \end{aligned} \quad (2.111)$$

where $p = 1, 2, 3, \dots$ and $n = 0, 1, \dots, 2^p - 1$.

How are the Walsh functions defined?

They are defined in various ways, all of which can be shown to be equivalent. We use here the definition from the recursive equation

$$W_{2j+q}(t) \equiv (-1)^{\lfloor \frac{j}{2} \rfloor + q} \{W_j(2t) + (-1)^{j+q} W_j(2t-1)\} \quad (2.112)$$

where $\lfloor \frac{j}{2} \rfloor$ means the largest integer which is smaller or equal to $\frac{j}{2}$, $q = 0$ or 1 , $j = 0, 1, 2, \dots$ and:

$$W_0(t) \equiv \begin{cases} 1 & \text{for } 0 \leq t < 1 \\ 0 & \text{elsewhere} \end{cases} \quad (2.113)$$

Different definitions (eg see Box 2.4) define these functions in different orders (see Box 2.5).

Box 2.4. Definition of Walsh functions in terms of the Rademacher functions

A **Rademacher function** of order n ($n \neq 0$) is defined as:

$$R_n(t) \equiv \text{sign} [\sin (2^n \pi t)] \quad \text{for } 0 \leq t \leq 1 \quad (2.114)$$

For $n = 0$:

$$R_0(t) \equiv 1 \quad \text{for } 0 \leq t \leq 1 \quad (2.115)$$

These functions look like square pulse versions of the sine function. The Walsh functions in terms of them are defined as

$$\tilde{W}_n(t) \equiv \prod_{i=1, b_i \neq 0}^{m+1} R_i(t) \quad (2.116)$$

where b_i are the digits of n when expressed as a binary number:

$$n = b_{m+1}2^m + b_m2^{m-1} + \cdots + b_22^1 + b_12^0 \quad (2.117)$$

For example, the binary expression for n when $n = 4$ is 100. This means that $m = 2$, $b_3 = 1$ and $b_2 = b_1 = 0$. Then:

$$\tilde{W}_4(t) = R_3(t) = \text{sign}[\sin(8\pi t)] \quad (2.118)$$

Figure 2.3 shows $\sin(8\pi t)$, $R_3(t)$ and $\tilde{W}_4(t)$.

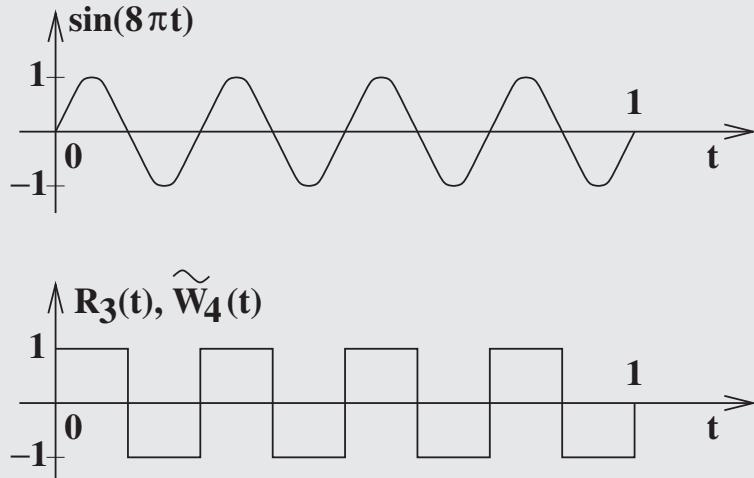


Figure 2.3: The sine function used to define the corresponding Rademacher function, which is Walsh function $\tilde{W}_4(t)$.

How can we use the Haar or Walsh functions to create image bases?

We saw that a unitary matrix has its columns forming an orthonormal set of vectors (=discrete functions). We can use the discretised Walsh or Haar functions as vectors that constitute such an orthonormal set. In other words, we can create transformation matrices that are made up from Walsh or Haar functions of different orders.

How can we create the image transformation matrices from the Haar and Walsh functions in practice?

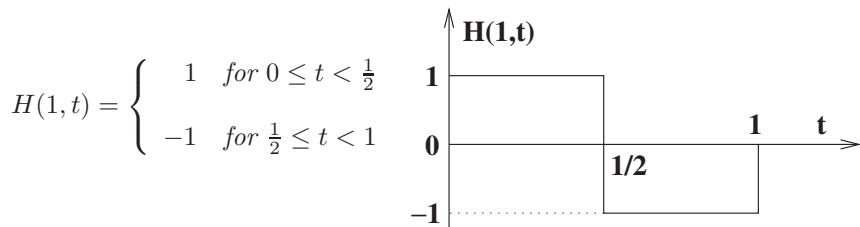
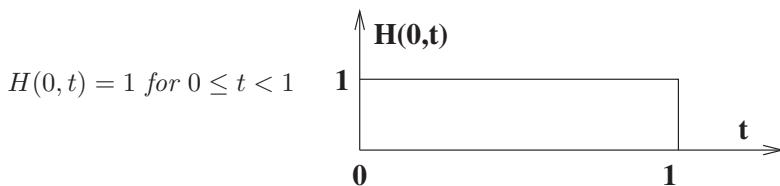
We first scale the independent variable t by the size of the matrix we want to create. Then we consider only its integer values i . Then $H_k(i)$ can be written in a matrix form for $k = 0, 1, 2, \dots, N - 1$ and $i = 0, 1, \dots, N - 1$ and be used for the transformation of a discrete $N \times N$ image. We work similarly for $W_k(i)$.

Note that the Haar/Walsh functions defined this way are not orthonormal. Each has to be normalised by being multiplied with $\frac{1}{\sqrt{T}}$ in the continuous case, or with $\frac{1}{\sqrt{N}}$ in the discrete case, if t takes up N equally spaced discrete values.

Example 2.19

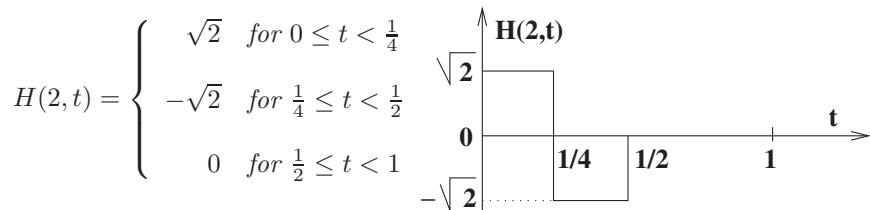
Derive the matrix which may be used to calculate the Haar transform of a 4×4 image.

First, by using equation (2.111), we shall calculate and plot the Haar functions of the continuous variable t which are needed for the calculation of the transformation matrix.

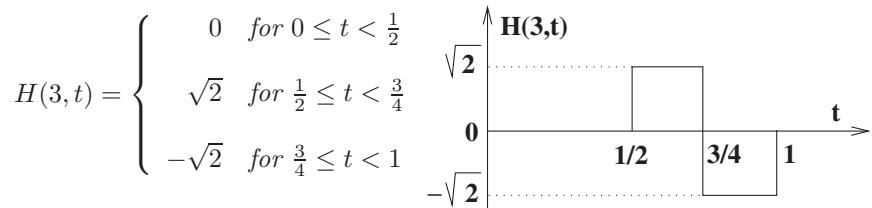


In the definition of the Haar functions, when $p = 1$, n takes the values 0 and 1.

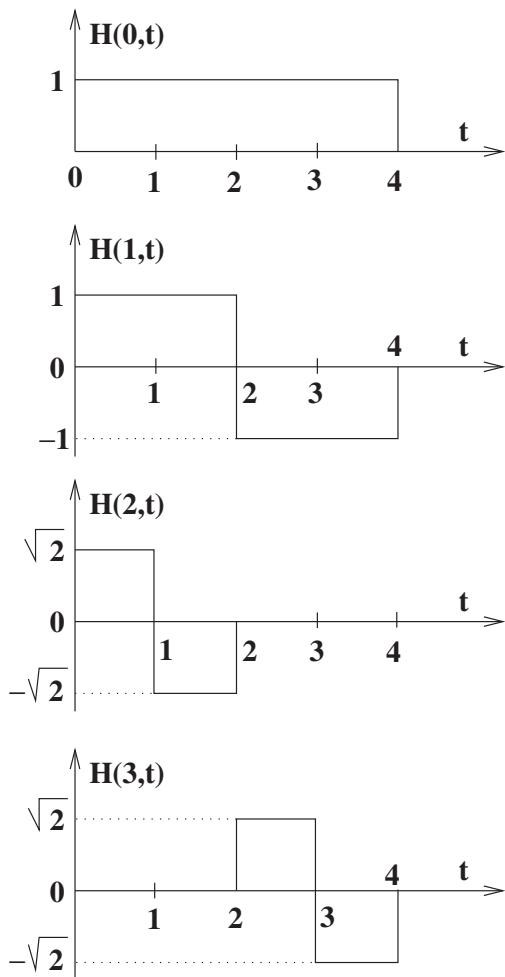
Case $p = 1$, $n = 0$:



Case $p = 1, n = 1$:



To transform a 4×4 image we need a 4×4 matrix. If we scale the t axis by multiplying it with 4 and take only the integer values of t (ie $t = 0, 1, 2, 3$), we can construct the transformation matrix. The plots of the scaled functions look like this:



The entries of the transformation matrix are the values of $H(s, t)$ where s and t take values 0, 1, 2, 3. Obviously then, the transformation matrix is:

$$H = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix} \quad (2.119)$$

Factor $\frac{1}{2}$ is introduced to normalise matrix H so that $HH^T = I$, the unit matrix.

Example 2.20

Calculate the Haar transform of image:

$$g = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad (2.120)$$

The Haar transform of image g is $A = HgH^T$. We use matrix H derived in example 2.19:

$$\begin{aligned} A &= \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & \sqrt{2} & 0 \\ 1 & 1 & -\sqrt{2} & 0 \\ 1 & -1 & 0 & \sqrt{2} \\ 1 & -1 & 0 & -\sqrt{2} \end{pmatrix} \\ &= \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix} \begin{pmatrix} 2 & 0 & -\sqrt{2} & \sqrt{2} \\ 2 & 0 & \sqrt{2} & -\sqrt{2} \\ 2 & 0 & \sqrt{2} & -\sqrt{2} \\ 2 & 0 & -\sqrt{2} & \sqrt{2} \end{pmatrix} \\ &= \frac{1}{4} \begin{pmatrix} 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & 4 \\ 0 & 0 & 4 & -4 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \end{aligned} \quad (2.121)$$

Example 2.21

Reconstruct the image of example 2.20 using an approximation of its Haar transform by setting its bottom right element equal to 0.

The approximate transformation matrix becomes:

$$\tilde{A} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.122)$$

The reconstructed image is given by $\tilde{g} = H^T \tilde{A} H$:

$$\begin{aligned} \tilde{g} &= \frac{1}{4} \begin{pmatrix} 1 & 1 & -\sqrt{2} & 0 \\ 1 & 1 & -\sqrt{2} & 0 \\ 1 & -1 & 0 & \sqrt{2} \\ 1 & -1 & 0 & -\sqrt{2} \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix} \\ &= \frac{1}{4} \begin{pmatrix} 1 & 1 & -\sqrt{2} & 0 \\ 1 & 1 & -\sqrt{2} & 0 \\ 1 & -1 & 0 & \sqrt{2} \\ 1 & -1 & 0 & -\sqrt{2} \end{pmatrix} \begin{pmatrix} 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 \\ -\sqrt{2} & \sqrt{2} & \sqrt{2} & -\sqrt{2} \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \end{pmatrix} \\ &= \frac{1}{4} \begin{pmatrix} 0 & 4 & 4 & 0 \\ 4 & 0 & 0 & 4 \\ 4 & 0 & 2 & 2 \\ 0 & 4 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0.5 & 0.5 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{aligned} \quad (2.123)$$

The square error is equal to:

$$0.5^2 + 0.5^2 + 1^2 = 1.5 \quad (2.124)$$

Note that the error is localised in the bottom-right corner of the reconstructed image.

What do the elementary images of the Haar transform look like?

Figure 2.4 shows the basis images for the expansion of an 8×8 image in terms of the Haar functions. Each of these images has been produced by taking the outer product of a discretised Haar function either with itself or with another one. The numbers along the left and on the top indicate the order of the function used along each row or column, respectively. The discrete values of each image have been scaled in the range $[0, 255]$ for displaying purposes.

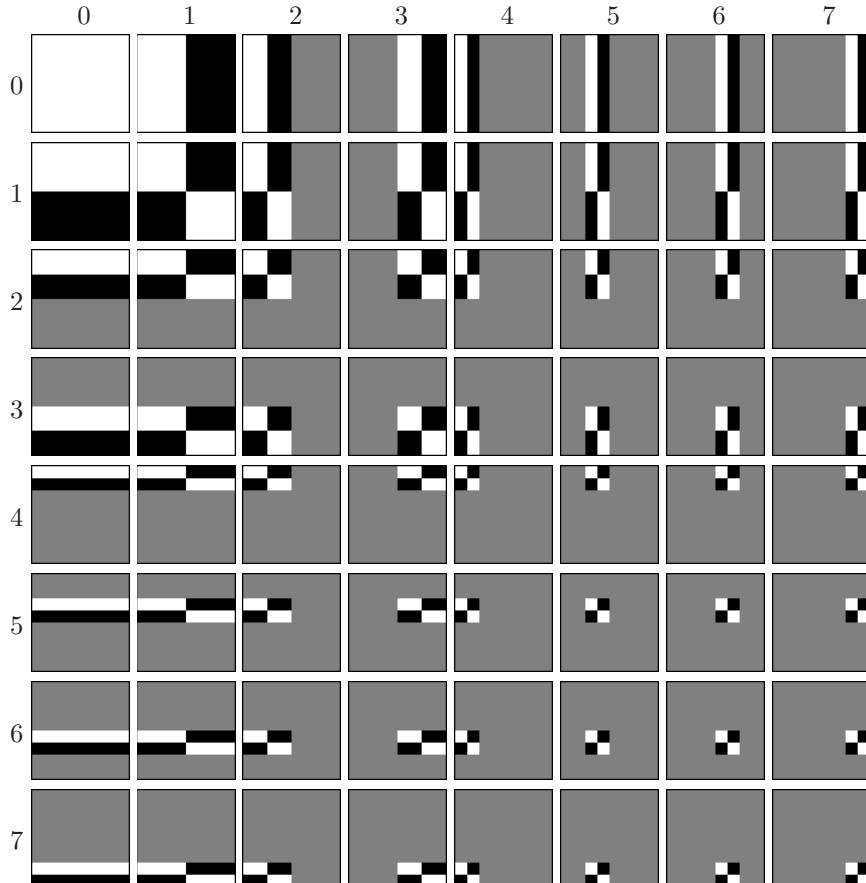


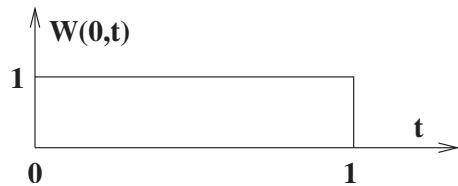
Figure 2.4: Haar transform basis images. In each image, grey means 0, black means a negative and white means a positive number. Note that each image has been scaled separately: black and white indicate different numbers from one image to the next. The vectors used to construct these images were constructed in the same way as the vectors in example 2.19, the only difference being that the t axis was scaled by multiplication with 8 instead of 4, and functions up to $H(7, t)$ had to be defined. Each image here is the outer product of two such vectors. For example, the image in row 4 and column 5 is the outer product of 8×1 vectors $H(4, t)H(5, t)^T$, for t in the range $[0, 8)$ sampled at values $0, 1, 2, \dots, 7$.

Example 2.22

Derive the matrix which can be used to calculate the Walsh transform of a 4×4 image.

First, by using equation (2.112), we calculate and plot the Walsh functions of the continuous variable t which are needed for the calculation of the transformation matrix.

$$W(0, t) = \begin{cases} 1 & \text{for } 0 \leq t < 1 \\ 0 & \text{elsewhere} \end{cases}$$



Case $j = 0, q = 1, \lfloor \frac{j}{2} \rfloor = 0$.

$$W(1, t) = - \left\{ W(0, 2t) - W \left(0, 2 \left(t - \frac{1}{2} \right) \right) \right\} \quad (2.125)$$

We must compute the values of $W(0, 2t)$ and $W \left(0, 2 \left(t - \frac{1}{2} \right) \right)$. We have to use the definition of $W(0, t)$ and examine the range of values of the expression that appears instead of t in the above two functions. For example, for $2t$ to be in the range $[0, 1]$, so that $W(0, 2t) \neq 0$, t must be in the range $[0, 1/2]$. So, we have to consider conveniently chosen ranges of t .

For $0 \leq t < \frac{1}{2}$:

$$0 \leq 2t < 1 \Rightarrow W(0, 2t) = 1 \quad (2.126)$$

$$-\frac{1}{2} \leq t - \frac{1}{2} < 0 \Rightarrow -1 \leq 2 \left(t - \frac{1}{2} \right) < 0 \Rightarrow W \left(0, 2 \left(t - \frac{1}{2} \right) \right) = 0 \quad (2.127)$$

Therefore:

$$W(1, t) = -1 \quad \text{for } 0 \leq t < \frac{1}{2} \quad (2.128)$$

For $\frac{1}{2} \leq t < 1$:

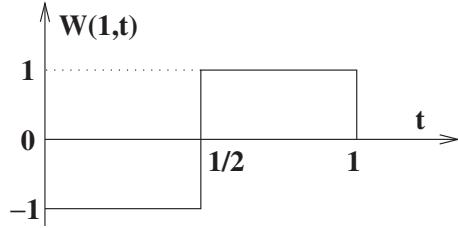
$$1 \leq 2t < 2 \Rightarrow W(0, 2t) = 0 \quad (2.129)$$

$$0 \leq t - \frac{1}{2} < \frac{1}{2} \Rightarrow 0 \leq 2(t - \frac{1}{2}) \leq 1 \Rightarrow W \left(0, 2 \left(t - \frac{1}{2} \right) \right) = 1 \quad (2.130)$$

Therefore:

$$W(1, t) = -(-1) = 1 \quad \text{for } \frac{1}{2} \leq t < 1 \quad (2.131)$$

$$W(1, t) = \begin{cases} -1 & \text{for } 0 \leq t < \frac{1}{2} \\ 1 & \text{for } \frac{1}{2} \leq t < 1 \end{cases}$$



Case $j = 1$, $q = 0$, $\lfloor \frac{j}{2} \rfloor = 0$.

$$W(2, t) = W(1, 2t) - W\left(1, 2\left(t - \frac{1}{2}\right)\right) \quad (2.132)$$

For $0 \leq t < \frac{1}{4}$:

$$0 \leq 2t < \frac{1}{2} \Rightarrow W(1, 2t) = -1 \quad (2.133)$$

$$-\frac{1}{2} \leq t - \frac{1}{2} < -\frac{1}{4} \Rightarrow -1 \leq 2\left(t - \frac{1}{2}\right) < -\frac{1}{2} \Rightarrow W\left(1, 2\left(t - \frac{1}{2}\right)\right) = 0 \quad (2.134)$$

Therefore:

$$W(2, t) = -1 \quad \text{for } 0 \leq t < \frac{1}{4} \quad (2.135)$$

For $\frac{1}{4} \leq t < \frac{1}{2}$:

$$\frac{1}{2} \leq 2t < 1 \Rightarrow W(1, 2t) = 1 \quad (2.136)$$

$$-\frac{1}{4} \leq t - \frac{1}{2} < 0 \Rightarrow -\frac{1}{2} \leq 2\left(t - \frac{1}{2}\right) < 0 \Rightarrow W\left(1, 2\left(t - \frac{1}{2}\right)\right) = 0 \quad (2.137)$$

Therefore:

$$W(2, t) = 1 \quad \text{for } \frac{1}{4} \leq t < \frac{1}{2} \quad (2.138)$$

For $\frac{1}{2} \leq t < \frac{3}{4}$:

$$1 \leq 2t < \frac{3}{2} \Rightarrow W(1, 2t) = 0 \quad (2.139)$$

$$0 \leq t - \frac{1}{2} < \frac{1}{4} \Rightarrow 0 \leq 2\left(t - \frac{1}{2}\right) < \frac{1}{2} \Rightarrow W\left(1, 2\left(t - \frac{1}{2}\right)\right) = -1 \quad (2.140)$$

Therefore:

$$W(2, t) = 1 \quad \text{for} \quad \frac{1}{2} \leq t < \frac{3}{4} \quad (2.141)$$

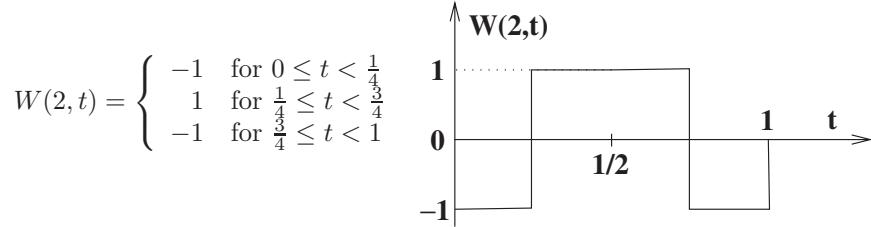
For $\frac{3}{4} \leq t < 1$:

$$\frac{3}{2} \leq 2t < 2 \Rightarrow W(1, 2t) = 0 \quad (2.142)$$

$$\frac{1}{4} \leq t - \frac{1}{2} < \frac{1}{2} \Rightarrow \frac{1}{2} \leq 2 \left(t - \frac{1}{2} \right) < 1 \Rightarrow W \left(1, 2 \left(t - \frac{1}{2} \right) \right) = 1 \quad (2.143)$$

Therefore:

$$W(2, t) = -1 \quad \text{for} \quad \frac{3}{4} \leq t < 1 \quad (2.144)$$



Case $j = 1, q = 1, \lfloor \frac{j}{2} \rfloor = 0$.

$$W(3, t) = - \left\{ W(1, 2t) + W \left(1, 2 \left(t - \frac{1}{2} \right) \right) \right\} \quad (2.145)$$

For $0 \leq t < \frac{1}{4}$:

$$W(1, 2t) = -1, \quad W \left(1, 2 \left(t - \frac{1}{2} \right) \right) = 0 \quad (2.146)$$

Therefore:

$$W(3, t) = 1 \quad \text{for} \quad 0 \leq t < \frac{1}{4} \quad (2.147)$$

For $\frac{1}{4} \leq t < \frac{1}{2}$:

$$W(1, 2t) = 1, \quad W \left(1, 2 \left(t - \frac{1}{2} \right) \right) = 0 \quad (2.148)$$

Therefore:

$$W(3, t) = -1 \quad \text{for } \frac{1}{4} \leq t < \frac{1}{2} \quad (2.149)$$

For $\frac{1}{2} \leq t < \frac{3}{4}$:

$$W(1, 2t) = 0, \quad W\left(1, 2\left(t - \frac{1}{2}\right)\right) = -1 \quad (2.150)$$

Therefore:

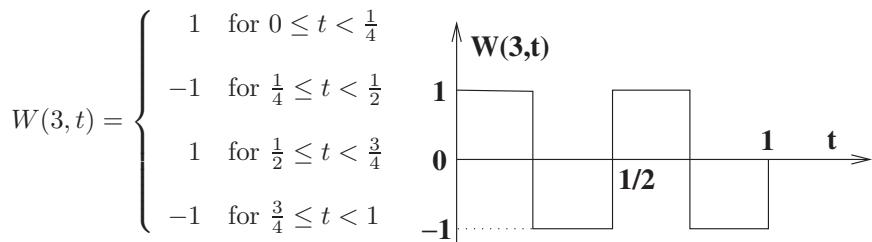
$$W(3, t) = 1 \quad \text{for } \frac{1}{2} \leq t < \frac{3}{4} \quad (2.151)$$

For $\frac{3}{4} \leq t < 1$:

$$W(1, 2t) = 0, \quad W\left(1, 2\left(t - \frac{1}{2}\right)\right) = 1 \quad (2.152)$$

Therefore:

$$W(3, t) = -1 \quad \text{for } \frac{3}{4} \leq t < 1 \quad (2.153)$$



To create a 4×4 matrix, we multiply t with 4 and consider only its integer values ie 0, 1, 2, 3. The first row of the matrix will be formed from $W(0, t)$. The second from $W(1, t)$, the third from $W(2, t)$ and so on:

$$W = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \quad (2.154)$$

This matrix has been normalised by multiplying it with $\frac{1}{2}$ so that $W^T W = I$, where I is the unit matrix.

Example 2.23

Calculate the Walsh transform of image:

$$g = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad (2.155)$$

In the general formula of a separable linear transform with real matrices U and V , $A = UgV^T$, use $U = V = W$ as derived in example 2.22:

$$\begin{aligned} A &= \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \end{pmatrix} \\ &= \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 2 & 0 \\ 2 & 0 & -2 & 0 \\ 2 & 0 & -2 & 0 \\ 2 & 0 & 2 & 0 \end{pmatrix} \\ &= \frac{1}{4} \begin{pmatrix} 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -8 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{aligned} \quad (2.156)$$

Can we define an orthogonal matrix with entries only +1 or -1?

Yes. These are the **Hadamard matrices** named after the mathematician who studied them in 1893. For a general size, these matrices have been shown to exist only for sizes up to 200×200 . Beyond this size, the Hadamard matrices are defined only for sizes that are powers of 2, using a recursive algorithm, as follows:

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{and} \quad H_{2N} = \begin{pmatrix} H_N & H_N \\ H_N & -H_N \end{pmatrix} \quad (2.157)$$

The rows of such matrices can be shown to be discretised Walsh functions. So the Walsh functions may be calculated from these matrices for $N = 2^n$, for n a positive integer.

Box 2.5. Ways of ordering the Walsh functions

Equation (2.112) defines the Walsh functions in what is called **sequency order**, or **Walsh order** or **Walsh-Kaczmarz order**.

Definition of the Walsh functions in terms of the Rademacher functions (see equation (2.116)) results in the Walsh functions being in **natural** or **normal** or **binary** or **dyadic** or **Paley order**. Let us denote these functions by \tilde{W} . Note that as all Rademacher functions start with a positive sign, no matter how many of them we multiply to create a Walsh function, the Walsh function created will always start with a positive value. So, some of the Walsh functions created that way will be equal to the negative of the corresponding Walsh function created by the difference equation (2.112).

The Walsh functions generated from the Hadamard matrices are said to be in **Kronecker** or **lexicographic** ordering. Let us denote these functions by $\tilde{\tilde{W}}$. All these functions also start from a positive value, so again, some of them will be equal to the negative of a Walsh function created by the difference equation (2.112). Because of that, we say that the Walsh functions created from the Rademacher functions and the Hadamard matrices “have positive phase”.

n	Binary	Gray	Nat. order of seq.	Bit-rev. of n	Nat. order of lex.	Relationship sequency -natural	Relationship lexicographic -natural
0	000	000	0	000	0	$W_0(t) = \tilde{W}_0(t)$	$\tilde{W}_0(t) = \tilde{W}_0(t)$
1	001	001	1	100	4	$W_1(t) = -\tilde{W}_1(t)$	$\tilde{W}_1(t) = \tilde{W}_4(t)$
2	010	011	3	010	2	$W_2(t) = -\tilde{W}_3(t)$	$\tilde{W}_2(t) = \tilde{W}_2(t)$
3	011	010	2	110	6	$W_3(t) = \tilde{W}_2(t)$	$\tilde{W}_3(t) = \tilde{W}_6(t)$
4	100	110	6	001	1	$W_4(t) = \tilde{W}_6(t)$	$\tilde{W}_4(t) = \tilde{W}_1(t)$
5	101	111	7	101	5	$W_5(t) = -\tilde{W}_7(t)$	$\tilde{W}_5(t) = \tilde{W}_5(t)$
6	110	101	5	011	3	$W_6(t) = -\tilde{W}_5(t)$	$\tilde{W}_6(t) = \tilde{W}_3(t)$
7	111	100	4	111	7	$W_7(t) = \tilde{W}_4(t)$	$\tilde{W}_7(t) = \tilde{W}_7(t)$

Table 2.1: n is either the sequency or the lexicographic order. Functions W are computed from equation (2.112); functions \tilde{W} are computed from equation (2.116), and functions $\tilde{\tilde{W}}$ are the successive rows of the 8×8 Hadamard matrix.

We can find the corresponding functions of the various orders as follows. Let us call n the sequency order of Walsh function W . We wish to find the natural order \tilde{n} of the same function. We write n in binary code and take its **Gray code**. The Gray code of a binary number is obtained if we add each bit i to bit $i + 1$ using modulo 2 addition. For example, if $n = 7 = 2^2 + 2^1 + 2^0$, and if we use 4-bit representations, its binary code is 0111. We read these digits from left to right. The first digit ($i = 1$) is 0, so adding it

to the second digit has no effect. The second digit is 1 and adding it to the third that is also 1, gives 2, which modulo 2 is 0. The third digit is again 1 and adding it to the fourth, that is also 1, gives again 2, which modulo 2 is 0. So, the Gray code of $n = 7$ is 0100. This number is $2^2 = 4$. Thus, the natural order of the Walsh function with sequency order 7 is 4. In other words, $W_7(t) = \tilde{W}_4(t)$.

To identify the correspondence of a lexicographic order we work as follows: we take the binary number of the lexicographic order and reverse the order of the bits. For example, in 3-bit representation, the binary version of 3 is 011. The bit-reverse version of it is 110. This is number 6 and it is the corresponding natural order. We say that the Walsh functions defined from the Hadamard matrix are “in bit-reverse natural order with positive phase”. In the above example, $\tilde{W}_3(t) = \tilde{W}_6(t)$. Here $\tilde{W}_3(t)$ is the 4th row of Hadamard matrix of size 8×8 . Table 2.1 lists the corresponding order of the first 8 Walsh functions.

Example B2.24

Compute the Walsh function with natural order $n = 7$.

We write n in binary code: $7 = 111$. In order to create the function with this natural order, we shall use equation (2.116), on page 75, with $m = 2$, $b_1 = b_2 = b_3 = 1$. Then:

$$\tilde{W}_7(t) = R_1(t)R_2(t)R_3(t) \quad (2.158)$$

Figure 2.5 shows the plots of the three Rademacher functions we have to multiply and the resultant Walsh function created this way. The formula for $\tilde{W}_7(t)$ is:

$$\tilde{W}_7(t) = \begin{cases} 1 & \text{for } 0 \leq t < \frac{1}{8} \\ -1 & \text{for } \frac{1}{8} \leq t < \frac{3}{8} \\ 1 & \text{for } \frac{3}{8} \leq t < \frac{4}{8} \\ -1 & \text{for } \frac{4}{8} \leq t < \frac{5}{8} \\ 1 & \text{for } \frac{5}{8} \leq t < \frac{7}{8} \\ -1 & \text{for } \frac{7}{8} \leq t < 1 \end{cases} \quad (2.159)$$

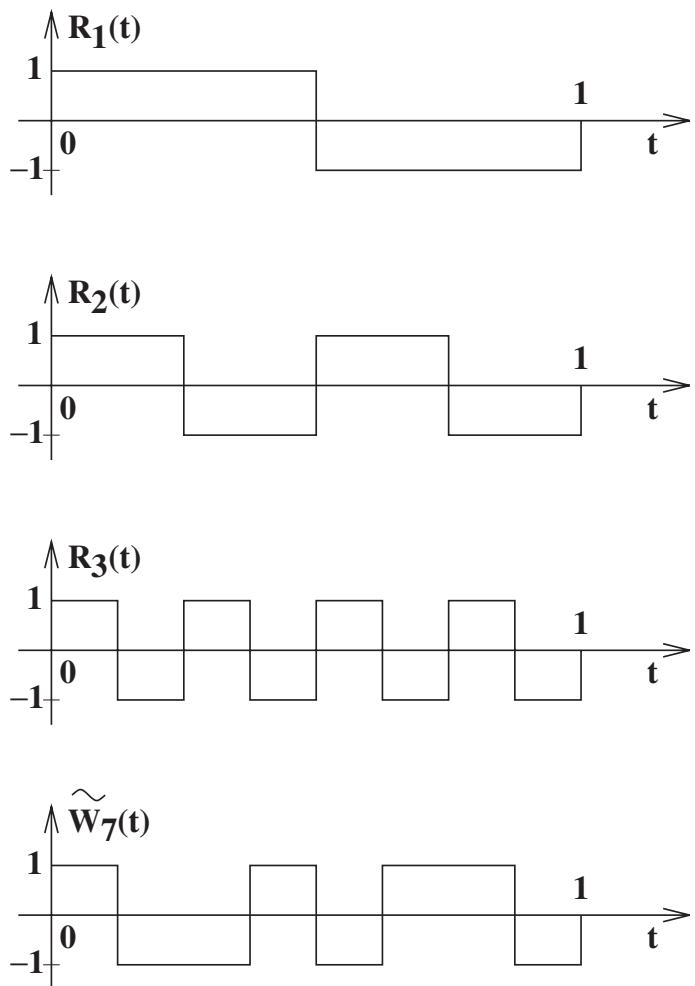


Figure 2.5: The top three functions are multiplied to produce function $\tilde{W}_7(t)$ at the bottom.

What do the basis images of the Hadamard/Walsh transform look like?

Figure 2.6 shows the basis images for the expansion of an 8×8 image in terms of Walsh functions in the order they are produced by applying equation (2.113), on page 74. The basis images were produced by taking the vector outer product of all possible pairs of the discretised 8-samples long Walsh functions.

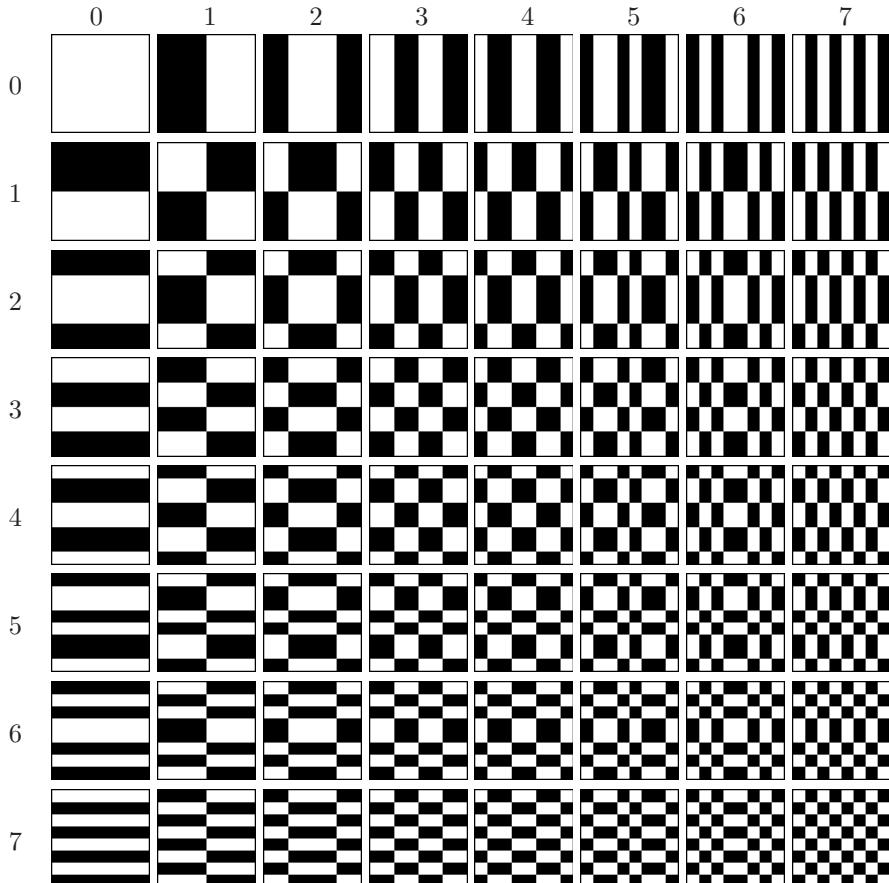


Figure 2.6: Hadamard/Walsh transform basis images. The vectors used to construct these images were constructed in the same way as the vectors in example 2.22, the only difference being that the t axis was scaled by multiplication with 8 instead of 4, and functions up to $W(7, t)$ had to be defined. Each image here is the outer product of two such vectors. For example, the image in row 4 and column 5 is the outer product of 8×1 vectors $W(4, t)W(5, t)^T$, for t in the range $[0, 8]$ sampled at values $0, 1, 2, \dots, 7$.

Example 2.25

Show the different stages of the Haar transform of the image of example 2.15, on page 69.

We can perform the reconstruction by keeping only basis images made up from one, up to eight Haar functions. Each such reconstruction will be an improved approximation

of the original image over the previous approximation. The series of images we obtain by these reconstructions are shown in figure 2.7. For example, figure 2.7b is the reconstructed image when only the coefficients that multiply the four basis images at the top left corner of figure 2.4 are retained. These four basis images are created from the first two Haar functions, $H(0, t)$ and $H(1, t)$. Image 2.7g is reconstructed when all the coefficients that multiply the basis images along the bottom row and the right column in figure 2.4 are set to 0. In other words, the basis images used for this reconstruction were created from the first seven Haar functions, ie $H(0, t), H(1, t), \dots, H(6, t)$.

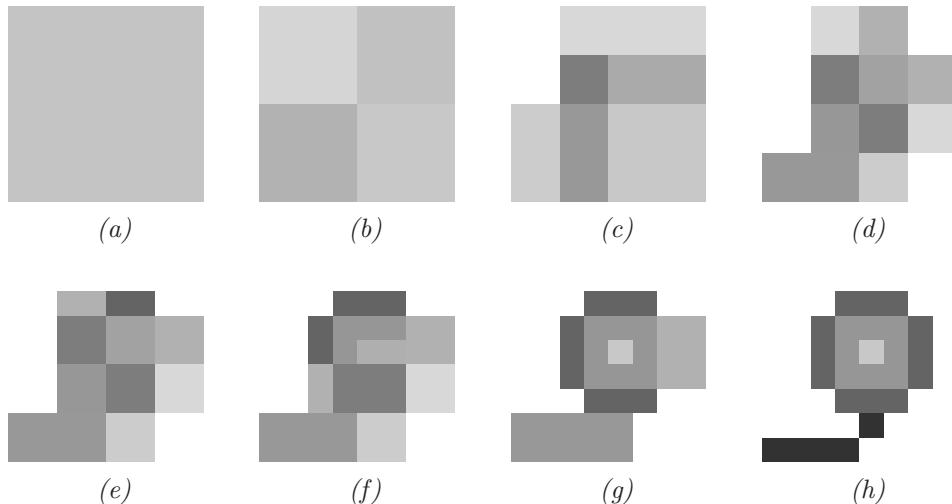


Figure 2.7: Reconstructed images when the basis images used are those created from the first one, two, three, ..., eight Haar functions, from top left to bottom right, respectively.

The sum of the square errors for each reconstructed image is as follows:

Square error for image 2.7a:	366394
Square error for image 2.7b:	356192
Square error for image 2.7c:	291740
Square error for image 2.7d:	222550
Square error for image 2.7e:	192518
Square error for image 2.7f:	174625
Square error for image 2.7g:	141100
Square error for image 2.7h:	0

Example 2.26

Show the different stages of the Walsh/Hadamard transform of the image of example 2.15, on page 69.

We can perform the reconstruction by keeping only basis images made up from one, up to eight Walsh functions. Each such reconstruction will be an improved approximation of the original image over the previous approximation. The series of images we obtain by these reconstructions are shown in figure 2.8. For example, figure 2.8f has been reconstructed from the inverse Walsh/Hadamard transform, by setting to 0 all elements of the transformation matrix that multiply the basis images in the bottom two rows and the two rightmost columns in figure 2.6. These omitted basis images are those that are created from functions $W(6, t)$ and $W(7, t)$.

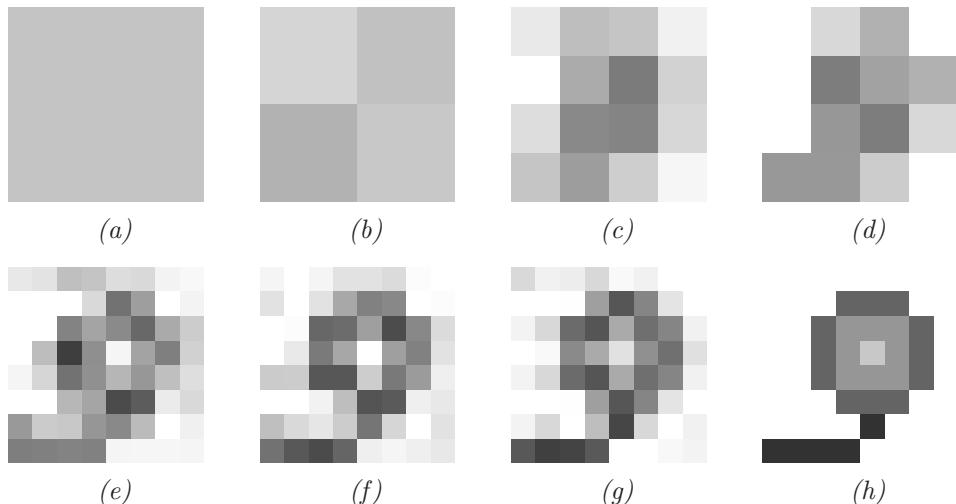


Figure 2.8: Reconstructed images when the basis images used are those created from the first one, two, three,..., eight Walsh functions, from top left to bottom right, respectively.

The sum of the square errors for each reconstructed image is as follows.

Square error for image 2.8a:	366394
Square error for image 2.8b:	356190
Square error for image 2.8c:	262206
Square error for image 2.8d:	222550
Square error for image 2.8e:	148029
Square error for image 2.8f:	92078
Square error for image 2.8g:	55905
Square error for image 2.8h:	0

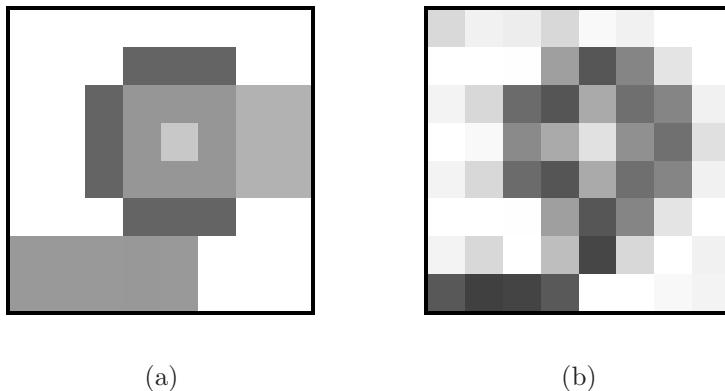


Figure 2.9: The flower image approximated with the same number of terms using (a) the Haar transform and (b) the Walsh transform. Note how the reconstruction with Haar has localised error, at the positions where the expansion coefficients have been set to 0, while the Walsh reconstruction distributes the error over the whole image.

What are the advantages and disadvantages of the Walsh and the Haar transforms?

From figure 2.4, on page 80, notice that the higher order Haar basis images use the same basic pattern that scans the whole image, as if every basis image attempts to capture more accurately the local characteristics of the image focusing every time at one place only. For example, all the 16 basis images in the bottom right quadrant of figure 2.4 use a window of 2×2 pixels to reproduce detail in various parts of the image. If we are not interested in that level of detail, we can set the corresponding 16 coefficients of the transform to zero. Alternatively, if, for example, we are not interested in the details only on the right part of the image, we may set to 0 all coefficients that multiply the basis images of the last column of figure 2.4. In other words, the Haar basis functions allow us to reconstruct with different levels of detail different parts of an image.

In contrast, higher order Walsh basis images try to approximate the image as a whole, with uniformly distributed detail structure. This is because Walsh functions cannot take the 0 value. Notice how this difference between the two bases is reflected in the reconstructed images: both images 2.7g (repeated here in figure 2.9a) and 2.8g (repeated here in figure 2.9b) have been reconstructed by retaining the same number of basis images. In figure 2.9a the flower has been almost fully reconstructed apart from some details on the right and at the bottom, because the omitted basis images were those that would describe the image in those locations, and the image happened to have significant detail there. That is why the reconstructed error in this case is higher for the Haar than the Walsh case. Notice that the error in the Walsh reconstruction is uniformly distributed over the whole image.

Walsh transforms have the advantage over Haar transforms that the Walsh functions take up only two values, namely +1 or -1, and thus they are easily implemented in a computer as their values correspond to binary logic.

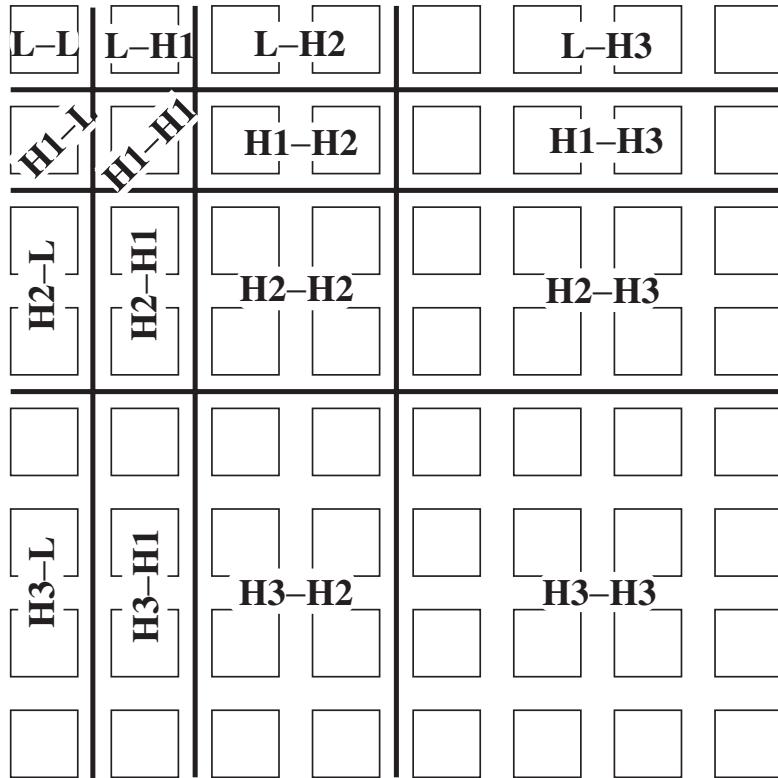


Figure 2.10: The empty panels shown correspond to the basis images shown in figure 2.4. The thick lines divide them into sets of elementary images of the same resolution. Letters L and H are used to indicate low and high resolution, respectively. The numbers next to letter H indicates which level of high resolution. The pairs of letters used indicate which resolution we have along the vertical and horizontal axis. For example, pair L-H2 indicates that the corresponding panels have low resolution along the vertical axis, but high second order resolution along the horizontal axis.

What is the Haar wavelet?

The property of the Haar basis functions to concentrate at one part of the image at a time is a characteristic property of a more general class of functions called **wavelets**. The **Haar wavelets** are all scaled and translated versions of the same function. For an 8×8 image they are shown in the 16 bottom right panels of figure 2.4 for the finest scale of resolution. The function represented by the top left panel of figure 2.4, ie the average flat image, is called the **scaling function**. The basis images represented by the other panels in the first column and the top row of figure 2.4 are produced from combinations of the scaling function and the wavelet. The rest of the panels correspond to intermediate scales and they may be grouped in sets of the same resolution panels, that cover the full image (figure 2.10). All panels together constitute a complete basis in terms of which any 8×8 image may be expanded.

2.3 Discrete Fourier Transform

What is the discrete version of the Fourier transform (DFT)?

The 1D discrete Fourier transform (DFT) of a function $f(k)$, defined at discrete points $k = 0, 1, \dots, N - 1$, is defined as:

$$F(m) \equiv \frac{1}{N} \sum_{k=0}^{N-1} f(k) \exp \left[-j \frac{2\pi mk}{N} \right] \quad (2.160)$$

The 2D discrete Fourier transform for an $N \times N$ image is defined as:

$$\alpha_{mn} \equiv \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g_{kl} e^{-j2\pi \frac{km+nl}{N}} \quad (2.161)$$

Note that in all previous sections the index of the elements of a signal or an image was taking values starting from 1. If we had retained that convention here, in the exponent of the exponential function in (2.160), instead of having k we should have had $(k - 1)$. For the sake of simplicity, in this section, we assume that for an N -sample long signal, the indices start from 0 and go up to $N - 1$, instead of starting from 1 and going up to N .

Unlike the other transforms that were developed directly in the discrete domain, this transform was initially developed in the continuous domain. To preserve this “historical” consistency, we shall go back into using function arguments rather than indices. Further, because we shall have to associate Fourier transforms of different functions, we shall use the convention of the Fourier transform being denoted by the same letter as the function, but with a hat on the top. Different numbers of hats will be used to distinguish the Fourier transforms that refer to different versions of the same function. The reason for this will become clear when the case arises. So, for the time being, we define the Fourier transform of an $M \times N$ digital image as follows:

$$\hat{g}(m, n) \equiv \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g(k, l) e^{-j2\pi \left[\frac{km}{M} + \frac{ln}{N} \right]} \quad (2.162)$$

We must think of this formula as a “slot machine”: when we slot in a function, out pops its DFT:

$$\underbrace{\dots}_{DFT} = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \underbrace{\dots}_{function} e^{-j2\pi \left[\frac{km}{M} + \frac{ln}{N} \right]} \quad (2.163)$$

This way of thinking will be very useful when we try to prove the various properties of the DFT.

Example B2.27

For S and t integers, show that:

$$\sum_{m=0}^{S-1} e^{j2\pi t \frac{m}{S}} = S\delta(t) \quad (2.164)$$

This is a geometric progression with S elements, first term 1 ($m = 0$) and ratio $q \equiv e^{j2\pi \frac{t}{S}}$. The sum of the first S terms of such a geometric progression is given by:

$$\sum_{m=0}^{S-1} q^m = \frac{q^S - 1}{q - 1} \quad \text{for } q \neq 1 \quad (2.165)$$

For $q \neq 1$, ie for $e^{j2\pi \frac{t}{S}} \neq 1$, ie for $t \neq 0$, sum (2.164) is, therefore, equal to:

$$\sum_{m=0}^{S-1} e^{j2\pi t \frac{m}{S}} = \frac{e^{j2\pi t} - 1}{e^{j2\pi \frac{t}{S}} - 1} = \frac{\cos(2\pi t) + j \sin(2\pi t) - 1}{e^{j2\pi \frac{t}{S}} - 1} = \frac{1 + j0 - 1}{e^{j2\pi \frac{t}{S}} - 1} = 0 \quad (2.166)$$

If, however, $t = 0$, all terms in (2.164) are equal to 1 and we have $\sum_{m=0}^{S-1} 1 = S$. So

$$\sum_{m=0}^{S-1} e^{j2\pi t \frac{m}{S}} = \begin{cases} S & \text{if } t = 0 \\ 0 & \text{if } t \neq 0 \end{cases} \quad (2.167)$$

and (2.164) follows.

Box 2.6. What is the inverse discrete Fourier transform?

To solve equation (2.162) for $g(k, l)$, we multiply both sides with $e^{j2\pi [\frac{qm}{M} + \frac{pn}{N}]}$ and sum over all m and n from 0 to $M - 1$ and $N - 1$, respectively. We get:

$$\begin{aligned} & \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \hat{g}(m, n) e^{j2\pi [\frac{qm}{M} + \frac{pn}{N}]} \\ &= \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g(k, l) e^{j2\pi [\frac{m(q-k)}{M} + \frac{n(p-l)}{N}]} \\ &= \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g(k, l) \sum_{m=0}^{M-1} e^{j2\pi m \frac{q-k}{M}} \sum_{n=0}^{N-1} e^{j2\pi n \frac{p-l}{N}} \end{aligned} \quad (2.168)$$

Applying formula (2.164) once for $t \equiv q - k$ and once for $t \equiv p - l$ and substituting into equation (2.168), we deduce that the right-hand side of (2.168) is

$$\frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g(k, l) M\delta(q - k) N\delta(p - l) \quad (2.169)$$

where $\delta(a - b)$ is 0 unless $a = b$. Therefore, the above expression is $g(q, p)$, ie:

$$g(q, p) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \hat{g}(m, n) e^{j2\pi[\frac{qm}{M} + \frac{pn}{N}]} \quad (2.170)$$

This is the inverse 2D discrete Fourier transform.

How can we write the discrete Fourier transform in a matrix form?

We construct matrix U with elements

$$U_{x\alpha} = \frac{1}{N} \exp \left[-j \frac{2\pi x\alpha}{N} \right] \quad (2.171)$$

where x takes values $0, 1, \dots, N - 1$ along each column and α takes the same values along each row. Notice that U is symmetric, ie $U^T = U$. Then, according to equation (2.3), on page 48, the 2D discrete Fourier transform of an image g is given by:

$$\hat{g} = UgU \quad (2.172)$$

Example 2.28

Derive the matrix with which the discrete Fourier transform of a 4×4 image may be obtained.

Apply formula (2.171) with $N = 4$, $0 \leq x \leq 3$, $0 \leq \alpha \leq 3$:

$$U = \frac{1}{4} \begin{pmatrix} e^{-j\frac{2\pi}{4} \times 0} & e^{-j\frac{2\pi}{4} \times 0} & e^{-j\frac{2\pi}{4} \times 0} & e^{-j\frac{2\pi}{4} \times 0} \\ e^{-j\frac{2\pi}{4} \times 0} & e^{-j\frac{2\pi}{4} \times 1} & e^{-j\frac{2\pi}{4} \times 2} & e^{-j\frac{2\pi}{4} \times 3} \\ e^{-j\frac{2\pi}{4} \times 0} & e^{-j\frac{2\pi}{4} \times 2} & e^{-j\frac{2\pi}{4} \times 4} & e^{-j\frac{2\pi}{4} \times 6} \\ e^{-j\frac{2\pi}{4} \times 0} & e^{-j\frac{2\pi}{4} \times 3} & e^{-j\frac{2\pi}{4} \times 6} & e^{-j\frac{2\pi}{4} \times 9} \end{pmatrix} \quad (2.173)$$

Or:

$$\begin{aligned} U &= \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & e^{-j\frac{\pi}{2}} & e^{-j\pi} & e^{-j\frac{3\pi}{2}} \\ 1 & e^{-j\pi} & e^{-j2\pi} & e^{-j3\pi} \\ 1 & e^{-j\frac{3\pi}{2}} & e^{-j3\pi} & e^{-j\frac{9\pi}{2}} \end{pmatrix} \\ &= \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & e^{-j\frac{\pi}{2}} & e^{-j\pi} & e^{-j\frac{3\pi}{2}} \\ 1 & e^{-j\pi} & 1 & e^{-j\pi} \\ 1 & e^{-j\frac{3\pi}{2}} & e^{-j\pi} & e^{-j\frac{\pi}{2}} \end{pmatrix} \end{aligned} \quad (2.174)$$

Recall that:

$$\begin{aligned} e^{-j\frac{\pi}{2}} &= \cos \frac{\pi}{2} - j \sin \frac{\pi}{2} = -j \\ e^{-j\pi} &= \cos \pi - j \sin \pi = -1 \\ e^{-j\frac{3\pi}{2}} &= \cos \frac{3\pi}{2} - j \sin \frac{3\pi}{2} = j \end{aligned} \quad (2.175)$$

Therefore:

$$U = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \quad (2.176)$$

Example 2.29

Use matrix U of example 2.28 to compute the discrete Fourier transform of the following image:

$$g = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.177)$$

Calculate first gU :

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \quad (2.178)$$

Multiply the result with U from the left to get $UgU = \hat{g}$ (the discrete Fourier transform of g):

$$\begin{aligned}
 & \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \frac{1}{4} \begin{pmatrix} 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \\
 = & \frac{1}{16} \begin{pmatrix} 4 & -4 & 4 & -4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.179}
 \end{aligned}$$

Example 2.30

Using the definition of DFT by formula (2.162), verify that the DFT values of image (2.177) for $m = 1$ and $n = 0$, and for $m = 0$ and $n = 1$ are as worked out in example 2.29.

Applying (2.162) for $M = N = 4$, we obtain:

$$\hat{g}(m, n) = \frac{1}{16} \sum_{k=0}^3 \sum_{l=0}^3 g(k, l) e^{-j2\pi \frac{km+nl}{4}} \tag{2.180}$$

For image (2.177) we have $g(0, 2) = g(1, 2) = g(2, 2) = g(3, 2) = 1$ and all other $g(k, l)$ values are 0. Then:

$$\hat{g}(m, n) = \frac{1}{16} \left[e^{-j2\pi \frac{2n}{4}} + e^{-j2\pi \frac{m+2n}{4}} + e^{-j2\pi \frac{2m+2n}{4}} + e^{-j2\pi \frac{3m+2n}{4}} \right] \tag{2.181}$$

For $m = 1$ and $n = 0$ we obtain:

$$\begin{aligned}
 \hat{g}(1, 0) &= \frac{1}{16} \left[e^0 + e^{-j2\pi \frac{1}{4}} + e^{-j2\pi \frac{2}{4}} + e^{-j2\pi \frac{3}{4}} \right] \\
 &= \frac{1}{16} \left[1 + \cos \frac{\pi}{2} - j \sin \frac{\pi}{2} + \cos \pi - j \sin \pi + \cos \frac{3\pi}{2} - j \sin \frac{3\pi}{2} \right] \\
 &= \frac{1}{16} [1 - j - 1 + j] = 0 \tag{2.182}
 \end{aligned}$$

For $m = 0$ and $n = 1$ we obtain:

$$\begin{aligned}
 \hat{g}(0, 1) &= \frac{1}{16} \left[e^{-j2\pi \frac{2}{4}} + e^{-j2\pi \frac{2}{4}} + e^{-j2\pi \frac{2}{4}} + e^{-j2\pi \frac{2}{4}} \right] \\
 &= \frac{1}{16} [4(\cos \pi - j \sin \pi)] = -\frac{1}{4} \tag{2.183}
 \end{aligned}$$

We note that both values deduced agree with the $\hat{g}(1, 0)$ and $\hat{g}(0, 1)$ we worked out using the matrix multiplication approach in example 2.29.

Is matrix U used for DFT unitary?

We must show that any row of this matrix is orthogonal to the complex conjugate of any other row¹. Using definition (2.171), the product of rows corresponding to $x = x_1$ and $x = x_2$ is given by

$$\frac{1}{N^2} \sum_{\alpha=0}^{N-1} e^{-j\frac{2\pi x_1 \alpha}{N}} e^{j\frac{2\pi x_2 \alpha}{N}} = \frac{1}{N^2} \sum_{\alpha=0}^{N-1} e^{j\frac{2\pi(x_2-x_1)\alpha}{N}} = \frac{1}{N^2} N\delta(x_2 - x_1) = \frac{1}{N}\delta(x_2 - x_1) \quad (2.184)$$

where we made use of equation (2.164), on page 95.

So, UU^H does not produce the unit matrix, but a diagonal matrix with all its elements along the diagonal equal to $1/N$. On the other hand, matrix

$$\tilde{U} \equiv \frac{1}{\sqrt{N}} U \quad (2.185)$$

is unitary. However, if we were to compute DFT using matrix \tilde{U} instead of U , the produced DFT would not have been identical with that produced by using the conventional definition formulae (2.160) and (2.161). To have full agreement between the matrix version of DFT and the formula version we should also redefine DFT as

$$F(m) \equiv \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} f(k) \exp \left[-j \frac{2\pi mk}{N} \right] \quad (2.186)$$

for 1D and as

$$\alpha_{mn} \equiv \frac{1}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g_{kl} e^{-j2\pi \frac{km+nl}{N}} \quad (2.187)$$

for 2D. These are perfectly acceptable alternative definitions. However, they have certain consequences for some theorems, in the sense that they alter some scaling constants. A scaling constant is not usually a problem in a transformation, as long as one is consistent in all subsequent manipulations and careful when taking the inverse transform. In other words, you either always use U given by (2.171) and definitions (2.160) and (2.161), remembering that U is unitary apart from a scaling constant, or you always use \tilde{U} given by (2.185) and definitions (2.186) and (2.187), remembering that some theorems may involve different multiplicative constants from those found in conventional books on DFT.

Example 2.31

Show that

$$e^{-j\frac{2\pi}{N} \times x} = e^{-j\frac{2\pi}{N} \times [\text{mod}_N(x)]} \quad (2.188)$$

where x is an integer.

For integer x we may write $x = qN + r$, where q is the integer number of times N fits in x and r is the residual. For example, if $N = 32$ and $x = 5$, $q = 0$ and $r = 5$. If

¹The need to take the complex conjugate of the second row arises because when we multiply imaginary numbers, in order to get 1, we have to multiply j with $-j$, ie its complex conjugate, rather than j with j .

$N = 32$ and $x = 36$, $q = 1$ and $r = 4$. The residual r is called **modulus of x over N** and is denoted as $\text{mod}_N(x)$.

A complex exponential $e^{j\phi}$ may be written as $\cos \phi + j \sin \phi$. We may, therefore, write:

$$\begin{aligned}
 e^{-j\frac{2\pi}{N} \times x} &= e^{-j\frac{2\pi}{N} \times (qN+r)} \\
 &= e^{-j\frac{2\pi}{N} qN - j\frac{2\pi}{N} r} \\
 &= e^{-j2\pi q} e^{-j\frac{2\pi}{N} r} \\
 &= [\cos(2\pi) - j \sin(2\pi)] e^{-j\frac{2\pi}{N} r} \\
 &= (1 - j \times 0) e^{-j\frac{2\pi}{N} r} \\
 &= e^{-j\frac{2\pi}{N} \times [\text{mod}_N(x)]}
 \end{aligned} \tag{2.189}$$

Example 2.32

Derive matrix U needed for the calculation of the DFT of an 8×8 image.

By applying formula (2.171) with $N = 8$ we obtain:

$$64U = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-j\frac{2\pi}{8} \times 1} & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 3} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 5} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 7} \\ 1 & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 8} & e^{-j\frac{2\pi}{8} \times 10} & e^{-j\frac{2\pi}{8} \times 12} & e^{-j\frac{2\pi}{8} \times 14} \\ 1 & e^{-j\frac{2\pi}{8} \times 3} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 9} & e^{-j\frac{2\pi}{8} \times 12} & e^{-j\frac{2\pi}{8} \times 15} & e^{-j\frac{2\pi}{8} \times 18} & e^{-j\frac{2\pi}{8} \times 21} \\ 1 & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 8} & e^{-j\frac{2\pi}{8} \times 12} & e^{-j\frac{2\pi}{8} \times 16} & e^{-j\frac{2\pi}{8} \times 20} & e^{-j\frac{2\pi}{8} \times 24} & e^{-j\frac{2\pi}{8} \times 28} \\ 1 & e^{-j\frac{2\pi}{8} \times 5} & e^{-j\frac{2\pi}{8} \times 10} & e^{-j\frac{2\pi}{8} \times 15} & e^{-j\frac{2\pi}{8} \times 20} & e^{-j\frac{2\pi}{8} \times 25} & e^{-j\frac{2\pi}{8} \times 30} & e^{-j\frac{2\pi}{8} \times 35} \\ 1 & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 12} & e^{-j\frac{2\pi}{8} \times 18} & e^{-j\frac{2\pi}{8} \times 24} & e^{-j\frac{2\pi}{8} \times 30} & e^{-j\frac{2\pi}{8} \times 36} & e^{-j\frac{2\pi}{8} \times 42} \\ 1 & e^{-j\frac{2\pi}{8} \times 7} & e^{-j\frac{2\pi}{8} \times 14} & e^{-j\frac{2\pi}{8} \times 21} & e^{-j\frac{2\pi}{8} \times 28} & e^{-j\frac{2\pi}{8} \times 35} & e^{-j\frac{2\pi}{8} \times 42} & e^{-j\frac{2\pi}{8} \times 49} \end{pmatrix}$$

Using formula (2.188), the above matrix may be simplified to:

$$U = \frac{1}{64} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-j\frac{2\pi}{8} \times 1} & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 3} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 5} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 7} \\ 1 & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 0} & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 6} \\ 1 & e^{-j\frac{2\pi}{8} \times 3} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 1} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 7} & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 5} \\ 1 & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 0} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 0} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 0} & e^{-j\frac{2\pi}{8} \times 4} \\ 1 & e^{-j\frac{2\pi}{8} \times 5} & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 7} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 1} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 3} \\ 1 & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 0} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 2} \\ 1 & e^{-j\frac{2\pi}{8} \times 7} & e^{-j\frac{2\pi}{8} \times 6} & e^{-j\frac{2\pi}{8} \times 5} & e^{-j\frac{2\pi}{8} \times 4} & e^{-j\frac{2\pi}{8} \times 3} & e^{-j\frac{2\pi}{8} \times 2} & e^{-j\frac{2\pi}{8} \times 1} \end{pmatrix} \tag{2.190}$$

Which are the elementary images in terms of which DFT expands an image?

As the kernel of DFT is a complex function, these images are complex. They may be created by taking the outer product of any two rows of matrix U . Figure 2.11 shows the real parts of these elementary images and figure 2.12 the imaginary parts, for the U matrix computed in example 2.32.

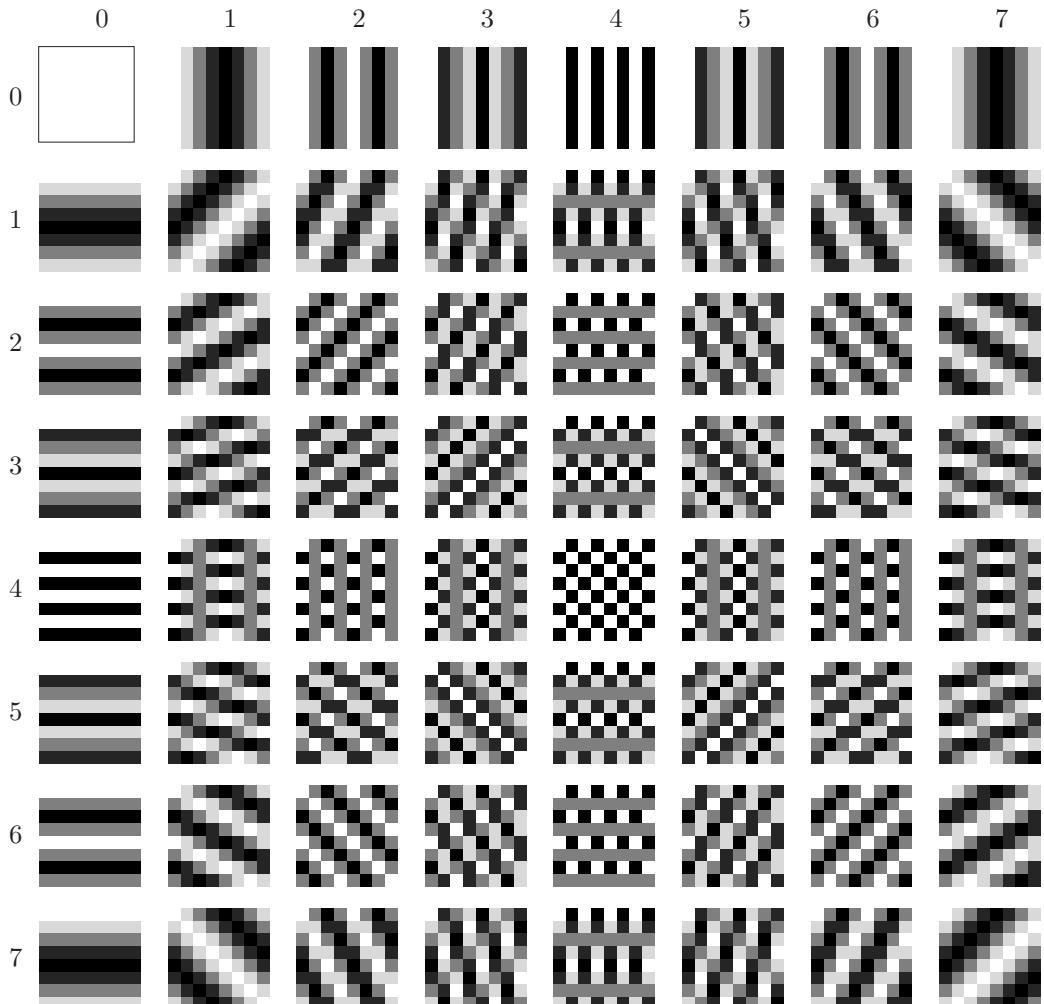


Figure 2.11: Real part of the Fourier transform basis images, appropriate for expanding an 8×8 image. All panels have been scaled together for presentation purposes.

The values of all the images have been linearly scaled to vary between 0 (black) and 255 (white). The numbers along the left and the top indicate which transposed row of matrix U was multiplied with which row to produce the corresponding image.

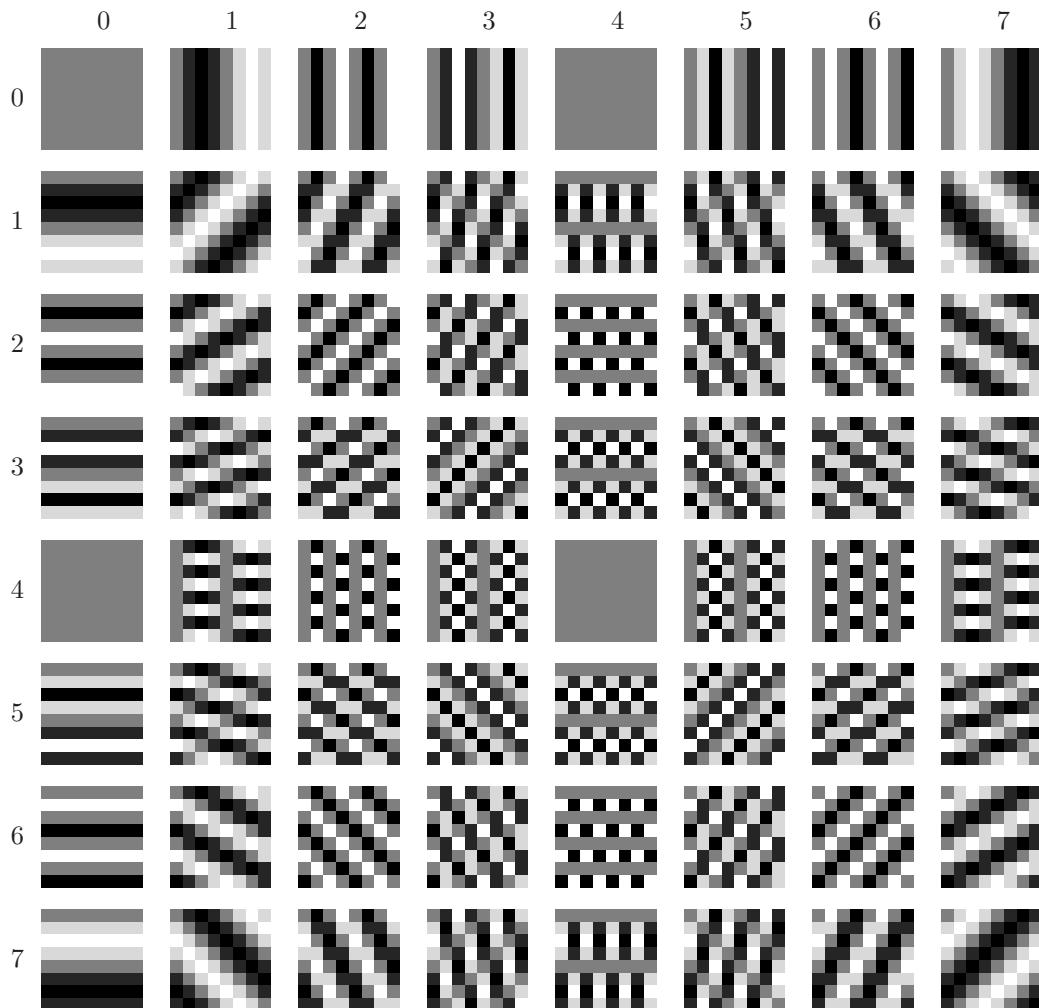


Figure 2.12: Imaginary part of the Fourier transform basis images, appropriate for expanding an 8×8 image. All panels have been scaled together for presentation purposes.

Example 2.33

Compute the real and imaginary parts of the discrete Fourier transform of image:

$$g = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.191)$$

We shall use matrix U of example 2.28. We have to compute $\hat{g} = UgU$. We start by computing first gU :

$$gU = \frac{1}{4} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 2 & -1-j & 0 & -1+j \\ 2 & -1-j & 0 & -1+j \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.192)$$

We then multiply this result with U from the left:

$$\begin{aligned} \hat{g} &= \frac{1}{16} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 2 & -1-j & 0 & -1+j \\ 2 & -1-j & 0 & -1+j \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ &= \frac{1}{16} \begin{pmatrix} 4 & -2-2j & 0 & -2+2j \\ -2-2j & 2j & 0 & 2 \\ 0 & 0 & 0 & 0 \\ -2+2j & 2 & 0 & -2j \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{4} & -\frac{1+j}{8} & 0 & \frac{-1+j}{8} \\ -\frac{1+j}{8} & \frac{j}{8} & 0 & \frac{1}{8} \\ 0 & 0 & 0 & 0 \\ \frac{-1+j}{8} & \frac{1}{8} & 0 & -\frac{j}{8} \end{pmatrix} \end{aligned} \quad (2.193)$$

Splitting the real and imaginary parts, we obtain:

$$\Re(A) = \begin{pmatrix} \frac{1}{4} & -\frac{1}{8} & 0 & -\frac{1}{8} \\ -\frac{1}{8} & 0 & 0 & \frac{1}{8} \\ 0 & 0 & 0 & 0 \\ -\frac{1}{8} & \frac{1}{8} & 0 & 0 \end{pmatrix} \quad \text{and} \quad \Im(A) = \begin{pmatrix} 0 & -\frac{1}{8} & 0 & \frac{1}{8} \\ -\frac{1}{8} & \frac{1}{8} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{8} & 0 & 0 & -\frac{1}{8} \end{pmatrix} \quad (2.194)$$

Example 2.34

Show the different stages of the approximation of the image of example 2.15, on page 69, by its Fourier transform.

The eight images shown in figure 2.13 are the reconstructed images when one, two, ..., eight lines of matrix U were used for the reconstruction. The sum of the squared errors for each reconstructed image are:

<i>Square error for image 2.13a:</i>	366394
<i>Square error for image 2.13b:</i>	285895
<i>Square error for image 2.13c:</i>	234539
<i>Square error for image 2.13d:</i>	189508
<i>Square error for image 2.13e:</i>	141481
<i>Square error for image 2.13f:</i>	119612
<i>Square error for image 2.13g:</i>	71908
<i>Square error for image 2.13h:</i>	0

Note that the reconstructed images are complex and in each case we consider only the real part of the reconstructed image.

From the basis images shown in figures 2.11 and 2.12, we can see that after the rows and columns marked with number 3, the basis images are symmetrically repeated. This is due to the nature of the complex exponential functions. This also means that by the 4th reconstruction shown in figure 2.13, the highest resolution details of the image have already been in place, and after that point, the extra components that are incorporated are gradually improving the details at various scales in the real part and gradually reduce the information in the imaginary part, which becomes exactly 0 for the full reconstruction.

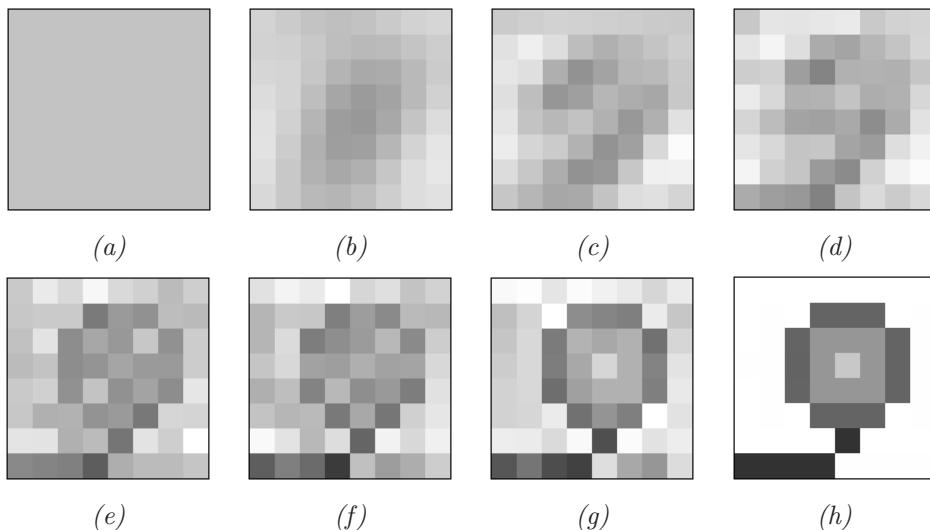


Figure 2.13: Reconstructed image when the basis images used are those created from the first one, two, ..., eight lines of matrix U of example 2.32, from top left to bottom right, respectively.

Why is the discrete Fourier transform more commonly used than the other transforms?

The major advantage of the discrete Fourier transform over the Walsh transform is that it obeys the **convolution theorem**. One may define a corresponding theorem for the Walsh functions, but the relationship between the Walsh transform and the convolution is not as simple and it cannot be implemented cheaply on a computer. The convolution theorem makes the Fourier transform by far the most attractive in image processing.

Apart from that, the Fourier transform uses very detailed basis functions, so in general it can approximate an image with smaller error than the other transforms for a fixed number of terms retained. This may be judged from the reconstruction errors of example 2.34, when compared with the reconstruction errors of examples 2.25 and 2.26. We must compare the errors for reconstructed images (a), (b) and (d) which correspond to keeping the first 2^0 , 2^1 , and 2^2 basis functions, respectively. In this particular example, however, the Walsh transform seems to produce better approximations for high numbers of retained coefficients, as judged by the square error, although the Fourier reconstructions appear visually more acceptable. This touches upon the problem of expressing the quality of an image by some function of its values: there are no really quantitative measures of image quality that correspond to the perceived quality of an image by human viewers.

In any case, we must remember that when we say we retained n number of basis images, in the case of the Fourier transform we actually require $2n$ coefficients for the reconstruction, while in the case of Haar and Walsh transforms we require only n coefficients. This is because the Fourier coefficients are complex and both their real and imaginary parts have to be stored or transmitted.

What does the convolution theorem state?

The convolution theorem states that: the Fourier transform of the convolution of two functions is *proportional* to the product of the individual Fourier transforms of the two functions. If the functions are images defined over a finite space, this theorem is true only if we assume that each image is repeated periodically in all directions.

Box 2.7. If a function is the convolution of two other functions, what is the relationship of its DFT with the DFTs of the two functions?

Assume that we convolve two discrete 2D functions $g(n, m)$ and $w(n, m)$ to produce another function $v(n, m)$:

$$v(n, m) = \sum_{n'=0}^{N-1} \sum_{m'=0}^{M-1} g(n - n', m - m') w(n', m') \quad (2.195)$$

Let us say that the discrete Fourier transforms of these three functions are \hat{v} , \hat{g} and \hat{w} , respectively. To find a relationship between them, we shall try to calculate the DFT of

$v(n, m)$. For this purpose, we multiply both sides of equation (2.195) with the kernel

$$\frac{1}{NM} \exp \left[-j2\pi \left(\frac{pn}{N} + \frac{qm}{M} \right) \right] \quad (2.196)$$

and sum over all m and n . Equation (2.195) then becomes:

$$\begin{aligned} & \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} v(n, m) e^{-j2\pi[\frac{pn}{N} + \frac{qm}{M}]} \\ &= \frac{1}{NM} \sum_{n'=0}^{N-1} \sum_{m'=0}^{M-1} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g(n-n', m-m') w(n', m') e^{-j2\pi[\frac{pn}{N} + \frac{qm}{M}]} \quad (2.197) \end{aligned}$$

We recognise the left-hand side of this expression to be the discrete Fourier transform of v , ie:

$$\hat{v}(p, q) = \frac{1}{NM} \sum_{n'=0}^{N-1} \sum_{m'=0}^{M-1} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g(n-n', m-m') w(n', m') e^{-j2\pi \frac{pn}{N}} e^{-j2\pi \frac{qm}{M}}$$

We would like to split the expression on the right-hand side into the product of two double sums, which eventually will be identified as the DFTs of g and w . To achieve this, we must have independent indices for g and w . We introduce new indices:

$$n - n' \equiv n'', \quad m - m' \equiv m'' \quad (2.198)$$

Then $n = n' + n''$, $m = m' + m''$ and we must find the limits of m'' and n'' . To do that, we map the area over which we sum in the (n, m) space into the corresponding area in the (n'', m'') space:

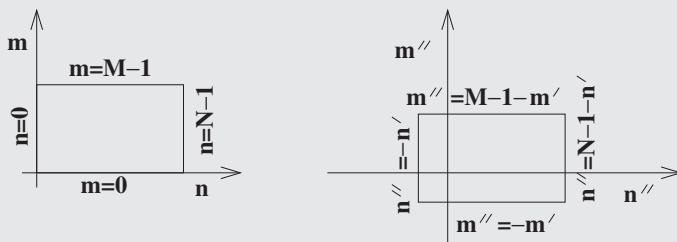


Figure 2.14: The area over which we sum is like a floating rectangle which is shifted about in the coordinate space we use, according to the change of variables we perform.

The area over which we sum in the (n, m) space is enclosed by four lines with equations given on the left-hand side of the list below. Each of these lines is transformed into a line in the (n'', m'') space, using equations (2.198). These transformed equations are given on the right-hand side of the list below. The transformed lines define the new

limits of summation.

$$\begin{aligned} m = 0 &\rightarrow m'' = -m' \\ m = M - 1 &\rightarrow m'' = M - 1 - m' \\ n = 0 &\rightarrow n'' = -n' \\ n = N - 1 &\rightarrow n'' = N - 1 - n' \end{aligned}$$

Then the last expression for $\hat{v}(p, q)$ becomes:

$$\begin{aligned} \hat{v}(p, q) &= \frac{1}{MN} \sum_{n'=0}^{N-1} \sum_{m'=0}^{M-1} w(n', m') e^{-j2\pi \left[\frac{pn'}{N} + \frac{qm'}{M} \right]} \\ &\quad \sum_{m''=-m'}^{M-1-m'} \sum_{n''=-n'}^{N-1-n'} g(n'', m'') e^{-j2\pi \left[\frac{pn''}{N} + \frac{qm''}{M} \right]} \end{aligned} \quad (2.199)$$

Let us concentrate on the last two sums of (2.199). Let us call them factor T . We may separate the negative from the positive indices of n'' and write:

$$\begin{aligned} T &\equiv \sum_{m''=-m'}^{M-1-m'} \left[\sum_{n''=-n'}^{-1} + \sum_{n''=0}^{N-1-n'} \right] g(n'', m'') e^{-j2\pi \left[\frac{pn''}{N} + \frac{qm''}{M} \right]} \\ &= \sum_{m''=-m'}^{M-1-m'} e^{-j2\pi \frac{pn''}{M}} \sum_{n''=-n'}^{-1} g(n'', m'') e^{-j2\pi \frac{qn''}{N}} + \\ &\quad \sum_{m''=-m'}^{M-1-m'} e^{-j2\pi \frac{pn''}{M}} \sum_{n''=0}^{N-1-n'} g(n'', m'') e^{-j2\pi \frac{qn''}{N}} \end{aligned} \quad (2.200)$$

Clearly the two images g and w are not defined for negative indices. We may choose to extend their definition for indices outside the range $[0, N - 1]$, $[0, M - 1]$ in any which way suits us. Let us examine the factor:

$$\sum_{n''=-n'}^{-1} g(n'', m'') e^{-j2\pi q \frac{n''}{N}} \quad (2.201)$$

We define a new variable $n''' \equiv N + n'' \Rightarrow n'' = n''' - N$. Then the above expression becomes:

$$\sum_{n'''=N-n'}^{N-1} g(n'' - N, m''') e^{-j2\pi q \frac{n'''}{N}} e^{-j2\pi q} \quad (2.202)$$

As q is an integer, $e^{-j2\pi q} = 1$. Now if we choose to define: $g(n'' - N, m''') \equiv g(n'', m''')$, the above sum is:

$$\sum_{n'''=N-n'}^{N-1} g(n'', m''') e^{-j2\pi q \frac{n'''}{N}} \quad (2.203)$$

Since n''' is a dummy index, we may call it anything we like. Let us call it n'' . Then the above expression becomes:

$$\sum_{n''=N-n'}^{N-1} g(n'', m'') e^{-j2\pi q \frac{n''}{N}} \quad (2.204)$$

This term is added to the term

$$\sum_{n''=0}^{N-n'-1} g(n'', m'') e^{-j2\pi q \frac{n''}{N}} \quad (2.205)$$

in (2.200) and the two together may be written as:

$$\sum_{n''=0}^{N-1} g(n'', m'') e^{-j2\pi q \frac{n''}{N}} \quad (2.206)$$

We can work in a similar way for the summation over index m'' and assume that g is periodic also in its first index with period M . Then, under the assumption we made about the definition of g outside its real area of definition, the double sum we called T is:

$$T = \sum_{m''=0}^{M-1} \sum_{n''=0}^{N-1} g(n'', m'') e^{-j2\pi \left[\frac{pn''}{N} + \frac{qm''}{M} \right]} \quad (2.207)$$

This does not contain indices m' , n' and therefore it is a factor that multiplies the double sum over n' and m' in (2.199). Further, it is recognised to be $MN\hat{g}(p, q)$. Similarly, in (2.199) we recognise the discrete Fourier transform of w and thus (2.199) becomes

$$\hat{v}(p, q) = MN\hat{g}(p, q)\hat{w}(p, q) \quad (2.208)$$

under the *assumptions* that:

$$\begin{aligned} g(n, m) &\equiv g(n - N, m - M) \\ w(n, m) &\equiv w(n - N, m - M) \\ g(n, m) &\equiv g(n, m - M) \\ w(n, m) &\equiv w(n, m - M) \\ g(n, m) &\equiv g(n - N, m) \\ w(n, m) &\equiv w(n - N, m) \end{aligned} \quad (2.209)$$

In other words, we assume that the image arrays g and w are defined in the whole (n, m) space periodically, with periods M and N in the two directions, respectively. This corresponds to the **time convolution theorem**.

The **frequency convolution theorem** would have exactly the same form. Because of the symmetry between the discrete Fourier transform and its inverse, this implies that the discrete Fourier transforms of these functions are also periodic in the whole (q, p) space, with periods N and M , respectively.

Example B2.35

You are given two $N \times M$ images $g(n, m)$ and $w(n, m)$. Their DFTs are $\hat{g}(p, q)$ and $\hat{w}(p, q)$, respectively. We create image $x(n, m)$ by multiplying the two images point by point:

$$x(n, m) = g(n, m) \times w(n, m) \quad (2.210)$$

Express the DFT, $\hat{x}(p, q)$, of $x(n, m)$, in terms of $\hat{g}(p, q)$ and $\hat{w}(p, q)$.

Let us take the DFT of both sides of equation (2.210):

$$\hat{x}(k, l) = \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g(n, m) w(n, m) e^{-j2\pi[\frac{kn}{N} + \frac{lm}{M}]} \quad (2.211)$$

We may express images g and w in terms of their DFTs \hat{g} and \hat{w} as follows:

$$\begin{aligned} g(n, m) &= \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} \hat{g}(p, q) e^{j2\pi[\frac{pn}{N} + \frac{qm}{M}]} \\ w(n, m) &= \sum_{s=0}^{N-1} \sum_{r=0}^{M-1} \hat{w}(s, r) e^{j2\pi[\frac{ns}{N} + \frac{rm}{M}]} \end{aligned} \quad (2.212)$$

Substituting these expressions into (2.211) we obtain:

$$\begin{aligned} \hat{x}(k, l) &= \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} \hat{g}(p, q) e^{j2\pi[\frac{pn}{N} + \frac{qm}{M}]} \\ &\quad \sum_{s=0}^{N-1} \sum_{r=0}^{M-1} \hat{w}(s, r) e^{j2\pi[\frac{ns}{N} + \frac{rm}{M}]} e^{-j2\pi[\frac{km}{M} + \frac{ln}{N}]} = \\ \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} \sum_{s=0}^{N-1} \sum_{r=0}^{M-1} \hat{g}(p, q) \hat{w}(s, r) e^{j2\pi[\frac{n(s+p)}{N} + \frac{m(r+q)}{M}]} e^{-j2\pi[\frac{kn}{N} + \frac{lm}{M}]} \end{aligned} \quad (2.213)$$

We notice that indices n and m do not appear in $\hat{g}(p, q)$ and $\hat{w}(s, r)$, so we may collect the terms that depend of n and m separately and sum over them:

$$\hat{x}(k, l) = \frac{1}{NM} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} \sum_{s=0}^{N-1} \sum_{r=0}^{M-1} \hat{g}(p, q) \hat{w}(s, r) \sum_{n=0}^{N-1} e^{j2\pi \frac{s+p-k}{N}} \sum_{m=0}^{M-1} e^{j2\pi \frac{r+q-l}{M}} \quad (2.214)$$

To compute the sums of the exponential functions we apply formula (2.164), on page 95, once for $S \equiv N$ and $t \equiv s + p - k$ and once for $S \equiv M$ and $t \equiv r + q - l$:

$$\hat{x}(k, l) = \frac{1}{NM} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} \sum_{s=0}^{N-1} \sum_{r=0}^{M-1} \hat{g}(p, q) \hat{w}(s, r) N \delta(s + p - k) M \delta(r + q - l) \quad (2.215)$$

The delta functions will pick from all values of s and r only the ones that may zero their arguments, ie they will only retain the terms for which $s = k - p$ and $r = l - q$. Therefore:

$$\hat{x}(k, l) = \underbrace{\sum_{p=0}^{N-1} \sum_{q=0}^{M-1} \hat{g}(p, q) \hat{w}(k-p, l-q)}_{\text{Convolution of } \hat{g} \text{ with } \hat{w}} \quad (2.216)$$

Example 2.36

Show that if $g(k, l)$ is an $M \times N$ image defined as a periodic function with periods M and N in the whole (k, l) space, its DFT $\hat{g}(m, n)$ is also periodic in the (m, n) space, with the same periods.

We must show that $\hat{g}(m + M, n + N) = \hat{g}(m, n)$. We start from the definition of $\hat{g}(m, n)$:

$$\hat{g}(m, n) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g(k, l) e^{-j2\pi[\frac{km}{M} + \frac{ln}{N}]} \quad (2.217)$$

Then

$$\begin{aligned} \hat{g}(m + M, n + N) &= \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g(k, l) e^{-j2\pi[\frac{k(m+M)}{M} + \frac{l(n+N)}{N}]} \\ &= \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g(k, l) e^{-j2\pi \frac{km}{M}} e^{-j2\pi \frac{ln}{N}} = \hat{g}(m, n) \end{aligned} \quad (2.218)$$

where we made use of $e^{j2\pi t} = \cos(2\pi t) + j \sin(2\pi t) = 1$, for t an integer.

Example B2.37

Show that if $v(n, m)$ is defined as

$$v(n, m) \equiv \sum_{n'=0}^{N-1} \sum_{m'=0}^{M-1} g(n - n', m - m') w(n', m') \quad (2.219)$$

where $g(n, m)$ and $w(n, m)$ are two periodically defined images with

periods N and M in the two variables respectively, $v(n, m)$ is also given by:

$$v(n, m) = \sum_{n'=0}^{N-1} \sum_{m'=0}^{M-1} w(n - n', m - m') g(n', m') \quad (2.220)$$

Define some new variables of summation k and l , so that:

$$\begin{aligned} k &\equiv n - n' \Rightarrow n' = n - k \\ l &\equiv m - m' \Rightarrow m' = m - l \end{aligned} \quad (2.221)$$

As n' takes values from 0 to $N-1$, k will take values from n to $n-N+1$. Similarly, as m' takes values from 0 to $M-1$, l will take values from m to $m-M+1$. Substituting in equation (2.219) we have:

$$v(n, m) = \sum_{k=n}^{n-N+1} \sum_{l=m}^{m-M+1} g(k, l) w(n - k, m - l) \quad (2.222)$$

Consider the sum:

$$\sum_{k=n}^{n-N+1} g(k, l) w(n - k, m - l) \quad (2.223)$$

First reverse the order of summation, with no consequence, and write it as:

$$\sum_{k=-N+n+1}^n g(k, l) w(n - k, m - l) \quad (2.224)$$

Next split the range of indices $[-N + n + 1, n]$ into the two ranges $[-N + n + 1, -1]$ and $[0, n]$:

$$\sum_{k=-N+n+1}^{-1} g(k, l) w(n - k, m - l) + \sum_{k=0}^n g(k, l) w(n - k, m - l) \quad (2.225)$$

Then note that the range of indices $[-N + n + 1, -1]$ is effectively indices $[-N, -1]$ minus indices $[-N, -N + n]$:

$$\underbrace{\sum_{k=-N}^{-1} g(k, l) w(n - k, m - l)}_{\text{Change variable } \tilde{k} \equiv k+N} - \underbrace{\sum_{k=-N}^{-N+n} g(k, l) w(n - k, m - l)}_{\text{Change variable } \tilde{k} \equiv k+N} + \sum_{k=0}^n g(k, l) w(n - k, m - l) \quad (2.226)$$

After the change of variables:

$$\sum_{\tilde{k}=0}^{N-1} g(\tilde{k} - N, l) w(n - \tilde{k} + N, m - l) - \sum_{\tilde{k}=0}^n g(\tilde{k} - N, l) w(n - \tilde{k} + N, m - l) \\ + \sum_{k=0}^n g(k, l) w(n - k, m - l) \quad (2.227)$$

$$\begin{aligned} g \text{ periodic} &\Rightarrow g(k - N, l) = g(k, l) \\ w \text{ periodic} &\Rightarrow w(s + N, t) = w(s, t) \end{aligned} \quad (2.228)$$

Therefore, the last two sums in (2.227) are identical and cancel each other, and the summation over k in (2.222) is from 0 to $N - 1$. Similarly, we can show that the summation over l in (2.222) is from 0 to $M - 1$, and thus prove equation (2.220).

How can we display the discrete Fourier transform of an image?

Assume that the discrete Fourier transform of an image is $\hat{g}(p, q)$. Scalars $\hat{g}(p, q)$ are the coefficients of the expansion of the image into discrete Fourier functions, each one of which corresponds to a different pair of spatial frequencies in the 2D (p, q) plane. As p and q increase, the contributions of these high frequencies to the image become less and less significant (in terms of the effect they have on the mean square error of the image when it is reconstructed without them) and thus the values of the corresponding coefficients $\hat{g}(p, q)$ become smaller. We may find difficult to display these coefficients, because their values span a great range. So, for displaying purposes only, we use the following logarithmic function:

$$d(p, q) \equiv \log_{10}(1 + |\hat{g}(p, q)|) \quad (2.229)$$

This function is then scaled into a displayable range of grey values and displayed instead of $\hat{g}(p, q)$. Notice that when $\hat{g}(p, q) = 0$, $d(p, q) = 0$ too. This function has the property of reducing the ratio between the high values of \hat{g} and the small ones, so that small and large values can be displayed in the same scale. For example, if $\hat{g}_{max} = 100$ and $\hat{g}_{min} = 0.1$, it is rather difficult to draw these numbers on the same graph, as their ratio is 1000. However, $\log_{10}(101) = 2.0043$ and $\log_{10}(1.1) = 0.0414$ and their ratio is only 48. So, both numbers can be drawn on the same scale more easily.

In order to display the values of $d(p, q)$ as a grey image, the scaling is done as follows. The minimum and the maximum values of $d(p, q)$ are identified and are denoted by d_{min} and d_{max} , respectively. Then each frequency sample (p, q) is assigned a new value $d_{new}(p, q)$ defined as:

$$d_{new}(p, q) \equiv \left\lfloor \frac{d(p, q) - d_{min}}{d_{max} - d_{min}} \times 255 + 0.5 \right\rfloor \quad (2.230)$$

Note that when $d(p, q) = d_{min}$, the fraction is 0, and taking the integer part of 0.5 yields 0. When $d(p, q) = d_{max}$, the fraction becomes 1 and multiplied with 255 yields 255. The term

0.5 is used to ensure that the real numbers that result from the division and multiplication with 255 are rounded to the nearest integer, rather than truncated to their integer part. For example, if the resultant number is 246.8, the integer part is 246 but if we add 0.5 first and then take the integer part, we get 247 which is an integer that represents 246.8 much better than 246.

What happens to the discrete Fourier transform of an image if the image is rotated?

We rewrite here the definition of the discrete Fourier transform, ie equation (2.162), on page 94, for a square image ($M = N$):

$$\hat{g}(m, n) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g(k, l) e^{-j2\pi \frac{km+ln}{N}} \quad (2.231)$$

We may introduce polar coordinates on the planes (k, l) and (m, n) , as follows: $k \equiv r \cos \theta$, $l \equiv r \sin \theta$, $m \equiv \omega \cos \phi$, $n \equiv \omega \sin \phi$. We note that $km + ln = r\omega(\cos \theta \cos \phi + \sin \theta \sin \phi) = r\omega \cos(\theta - \phi)$. Then equation (2.231) becomes:

$$\hat{g}(\omega, \phi) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g(r, \theta) e^{-j2\pi \frac{r\omega \cos(\theta - \phi)}{N}} \quad (2.232)$$

Variables k and l , over which we sum, do not appear in the summand explicitly. However, they are there implicitly and the summation is supposed to happen over all relevant points. From the values of k and l , we are supposed to find the corresponding values of r and θ .

Assume now that we rotate $g(r, \theta)$ by an angle θ_0 . It becomes $g(r, \theta + \theta_0)$. We want to find the discrete Fourier transform of this rotated function. Formula (2.232) is another “slot machine”. We slot in the appropriate place the function, the transform of which we require, and out comes its DFT. Therefore, we shall use formula (2.232) to calculate the DFT of $g(r, \theta + \theta_0)$ by simply replacing $g(r, \theta)$ with $g(r, \theta + \theta_0)$. We denote the DFT of $g(r, \theta + \theta_0)$ as $\hat{g}(\omega, \phi)$. We get:

$$\hat{g}(\omega, \phi) = \frac{1}{N^2} \underbrace{\sum_{\text{all points}}}_{\text{all points}} g(r, \theta + \theta_0) e^{-j2\pi \frac{r\omega \cos(\theta - \phi)}{N}} \quad (2.233)$$

To find the relationship between $\hat{g}(\omega, \phi)$ and $\hat{g}(\omega, \phi)$ we have somehow to make $g(r, \theta)$ appear on the right-hand side of this expression. For this purpose, we introduce a new variable, $\tilde{\theta} \equiv \theta + \theta_0$ and replace θ by $\tilde{\theta} - \theta_0$ in (2.233):

$$\hat{g}(\omega, \phi) = \frac{1}{N^2} \underbrace{\sum_{\text{all points}}}_{\text{all points}} g(r, \tilde{\theta}) e^{-j2\pi \frac{r\omega \cos(\tilde{\theta} - \theta_0 - \phi)}{N}} \quad (2.234)$$

Then on the right-hand side we recognise the DFT of the unrotated image calculated at $\phi + \theta_0$ instead of ϕ : $\hat{g}(\omega, \phi + \theta_0)$. That is, we have:

$$\hat{g}(\omega, \phi) = \hat{g}(\omega, \phi + \theta_0) \quad (2.235)$$

We conclude that:

The DFT of the image rotated by θ_0 = the DFT of the unrotated image rotated by the same angle θ_0 .

Example 2.38

Rotate the image of example 2.29, on page 97, clockwise by 90° about its top left corner and recalculate its discrete Fourier transform. Thus, verify the relationship between the discrete Fourier transform of a 2D image and the discrete Fourier transform of the same image rotated by angle θ_0 .

The rotated by 90° image is:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.236)$$

To calculate its DFT we multiply it first from the right with matrix U of example 2.28,

$$\frac{1}{4} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.237)$$

and then multiply the result from the left with the same matrix U :

$$\frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ -1/4 & 0 & 0 & 0 \\ 1/4 & 0 & 0 & 0 \\ -1/4 & 0 & 0 & 0 \end{pmatrix} \quad (2.238)$$

By comparing the above result with the result of example 2.29 we see that the discrete Fourier transform of the rotated image is the discrete Fourier transform of the unrotated image rotated clockwise by 90° .

What happens to the discrete Fourier transform of an image if the image is shifted?

Assume that we shift the image to the point (k_0, l_0) , so that it becomes $g(k - k_0, l - l_0)$. To calculate the DFT of the shifted image, we slot this function into formula (2.231). We denote the DFT of $g(k - k_0, l - l_0)$ as $\hat{g}(m, n)$ and obtain:

$$\hat{g}(m, n) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g(k - k_0, l - l_0) e^{-j2\pi \frac{km+ln}{N}} \quad (2.239)$$

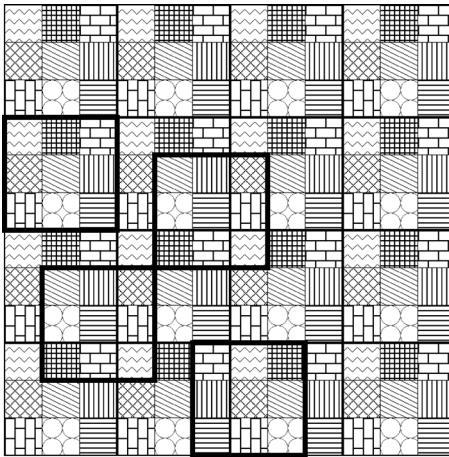


Figure 2.15: A 3×3 image repeated ad infinitum in both directions. Any 3×3 floating window (depicted with the thick black line) will pick up exactly the same pixels wherever it is placed. If each pattern represents a different number, the average inside each black frame will always be the same. When we take a weighted average, as long as the weights are also periodic with the same period as the image and are shifted in the same way as the elements inside each window, the result will also be always the same. Kernel $e^{-j2\pi m/3}$ used for DFT has such properties and that is why the range of indices over which we sum does not matter, as long as they are consecutive and equal in number to the size of the image.

To find a relationship between $\hat{g}(m, n)$ and $\hat{g}(m, n)$, we must somehow make $g(k, l)$ appear on the right-hand side of this expression. For this purpose, we define new variables $k' \equiv k - k_0$ and $l' \equiv l - l_0$. Then:

$$\hat{g}(m, n) = \frac{1}{N^2} \sum_{k'=-k_0}^{N-1-k_0} \sum_{l'=-l_0}^{N-1-l_0} g(k', l') e^{-j2\pi \frac{k'm+l'n}{N}} e^{-j2\pi \frac{k_0 m + l_0 n}{N}} \quad (2.240)$$

Because of the assumed periodic repetition of the image in both directions and the easily proven periodicity of the exponential kernel, also in both directions with the same period N , where exactly we perform the summation (ie between which indices) does not really matter, as long as a window of the right size is used for the summation (see figure 2.15). In other words, as long as summation indices k' and l' are allowed to take N consecutive values each, it does not matter where they start from. So, we may assume in the expression above that k' and l' take values from 0 to $N - 1$. We also notice that factor $e^{-j2\pi \frac{k_0 m + l_0 n}{N}}$ is independent of k' and l' and, therefore, it can come out of the summation. Then, we recognise in (2.240) the DFT of $g(k, l)$ appearing on the right-hand side. (Note that k', l' are dummy indices and it makes no difference whether we call them k', l' or k, l .) We have, therefore:

$$\hat{g}(m, n) = \hat{g}(m, n) e^{-j2\pi \frac{k_0 m + l_0 n}{N}} \quad (2.241)$$

The DFT of the shifted image = the DFT of the unshifted image $\times e^{-j2\pi \frac{k_0 m + l_0 n}{N}}$

Similarly, one can show that

$$\text{The shifted DFT of an image} = \text{the DFT of } \left[\text{image} \times e^{j2\pi \frac{m_0 k + n_0 l}{N}} \right]$$

or

$$\hat{g}(m - m_0, n - n_0) = \text{DFT of } \left[\text{image} \times e^{j2\pi \frac{m_0 k + n_0 l}{N}} \right]$$

Example 2.39

Image (2.191), on page 102, is shifted so that its top left coordinate, instead of being at position $(0, 0)$, is at position $(-3/2, -3/2)$. Use the result of example 2.33 to work out the DFT of the shifted image.

The shifting parameters are $k_0 = l_0 = -3/2$ and $N = 4$. Then each term of the DFT of (2.191), given by equation (2.193), will have to be multiplied with factor

$$F \equiv e^{j2\pi \frac{3m+3n}{8}} \quad (2.242)$$

We compute factor (2.242) for all allowed combinations of values of m and n :

$$m = 0 \quad n = 0 \quad F = 1$$

$$m = 0 \quad n = 1 \quad F = e^{j2\pi \frac{3}{8}} = \cos \frac{3\pi}{4} + j \sin \frac{3\pi}{4} = -\frac{\sqrt{2}}{2} + j \frac{\sqrt{2}}{2} = \frac{\sqrt{2}}{2}(-1 + j)$$

$$m = 0 \quad n = 2 \quad F = e^{j2\pi \frac{6}{8}} = \cos \frac{3\pi}{4} + j \sin \frac{3\pi}{4} = -j$$

$$m = 0 \quad n = 3 \quad F = e^{j2\pi \frac{9}{8}} = e^{j\frac{9\pi}{4}} = \cos \frac{\pi}{4} + j \sin \frac{\pi}{4} = \frac{\sqrt{2}}{2} + j \frac{\sqrt{2}}{2} = \frac{\sqrt{2}}{2}(1 + j)$$

$$m = 1 \quad n = 0 \quad F = e^{j2\pi \frac{3}{8}} = \frac{\sqrt{2}}{2}(-1 + j)$$

$$m = 1 \quad n = 1 \quad F = e^{j2\pi \frac{6}{8}} = -j$$

$$m = 1 \quad n = 2 \quad F = e^{j2\pi \frac{9}{8}} = \frac{\sqrt{2}}{2}(1 + j)$$

$$m = 1 \quad n = 3 \quad F = e^{j2\pi \frac{12}{8}} = e^{j\pi} = \cos \pi + j \sin \pi = -1$$

$$m = 2 \quad n = 0 \quad F = e^{j2\pi \frac{6}{8}} = -j$$

$$m = 2 \quad n = 1 \quad F = e^{j2\pi \frac{9}{8}} = \frac{\sqrt{2}}{2}(1 + j)$$

$$m = 2 \quad n = 2 \quad F = e^{j2\pi \frac{12}{8}} = -1$$

$$m = 2 \quad n = 3 \quad F = e^{j2\pi \frac{15}{8}} = e^{j\frac{7\pi}{4}} = \cos \frac{7\pi}{4} + j \sin \frac{7\pi}{4} = \frac{\sqrt{2}}{2}(1 - j)$$

$$m = 3 \quad n = 0 \quad F = e^{j2\pi \frac{9}{8}} = \frac{\sqrt{2}}{2}(1 + j)$$

$$\begin{aligned}
 m = 3 & \quad n = 1 \quad F = e^{j2\pi \frac{12}{8}} = -1 \\
 m = 3 & \quad n = 2 \quad F = e^{j2\pi \frac{15}{8}} = \frac{\sqrt{2}}{2}(1-j) \\
 m = 3 & \quad n = 3 \quad F = e^{j2\pi \frac{18}{8}} = e^{j\frac{\pi}{2}} = \cos \frac{\pi}{2} + j \sin \frac{\pi}{2} = j
 \end{aligned} \tag{2.243}$$

So, the DFT of the shifted function is given by (2.193) if we multiply each element of that matrix with the corresponding correction factor:

$$\begin{aligned}
 \hat{g}_{shifted} &= \\
 & \left(\begin{array}{cccc}
 \frac{1}{4} \times 1 & -\frac{1+j}{8} \times \frac{\sqrt{2}}{2}(-1+j) & 0 \times (-j) & \frac{j-1}{8} \times \frac{\sqrt{2}}{2}(1+j) \\
 -\frac{1+j}{8} \times \frac{\sqrt{2}}{2}(-1+j) & \frac{j}{8} \times (-j) & 0 \times \frac{\sqrt{2}}{2}(1+j) & \frac{1}{8} \times (-1) \\
 0 \times (-j) & 0 \times \frac{\sqrt{2}}{2}(1+j) & 0 \times (-1) & 0 \times \frac{\sqrt{2}}{2}(1-j) \\
 \frac{j-1}{8} \times \frac{\sqrt{2}}{2}(1+j) & \frac{1}{8} \times (-1) & 0 \times \frac{\sqrt{2}}{2}(1-j) & -\frac{j}{8} \times j
 \end{array} \right) \\
 &= \left(\begin{array}{cccc}
 \frac{1}{4} & \frac{\sqrt{2}}{8} & 0 & -\frac{\sqrt{2}}{8} \\
 \frac{\sqrt{2}}{8} & \frac{1}{8} & 0 & -\frac{1}{8} \\
 0 & 0 & 0 & 0 \\
 -\frac{\sqrt{2}}{8} & -\frac{1}{8} & 0 & \frac{1}{8}
 \end{array} \right) \tag{2.244}
 \end{aligned}$$

Example 2.40

Compute the DFT of image (2.191), on page 102, using formula (2.162) and assuming that the centre of the axes is in the centre of the image.

If the centre of the axes is in the centre of the image, the only nonzero elements of this image are at half-integer positions. They are:

$$g\left(-\frac{1}{2}, -\frac{1}{2}\right) = g\left(-\frac{1}{2}, \frac{1}{2}\right) = g\left(\frac{1}{2}, \frac{1}{2}\right) = g\left(\frac{1}{2}, -\frac{1}{2}\right) = 1 \tag{2.245}$$

Applying formula (2.162) then for k and l values from the set $\{-1/2, 1/2\}$, we obtain:

$$\hat{g}(m, n) = \frac{1}{16} \left[e^{-j\frac{2\pi}{4}\frac{-m-n}{2}} + e^{-j\frac{2\pi}{4}\frac{-m+n}{2}} + e^{-j\frac{2\pi}{4}\frac{m-n}{2}} + e^{-j\frac{2\pi}{4}\frac{m+n}{2}} \right] \tag{2.246}$$

We apply this formula now to work out the elements of the DFT of the image. For $m = n = 1$ we obtain:

$$\hat{g}(1, 1) = \frac{1}{16} [e^{j\frac{\pi}{2}} + 1 + 1 + e^{-j\frac{\pi}{2}}] = \frac{1}{16}[j + 2 - j] = \frac{1}{8} \tag{2.247}$$

For $m = 0$ and $n = 1$ we obtain:

$$\begin{aligned}
 \hat{g}(0,1) &= \frac{1}{16} [e^{j\frac{\pi}{4}} + e^{-j\frac{\pi}{4}} + e^{j\frac{\pi}{4}} + e^{-j\frac{\pi}{4}}] \\
 &= \frac{1}{16} \left[\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} \right] \\
 &= \frac{\sqrt{2}}{8}
 \end{aligned} \tag{2.248}$$

We work similarly for the other terms. Eventually we obtain the same DFT we obtained in example 2.39, given by equation (2.244), where we applied the shifting property of the Fourier transform.

What is the relationship between the average value of the image and its DFT?

The average value of the image is given by:

$$\bar{g} = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g(k, l) \tag{2.249}$$

If we set $m = n = 0$ in (2.231), we get:

$$\hat{g}(0,0) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g(k, l) \tag{2.250}$$

Therefore, the mean of an image and the **direct component** (or **dc**) of its DFT (ie the component at frequency $(0, 0)$) are equal:

$$\boxed{\bar{g} = \hat{g}(0,0)} \tag{2.251}$$

Example 2.41

Confirm the relationship between the average of image

$$g = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.252}$$

and its discrete Fourier transform.

Apply the discrete Fourier transform formula (2.162) for $N = M = 4$ and for

$m = n = 0$:

$$\begin{aligned}\hat{g}(0, 0) &= \frac{1}{16} \sum_{k=0}^3 \sum_{l=0}^3 g(k, l) = \frac{1}{16} (0 + 0 + 0 + 0 + 0 + 0 + 1 + 1 + 0 + 0 + 1 + 1 + 0 \\ &\quad + 0 + 0 + 0 + 0) = \frac{1}{4}\end{aligned}\tag{2.253}$$

The mean of g is:

$$\begin{aligned}\bar{g} &\equiv \frac{1}{16} \sum_{k=0}^3 \sum_{l=0}^3 g(k, l) = \frac{1}{16} (0 + 0 + 0 + 0 + 0 + 0 + 1 + 1 + 0 + 0 + 1 + 1 + 0 \\ &\quad + 0 + 0 + 0 + 0) = \frac{4}{16} = \frac{1}{4}\end{aligned}\tag{2.254}$$

Thus (2.251) is confirmed.

What happens to the DFT of an image if the image is scaled?

When we take the average of a discretised function over an area over which this function is defined, we implicitly perform the following operation: we divide the area into small elementary areas of size $\Delta x \times \Delta y$ say, take the value of the function at the centre of each of these little tiles and assume that it represents the value of the function over the whole tile. Thus, we sum and divide by the total number of tiles. So, really the average of a function is defined as:

$$\bar{g} = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x, y) \Delta x \Delta y\tag{2.255}$$

We simply omit Δx and Δy because x and y are incremented by 1 at a time, so $\Delta x = \Delta y = 1$. We also notice, from the definition of the discrete Fourier transform, that really, the discrete Fourier transform is a weighted average, where the value of $g(k, l)$ is multiplied with a different weight inside each little tile. Seeing the DFT that way, we realise that the correct definition of the discrete Fourier transform should include a factor $\Delta k \times \Delta l$ too, as the area of the little tile over which we assume the value of the function g to be constant. We omit it because $\Delta k = \Delta l = 1$. So, the formula for DFT that explicitly states this is:

$$\hat{g}(m, n) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g(k, l) e^{-j2\pi \frac{km+ln}{N}} \Delta k \Delta l\tag{2.256}$$

Now assume that we change the scales in the (k, l) plane and $g(k, l)$ becomes $g(\alpha k, \beta l)$. We denote the discrete Fourier transform of the scaled g as $\hat{g}(m, n)$. In order to calculate it, we must slot function $g(\alpha k, \beta l)$ in place of $g(k, l)$ in formula (2.256). We obtain:

$$\hat{g}(m, n) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} g(\alpha k, \beta l) e^{-j2\pi \frac{km+ln}{N}} \Delta k \Delta l\tag{2.257}$$

We wish to find a relationship between $\hat{g}(m, n)$ and $\hat{g}(m, n)$. Therefore, somehow we must make $g(k, l)$ appear on the right-hand side of equation (2.257). For this purpose, we define new variables of summation $k' \equiv \alpha k$ and $l' \equiv \beta l$. Then:

$$\hat{g}(m, n) = \frac{1}{N^2} \sum_{k'=0}^{\alpha(N-1)} \sum_{l'=0}^{\beta(N-1)} g(k', l') e^{-j2\pi \frac{k' \frac{m}{\alpha} + l' \frac{n}{\beta}}{N}} \frac{\Delta k'}{\alpha} \frac{\Delta l'}{\beta} \quad (2.258)$$

The summation that appears in this expression spans all points over which function $g(k', l')$ is defined, except that the summation variables k' and l' are not incremented by 1 in each step. We recognise again the DFT of $g(k, l)$ on the right-hand side of (2.258), calculated, not at point (m, n) , but at point $(\frac{m}{\alpha}, \frac{n}{\beta})$. Therefore, we may write:

$$\hat{g}(m, n) = \frac{1}{\alpha\beta} \hat{g}\left(\frac{m}{\alpha}, \frac{n}{\beta}\right) \quad (2.259)$$

The DFT of the scaled function = $\frac{1}{|\text{product of scaling factors}|} \times$ the DFT of the unscaled function calculated at the same point inversely scaled.

Example 2.42

You are given a continuous function $f(x, y)$ where $-\infty < x < +\infty$ and $-\infty < y < +\infty$, defined as

$$f(x, y) = \begin{cases} 1 & \text{for } -0.5 < x < 4.5 \text{ and } 0.5 < y < 1.5 \\ 0 & \text{elsewhere} \end{cases} \quad (2.260)$$

Sample this function at integer positions (i, j) , where $0 \leq i, j \leq 4$, to create a 4×4 digital image.

Figure 2.16 on the left shows a plot of this function. On the right it shows the sampling points (i, j) marked as black dots. The region highlighted with grey is the area where the function has value 1. At all other points the function has value 0. So, the image created by sampling this function at the marked points is:

$$g = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.261)$$

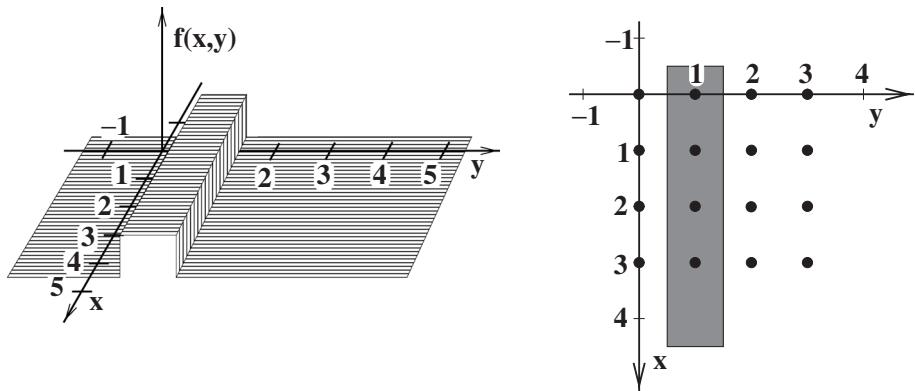


Figure 2.16: On the left, the plot of a continuous function and on the right the region where the function takes nonzero values highlighted in grey. The black dots are the sampling points we use to create a digital image out of this function.

Example 2.43

Scale function $f(x, y)$ of example 2.42 to produce function $\tilde{f}(\alpha x, \beta y)$ where $\alpha = \beta = 2$. Plot the scaled function and sample it at points (i, j) where i and j take all possible values from the set $\{0, 0.5, 1, 1.5\}$.

We note that function $\tilde{f}(\alpha x, \beta y)$ will be nonzero when $-0.5 < \alpha x < 4.5$ and $0.5 < \beta y < 1.5$. That is $\tilde{f}(\alpha x, \beta y)$ will be nonzero when $-0.5/\alpha < x < 4.5/\alpha$ and $0.5/\beta < y < 1.5/\beta$. So, for $-0.25 < x < 2.25$ and $0.25 < y < 0.75$, $\tilde{f}(\alpha x, \beta y)$ will be 1 and it will be 0 for all other values of its argument. This function is plotted in the left panel of figure 2.17. On the right we can see how this plot looks from above, marking with grey the region where the function takes value 1, and with black dots the points where it will be sampled to create a digital version of it. The digital image we create this way is:

$$\tilde{g} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.262)$$

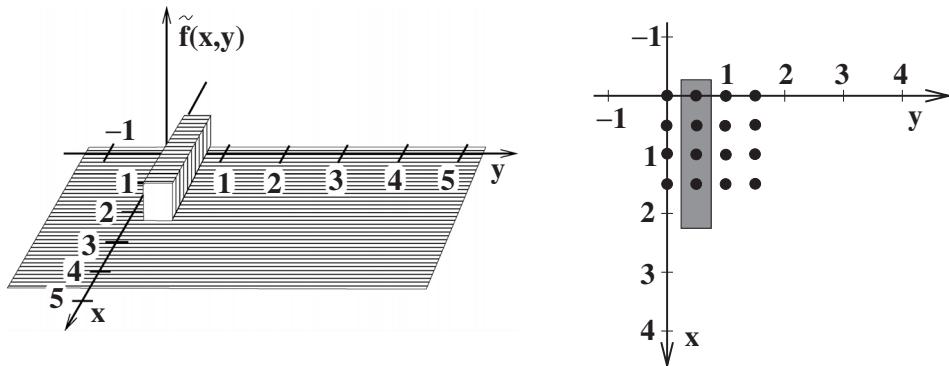


Figure 2.17: On the left, the plot of the scaled function and on the right the region where the function takes nonzero values highlighted in grey. The black dots are the sampling points we use to create a digital image out of this function.

Example 2.44

Use formula (2.256) to compute the DFT of the digital image you created in example 2.43.

Here:

$$N = 4, \quad \Delta k = \Delta l = \frac{1}{2} \quad (2.263)$$

So, formula (2.256) takes the form:

$$\begin{aligned} \hat{g}(m, n) &= \frac{1}{16} \sum_{k \in \{0, 0.5, 1, 1.5\}} \sum_{l \in \{0, 0.5, 1, 1.5\}} g(k, l) e^{-j2\pi \frac{km+ln}{4}} \frac{1}{4} \\ &= \frac{1}{64} \left[g(0, 0.5) e^{-j2\pi \frac{n}{8}} + g(0.5, 0.5) e^{-j2\pi \frac{m+n}{4}} + g(1, 0.5) e^{-j2\pi \frac{2m+n}{8}} \right. \\ &\quad \left. + g(1.5, 0.5) e^{-j2\pi \frac{3m+n}{8}} \right] \\ &= \frac{1}{64} \left[e^{-j\pi \frac{n}{4}} + e^{-j\pi \frac{m+n}{4}} + e^{-j\pi \frac{2m+n}{4}} + e^{-j\pi \frac{3m+n}{4}} \right] \end{aligned} \quad (2.264)$$

This is the DFT of the image.

Example 2.45

Compute the DFT of image (2.261). Then use the result of example 2.44 to verify formula (2.259).

We use formula (2.256) with $N = 4$ and $\Delta k = \Delta l = 1$ to obtain the DFT of (2.261):

$$\begin{aligned}\hat{g}(m, n) &= \frac{1}{16} \sum_{k=0}^3 \sum_{l=0}^3 g(k, l) e^{-j2\pi \frac{km+ln}{4}} \\ &= \frac{1}{16} \left[e^{-j2\pi \frac{n}{4}} + e^{-j2\pi \frac{m+n}{4}} + e^{-j2\pi \frac{2m+n}{4}} + e^{-j2\pi \frac{3m+n}{4}} \right] \\ &= \frac{1}{16} \left[e^{-j\pi \frac{n}{2}} + e^{-j\pi \frac{m+n}{2}} + e^{-j\pi \frac{2m+n}{2}} + e^{-j\pi \frac{3m+n}{2}} \right]\end{aligned}\quad (2.265)$$

For $\alpha = \beta = 2$, according to formula (2.259), we must have:

$$\hat{\tilde{g}}(m, n) = \frac{1}{4} \hat{g}\left(\frac{m}{2}, \frac{n}{2}\right) \quad (2.266)$$

By comparing (2.264) and (2.265) we see that (2.266) is verified.

Example B2.46

If $w_N \equiv e^{-j2\pi/N}$, show that

$$w_{2M}^{2t} = w_M^t \quad w_{2M}^{2ut+u} = w_M^{ut} w_{2M}^u \quad (2.267)$$

where N, M, t and u are integers.

By definition:

$$w_{2M}^{2t} = \left(e^{-j2\pi/2M}\right)^{2t} = e^{-j2\pi 2t/2M} = e^{-j2\pi t/M} = \left(e^{-j2\pi/M}\right)^t = w_M^t \quad (2.268)$$

Similarly:

$$\begin{aligned}w_{2M}^{2ut+u} &= \left(e^{-j2\pi/2M}\right)^{2ut+u} = e^{-j2\pi(2ut+u)/2M} \\ &= e^{-j2\pi 2ut/2M} e^{-j2\pi u/2M} = e^{-j2\pi ut/M} w_{2M}^u = w_M^{ut} w_{2M}^u\end{aligned}\quad (2.269)$$

Example B2.47

If $w_M \equiv e^{\frac{-j2\pi}{M}}$, show that

$$\begin{aligned} w_M^{u+M} &= w_M^u \\ w_{2M}^{u+M} &= -w_{2M}^u \end{aligned} \quad (2.270)$$

where M and u are integers.

By definition:

$$w_M^{u+M} = e^{\frac{-j2\pi(u+M)}{M}} = e^{\frac{-j2\pi u}{M}} e^{\frac{-j2\pi M}{M}} = w_M^u e^{-j2\pi} = w_M^u \quad (2.271)$$

Also:

$$w_{2M}^{u+M} = e^{\frac{-j2\pi(u+M)}{2M}} = e^{\frac{-j2\pi u}{2M}} e^{\frac{-j2\pi M}{2M}} = w_{2M}^u e^{-j\pi} = -w_{2M}^u \quad (2.272)$$

Box 2.8. What is the Fast Fourier Transform?

All the transforms we have dealt with so far are separable. This means that they may be computed as two 1D transforms as opposed to one 2D transform. The discrete Fourier transform in 2D may be computed as two discrete Fourier transforms in 1D, using special algorithms which have been especially designed for speed and efficiency. Such algorithms are called **Fast Fourier Transforms (FFT)**. We shall describe briefly here the Fast Fourier Transform algorithm called **successive doubling**. We shall work in 1D. The discrete Fourier transform is defined as

$$\hat{f}(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) w_N^{ux} \quad (2.273)$$

where $w_N \equiv e^{\frac{-j2\pi}{N}}$. Assume now that $N = 2^n$. Then we may write N as $2M$ and substitute in (2.273):

$$\hat{f}(u) = \frac{1}{2M} \sum_{x=0}^{2M-1} f(x) w_{2M}^{ux} \quad (2.274)$$

We may separate the odd and even values of the argument of f . Let us express that by writing:

$$\begin{aligned} x &\equiv 2y && \text{when } x \text{ is even} \\ x &\equiv 2y + 1 && \text{when } x \text{ is odd} \end{aligned} \quad (2.275)$$

Then:

$$\hat{f}(u) = \frac{1}{2} \left\{ \frac{1}{M} \sum_{y=0}^{M-1} f(2y) w_{2M}^{u(2y)} + \frac{1}{M} \sum_{y=0}^{M-1} f(2y+1) w_{2M}^{u(2y+1)} \right\} \quad (2.276)$$

From example 2.46 we know that $w_{2M}^{2uy} = w_M^{uy}$ and $w_{2M}^{2uy+u} = w_M^{uy} w_{2M}^u$. Then:

$$\hat{f}(u) = \frac{1}{2} \left\{ \frac{1}{M} \sum_{y=0}^{M-1} f(2y) w_M^{uy} + \frac{1}{M} \sum_{y=0}^{M-1} f(2y+1) w_M^{uy} w_{2M}^u \right\} \quad (2.277)$$

We may write

$$\hat{f}(u) \equiv \frac{1}{2} \left\{ \hat{f}_{even}(u) + \hat{f}_{odd}(u) w_{2M}^u \right\} \quad (2.278)$$

where we have defined $\hat{f}_{even}(u)$ to be the DFT of the even samples of function f and \hat{f}_{odd} to be the DFT of the odd samples of function f :

$$\begin{aligned} \hat{f}_{even}(u) &\equiv \frac{1}{M} \sum_{y=0}^{M-1} f(2y) w_M^{uy} \\ \hat{f}_{odd}(u) &\equiv \frac{1}{M} \sum_{y=0}^{M-1} f(2y+1) w_M^{uy} \end{aligned} \quad (2.279)$$

Formula (2.278), however, defines $\hat{f}(u)$ only for $u < M$ because definitions (2.279) are valid for $0 \leq u < M$, being the DFTs of M -sample long functions. We need to define $\hat{f}(u)$ for $u = 0, 1, \dots, N-1$, ie for u up to $2M-1$. For this purpose, we apply formula (2.277) with $u+M$ as the argument of \hat{f} :

$$\hat{f}(u+M) = \frac{1}{2} \left\{ \frac{1}{M} \sum_{y=0}^{M-1} f(2y) w_M^{uy+My} + \frac{1}{M} \sum_{y=0}^{M-1} f(2y+1) w_M^{uy+My} w_{2M}^{u+M} \right\} \quad (2.280)$$

Making use of equations (2.270) we obtain:

$$\hat{f}(u+M) = \frac{1}{2} \left\{ \hat{f}_{even}(u) - \hat{f}_{odd}(u) w_{2M}^u \right\} \quad (2.281)$$

We note that formulae (2.278) and (2.281) with definitions (2.279) fully define $\hat{f}(u)$. Thus, an N point transform may be computed as two $N/2$ point transforms given by equations (2.279). Then equations (2.278) and (2.281) may be used to calculate the full transform. It can be shown that the number of operations required reduces from being proportional to N^2 to being proportional to $N \log_2 N$. This is another reason why images with dimensions powers of 2 are preferred.

What are the advantages and disadvantages of DFT?

DFT offers a much richer representation of an image than the Walsh or the Haar transforms. However, it achieves that at the expense of using complex numbers. So, although in theory the approximation of an image in terms of Fourier coefficients is more accurate than its approximation in terms of Walsh transforms for a fixed number of terms retained, the Fourier coefficients are complex numbers and each one requires twice as many bits to be represented as a Walsh coefficient. So, it is not fair to compare the error of the DFT with that of the other transforms for a fixed number of terms retained, but rather the error of the DFT for K terms retained with that of the other transforms for $2K$ terms retained.

Another disadvantage of DFT is shared with the Walsh transform when these two transforms are compared with the Haar transform: they are both global transforms as can be inferred from the structure of the basis functions in terms of which they expand an image. So, neither DFT nor the Walsh transform allows the preferential reconstruction of an image at certain localities like Haar transform does. This situation, however, is mitigated by applying the DFT in local windows of the image. This leads to **Gabor functions** which are extensively examined in Book II and are beyond the scope of this book.

Can we have a real valued DFT?

Yes, if the signal is real and symmetric, defined over a symmetric range of values.

Let us consider the DFT of a symmetric signal $f(k)$ that is defined over a symmetric range of indices. Let us say that this signal consists of N samples where N is even, so that we may write $N \equiv 2J$.

We have to be careful now about the values the indices of this signal take when we use them in the exponent of the kernel of the DFT. As the origin of the axes is in between the two central samples of the signal, all samples are at half integer locations, starting from $\pm\frac{1}{2}$ and going up to $\pm(J - \frac{1}{2})$. For example, in example 2.40, on page 117, we have a 2D image. One line of it may be treated as a 1D signal with $N = 4$ and therefore $J = 2$, with the indices of the available samples taking values $\{-2, -1, 0, 1\}$, corresponding to true coordinate locations $\{-\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}\}$. It is these coordinate locations that will have to be used in the exponent of the DFT kernel to weigh the corresponding sample values.

So, the DFT of signal $f(k)$ will be given by:

$$\begin{aligned}
 F(m) &= \frac{1}{2J} \sum_{k=-J}^{J-1} f(k) e^{-j \frac{2\pi}{2J} m (k + \frac{1}{2})} \\
 &= \frac{1}{2J} \sum_{k=-J}^{J-1} f(k) \cos \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] - j \frac{1}{2J} \underbrace{\sum_{k=-J}^{J-1} f(k) \sin \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right]}_S
 \end{aligned} \tag{2.282}$$

It can be shown that $S = 0$ (see example 2.48).

Example B2.48

Show that the imaginary part of (2.282) is zero.

Let us split S into two parts, made up from the negative and non-negative indices:

$$S = \sum_{k=-J}^{-1} f(k) \sin \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] + \sum_{k=0}^{J-1} f(k) \sin \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] \quad (2.283)$$

In the first sum on the right-hand side of the above equation, let us define a new summation variable $k' \equiv -k - 1 \Rightarrow k = -k' - 1$. The limits of summation over k' then will be from $J - 1$ to 0. As the order by which we sum does not matter, we may exchange the lower with the upper limit. We shall then have:

$$\begin{aligned} S &= \sum_{k'=0}^{J-1} f(-k' - 1) \sin \left[\frac{2\pi m}{2J} \left(-k' - 1 + \frac{1}{2} \right) \right] + \sum_{k=0}^{J-1} f(k) \sin \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] \\ &= \sum_{k'=0}^{J-1} f(-k' - 1) \sin \left[\frac{2\pi m}{2J} \left(-k' - \frac{1}{2} \right) \right] + \sum_{k=0}^{J-1} f(k) \sin \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] \end{aligned} \quad (2.284)$$

Function $f(k)$ is symmetric, so the values at the negative indices $[-J, -J + 1, \dots, -2, -1]$ are mirrored in the values of the non-negative indices $[0, 1, \dots, J - 2, J - 1]$. This means that $f(-k' - 1) = f(k')$. We also remember that for any real x , $\sin(-x) = -\sin x$ and that as k' is a dummy summation index, it may be called anything we like, including k . Then we can deduce that the imaginary part of (2.282) is zero:

$$S = - \sum_{k'=0}^{J-1} f(k') \sin \left[\frac{2\pi m}{2J} \left(k' + \frac{1}{2} \right) \right] + \sum_{k=0}^{J-1} f(k) \sin \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] = 0 \quad (2.285)$$

Example 2.49

Show that a real symmetric signal $f(k)$ made up from an odd number of samples $2J + 1$, defined over a symmetric range of indices, has a real DFT.

The signal is defined over indices $[-J, -J + 1, \dots, -1, 0, 1, \dots, J - 1, J]$. Let us take its DFT:

$$F(m) = \frac{1}{2J + 1} \sum_{k=-J}^J f(k) e^{-j2\pi \frac{mk}{2J+1}} \quad (2.286)$$

We may separate the real and imaginary parts, to write:

$$F(m) = \frac{1}{2J+1} \sum_{k=-J}^J f(k) \cos \frac{2\pi mk}{2J+1} - j \underbrace{\frac{1}{2J+1} \sum_{k=-J}^J f(k) \sin \frac{2\pi mk}{2J+1}}_S \quad (2.287)$$

Let us concentrate on the imaginary part and let us split the sum into three terms, the negative indices, the 0 index and the positive indices:

$$S = \sum_{k=-J}^{-1} f(k) \sin \frac{2\pi mk}{2J+1} + f(0) \sin 0 + \sum_{k=1}^J f(k) \sin \frac{2\pi mk}{2J+1} \quad (2.288)$$

In the first sum, we change summation variable from k to $k' \equiv -k \Rightarrow k = -k'$. The summation limits then become from J to 1, and since in summation the order of the summands does not matter, we may say that the summation over k' runs from 1 to J :

$$S = \sum_{k'=1}^J f(-k') \sin \left(-\frac{2\pi mk'}{2J+1} \right) + \sum_{k=1}^J f(k) \sin \frac{2\pi mk}{2J+1} \quad (2.289)$$

If $f(k)$ is symmetric, $f(-k') = f(k')$. We also know that $\sin(-x) = -\sin x$ for any real x , so we may write:

$$S = - \sum_{k'=1}^J f(k') \sin \frac{2\pi mk'}{2J+1} + \sum_{k=1}^J f(k) \sin \frac{2\pi mk}{2J+1} \quad (2.290)$$

As k' is a dummy summation variable, we may replace it with anything we like, say k . Then it becomes clear that the two sums on the right-hand side of (2.290) cancel out, and the imaginary part of (2.287) disappears.

Example 2.50

Show that the DFT of a real symmetric signal $f(k)$, made up from an odd number of samples $2J+1$, defined over a symmetric range of indices, may be written as:

$$F(m) = \frac{1}{2J+1} \left[f(0) + 2 \sum_{k=1}^J f(k) \cos \frac{2\pi mk}{2J+1} \right] \quad (2.291)$$

From example 2.49 we know that the DFT of such a signal is:

$$F(m) = \frac{1}{2J+1} \underbrace{\sum_{k=-J}^J f(k) \cos \frac{2\pi mk}{2J+1}}_S \quad (2.292)$$

Let us split the sum into three terms, the negative indices, the 0 index and the positive indices:

$$S = \sum_{k=-J}^{-1} f(k) \cos \frac{2\pi mk}{2J+1} + f(0) \cos 0 + \sum_{k=1}^J f(k) \cos \frac{2\pi mk}{2J+1} \quad (2.293)$$

In the first sum we change summation variable from k to $k' \equiv -k \Rightarrow k = -k'$. The summation limits then become from J to 1, and since in summation the order of the summands does not matter, we may say that the summation over k' runs from 1 to J :

$$S = f(0) + \sum_{k'=1}^J f(-k') \cos \left(-\frac{2\pi mk'}{2J+1} \right) + \sum_{k=1}^J f(k) \cos \frac{2\pi mk}{2J+1} \quad (2.294)$$

If $f(k)$ is symmetric, $f(-k') = f(k')$. We also know that $\cos(-x) = \cos x$ for any real x , so we may write:

$$S = f(0) + \sum_{k'=1}^J f(k') \cos \frac{2\pi mk'}{2J+1} + \sum_{k=1}^J f(k) \cos \frac{2\pi mk}{2J+1} \quad (2.295)$$

As k' is a dummy summation variable, we may replace it with anything we like, say k . Then formula (2.291) follows.

Example 2.51

Show that the DFT of a real symmetric signal $f(k)$ made up from an even number of samples $2J$, defined over a symmetric range of indices, may be written as:

$$F(m) = \frac{1}{J} \sum_{k=0}^{J-1} f(k) \cos \left[\frac{\pi m}{J} \left(k + \frac{1}{2} \right) \right] \quad (2.296)$$

According to equation (2.282) the DFT of such a signal is given by:

$$F(m) = \frac{1}{2J} \underbrace{\sum_{k=-J}^{J-1} f(k) \cos \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right]}_S \quad (2.297)$$

Let us split the sum into two parts, made up from the negative and non-negative indices:

$$S = \sum_{k=-J}^{-1} f(k) \cos \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] + \sum_{k=0}^{J-1} f(k) \cos \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] \quad (2.298)$$

In the first sum on the right-hand side of the above equation, let us define a new summation variable $k' \equiv -k - 1 \Rightarrow k = -k' - 1$. The limits of summation over k' then will be from $J - 1$ to 0. As the order by which we sum does not matter, we may exchange the lower with the upper limit. We shall then have:

$$\begin{aligned} S &= \sum_{k'=0}^{J-1} f(-k' - 1) \cos \left[\frac{2\pi m}{2J} \left(-k' - 1 + \frac{1}{2} \right) \right] + \sum_{k=0}^{J-1} f(k) \cos \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] \\ &= \sum_{k'=0}^{J-1} f(-k' - 1) \cos \left[\frac{2\pi m}{2J} \left(-k' - \frac{1}{2} \right) \right] + \sum_{k=0}^{J-1} f(k) \cos \left[\frac{2\pi m}{2J} \left(k + \frac{1}{2} \right) \right] \end{aligned} \quad (2.299)$$

Function $f(k)$ is symmetric, so the values at the negative indices $[-J, -J + 1, \dots, -2, -1]$ are mirrored in the values of the non-negative indices $[0, 1, \dots, J - 2, J - 1]$. This means that $f(-k' - 1) = f(k')$. We also remember that for any real x $\cos(-x) = \cos x$ and that as k' is a dummy summation index, it may be called anything we like, including k . Then formula (2.296) follows.

Can we have a purely imaginary DFT?

Yes, if the signal is real and antisymmetric, defined over a symmetric range of values.

Let us consider first the case when signal $f(k)$ is defined for an even set of indices $2J$ over a symmetric range of values. If $f(k)$ is antisymmetric, $f(-k' - 1) = -f(k')$ in (2.284) and the two sums instead of cancelling out, are identical. On the contrary, the two sums in (2.299) cancel out instead of being equal. So, the DFT of an antisymmetric signal, made up from an even number $2J$ of samples, is given by:

$$F(m) = -j \frac{1}{J} \sum_{k=0}^{J-1} f(k) \sin \left[\frac{\pi m}{J} \left(k + \frac{1}{2} \right) \right] \quad (2.300)$$

In the case of an antisymmetric signal defined over an odd set of indices $2J + 1$, we have $f(-k') = -f(k')$, so the two sums in (2.289), instead of cancelling out, are identical. On the other hand, the two sums in (2.294), instead of adding up, cancel out. In addition, if $f(-k') = -f(k')$, $f(0)$ has to be 0 as no other number is equal to its opposite. Further, the two sums in (2.290), instead of cancelling out, are identical. According to all these observations then, the DFT of such a signal is given by:

$$F(m) = -j \frac{2}{2J + 1} \sum_{k=1}^J f(k) \sin \frac{2\pi mk}{2J + 1} \quad (2.301)$$

Example B2.52

A 2D function $f(k, l)$ is defined for k taking integer values in the range $[-M, M - 1]$ and l taking integer values in the range $[-N, N - 1]$, and it has the following properties:

$$f(-k-1, -l-1) = f(k, l) \quad f(-k-1, l) = f(k, l) \quad f(k, -l-1) = f(k, l) \quad (2.302)$$

Work out the DFT of this function.

Applying (2.282) to 2D, we obtain:

$$\begin{aligned} F(m, n) &= \frac{1}{2MN} \sum_{k=-M}^{M-1} \sum_{l=-N}^{N-1} f(k, l) e^{-j\frac{2\pi}{2M}m(k+\frac{1}{2})} e^{-j\frac{2\pi}{2N}n(l+\frac{1}{2})} \\ &= \frac{1}{4MN} \sum_{k=-M}^{M-1} \sum_{l=-N}^{N-1} f(k, l) e^{-j\pi[\frac{m}{M}(k+\frac{1}{2}) + \frac{n}{N}(l+\frac{1}{2})]} \end{aligned} \quad (2.303)$$

We split the negative from the non-negative indices in each fraction:

$$\begin{aligned} F(m, n) &= \frac{1}{4MN} \times \quad (2.304) \\ &\left\{ \underbrace{\sum_{k=-M}^{-1} \sum_{l=-N}^{-1}}_{A_1} + \underbrace{\sum_{k=-M}^{-1} \sum_{l=0}^{N-1}}_{A_2} + \underbrace{\sum_{k=0}^{M-1} \sum_{l=-N}^{-1}}_{A_3} + \underbrace{\sum_{k=0}^{M-1} \sum_{l=0}^{N-1}}_{A_4} \right\} f(k, l) e^{-j\pi[\frac{m}{M}(k+\frac{1}{2}) + \frac{n}{N}(l+\frac{1}{2})]} \end{aligned}$$

We shall change variables of summation in A_1 to $\tilde{k} \equiv -k - 1 \Rightarrow k = -\tilde{k} - 1$ and $\tilde{l} \equiv -l - 1 \Rightarrow l = -\tilde{l} - 1$. We shall also use the first of properties (2.302), and the trigonometric identities $\cos(a+b) = \cos a \cos b - \sin a \sin b$ and $\sin(a+b) = \sin a \cos b + \sin b \cos a$. Further, we remember that $\cos(-a) = \cos a$ and $\sin(-a) = -\sin a$. Then:

$$A_1 = \sum_{\tilde{k}=0}^{M-1} \sum_{\tilde{l}=0}^{N-1} f(-\tilde{k} - 1, -\tilde{l} - 1) e^{-j\pi[\frac{m}{M}(-\tilde{k}-\frac{1}{2}) + \frac{n}{N}(-\tilde{l}-\frac{1}{2})]} \quad (2.305)$$

Or:

$$\begin{aligned}
A_1 &= \sum_{\tilde{k}=0}^{M-1} \sum_{\tilde{l}=0}^{N-1} f(\tilde{k}, \tilde{l}) \left\{ \cos \left[\frac{m\pi}{M} \left(-\tilde{k} - \frac{1}{2} \right) + \frac{n\pi}{N} \left(-\tilde{l} - \frac{1}{2} \right) \right] \right. \\
&\quad \left. - j \sin \left[\frac{m\pi}{M} \left(-\tilde{k} - \frac{1}{2} \right) + \frac{n\pi}{N} \left(-\tilde{l} - \frac{1}{2} \right) \right] \right\} \\
&= \sum_{\tilde{k}=0}^{M-1} \sum_{\tilde{l}=0}^{N-1} f(\tilde{k}, \tilde{l}) \left\{ \cos \left[\frac{m\pi}{M} \left(-\tilde{k} - \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(-\tilde{l} - \frac{1}{2} \right) \right] \right. \\
&\quad - \sin \left[\frac{m\pi}{M} \left(-\tilde{k} - \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(-\tilde{l} - \frac{1}{2} \right) \right] \\
&\quad - j \sin \left[\frac{m\pi}{M} \left(-\tilde{k} - \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(-\tilde{l} - \frac{1}{2} \right) \right] \\
&\quad \left. - j \cos \left[\frac{m\pi}{M} \left(-\tilde{k} - \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(-\tilde{l} - \frac{1}{2} \right) \right] \right\} \\
&= \sum_{\tilde{k}=0}^{M-1} \sum_{\tilde{l}=0}^{N-1} f(\tilde{k}, \tilde{l}) \left\{ \cos \left[\frac{m\pi}{M} \left(\tilde{k} + \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(\tilde{l} + \frac{1}{2} \right) \right] \right. \\
&\quad - \sin \left[\frac{m\pi}{M} \left(\tilde{k} + \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(\tilde{l} + \frac{1}{2} \right) \right] \\
&\quad + j \sin \left[\frac{m\pi}{M} \left(\tilde{k} + \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(\tilde{l} + \frac{1}{2} \right) \right] \\
&\quad \left. + j \cos \left[\frac{m\pi}{M} \left(\tilde{k} + \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(\tilde{l} + \frac{1}{2} \right) \right] \right\} \tag{2.306}
\end{aligned}$$

Term A_4 may be written as:

$$\begin{aligned}
A_4 &= \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f(k, l) \left\{ \cos \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{n}{N} \left(l + \frac{1}{2} \right) \right] \right. \\
&\quad - \sin \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \sin \left[\frac{n}{N} \left(l + \frac{1}{2} \right) \right] \\
&\quad - j \sin \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{n}{N} \left(l + \frac{1}{2} \right) \right] \\
&\quad \left. - j \cos \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \sin \left[\frac{n}{N} \left(l + \frac{1}{2} \right) \right] \right\} \tag{2.307}
\end{aligned}$$

We observe that $A_1 + A_4$ may be written as:

$$\begin{aligned}
A_1 + A_4 &= \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} 2f(k, l) \left\{ \cos \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{n}{N} \left(l + \frac{1}{2} \right) \right] \right. \\
&\quad - \sin \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \sin \left[\frac{n}{N} \left(l + \frac{1}{2} \right) \right] \left. \right\} \tag{2.308}
\end{aligned}$$

Working in a similar way and changing variable of summation to $\tilde{k} \equiv -k - 1 \Rightarrow k = -\tilde{k} - 1$ in term A_2 , we deduce:

$$\begin{aligned} A_2 &= \sum_{\tilde{k}=0}^{M-1} \sum_{l=0}^{N-1} f(-\tilde{k} - 1, l) e^{-j\pi[\frac{m}{M}(-\tilde{k}-\frac{1}{2}) + \frac{n}{N}(l+\frac{1}{2})]} \\ &= \sum_{\tilde{k}=0}^{M-1} \sum_{l=0}^{N-1} f(\tilde{k}, l) \left\{ \cos \left[\frac{m\pi}{M} \left(\tilde{k} + \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \right. \\ &\quad + \sin \left[\frac{m\pi}{M} \left(\tilde{k} + \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \\ &\quad + j \sin \left[\frac{m\pi}{M} \left(-\tilde{k} - \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \\ &\quad \left. - j \cos \left[\frac{m\pi}{M} \left(-\tilde{k} - \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \right\} \end{aligned} \quad (2.309)$$

Working in a similar way and changing variable of summation to $\tilde{l} \equiv -l - 1 \Rightarrow l = -\tilde{l} - 1$ in term A_3 , we deduce:

$$\begin{aligned} A_3 &= \sum_{k=0}^{M-1} \sum_{\tilde{l}=0}^{N-1} f(k, \tilde{l}) \left\{ \cos \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(\tilde{l} + \frac{1}{2} \right) \right] \right. \\ &\quad + \sin \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(\tilde{l} + \frac{1}{2} \right) \right] \\ &\quad - j \sin \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(\tilde{l} + \frac{1}{2} \right) \right] \\ &\quad \left. + j \cos \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(\tilde{l} + \frac{1}{2} \right) \right] \right\} \end{aligned} \quad (2.310)$$

Sum $A_2 + A_3$ then is:

$$\begin{aligned} A_2 + A_3 &= \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} 2f(k, l) \left\{ \cos \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \right. \\ &\quad \left. + \sin \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \right\} \end{aligned} \quad (2.311)$$

Combining the sums (2.308) and (2.311) into (2.304), we obtain:

$$F(m, n) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f(k, l) \cos \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \quad (2.312)$$

Example B2.53

A 2D function $f(k, l)$ is defined for k taking integer values in the range $[-M, M - 1]$ and l taking integer values in the range $[-N, N - 1]$, and has the following properties:

$$f(-k - 1, -l - 1) = f(k, l) \quad f(-k - 1, l) = -f(k, l) \quad f(k, -l - 1) = -f(k, l) \quad (2.313)$$

Work out the DFT of this function.

This case is similar to that of example 2.52, except for the last two properties of function $f(k, l)$. The antisymmetry of the function in terms of each of its arguments separately, does not affect the sum of terms $A_1 + A_4$ of the DFT, given by (2.308). However, because the function that appears in terms A_2 and A_3 changes sign with the change of summation variable, the sum of terms A_2 and A_3 now has the opposite sign:

$$\begin{aligned} A_2 + A_3 &= \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} 2f(k, l) \left\{ \cos \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \right. \\ &\quad \left. + \sin \left[\frac{m}{M} \left(k + \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \right\} \end{aligned} \quad (2.314)$$

Combining sums (2.308) and (2.314) into (2.304), we obtain:

$$F(m, n) = -\frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f(k, l) \sin \left[\frac{m\pi}{M} \left(k + \frac{1}{2} \right) \right] \sin \left[\frac{n\pi}{N} \left(l + \frac{1}{2} \right) \right] \quad (2.315)$$

Example B2.54

A 2D function $f(k, l)$ is defined for k taking integer values in the range $[-M, M]$ and l taking integer values in the range $[-N, N]$, and has the following properties:

$$f(-k, -l) = f(k, l) \quad f(-k, l) = f(k, l) \quad f(k, -l) = f(k, l) \quad (2.316)$$

Work out the DFT of this function.

Applying equation (2.286) for 2D, we obtain:

$$F(m, n) = \frac{1}{(2M+1)(2N+1)} \sum_{k=-M}^M \sum_{l=-N}^N f(k, l) e^{-j \frac{2\pi m k}{2M+1}} e^{-j \frac{2\pi n l}{2N+1}} \quad (2.317)$$

We may separate the negative from the zero and the positive indices in the double sum:

$$F(m, n) = \frac{1}{(2M+1)(2N+1)} \left\{ \underbrace{\sum_{k=-M}^{-1} \sum_{l=-N}^{-1}}_{A_1} + \underbrace{\sum_{k=0}^0 \sum_{l=-N}^{-1}}_{A_2} + \underbrace{\sum_{k=1}^M \sum_{l=-N}^{-1}}_{A_3} + \underbrace{\sum_{k=-M}^{-1} \sum_{l=0}^0}_{A_4} + \right. \\ \left. \underbrace{\sum_{k=0}^0 \sum_{l=0}^0}_{A_5} + \underbrace{\sum_{k=1}^M \sum_{l=0}^0}_{A_6} + \underbrace{\sum_{k=-M}^{-1} \sum_{l=1}^N}_{A_7} + \underbrace{\sum_{k=0}^0 \sum_{l=1}^N}_{A_8} + \underbrace{\sum_{k=1}^M \sum_{l=1}^N}_{A_9} f(k, l) e^{-j[\frac{2\pi mk}{2M+1} + \frac{2\pi nl}{2N+1}]} \right\} \quad (2.318)$$

We shall use the identities $\cos(a+b) = \cos a \cos b - \sin a \sin b$ and $\sin(a+b) = \sin a \cos b + \cos a \sin b$, and the fact that $\cos(-a) = \cos a$ and $\sin(-a) = -\sin a$, and express the complex exponential in terms of trigonometric functions. Then:

$$A_9 = \sum_{k=1}^M \sum_{l=1}^N f(k, l) \left[\cos \frac{2\pi mk}{2M+1} \cos \frac{2\pi nl}{2N+1} - \sin \frac{2\pi mk}{2M+1} \sin \frac{2\pi nl}{2N+1} \right. \\ \left. - j \sin \frac{2\pi mk}{2M+1} \cos \frac{2\pi nl}{2N+1} - j \cos \frac{2\pi mk}{2M+1} \sin \frac{2\pi nl}{2N+1} \right] \quad (2.319)$$

In A_1 , change summation variables to $\tilde{k} \equiv -k \Rightarrow k = -\tilde{k}$ and $\tilde{l} \equiv -l \Rightarrow l = -\tilde{l}$. Also, use the first of properties (2.316):

$$A_1 = \sum_{\tilde{k}=1}^M \sum_{\tilde{l}=1}^N f(\tilde{k}, \tilde{l}) \left[\cos \frac{2\pi m\tilde{k}}{2M+1} \cos \frac{2\pi n\tilde{l}}{2N+1} - \sin \frac{2\pi m\tilde{k}}{2M+1} \sin \frac{2\pi n\tilde{l}}{2N+1} \right. \\ \left. + j \sin \frac{2\pi m\tilde{k}}{2M+1} \cos \frac{2\pi n\tilde{l}}{2N+1} + j \cos \frac{2\pi m\tilde{k}}{2M+1} \sin \frac{2\pi n\tilde{l}}{2N+1} \right] \quad (2.320)$$

Then:

$$A_1 + A_9 = 2 \sum_{k=1}^M \sum_{l=1}^N f(k, l) \left[\cos \frac{2\pi mk}{2M+1} \cos \frac{2\pi nl}{2N+1} - \sin \frac{2\pi mk}{2M+1} \sin \frac{2\pi nl}{2N+1} \right] \quad (2.321)$$

We observe that:

$$A_8 = \sum_{l=1}^N f(0, l) \left[\cos \frac{2\pi nl}{2N+1} - j \sin \frac{2\pi nl}{2N+1} \right] \quad (2.322)$$

In A_2 we set $k = 0$ and $\tilde{l} \equiv -l \Rightarrow l = -\tilde{l}$:

$$A_2 = \sum_{\tilde{l}=1}^N f(0, \tilde{l}) \left[\cos \frac{2\pi n\tilde{l}}{2N+1} + j \sin \frac{2\pi n\tilde{l}}{2N+1} \right] \quad (2.323)$$

Then:

$$A_2 + A_8 = 2 \sum_{l=1}^N f(0, l) \cos \frac{2\pi nl}{2N+1} \quad (2.324)$$

In A_3 , we set $\tilde{l} \equiv -l \Rightarrow l = -\tilde{l}$ and remember that $f(k, -\tilde{l}) = f(k, \tilde{l})$:

$$\begin{aligned} A_3 &= \sum_{k=1}^M \sum_{\tilde{l}=1}^N f(k, \tilde{l}) \left[\cos \frac{2\pi mk}{2M+1} \cos \frac{2\pi n\tilde{l}}{2N+1} + \sin \frac{2\pi mk}{2M+1} \sin \frac{2\pi n\tilde{l}}{2N+1} \right. \\ &\quad \left. - j \sin \frac{2\pi mk}{2M+1} \cos \frac{2\pi n\tilde{l}}{2N+1} + j \cos \frac{2\pi mk}{2M+1} \sin \frac{2\pi n\tilde{l}}{2N+1} \right] \end{aligned} \quad (2.325)$$

In A_7 , we set $\tilde{k} \equiv -k \Rightarrow k = -\tilde{k}$ and remember that $f(-\tilde{k}, l) = f(\tilde{k}, l)$:

$$\begin{aligned} A_7 &= \sum_{\tilde{k}=1}^M \sum_{l=1}^N f(\tilde{k}, l) \left[\cos \frac{2\pi m\tilde{k}}{2M+1} \cos \frac{2\pi nl}{2N+1} + \sin \frac{2\pi m\tilde{k}}{2M+1} \sin \frac{2\pi nl}{2N+1} \right. \\ &\quad \left. + j \sin \frac{2\pi m\tilde{k}}{2M+1} \cos \frac{2\pi nl}{2N+1} - j \cos \frac{2\pi m\tilde{k}}{2M+1} \sin \frac{2\pi nl}{2N+1} \right] \end{aligned} \quad (2.326)$$

Then:

$$A_3 + A_7 = 2 \sum_{k=1}^M \sum_{l=1}^N f(k, l) \left[\cos \frac{2\pi mk}{2M+1} \cos \frac{2\pi nl}{2N+1} + \sin \frac{2\pi mk}{2M+1} \sin \frac{2\pi nl}{2N+1} \right] \quad (2.327)$$

We observe that:

$$A_6 = \sum_{k=1}^M f(k, 0) \left[\cos \frac{2\pi mk}{2M+1} - j \sin \frac{2\pi mk}{2M+1} \right] \quad (2.328)$$

In A_4 , we set $\tilde{k} \equiv -k \Rightarrow k = -\tilde{k}$ and $l = 0$, and we observe that $f(-\tilde{k}, 0) = f(\tilde{k}, 0)$:

$$A_4 = \sum_{\tilde{k}=1}^M f(\tilde{k}, 0) \left[\cos \frac{2\pi m\tilde{k}}{2M+1} + j \sin \frac{2\pi m\tilde{k}}{2M+1} \right] \quad (2.329)$$

Then:

$$A_4 + A_6 = 2 \sum_{k=1}^M f(k, 0) \cos \frac{2\pi mk}{2M+1} \quad (2.330)$$

Finally, we observe that $A_5 = f(0, 0)$.

Putting all these together, we deduce that:

$$\begin{aligned}
 F(m, n) = & \frac{1}{(2M+1)(2N+1)} \left\{ 4 \sum_{k=1}^M \sum_{l=1}^N f(k, l) \cos \frac{2\pi mk}{2M+1} \cos \frac{2\pi nl}{2N+1} \right. \\
 & \left. + 2 \sum_{k=1}^M f(k, 0) \cos \frac{2\pi mk}{2M+1} + 2 \sum_{l=1}^N f(0, l) + f(0, 0) \right\} \quad (2.331)
 \end{aligned}$$

Example B2.55

A 2D function $f(k, l)$ is defined for k taking integer values in the range $[-M, M]$ and l taking integer values in the range $[-N, N]$, and has the following properties:

$$\begin{aligned}
 f(-k, -l) &= f(k, l) \\
 f(-k, l) &= -f(k, l) \\
 f(k, -l) &= -f(k, l) \\
 f(0, l) &= f(k, 0) = f(0, 0) = 0 \quad (2.332)
 \end{aligned}$$

Work out the DFT of this function.

We work as we did in example 2.54. However, due to the different properties of the function, now terms $A_2 = A_4 = A_5 = A_6 = A_8 = 0$. Further, sum $A_3 + A_7$ now has the opposite sign. This results to:

$$F(m, n) = -4 \frac{1}{(2M+1)(2N+1)} \sum_{k=1}^M \sum_{l=1}^N f(k, l) \sin \frac{2\pi mk}{2M+1} \sin \frac{2\pi nl}{2N+1} \quad (2.333)$$

Can an image have a purely real or a purely imaginary valued DFT?

In general no. The image has to be symmetric about both axes in order to have a real valued DFT (see example 2.40) and antisymmetric about both axes in order to have an imaginary valued DFT. In general this is not the case. However, we may double the size of the image by reflecting it about its axes in order to form a symmetric or an antisymmetric image four times the size of the original image. We may then take the DFT of the enlarged image which will be guaranteed to be real or imaginary, accordingly. This will result in the so called **even symmetric discrete cosine transform**, or the **odd symmetric discrete cosine transform**, or the **even antisymmetric discrete sine transform** or the **odd antisymmetric discrete sine transform**.

2.4 The even symmetric discrete cosine transform (EDCT)

What is the even symmetric discrete cosine transform?

Assume that we have an $M \times N$ image f and reflect it about its left and top border so that we have a $2M \times 2N$ image. The DFT of the $2M \times 2N$ image will be real (see example 2.52, on page 131) and given by:

$$\hat{f}_{ec}(m, n) \equiv \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f(k, l) \cos \left[\frac{\pi m}{M} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{\pi n}{N} \left(l + \frac{1}{2} \right) \right] \quad (2.334)$$

This is the **even symmetric cosine transform (EDCT)** of the original image.

Example 2.56

Compute matrix U_{ec} appropriate for multiplying from left and right an 8×8 image with the origin of the axes in its centre, in order to obtain its DFT.

When the origin of the axis of a 1D signal is in the middle of the signal, the kernel of the DFT is

$$\frac{1}{2J} e^{-j\frac{\pi}{J}m(k+\frac{1}{2})} \quad (2.335)$$

where m is the frequency index taking integer values from 0 to 7 and k is the signal index, taking integer values from -4 to +3 (see equation (2.282)). As J is half the size of the signal, here $J = 4$. Matrix U_{ec} is not symmetric in its arguments, so the general DFT transform of an image g will be $U_{ec}gU_{ec}^T$, instead of UGU we had for the case of the DFT using matrix U of equation (2.190), on page 100.

We may then construct matrix U_{ec} by allowing k to take all its possible values along each row and m to take all its possible values along each column. Then matrix $8U_{ec}$ is:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & \dots & 1 \\ e^{-j\frac{\pi}{4}(-\frac{7}{2})} & e^{-j\frac{\pi}{4}(-\frac{5}{2})} & e^{-j\frac{\pi}{4}(-\frac{3}{2})} & e^{-j\frac{\pi}{4}(-\frac{1}{2})} & e^{-j\frac{\pi}{4}\frac{1}{2}} & \dots & e^{-j\frac{\pi}{4}\frac{7}{2}} \\ e^{-j\frac{\pi}{4}2(-\frac{7}{2})} & e^{-j\frac{\pi}{4}2(-\frac{5}{2})} & e^{-j\frac{\pi}{4}2(-\frac{3}{2})} & e^{-j\frac{\pi}{4}2(-\frac{1}{2})} & e^{-j\frac{\pi}{4}2\frac{1}{2}} & \dots & e^{-j\frac{\pi}{4}2\frac{7}{2}} \\ e^{-j\frac{\pi}{4}3(-\frac{7}{2})} & e^{-j\frac{\pi}{4}3(-\frac{5}{2})} & e^{-j\frac{\pi}{4}3(-\frac{3}{2})} & e^{-j\frac{\pi}{4}3(-\frac{1}{2})} & e^{-j\frac{\pi}{4}3\frac{1}{2}} & \dots & e^{-j\frac{\pi}{4}3\frac{7}{2}} \\ e^{-j\frac{\pi}{4}4(-\frac{7}{2})} & e^{-j\frac{\pi}{4}4(-\frac{5}{2})} & e^{-j\frac{\pi}{4}4(-\frac{3}{2})} & e^{-j\frac{\pi}{4}4(-\frac{1}{2})} & e^{-j\frac{\pi}{4}4\frac{1}{2}} & \dots & e^{-j\frac{\pi}{4}4\frac{7}{2}} \\ e^{-j\frac{\pi}{4}5(-\frac{7}{2})} & e^{-j\frac{\pi}{4}5(-\frac{5}{2})} & e^{-j\frac{\pi}{4}5(-\frac{3}{2})} & e^{-j\frac{\pi}{4}5(-\frac{1}{2})} & e^{-j\frac{\pi}{4}5\frac{1}{2}} & \dots & e^{-j\frac{\pi}{4}5\frac{7}{2}} \\ e^{-j\frac{\pi}{4}6(-\frac{7}{2})} & e^{-j\frac{\pi}{4}6(-\frac{5}{2})} & e^{-j\frac{\pi}{4}6(-\frac{3}{2})} & e^{-j\frac{\pi}{4}6(-\frac{1}{2})} & e^{-j\frac{\pi}{4}6\frac{1}{2}} & \dots & e^{-j\frac{\pi}{4}6\frac{7}{2}} \\ e^{-j\frac{\pi}{4}7(-\frac{7}{2})} & e^{-j\frac{\pi}{4}7(-\frac{5}{2})} & e^{-j\frac{\pi}{4}7(-\frac{3}{2})} & e^{-j\frac{\pi}{4}7(-\frac{1}{2})} & e^{-j\frac{\pi}{4}7\frac{1}{2}} & \dots & e^{-j\frac{\pi}{4}7\frac{7}{2}} \end{pmatrix} \quad (2.336)$$

After simplification, this matrix becomes:

$$U_{ec} = \frac{1}{8} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ e^{j\frac{7\pi}{8}} & e^{j\frac{5\pi}{8}} & e^{j\frac{3\pi}{8}} & e^{j\frac{\pi}{8}} & e^{-j\frac{\pi}{8}} & e^{-j\frac{3\pi}{8}} & e^{-j\frac{5\pi}{8}} & e^{-j\frac{7\pi}{8}} \\ e^{j\frac{7\pi}{4}} & e^{j\frac{5\pi}{4}} & e^{j\frac{3\pi}{4}} & e^{j\frac{\pi}{4}} & e^{-j\frac{\pi}{4}} & e^{-j\frac{3\pi}{4}} & e^{-j\frac{5\pi}{4}} & e^{-j\frac{7\pi}{4}} \\ e^{j\frac{5\pi}{8}} & e^{j\frac{15\pi}{8}} & e^{j\frac{9\pi}{8}} & e^{j\frac{3\pi}{8}} & e^{-j\frac{3\pi}{8}} & e^{-j\frac{9\pi}{8}} & e^{-j\frac{15\pi}{8}} & e^{-j\frac{5\pi}{8}} \\ e^{j\frac{5\pi}{2}} & e^{j\frac{5\pi}{2}} & e^{j\frac{3\pi}{2}} & e^{j\frac{\pi}{2}} & e^{-j\frac{\pi}{2}} & e^{-j\frac{3\pi}{2}} & e^{-j\frac{5\pi}{2}} & e^{-j\frac{7\pi}{2}} \\ e^{j\frac{3\pi}{8}} & e^{j\frac{9\pi}{8}} & e^{j\frac{15\pi}{8}} & e^{j\frac{5\pi}{8}} & e^{-j\frac{5\pi}{8}} & e^{-j\frac{15\pi}{8}} & e^{-j\frac{9\pi}{8}} & e^{-j\frac{3\pi}{8}} \\ e^{j\frac{5\pi}{4}} & e^{j\frac{7\pi}{4}} & e^{j\frac{\pi}{4}} & e^{j\frac{3\pi}{4}} & e^{-j\frac{3\pi}{4}} & e^{-j\frac{\pi}{4}} & e^{-j\frac{7\pi}{4}} & e^{-j\frac{5\pi}{4}} \\ e^{j\frac{\pi}{8}} & e^{j\frac{3\pi}{8}} & e^{j\frac{5\pi}{8}} & e^{j\frac{7\pi}{8}} & e^{-j\frac{7\pi}{8}} & e^{-j\frac{5\pi}{8}} & e^{-j\frac{3\pi}{8}} & e^{-j\frac{\pi}{8}} \end{pmatrix} \quad (2.337)$$

Example 2.57

Compute the even symmetric cosine transform of image

$$g = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 2 & 2 & 0 \end{pmatrix} \quad (2.338)$$

by taking the DFT of the corresponding image of size 8×8 .

We start by creating first the corresponding large image of size 8×8 :

$$\tilde{g} = \begin{pmatrix} 0 & 2 & 2 & 1 & 1 & 2 & 2 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & 1 & 2 & 0 & 1 \\ 1 & 0 & 2 & 1 & 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 2 & 2 & 1 & 1 & 2 & 2 & 0 \end{pmatrix} \quad (2.339)$$

To take the DFT of this image we multiply it from the left with matrix (2.337) and from the right with the transpose of the same matrix. The result is:

$$\tilde{G}_{ec} = \begin{pmatrix} 0.875 & 0 & -0.088 & 0 & 0 & 0 & 0.088 & 0 \\ -0.129 & 0.075 & 0.139 & -0.146 & 0 & 0.146 & -0.139 & -0.075 \\ 0.177 & 0.149 & -0.125 & -0.129 & 0 & 0.129 & 0.125 & -0.149 \\ 0.149 & -0.208 & 0.010 & -0.013 & 0 & 0.013 & -0.010 & 0.208 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.149 & 0.208 & -0.010 & 0.013 & 0 & -0.013 & 0.010 & -0.208 \\ -0.177 & -0.149 & 0.125 & 0.129 & 0 & -0.129 & -0.125 & 0.149 \\ 0.129 & -0.075 & -0.139 & 0.146 & 0 & -0.146 & 0.139 & 0.075 \end{pmatrix} \quad (2.340)$$

Example 2.58

Compute the $(1, 2)$ element of the even symmetric cosine transform of image (2.338) by using formula (2.334). Compare your answer with that of example 2.57.

Applying the formula for $m = 1$, $n = 2$ and $M = N = 4$, we obtain:

$$\begin{aligned} \hat{g}_{ec}(1, 2) &= \frac{1}{16} \sum_{k=0}^3 \sum_{l=0}^3 g(k, l) \cos \left[\frac{\pi}{4} \left(k + \frac{1}{2} \right) \right] \cos \left[\frac{\pi 2}{4} \left(l + \frac{1}{2} \right) \right] \quad (2.341) \\ &= \frac{1}{16} \left\{ g(0, 0) \cos \frac{\pi}{8} \cos \frac{\pi}{4} + g(0, 1) \cos \frac{\pi}{8} \cos \frac{3\pi}{4} + g(0, 3) \cos \frac{\pi}{8} \cos \frac{7\pi}{4} + \right. \\ &\quad g(1, 0) \cos \frac{3\pi}{8} \cos \frac{\pi}{4} + g(2, 2) \cos \frac{5\pi}{8} \cos \frac{5\pi}{4} + g(2, 3) \cos \frac{5\pi}{8} \cos \frac{7\pi}{4} + \\ &\quad \left. g(3, 0) \cos \frac{7\pi}{8} \cos \frac{\pi}{4} + g(3, 1) \cos \frac{7\pi}{8} \cos \frac{3\pi}{4} + g(3, 2) \cos \frac{7\pi}{8} \cos \frac{5\pi}{4} \right\} \end{aligned}$$

Here we omitted terms for which $g(k, l) = 0$. Substituting the values of $g(k, l)$ in (2.341) and doing the calculation, we deduce that $\hat{g}_{ec}(1, 2) = 0.139402656$. We observe from (2.340) that $\tilde{G}_{ec}(1, 2) = 0.139$, so the two values agree.

Example B2.59

The even symmetric cosine transform of an M -sample long signal $f(k)$ is defined as:

$$\hat{f}_{ec}(m) \equiv \frac{1}{M} \sum_{k=0}^{M-1} f(k) \cos \frac{\pi m(2k+1)}{2M} \quad (2.342)$$

Identify the period of $\hat{f}_{ec}(m)$.

The period of a function is the smallest number X for which $\hat{f}_{ec}(m + X) = \hat{f}_{ec}(m)$, for all m . If $X \rightarrow +\infty$, the function is not periodic.

Using definition (2.342), we have:

$$\begin{aligned}\hat{f}_{ec}(m + X) &= \frac{1}{M} \sum_{k=0}^{M-1} f(k) \cos \frac{\pi(m+X)(2k+1)}{2M} \\ &= \frac{1}{M} \sum_{k=0}^{M-1} f(k) \cos \left(\underbrace{\frac{\pi m(2k+1)}{2M}}_{\phi} + \frac{\pi X(2k+1)}{2M} \right)\end{aligned}\quad (2.343)$$

In order to have $\hat{f}_{ec}(m + X) = \hat{f}_{ec}(m)$, we must have

$$\cos \left(\phi + \frac{\pi X(2k+1)}{2M} \right) = \cos \phi \quad (2.344)$$

This is only true if $\pi X(2k+1)/(2M)$ is an integer multiple of 2π . The first number for which this is guaranteed is for $X = 4M$. So, $\hat{f}_{ec}(m)$ is periodic with period $4M$.

Example B2.60

You are given a 5-sample long signal with the following values: $f(0) = 0$, $f(1) = 1$, $f(2) = 2$, $f(3) = 3$ and $f(4) = 4$. Compute its EDCT $\hat{f}_{ec}(m)$ and plot both the extended signal and its EDCT for 60 consecutive samples.

To compute the EDCT of the data we apply formula (2.342) for $M = 5$. According to example 2.59, $\hat{f}_{ec}(m)$ is periodic with period $4M = 20$. So, we work out its values for $m = 0, 1, 2, \dots, 19$. The values of $\hat{f}_{ec}(m)$ for one period are:

$$(-1, 0, -0.09, 0, 0, 0, 0.09, 0, 1, -2, 1, 0, 0.09, 0, 0, 0, -0.09, 0, -1, 2) \quad (2.345)$$

The extended version of the given signal is a 10-sample long signal formed by reflecting the original signal about its origin. The added samples are: $f(-5) = 4$, $f(-4) = 3$, $f(-3) = 2$, $f(-2) = 1$ and $f(-1) = 0$. So, the DFT of signal $(4, 3, 2, 1, 0, 0, 1, 2, 3, 4)$ is the EDCT of the original signal. The DFT “sees” the extended signal repeated ad infinitum. Figure 2.18 shows on the left the plot of 60 samples of this signal, and on the right three periods of the EDCT of the original data.

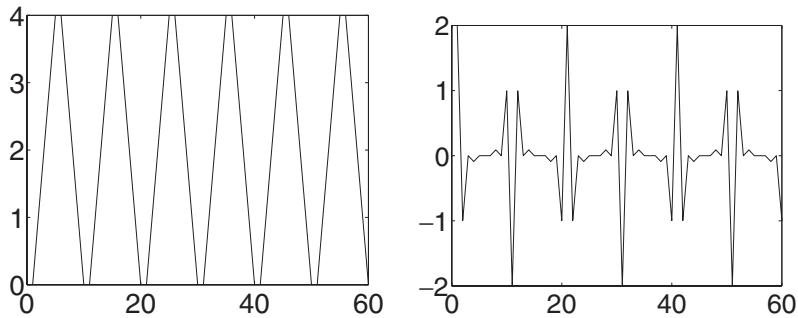


Figure 2.18: On the left, 60 consecutive samples of the extended signal seen by the DFT. On the right, the EDCT of the original 5-sample long signal also for 60 consecutive samples.

Example B2.61

Use definition (2.342) to show that $\hat{f}_{ec}(-m) = \hat{f}_{ec}(m)$.

By applying definition (2.342), we may write:

$$\begin{aligned}\hat{f}_{ec}(-m) &= \frac{1}{M} \sum_{k=0}^{M-1} f(k) \cos \frac{\pi(-m)(2k+1)}{2M} \\ &= \frac{1}{M} \sum_{k=0}^{M-1} f(k) \cos \frac{\pi m(2k+1)}{2M} = \hat{f}_{ec}(m)\end{aligned}\quad (2.346)$$

Example B2.62

If t is an integer, show that:

$$\sum_{m=-M}^{M-1} e^{j \frac{\pi t m}{M}} = 2M\delta(t) \quad (2.347)$$

We define a new variable of summation $\tilde{m} \equiv m + M \Rightarrow m = \tilde{m} - M$. Then:

$$\sum_{m=-M}^{M-1} e^{j\frac{\pi t m}{M}} = \sum_{\tilde{m}=0}^{2M-1} e^{j\frac{\pi t(\tilde{m}-M)}{M}} \quad (2.348)$$

We observe that $e^{j\frac{\pi t(\tilde{m}-M)}{M}} = e^{j\frac{\pi t \tilde{m}}{M}} e^{-j\pi t}$. Since $e^{-j\pi t} = \cos(\pi t) - j \sin(\pi t) = (-1)^t$, we may write:

$$\sum_{m=-M}^{M-1} e^{j\frac{\pi t m}{M}} = (-1)^t \sum_{\tilde{m}=0}^{2M-1} e^{j\frac{\pi t \tilde{m}}{M}} \quad (2.349)$$

The sum on the right-hand side of the above equation is a geometric progression with first term 1 and ratio $q \equiv e^{j\frac{\pi t}{M}}$. We apply formula (2.165), on page 95, to compute the sum of the first $2M$ terms of it, when $q \neq 1$, ie when $t \neq 0$, and obtain:

$$\sum_{m=-M}^{M-1} e^{j\frac{\pi t m}{M}} = (-1)^t \frac{\left(e^{j\frac{\pi t}{M}}\right)^{2M} - 1}{e^{j\frac{\pi t}{M}} - 1} = (-1)^t \frac{e^{j2\pi t} - 1}{e^{j\frac{\pi t}{M}} - 1} = 0 \quad (2.350)$$

This is because $e^{j2\pi t} = \cos(2\pi t) + j \sin(2\pi t) = 1$.

If $t = 0$, all terms in the sum on the left-hand side of (2.347) are equal to 1, so the sum is equal to $2M$. This completes the proof of (2.347).

Box 2.9. Derivation of the inverse 1D even discrete cosine transform

The 1D EDCT is defined by (2.342). Let us define $f(-k-1) \equiv f(k)$ for all values of $k = 0, 1, \dots, M-1$. We also note that:

$$\cos \frac{\pi m[2(-k-1)+1]}{2M} = \cos \frac{\pi m(-2k-1)}{2M} = \cos \frac{-\pi m(2k+1)}{2M} = \cos \frac{\pi m(2k+1)}{2M} \quad (2.351)$$

Then:

$$\sum_{k=-M}^{-1} f(k) \cos \frac{\pi m(2k+1)}{2M} = \sum_{k=0}^{M-1} f(k) \cos \frac{\pi m(2k+1)}{2M} \quad (2.352)$$

We can see that easily by changing variable of summation in the sum on the left-hand side from k to $\tilde{k} \equiv -k-1$. The limits of summation will become from $M-1$ to 0 and the summand will not change, as $f(-\tilde{k}-1) = f(\tilde{k})$ and similarly for the cosine factor. Replacing \tilde{k} then by k proves the equation. This means that we may replace definition (2.342) with:

$$\hat{f}_{ec}(m) \equiv \frac{1}{2M} \sum_{k=-M}^{M-1} f(k) \cos \frac{\pi m(2k+1)}{2M} \quad (2.353)$$

To derive the inverse transform we must solve this equation for $f(k)$. To achieve this, we multiply both sides of the equation with $\cos \frac{\pi m(2p+1)}{2M}$ and sum over m from $-M$ to $M - 1$:

$$\underbrace{\sum_{m=-M}^{M-1} \hat{f}_{ec}(m) \cos \frac{\pi m(2p+1)}{2M}}_S = \frac{1}{2M} \sum_{m=-M}^{M-1} \sum_{k=-M}^{M-1} f(k) \cos \frac{\pi m(2k+1)}{2M} \cos \frac{\pi m(2p+1)}{2M} \quad (2.354)$$

On the right-hand side we replace the trigonometric functions by using formula $\cos \phi \equiv (e^{j\phi} + e^{-j\phi})/2$, where ϕ is real. We also exchange the order of summations, observing that summation over m applies only to the kernel functions:

$$\begin{aligned} S &= \frac{1}{8M} \sum_{k=-M}^{M-1} f(k) \sum_{m=-M}^{M-1} \left[e^{j\frac{\pi m(2k+1)}{2M}} + e^{-j\frac{\pi m(2k+1)}{2M}} \right] \left[e^{j\frac{\pi m(2p+1)}{2M}} + e^{-j\frac{\pi m(2p+1)}{2M}} \right] \\ &= \frac{1}{8M} \sum_{k=-M}^{M-1} f(k) \sum_{m=-M}^{M-1} \left[e^{j\frac{\pi m(2k+2p+2)}{2M}} + e^{j\frac{\pi m(2k-2p)}{2M}} \right. \\ &\quad \left. + e^{j\frac{\pi m(-2k+2p)}{2M}} + e^{j\frac{\pi m(-2k-2p-2)}{2M}} \right] \\ &= \frac{1}{8M} \sum_{k=-M}^{M-1} f(k) \sum_{m=-M}^{M-1} \left[e^{j\frac{\pi m(k+p+1)}{M}} + e^{j\frac{\pi m(k-p)}{M}} \right. \\ &\quad \left. + e^{j\frac{\pi m(-k+p)}{M}} + e^{j\frac{\pi m(-k-p-1)}{M}} \right] \end{aligned} \quad (2.355)$$

To compute the sums over m , we make use of (2.347):

$$\begin{aligned} S &= \frac{1}{8M} \sum_{k=-M}^{M-1} f(k) [2M\delta(k+p+1) + 2M\delta(k-p) \\ &\quad + 2M\delta(-k+p) + 2M\delta(-k-p-1)] \\ &= \frac{1}{4} \sum_{k=-M}^{M-1} f(k) [\delta(k+p+1) + \delta(k-p) + \delta(-k+p) + \delta(-k-p-1)] \\ &= \frac{1}{4} \sum_{k=-M}^{M-1} f(k) [2\delta(k+p+1) + 2\delta(k-p)] \\ &= \frac{1}{2} \sum_{k=-M}^{M-1} f(k) [\delta(k+p+1) + \delta(k-p)] \end{aligned} \quad (2.356)$$

We used here the property of the delta function that $\delta(x) = \delta(-x)$. We note that, from all the terms in the sum, only two will survive, namely the one for $k = -p - 1$ and the one for $k = p$. Given that we defined $f(-k - 1) = f(k)$, both these terms will be equal,

ie $f(-p - 1) = f(p)$, and so we shall have that $S = f(p)$. This allows us to write the 1D inverse EDCT as:

$$f(p) = \sum_{m=-M}^{M-1} \hat{f}_{ec}(m) \cos \frac{\pi m(2p+1)}{2M} \quad (2.357)$$

We split the negative from the non-negative indices in the above sum:

$$f(p) = \underbrace{\sum_{m=-M}^{-1} \hat{f}_{ec}(m) \cos \frac{\pi m(2p+1)}{2M}}_S + \sum_{m=0}^{M-1} \hat{f}_{ec}(m) \cos \frac{\pi m(2p+1)}{2M} \quad (2.358)$$

In the first sum we change variable of summation from m to $\tilde{m} \equiv -m \Rightarrow m = -\tilde{m}$. The summation limits over \tilde{m} are then from M to 1, or from 1 to M :

$$S = \sum_{\tilde{m}=1}^M \hat{f}_{ec}(-\tilde{m}) \cos \frac{\pi(-\tilde{m})(2p+1)}{2M} = \sum_{\tilde{m}=1}^M \hat{f}_{ec}(\tilde{m}) \cos \frac{\pi\tilde{m}(2p+1)}{2M} \quad (2.359)$$

Here we made use of the result of example 2.61. Using (2.359) in (2.358), we may write:

$$f(p) = \hat{f}_{ec}(0) \cos \frac{\pi 0(2p+1)}{2M} + \hat{f}_{ec}(M) \cos \frac{\pi M(2p+1)}{2M} + 2 \sum_{m=1}^{M-1} \hat{f}_{ec}(m) \cos \frac{\pi m(2p+1)}{2M} \quad (2.360)$$

We note that

$$\cos \frac{\pi M(2p+1)}{2M} = \cos \frac{\pi(2p+1)}{2} = 0 \quad (2.361)$$

since the cosine of an odd multiple of $\pi/2$ is always 0. Finally, we may write for the inverse 1D EDCT

$$f(p) = \hat{f}_{ec}(0) + 2 \sum_{m=1}^{M-1} \hat{f}_{ec}(m) \cos \frac{\pi m(2p+1)}{2M} = \sum_{m=0}^{M-1} C(m) \hat{f}_{ec}(m) \cos \frac{\pi m(2p+1)}{2M} \quad (2.362)$$

where $C(0) = 1$ and $C(m) = 2$ for $m = 1, 2, \dots, M-1$.

What is the inverse 2D even cosine transform?

The inverse of equation (2.334) is

$$f(k, l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} C(m) C(n) \hat{f}_{ec}(m, n) \cos \frac{\pi m(2k+1)}{2M} \cos \frac{\pi n(2l+1)}{2N} \quad (2.363)$$

where $C(0) = 1$ and $C(m) = C(n) = 2$ for $m, n \neq 0$.

What are the basis images in terms of which the even cosine transform expands an image?

In equation (2.363), we may view function

$$T_m(k) \equiv C(m) \cos \frac{\pi m(2k+1)}{2M} \quad (2.364)$$

as a function of k with parameter m . Then the basis functions in terms of which an $M \times N$ image is expanded are the vector outer products of vector functions $T_m(k)T_n^T(l)$, where $k = 0, \dots, M-1$ and $l = 0, \dots, N-1$. For fixed (m, n) this vector outer product creates an elementary image of size $M \times N$. Coefficient $\hat{f}_{ec}(m, n)$ in (2.363) tells us the degree to which this elementary image is present in the original image $f(k, l)$.

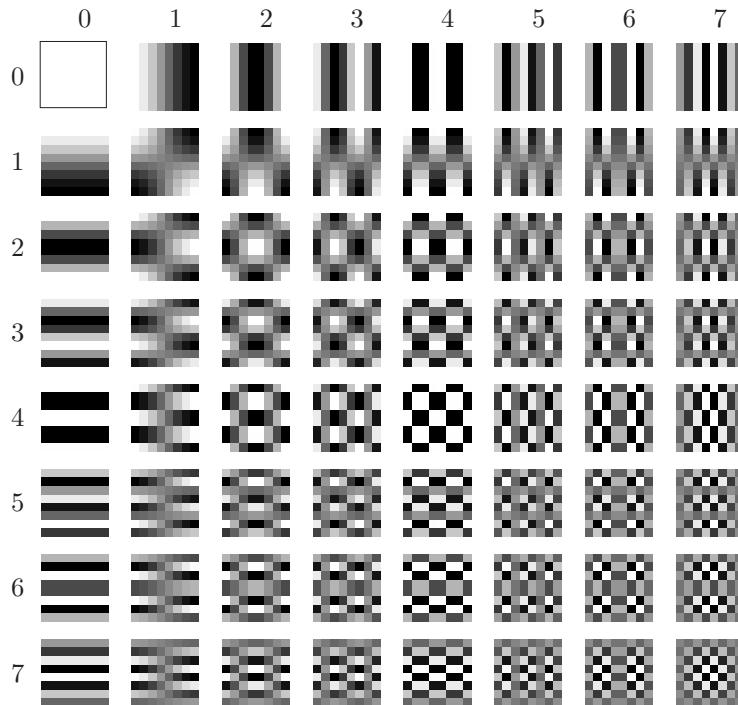


Figure 2.19: The basis images in terms of which any 8×8 image is expanded by EDCT. The numbers on the left and at the top indicate the indices of the $T_m(k)$ functions, the outer product of which resulted in the corresponding basis image. For example, the image in line 3 and column 0 corresponds to $T_3 T_0^T$, where the elements of these vectors are given by (2.364) for $k = 0, 1, \dots, 7$.

Figure 2.19 shows the elementary images in terms of which any 8×8 image is expanded by the EDCT. These images have been produced by setting $M = 8$ in (2.364) and allowing parameter m to take values $0, \dots, 7$. For every value of m we have a different function $T_m(k)$. Each one of these functions is then sampled at values of $k = 0, \dots, 7$ to form an 8×1 vector. The plots of these eight functions are shown in figure 2.20.

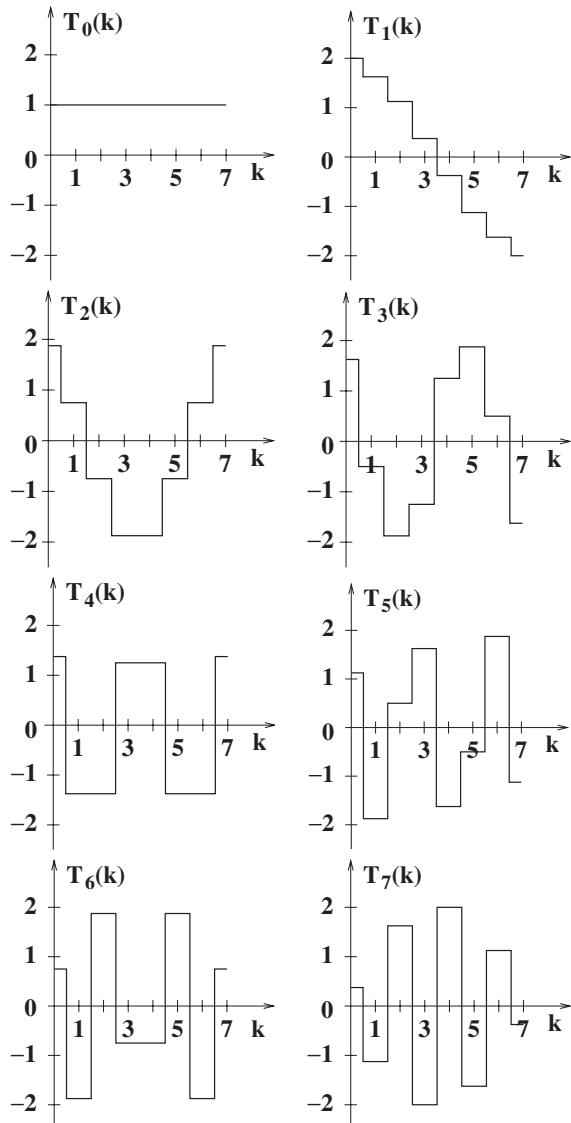


Figure 2.20: These plots are the digitised versions of $T_m(k)$ for $m = 0, 1, \dots, 7$, from top left to bottom right, respectively. Continuous valued functions $T_m(k)$ defined by equation (2.364) are sampled for integer values of k to form vectors. The outer product of these vectors in all possible combinations form the basis images of figure 2.19. In these plots the values of the functions at non-integer arguments are rounded to the nearest integer.

Figure 2.19 shows along the left and at the top which function $T_m(k)$ (identified by index m) was multiplied with which other function to create the corresponding elementary image. Each one of these elementary images is then scaled individually so that its grey values range from 1 to 255.

Example 2.63

Take the EDCT transform of image (2.103), on page 69, and show the various approximations of it by reconstructing it using only the first 1, 4, 9, etc elementary images in terms of which it is expanded.

The eight images shown in figure 2.21 are the reconstructed images when for the reconstruction the basis images created from the first one, two, ..., eight functions $T_m(k)$ are used. For example, figure 2.21f has been reconstructed from the inverse EDCT transform, by setting to 0 all elements of the transformation matrix that multiply the basis images in the bottom two rows and the two right-most columns in figure 2.19. These omitted basis images are those that are created from functions $T_6(k)$ and $T_7(k)$.

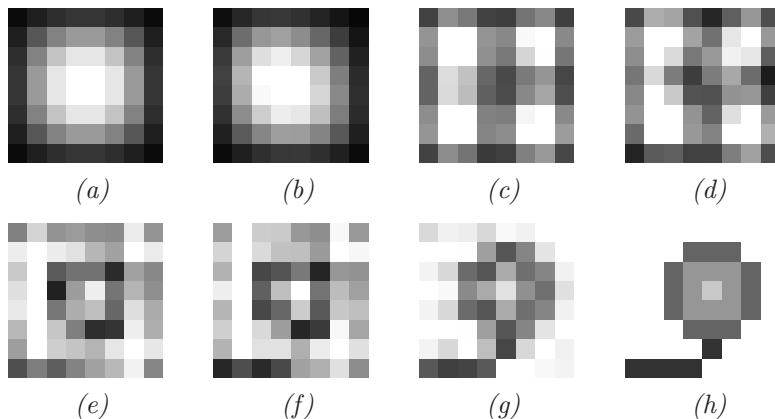


Figure 2.21: Approximate reconstructions of the flower image, by keeping only the coefficients that multiply the basis images produced by the outer products of the $T_m T_n^T$ vectors defined by equation (2.364), for all possible combinations of m and n , when m and n are allowed to take the value of 0 only, values $\{0, 1\}$, values $\{0, 1, 2\}$, etc, from top left to bottom right, respectively. In these reconstructions, values smaller than 0 and larger than 255 were truncated to 0 and 255, respectively, for displaying purposes.

The sum of the square errors for each reconstructed image is as follows.

Square error for image 2.21a:	366394
Square error for image 2.21b:	338683
Square error for image 2.21c:	216608
Square error for image 2.21d:	173305
Square error for image 2.21e:	104094
Square error for image 2.21f:	49179
Square error for image 2.21g:	35662
Square error for image 2.21h:	0

2.5 The odd symmetric discrete cosine transform (ODCT)

What is the odd symmetric discrete cosine transform?

Assume that we have an $M \times N$ image f and reflect it about its left-most column and about its topmost row so that we have a $(2M-1) \times (2N-1)$ image. The DFT of the $(2M-1) \times (2N-1)$ image will be real (see example 2.54) and given by:

$$\hat{f}_{oc}(m, n) \equiv \frac{1}{(2M-1)(2N-1)} \left[f(0, 0) + 4 \sum_{k=1}^{M-1} \sum_{l=1}^{N-1} f(k, l) \cos \frac{2\pi mk}{2M-1} \cos \frac{2\pi nl}{2N-1} + 2 \sum_{k=1}^{M-1} f(k, 0) \cos \frac{2\pi mk}{2M-1} + 2 \sum_{l=1}^{N-1} f(0, l) \cos \frac{2\pi nl}{2N-1} \right] \quad (2.365)$$

In a more concise way, this may be written as

$$\hat{f}_{oc}(m, n) \equiv \frac{1}{(2M-1)(2N-1)} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} C(k)C(l)f(k, l) \cos \frac{2\pi mk}{2M-1} \cos \frac{2\pi nl}{2N-1} \quad (2.366)$$

where $C(0) = 1$ and $C(k) = C(l) = 2$ for $k, l \neq 0$. This is the **odd symmetric discrete cosine transform (ODCT)** of the original image.

Example 2.64

Compute the odd symmetric cosine transform of image

$$g = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 2 & 2 & 0 \end{pmatrix} \quad (2.367)$$

by taking the DFT of the corresponding image of size 7×7 .

We start by creating first the corresponding large image of size 7×7 :

$$\tilde{g} = \begin{pmatrix} 0 & 2 & 2 & 1 & 2 & 2 & 0 \\ 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 2 & 2 & 1 & 2 & 2 & 0 \end{pmatrix} \quad (2.368)$$

To take the DFT of this image we have to multiply it from left and right with the appropriate matrix U for images of these dimensions. We create this matrix using definition (2.286). Here $J = 3$ and the elements of matrix U are given by $\frac{1}{7}e^{-j2\pi mk/7}$, where k takes values $-3, -2, -1, 0, 1, 2, 3$ along each row and m takes values $0, 1, 2, 3, 4, 5, 6$ along each column.

$$U_{oc} = \frac{1}{7} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ e^{j\frac{6\pi}{7}} & e^{j\frac{4\pi}{7}} & e^{j\frac{2\pi}{7}} & 1 & e^{-j\frac{2\pi}{7}} & e^{-j\frac{4\pi}{7}} & e^{-j\frac{6\pi}{7}} \\ e^{j\frac{12\pi}{7}} & e^{j\frac{8\pi}{7}} & e^{j\frac{4\pi}{7}} & 1 & e^{-j\frac{4\pi}{7}} & e^{-j\frac{8\pi}{7}} & e^{-j\frac{12\pi}{7}} \\ e^{j\frac{4\pi}{7}} & e^{j\frac{12\pi}{7}} & e^{j\frac{6\pi}{7}} & 1 & e^{-j\frac{6\pi}{7}} & e^{-j\frac{12\pi}{7}} & e^{-j\frac{4\pi}{7}} \\ e^{j\frac{10\pi}{7}} & e^{j\frac{2\pi}{7}} & e^{j\frac{8\pi}{7}} & 1 & e^{-j\frac{8\pi}{7}} & e^{-j\frac{2\pi}{7}} & e^{-j\frac{10\pi}{7}} \\ e^{j\frac{2\pi}{7}} & e^{j\frac{6\pi}{7}} & e^{j\frac{10\pi}{7}} & 1 & e^{-j\frac{10\pi}{7}} & e^{-j\frac{6\pi}{7}} & e^{-j\frac{2\pi}{7}} \\ e^{j\frac{8\pi}{7}} & e^{j\frac{10\pi}{7}} & e^{j\frac{12\pi}{7}} & 1 & e^{-j\frac{12\pi}{7}} & e^{-j\frac{10\pi}{7}} & e^{-j\frac{8\pi}{7}} \end{pmatrix} \quad (2.369)$$

We use this matrix to compute $U\tilde{g}U^T$. We keep separate the real and imaginary parts. We observe that the imaginary part turns out to be 0.

$$\tilde{G}_{oc} = \begin{pmatrix} 0.878 & -0.002 & -0.119 & 0.040 & 0.040 & -0.119 & -0.002 \\ -0.235 & 0.005 & 0.192 & -0.047 & -0.047 & 0.192 & 0.005 \\ 0.069 & 0.257 & -0.029 & -0.133 & -0.133 & -0.029 & 0.257 \\ 0.228 & -0.140 & -0.006 & -0.057 & -0.057 & -0.006 & -0.140 \\ 0.228 & -0.140 & -0.006 & -0.057 & -0.057 & -0.006 & -0.140 \\ 0.069 & 0.257 & -0.029 & -0.133 & -0.133 & -0.029 & 0.257 \\ -0.235 & 0.005 & 0.192 & -0.047 & -0.047 & 0.192 & 0.005 \end{pmatrix} \quad (2.370)$$

Example 2.65

Compute the $(1, 2)$ element of the odd symmetric cosine transform of image (2.367) by using formula (2.365). Compare your answer with that of example 2.64.

Applying the formula for $m = 1$, $n = 2$ and $M = N = 4$, we obtain

$$\hat{g}_{oc}(1, 2) \equiv \frac{1}{49} \sum_{k=0}^3 \sum_{l=0}^3 C(k)C(l)g(k, l) \cos \frac{2\pi k}{7} \cos \frac{2\pi 2l}{7} \quad (2.371)$$

where $C(0) = 1$ and $C(k) = C(l) = 2$ for $k, l \neq 0$. Expanding the sums and keeping only the nonzero elements, we deduce:

$$\begin{aligned}\hat{g}_{oc}(1, 2) = & \frac{1}{49} \left\{ g(0, 0) + 2g(0, 1) \cos \frac{4\pi}{7} + 2g(0, 3) \cos \frac{12\pi}{7} + 2g(1, 0) \cos \frac{2\pi}{7} + \right. \\ & 4g(2, 2) \cos \frac{4\pi}{7} \cos \frac{8\pi}{7} + 4g(2, 3) \cos \frac{4\pi}{7} \cos \frac{12\pi}{7} + 2g(3, 0) \cos \frac{6\pi}{7} + \\ & \left. 4g(3, 1) \cos \frac{6\pi}{7} \cos \frac{4\pi}{7} + 4g(3, 2) \cos \frac{6\pi}{7} \cos \frac{8\pi}{7} \right\} \quad (2.372)\end{aligned}$$

Substituting the values of $g(k, l)$ and performing the calculation, we deduce that $\hat{g}_{oc}(1, 2) = 0.191709$.

We see from (2.370) that the value of $\tilde{G}_{oc}(1, 2) = 0.192$.

Example B2.66

The odd symmetric cosine transform of an M -sample long signal $f(k)$ is defined as

$$\hat{f}_{oc}(m) \equiv \frac{1}{2M-1} \sum_{k=0}^{M-1} C(k) f(k) \cos \frac{2\pi mk}{2M-1} \quad (2.373)$$

where $C(0) = 1$ and $C(k) = 2$ for $k \neq 0$. Identify the period of $\hat{f}_{oc}(m)$.

The period of a function is the smallest number X for which $\hat{f}_{oc}(m+X) = \hat{f}_{oc}(m)$, for all m .

Using definition (2.373), we have:

$$\begin{aligned}\hat{f}_{oc}(m+X) &= \frac{1}{2M-1} \sum_{k=0}^{M-1} C(k) f(k) \cos \frac{2\pi(m+X)k}{2M-1} \\ &= \frac{1}{2M-1} \sum_{k=0}^{M-1} C(k) f(k) \cos \left(\underbrace{\frac{2\pi mk}{2M-1}}_{\phi} + \frac{2\pi Xk}{2M-1} \right) \quad (2.374)\end{aligned}$$

In order to have $\hat{f}_{oc}(m+X) = \hat{f}_{oc}(m)$, we must have:

$$\cos \left(\phi + \frac{2\pi Xk}{2M-1} \right) = \cos \phi \quad (2.375)$$

This is only true if $2\pi Xk/(2M-1)$ is an integer multiple of 2π . The first number for which this is guaranteed is for $X = 2M-1$. So, $\hat{f}_{oc}(m)$ is periodic with period $2M-1$.

Example B2.67**Show that**

$$\sum_{m=-M+1}^{M-1} e^{j \frac{2\pi t m}{2M-1}} = (2M-1)\delta(t) \quad (2.376)$$

for t integer.

We define a new summation variable $\tilde{m} \equiv m + M - 1 \Rightarrow m = \tilde{m} - M + 1$. Then we have

$$\sum_{m=-M+1}^{M-1} e^{j \frac{2\pi t m}{2M-1}} = \sum_{\tilde{m}=0}^{2M-2} e^{j \frac{2\pi t (\tilde{m}-M+1)}{2M-1}} = e^{j \frac{2\pi t (-M+1)}{2M-1}} \sum_{\tilde{m}=0}^{2M-2} e^{j \frac{2\pi t \tilde{m}}{2M-1}} = (2M-1)\delta(t) \quad (2.377)$$

where we made use of (2.164), on page 95, with $S = 2M - 1$ and the fact that for $t = 0$, $e^{j \frac{2\pi t (-M+1)}{2M-1}} = 1$.

Box 2.10. Derivation of the inverse 1D odd discrete cosine transform

The 1D ODCT is defined by (2.373). Let us define $f(-k) \equiv f(k)$ for values of $k = 1, \dots, M - 1$. As the cosine function is an even function with respect to its argument k , we may rewrite definition (2.373) as:

$$\hat{f}_{oc}(m) = \frac{1}{2M-1} \left\{ f(0) + 2 \sum_{k=1}^{M-1} f(k) \cos \frac{2\pi m k}{2M-1} \right\} = \frac{1}{2M-1} \sum_{k=-M+1}^{M-1} f(k) \cos \frac{2\pi m k}{2M-1} \quad (2.378)$$

To derive the inverse transform we must solve this equation for $f(k)$. To achieve this, we multiply both sides of the equation with $\cos \frac{2\pi m p}{2M-1}$ and sum over m from $-M + 1$ to $M - 1$:

$$\underbrace{\sum_{m=-M+1}^{M-1} \hat{f}_{oc}(m) \cos \frac{2\pi m p}{2M-1}}_S = \frac{1}{2M-1} \sum_{m=-M+1}^{M-1} \sum_{k=-M+1}^{M-1} f(k) \cos \frac{2\pi m k}{2M-1} \cos \frac{2\pi m p}{2M-1} \quad (2.379)$$

On the right-hand side we replace the trigonometric functions by using formula $\cos \phi \equiv (e^{j\phi} + e^{-j\phi})/2$, where ϕ is real. We also exchange the order of summations, observing that summation over m applies only to the kernel functions:

$$\begin{aligned}
S &= \frac{1}{4(2M-1)} \sum_{k=-M+1}^{M-1} f(k) \sum_{m=-M+1}^{M-1} \left[e^{j\frac{2\pi mk}{2M-1}} + e^{-j\frac{2\pi mk}{2M-1}} \right] \left[e^{j\frac{2\pi mp}{2M-1}} + e^{-j\frac{2\pi mp}{2M-1}} \right] \\
&= \frac{1}{4(2M-1)} \sum_{k=-M+1}^{M-1} f(k) \sum_{m=-M+1}^{M-1} \left[e^{j\frac{2\pi m(k+p)}{2M-1}} + e^{j\frac{2\pi m(k-p)}{2M-1}} \right. \\
&\quad \left. + e^{j\frac{2\pi m(-k+p)}{2M-1}} + e^{j\frac{2\pi m(-k-p)}{2M-1}} \right]
\end{aligned} \tag{2.380}$$

To compute the sums over m , we make use of (2.376):

$$\begin{aligned}
S &= \frac{1}{4(2M-1)} \sum_{k=-M+1}^{M-1} f(k) [(2M-1)\delta(k+p) + (2M-1)\delta(k-p) \\
&\quad + (2M-1)\delta(-k+p) + (2M-1)\delta(-k-p)] \\
&= \frac{1}{4} \sum_{k=-M+1}^{M-1} f(k) [\delta(k+p) + \delta(k-p) + \delta(-k+p) + \delta(-k-p)] \\
&= \frac{1}{4} \sum_{k=-M+1}^{M-1} f(k) [2\delta(k+p) + 2\delta(k-p)] \\
&= \frac{1}{2} \sum_{k=-M+1}^{M-1} f(k) [\delta(k+p) + \delta(k-p)]
\end{aligned} \tag{2.381}$$

We used here the property of the delta function that $\delta(x) = \delta(-x)$. We note that, from all the terms in the sum, only two will survive, namely the one for $k = -p$ and the one for $k = p$. Given that we defined $f(-k) = f(k)$, both these terms will be equal and so we shall have $S = f(p)$. This allows us to write the 1D inverse ODCT as:

$$f(p) = \sum_{m=-M+1}^{M-1} \hat{f}_{oc}(m) \cos \frac{2\pi mp}{2M-1} \tag{2.382}$$

From definition (2.373) it is obvious that $\hat{f}_{oc}(-m) = \hat{f}_{oc}(m)$. The cosine function is also an even function of m , so we may write

$$f(p) = \hat{f}_{oc}(0) + 2 \sum_{m=1}^{M-1} \hat{f}_{oc}(m) \cos \frac{2\pi mp}{2M-1} \tag{2.383}$$

or, in a more concise way,

$$f(p) = \sum_{m=0}^{M-1} C(m) \hat{f}_{oc}(m) \cos \frac{2\pi mp}{2M-1} \tag{2.384}$$

where $C(0) = 1$ and $C(m) = 2$ for $m \neq 0$.

What is the inverse 2D odd discrete cosine transform?

The inverse of equation (2.365) is

$$f(k, l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} C(m) C(n) \hat{f}_{oc}(m, n) \cos \frac{2\pi m k}{2M-1} \cos \frac{2\pi n l}{2N-1} \quad (2.385)$$

where $C(0) = 1$ and $C(m) = C(n) = 2$ for $m, n \neq 0$.

What are the basis images in terms of which the odd discrete cosine transform expands an image?

In equation (2.385), we may view function

$$U_m(k) \equiv C(m) \cos \frac{2\pi m k}{2M-1} \quad (2.386)$$

as a function of k with parameter m . Then the basis functions, in terms of which an $M \times N$ image is expanded, are the vector outer products of vector functions $U_m(k)U_n^T(l)$, where $k = 0, \dots, M-1$ and $l = 0, \dots, N-1$. For fixed (m, n) , each such vector outer product creates an elementary image of size $M \times N$. Coefficient $\hat{f}_{oc}(m, n)$ in (2.385) tells us the degree to which this elementary image is present in the original image $f(k, l)$.

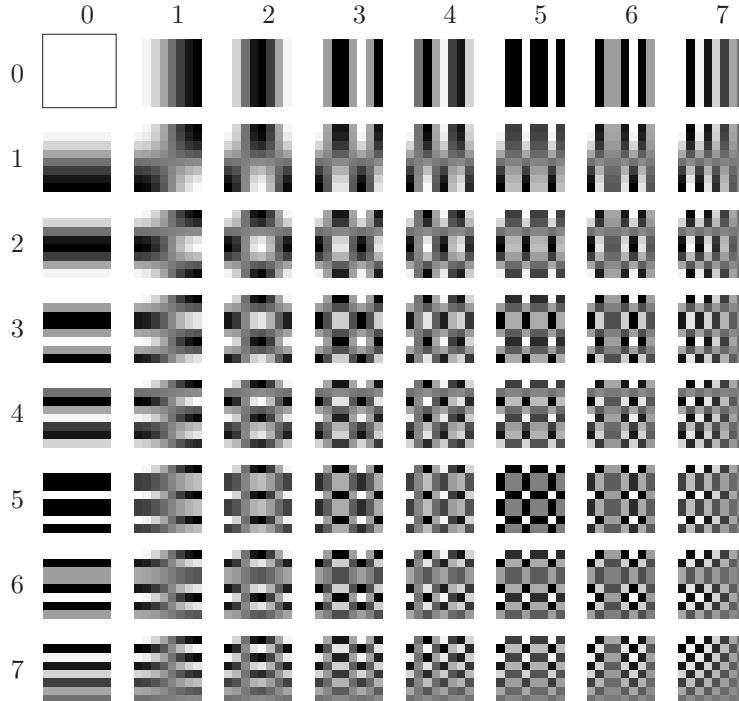


Figure 2.22: The basis images in terms of which ODCT expands any 8×8 image. The numbers on the left and at the top are the indices n and m , respectively, of the functions defined by (2.386), the vector outer product of which, $U_m U_n^T$, is the corresponding elementary image.

Figure 2.22 shows the elementary images in terms of which any 8×8 image is expanded by the ODCT. These images have been produced by setting $M = 8$ in (2.386) and allowing parameter m to take values $0, \dots, 7$. For every value of m we have a different function $U_m(k)$. Each one of these functions is then sampled at values of $k = 0, \dots, 7$ to form an 8×1 vector. The plots of these eight functions are shown in figure 2.23.

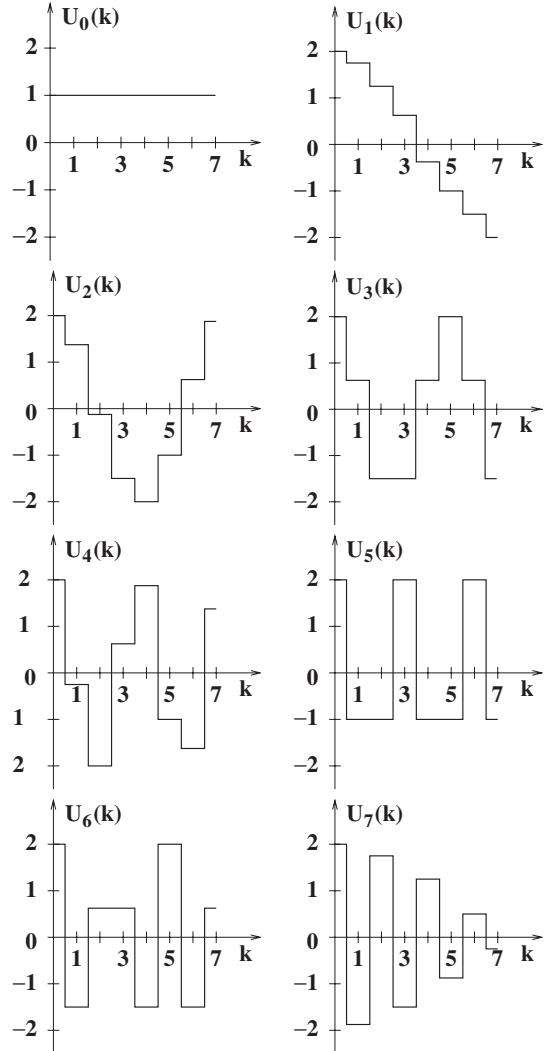


Figure 2.23: Functions $U_m(k)$ defined by (2.386), for $m = 0, 1, \dots, 7$, from top left to bottom right, respectively. Values of non-integer arguments are rounded to the nearest integer. The sampled versions of these functions at integer values of k are used to create the basis images of figure 2.22, by taking their vector outer product in all possible combinations.

Figure 2.22 shows along the left and at the top which function $U_m(k)$ (identified by index m) was multiplied with which other function to create the corresponding elementary image. Each one of these elementary images is then scaled individually so that its grey values range from 1 to 255.

Example 2.68

Take the ODCT transform of image (2.103), on page 69, and show the various approximations of it, by reconstructing it using only the first 1, 4, 9, etc elementary images in terms of which it is expanded.

The eight images shown in figure 2.24 are the reconstructed images, when, for the reconstruction, the basis images created from the first one, two, ..., eight functions $U_m(k)$ are used. For example, figure 2.24f has been reconstructed from the inverse ODCT transform, by setting to 0 all elements of the transformation matrix that multiply the basis images in the bottom two rows and the two right-most columns in figure 2.22. The omitted basis images are those that are created from functions $U_6(k)$ and $U_7(k)$.

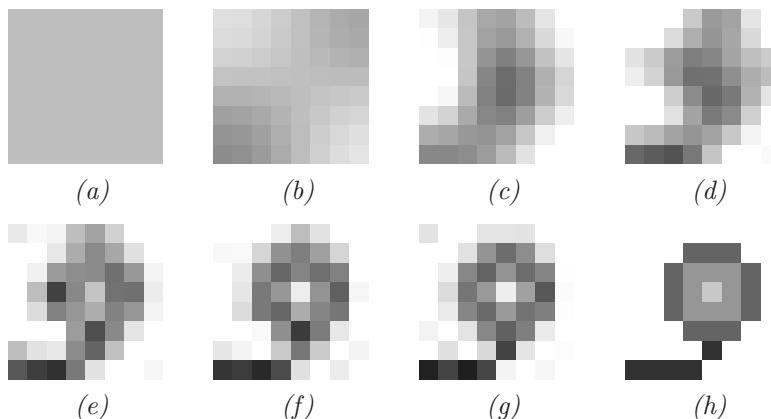


Figure 2.24: Gradually improved approximations of the flower image as more and more terms in its expansion, in terms of the basis images of figure 2.22, are retained, starting from the flat image at the top left corner and gradually adding one row and one column of images at a time, until the bottom-most row and the right-most column are added. In the approximate reconstructions, negative values and values larger than 255 were truncated to 0 and 255, respectively, for displaying purposes.

The sum of the square errors for each reconstructed image is as follows.

Square error for image 2.24a:	368946
Square error for image 2.24b:	342507
Square error for image 2.24c:	221297
Square error for image 2.24d:	175046
Square error for image 2.24e:	96924
Square error for image 2.24f:	55351
Square error for image 2.24g:	39293
Square error for image 2.24h:	0

2.6 The even antisymmetric discrete sine transform (EDST)

What is the even antisymmetric discrete sine transform?

Assume that we have an $M \times N$ image f , change its sign and reflect it about its left and top border so that we have a $2M \times 2N$ image. The DFT of the $2M \times 2N$ image will be real and given by (see example 2.53):

$$\hat{f}_{es}(m, n) \equiv -\frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f(k, l) \sin \frac{\pi m(2k+1)}{2M} \sin \frac{\pi n(2l+1)}{2N} \quad (2.387)$$

This is the **even antisymmetric discrete sine transform (EDST)** of the original image.

Example 2.69

Compute the even antisymmetric sine transform of image

$$g = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 2 & 2 & 0 \end{pmatrix} \quad (2.388)$$

by taking the DFT of the corresponding enlarged image of size 8×8 .

We start by creating first the corresponding large image of size 8×8 :

$$\tilde{g} = \begin{pmatrix} 0 & 2 & 2 & 1 & -1 & -2 & -2 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & -2 & -2 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & -1 & -2 & 0 & -1 \\ -1 & 0 & -2 & -1 & 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & -2 & -2 & -1 & 1 & 2 & 2 & 0 \end{pmatrix} \quad (2.389)$$

To take the DFT of this image we multiply it from the left with matrix U given by (2.337) and from the right with the transpose of the same matrix. The result is:

$$\hat{G}_{es} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.333 & 0.072 & -0.127 & -0.034 & -0.127 & 0.072 & -0.333 \\ 0 & 0.163 & -0.188 & -0.068 & 0.088 & -0.068 & -0.188 & 0.163 \\ 0 & -0.315 & -0.173 & 0.021 & 0.082 & 0.021 & -0.173 & -0.315 \\ 0 & -0.048 & 0.177 & -0.115 & 0.250 & -0.115 & 0.177 & -0.048 \\ 0 & -0.315 & -0.173 & 0.021 & 0.082 & 0.021 & -0.173 & -0.315 \\ 0 & 0.163 & -0.188 & -0.068 & 0.088 & -0.068 & -0.188 & 0.163 \\ 0 & -0.333 & 0.072 & -0.127 & -0.034 & -0.127 & 0.072 & -0.333 \end{pmatrix} \quad (2.390)$$

Example 2.70

Compute the $(1, 2)$ element of the even antisymmetric sine transform of image (2.388) by using formula (2.387). Compare your answer with that of example 2.69.

Applying the formula for $m = 1$, $n = 2$ and $M = N = 4$, we obtain:

$$\begin{aligned} \hat{g}_{es}(1, 2) &= -\frac{1}{16} \sum_{k=0}^3 \sum_{l=0}^3 g(k, l) \sin \frac{\pi(2k+1)}{8} \sin \frac{\pi(2l+1)}{8} \\ &= -\frac{1}{16} \left\{ g(0, 0) \sin \frac{\pi}{8} \sin \frac{\pi}{4} + g(0, 1) \sin \frac{\pi}{8} \sin \frac{3\pi}{4} \right. \\ &\quad + g(0, 3) \sin \frac{\pi}{8} \sin \frac{7\pi}{4} + g(1, 0) \sin \frac{3\pi}{8} \sin \frac{\pi}{4} \\ &\quad + g(2, 2) \sin \frac{5\pi}{8} \sin \frac{5\pi}{4} + g(2, 3) \sin \frac{5\pi}{8} \sin \frac{7\pi}{4} \\ &\quad \left. + g(3, 0) \sin \frac{7\pi}{8} \sin \frac{\pi}{4} + g(3, 1) \sin \frac{7\pi}{8} \sin \frac{3\pi}{4} \right. \\ &\quad \left. + g(3, 2) \sin \frac{7\pi}{8} \sin \frac{5\pi}{4} \right\} \end{aligned} \quad (2.391)$$

Here we omitted terms for which $g(k, l) = 0$. Substituting the values of $g(k, l)$ in (2.391) and performing the calculation, we deduce that $\hat{g}_{es}(1, 2) = 0.0718$.

We note from (2.390) that the $(1, 2)$ element of \hat{G}_{es} is 0.072.

Example B2.71

The even antisymmetric sine transform of an M -sample long signal $f(k)$ is defined as:

$$\hat{f}_{es}(m) \equiv -j \frac{1}{M} \sum_{k=0}^{M-1} f(k) \sin \frac{\pi m(2k+1)}{2M} \quad (2.392)$$

Identify the period of $\hat{f}_{es}(m)$.

The period of a function is the smallest number X for which $\hat{f}_{es}(m+X) = \hat{f}_{es}(m)$, for all m .

Using definition (2.392), we have:

$$\begin{aligned} \hat{f}_{es}(m+X) &= -j \frac{1}{M} \sum_{k=0}^{M-1} f(k) \sin \frac{\pi(m+X)(2k+1)}{2M} \\ &= -j \frac{1}{M} \sum_{k=0}^{M-1} f(k) \sin \left(\underbrace{\frac{\pi m(2k+1)}{2M}}_{\phi} + \frac{\pi X(2k+1)}{2M} \right) \end{aligned} \quad (2.393)$$

In order to have $\hat{f}_{es}(m+X) = \hat{f}_{es}(m)$, we must have:

$$\sin \left(\phi + \frac{\pi X(2k+1)}{2M} \right) = \sin \phi \quad (2.394)$$

This is only true if $\pi X(2k+1)/(2M)$ is an integer multiple of 2π . The first number for which this is guaranteed is for $X = 4M$. So, $\hat{f}_{es}(m)$ is periodic with period $4M$.

Example B2.72

You are given a 5-sample long signal with the following values: $f(0) = 0$, $f(1) = 1$, $f(2) = 2$, $f(3) = 3$ and $f(4) = 4$. Compute its EDST $\hat{f}_{es}(m)$ and plot both the extended signal and its EDST, for 50 consecutive samples.

The extended signal we create is $-4, -3, -2, -1, 0, 0, 1, 2, 3, 4$. DFT sees this signal repeated ad infinitum. Since $M = 5$ here, the EDST of the original signal has period 20. The values of $\hat{f}_{es}(m)$ for one period are:

$$(-1.29, 0.85, -0.49, 0.53, -0.4, 0.53, -0.49, 0.85, -1.29, 0, \\ 1.29, -0.85, 0.49, -0.53, 0.4, -0.53, 0.49, -0.85, 1.29, 0)$$

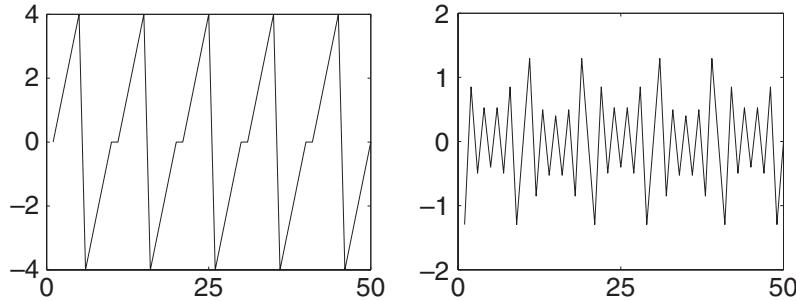


Figure 2.25: On the left, 50 consecutive samples of the extended signal as seen by the DFT. On the right, the EDST of the original 5-sample long signal, also for 50 consecutive samples.

Figure 2.25 shows 50 samples of the signal as seen by the DFT and 2.5 periods of the EDST of the original signal.

Box 2.11. Derivation of the inverse 1D even discrete sine transform

The 1D EDST is defined by (2.392). Let us define $f(-k-1) \equiv -f(k)$ for all values of $k = 0, 1, \dots, M-1$. We also note that:

$$\sin \frac{\pi m(2(-k-1)+1)}{2M} = \sin \frac{\pi m(-2k-1)}{2M} = \sin \frac{-\pi m(2k+1)}{2M} = -\sin \frac{\pi m(2k+1)}{2M} \quad (2.395)$$

Then:

$$\sum_{k=-M}^{-1} f(k) \sin \frac{\pi m(2k+1)}{2M} = \sum_{k=0}^{M-1} f(k) \sin \frac{\pi m(2k+1)}{2M} \quad (2.396)$$

We can see that easily by changing variable of summation in the sum on the left-hand side from k to $\tilde{k} \equiv -k-1$. The limits of summation will become from $M-1$ to 0 and the summand will not change, as $f(-\tilde{k}-1) = -f(\tilde{k})$ and at the same time the sine factor changes sign too. Replacing \tilde{k} then by k proves the equation. This means that

we may replace definition (2.392) with:

$$\hat{f}_{es}(m) \equiv -j \frac{1}{2M} \sum_{k=-M}^{M-1} f(k) \sin \frac{\pi m(2k+1)}{2M} \quad (2.397)$$

To derive the inverse transform we must solve this equation for $f(k)$. To achieve this we multiply both sides of the equation with $j \sin \frac{\pi m(2p+1)}{2M}$ and sum over m from $-M$ to $M-1$:

$$\underbrace{j \sum_{m=-M}^{M-1} \hat{f}_{es}(m) \sin \frac{\pi m(2p+1)}{2M}}_S = \frac{1}{2M} \sum_{m=-M}^{M-1} \sum_{k=-M}^{M-1} f(k) \sin \frac{\pi m(2k+1)}{2M} \sin \frac{\pi m(2p+1)}{2M} \quad (2.398)$$

On the right-hand side we replace the trigonometric functions by using formula $\sin \phi \equiv (e^{j\phi} - e^{-j\phi})/(2j)$, where ϕ is real. We also exchange the order of summations, observing that summation over m applies only to the kernel functions:

$$\begin{aligned} S &= -\frac{1}{8M} \sum_{k=-M}^{M-1} f(k) \sum_{m=-M}^{M-1} \left[e^{j \frac{\pi m(2k+1)}{2M}} - e^{-j \frac{\pi m(2k+1)}{2M}} \right] \left[e^{j \frac{\pi m(2p+1)}{2M}} - e^{-j \frac{\pi m(2p+1)}{2M}} \right] \\ &= -\frac{1}{8M} \sum_{k=-M}^{M-1} f(k) \sum_{m=-M}^{M-1} \left[e^{j \frac{\pi m(2k+2p+2)}{2M}} - e^{j \frac{\pi m(2k-2p)}{2M}} \right. \\ &\quad \left. - e^{j \frac{\pi m(-2k+2p)}{2M}} + e^{j \frac{\pi m(-2k-2p-2)}{2M}} \right] \\ &= -\frac{1}{8M} \sum_{k=-M}^{M-1} f(k) \sum_{m=-M}^{M-1} \left[e^{j \frac{\pi m(k+p+1)}{M}} - e^{j \frac{\pi m(k-p)}{M}} \right. \\ &\quad \left. - e^{j \frac{\pi m(-k+p)}{M}} + e^{j \frac{\pi m(-k-p-1)}{M}} \right] \end{aligned} \quad (2.399)$$

To compute the sums over m , we make use of (2.347), on page 142:

$$\begin{aligned} S &= -\frac{1}{8M} \sum_{k=-M}^{M-1} f(k) [2M\delta(k+p+1) - 2M\delta(k-p) \\ &\quad - 2M\delta(-k+p) + 2M\delta(-k-p-1)] \\ &= -\frac{1}{4} \sum_{k=-M}^{M-1} f(k) [\delta(k+p+1) - \delta(k-p) - \delta(-k+p) + \delta(-k-p-1)] \\ &= -\frac{1}{4} \sum_{k=-M}^{M-1} f(k) [2\delta(k+p+1) - 2\delta(k-p)] \\ &= -\frac{1}{2} \sum_{k=-M}^{M-1} f(k) [\delta(k+p+1) - \delta(k-p)] \end{aligned} \quad (2.400)$$

We used here the property of the delta function that $\delta(x) = \delta(-x)$. We note that, from all the terms in the sum, only two will survive, namely the one for $k = -p - 1$ and the one for $k = p$. Given that we defined $f(-k - 1) = -f(k)$, both these terms will be equal, ie $f(-p - 1) = -f(p)$, and so we shall obtain $S = 2f(p)$. This allows us to write the 1D inverse EDST as:

$$f(p) = j \sum_{m=-M}^{M-1} \hat{f}_{es}(m) \sin \frac{\pi m(2p+1)}{2M} \quad (2.401)$$

We split the negative from the non-negative indices in the above sum:

$$f(p) = j \left\{ \underbrace{\sum_{m=-M}^{-1} \hat{f}_{es}(m) \sin \frac{\pi m(2p+1)}{2M}}_S + \sum_{m=0}^{M-1} \hat{f}_{es}(m) \sin \frac{\pi m(2p+1)}{2M} \right\} \quad (2.402)$$

In the first sum we change the variable of summation from m to $\tilde{m} \equiv -m \Rightarrow m = -\tilde{m}$. The summation limits over \tilde{m} are then from M to 1, or from 1 to M :

$$S = \sum_{\tilde{m}=1}^M \hat{f}_{es}(-\tilde{m}) \sin \frac{\pi(-\tilde{m})(2p+1)}{2M} = \sum_{\tilde{m}=1}^M \hat{f}_{es}(\tilde{m}) \sin \frac{\pi \tilde{m}(2p+1)}{2M} \quad (2.403)$$

Here we made use of the fact that the sine function is antisymmetric with respect to \tilde{m} and so is $\hat{f}_{es}(\tilde{m})$ if we look at its definition. So their product is symmetric with respect to change of sign of \tilde{m} . Using (2.403) into (2.402), we may write:

$$f(p) = j \hat{f}_{es}(0) \sin \frac{\pi 0(2p+1)}{2M} + j \hat{f}_{es}(M) \sin \frac{\pi M(2p+1)}{2M} + j 2 \sum_{m=1}^{M-1} \hat{f}_{es}(m) \sin \frac{\pi m(2p+1)}{2M} \quad (2.404)$$

Finally, we may write for the inverse 1D EDST

$$f(p) = j \sum_{m=1}^M S(m) \hat{f}_{es}(m) \sin \frac{\pi m(2p+1)}{2M} \quad (2.405)$$

where $S(M) = 1$ and $S(m) = 2$ for $m \neq M$.

What is the inverse 2D even sine transform?

The inverse of equation (2.387) is

$$f(k, l) = - \sum_{m=1}^M \sum_{n=1}^N S(m) S(n) \hat{f}_{es}(m, n) \sin \frac{\pi m(2k+1)}{2M} \sin \frac{\pi n(2l+1)}{2N} \quad (2.406)$$

where $S(M) = 1$, $S(N) = 1$, and $S(m) = S(n) = 2$ for $m \neq M$, $n \neq N$.

What are the basis images in terms of which the even sine transform expands an image?

In equation (2.406), we may view function

$$V_m(k) \equiv jS(m) \sin \frac{\pi m(2k+1)}{2M} \quad (2.407)$$

as a function of k with parameter m . Then the basis functions, in terms of which an $M \times N$ image is expanded, are the vector outer products of vector functions $V_m(k)V_n^T(l)$, where $k = 0, \dots, M-1$ and $l = 0, \dots, N-1$. For fixed (m, n) , such a vector outer product creates an elementary image of size $M \times N$. Coefficient $\hat{f}_{es}(m, n)$ in (2.406) tells us the degree to which this elementary image is present in the original image $f(k, l)$.

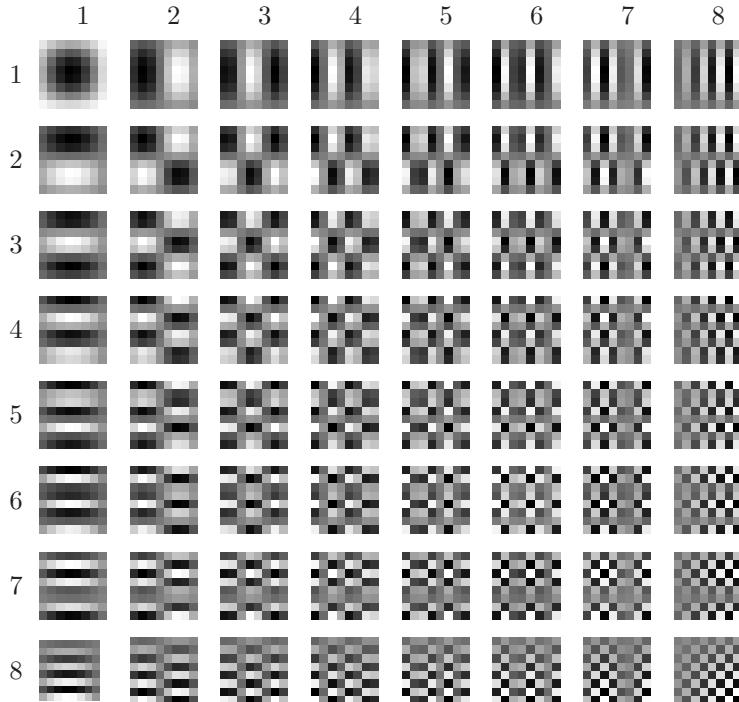


Figure 2.26: The basis images in terms of which EDST expands an 8×8 image. These basis images are the vector outer products of imaginary functions. The numbers on the left and at the top are indices m in (2.407), identifying which functions produced the corresponding basis image. Note that this basis does not include a flat image, ie there is no dc component. This means that, for best results, the mean of the image that is to be expanded in terms of these functions should be removed before the expansion.

Figure 2.26 shows the elementary images in terms of which any 8×8 image is expanded by the EDST. These images have been produced by setting $M = 8$ in (2.407) and allowing parameter m to take values $1, \dots, 8$. For every value of m , we have a different function $V_m(k)$. Each one of these functions is then sampled at values of $k = 0, \dots, 7$ to form an 8×1 vector. The plots of these eight functions are shown in figure 2.27.

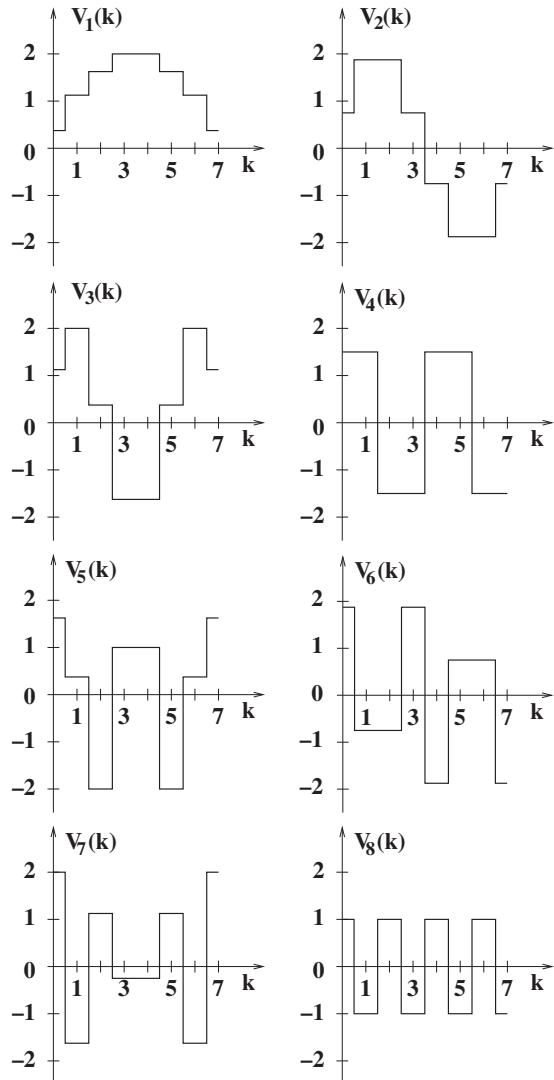


Figure 2.27: These are the discretised versions of imaginary functions $V_m(k)$, given by (2.407). The vector outer product of all possible combinations of them produce the basis images shown in figure 2.26, useful for the expansion of any 8×8 image. Note that the basis images are real because they are the products of the multiplications of two purely imaginary functions.

Figure 2.26 shows along the left and at the top which function $V_m(k)$ (identified by index m) was multiplied with which other function to create the corresponding elementary image. Each one of these elementary images is then scaled individually so that its grey values range from 1 to 255.

Example 2.73

Take the EDST transform of image (2.103), on page 69, and show the various approximations of it, by reconstructing it using only the first 1, 4, 9, etc elementary images in terms of which it is expanded.

Before we apply EDST, we remove the mean from all pixels of the image. After each reconstruction, and before we calculate the reconstruction error, we add the mean to all pixels. The eight images shown in figure 2.28 are the reconstructed images when, for the reconstruction, the basis images created from the first one, two, ..., eight functions $V_m(k)$ are used. For example, figure 2.28f has been reconstructed from the inverse EDST transform, by setting to 0 all elements of the transformation matrix that multiply the basis images in the bottom two rows and the two right-most columns in figure 2.26. The omitted basis images are those that are created from functions $V_7(k)$ and $V_8(k)$.

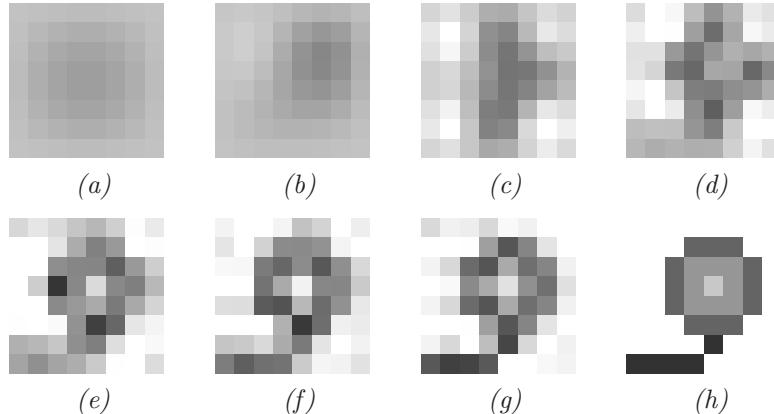


Figure 2.28: Successive approximations of the flower image by retaining an increasing number of basis functions, from $m = 1$ to $m = 8$, from top left to bottom right, respectively. For example, panel (b) was created by keeping only the coefficients that multiply the four basis images at the top left corner of figure 2.26. Values smaller than 0 and larger than 255 were truncated to 0 and 255, respectively, for displaying purposes.

The sum of the square errors for each reconstructed image is as follows.

Square error for image 2.28a:	341243
Square error for image 2.28b:	328602
Square error for image 2.28c:	259157
Square error for image 2.28d:	206923
Square error for image 2.28e:	153927
Square error for image 2.28f:	101778
Square error for image 2.28g:	55905
Square error for image 2.28h:	0

What happens if we do not remove the mean of the image before we compute its EDST?

The algorithm will work perfectly well even if we do not remove the mean of the image, but the approximation error, at least for the reconstructions that are based only on the first few components, will be very high. Figure 2.29 shows the successive reconstructions of the flower image without removing the mean before the transformation is taken. The various approximations of the image should be compared with those shown in figure 2.28. The corresponding approximation errors are:

Square error for image 2.29a:	1550091
Square error for image 2.29b:	1537450
Square error for image 2.29c:	749053
Square error for image 2.29d:	696820
Square error for image 2.29e:	342055
Square error for image 2.29f:	289906
Square error for image 2.29g:	55905
Square error for image 2.29h:	0

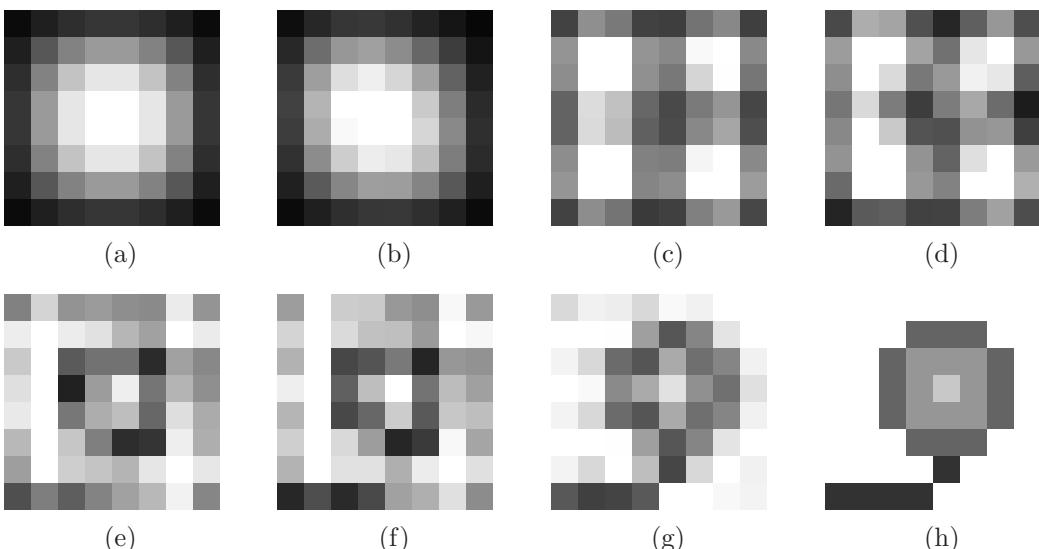


Figure 2.29: Successive approximations of the flower image by retaining an increasing number of basis functions, from $m = 1$ to $m = 8$, from top left to bottom right, respectively. In this case the mean value of the image was not removed before the transformation was computed.

2.7 The odd antisymmetric discrete sine transform (ODST)

What is the odd antisymmetric discrete sine transform?

Assume that we have an $M \times N$ image f , change sign and reflect it about its left and top border and also insert a row and a column of 0s along the reflection lines, so that we have a $(2M + 1) \times (2N + 1)$ image. The DFT of the $(2M + 1) \times (2N + 1)$ image will be real (see example 2.55) and given by:

$$-\frac{4}{(2M+1)(2N+1)} \sum_{\tilde{k}=1}^M \sum_{\tilde{l}=1}^N f(\tilde{k}, \tilde{l}) \sin \frac{2\pi m \tilde{k}}{2M+1} \sin \frac{2\pi n \tilde{l}}{2N+1} \quad (2.408)$$

Note that here indices \tilde{k} and \tilde{l} are not the indices of the original image, which were running from 0 to $M - 1$ and $N - 1$, respectively. Because of the insertion of the row and column of 0s, the indices have been shifted by 1. In order to retain the original indices, we define the **odd discrete sine transform (ODST)** of the original image as:

$$\hat{f}_{os}(m, n) \equiv -\frac{4}{(2M+1)(2N+1)} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f(k, l) \sin \frac{2\pi m(k+1)}{2M+1} \sin \frac{2\pi n(l+1)}{2N+1} \quad (2.409)$$

Example 2.74

Compute the odd antisymmetric sine transform of image

$$g = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 2 & 2 & 0 \end{pmatrix} \quad (2.410)$$

by taking the DFT of the corresponding image of size 9×9 .

We start by creating first the corresponding large image of size 9×9 :

$$\tilde{g} = \begin{pmatrix} 0 & 2 & 2 & 1 & 0 & -1 & -2 & -2 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & 0 & -2 & -2 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & 0 & -1 & -2 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -2 & -1 & 0 & 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & -2 & -2 & -1 & 0 & 1 & 2 & 2 & 0 \end{pmatrix} \quad (2.411)$$

To take the DFT of this image we multiply it from the left with the appropriate matrix U and from the right with its transpose. We create this matrix using definition (2.286), on page 127. Here $J = 4$ and the elements of matrix U are given by $\frac{1}{9}e^{-j2\pi mk/9}$ where k takes values $-4, -3, -2, -1, 0, 1, 2, 3, 4$ along each row and m takes values $0, 1, 2, 3, 4, 5, 6, 7, 8$ along each column.

$$U_{os} = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ e^{j\frac{8\pi}{9}} & e^{j\frac{6\pi}{9}} & e^{j\frac{4\pi}{9}} & e^{j\frac{2\pi}{9}} & 1 & e^{-j\frac{2\pi}{9}} & e^{-j\frac{4\pi}{9}} & e^{-j\frac{6\pi}{9}} & e^{-j\frac{8\pi}{9}} \\ e^{j\frac{16\pi}{9}} & e^{j\frac{12\pi}{9}} & e^{j\frac{8\pi}{9}} & e^{j\frac{4\pi}{9}} & 1 & e^{-j\frac{4\pi}{9}} & e^{-j\frac{8\pi}{9}} & e^{-j\frac{12\pi}{9}} & e^{-j\frac{16\pi}{9}} \\ e^{j\frac{6\pi}{9}} & e^{j\frac{18\pi}{9}} & e^{j\frac{12\pi}{9}} & e^{j\frac{6\pi}{9}} & 1 & e^{-j\frac{6\pi}{9}} & e^{-j\frac{12\pi}{9}} & e^{-j\frac{18\pi}{9}} & e^{-j\frac{6\pi}{9}} \\ e^{j\frac{14\pi}{9}} & e^{j\frac{8\pi}{9}} & e^{j\frac{16\pi}{9}} & e^{j\frac{8\pi}{9}} & 1 & e^{-j\frac{8\pi}{9}} & e^{-j\frac{16\pi}{9}} & e^{-j\frac{6\pi}{9}} & e^{-j\frac{14\pi}{9}} \\ e^{j\frac{4\pi}{9}} & e^{j\frac{12\pi}{9}} & e^{j\frac{2\pi}{9}} & e^{j\frac{10\pi}{9}} & 1 & e^{-j\frac{10\pi}{9}} & e^{-j\frac{2\pi}{9}} & e^{-j\frac{12\pi}{9}} & e^{-j\frac{4\pi}{9}} \\ e^{j\frac{12\pi}{9}} & e^{j\frac{18\pi}{9}} & e^{j\frac{6\pi}{9}} & e^{j\frac{12\pi}{9}} & 1 & e^{-j\frac{12\pi}{9}} & e^{-j\frac{6\pi}{9}} & e^{-j\frac{18\pi}{9}} & e^{-j\frac{12\pi}{9}} \\ e^{j\frac{2\pi}{9}} & e^{j\frac{6\pi}{9}} & e^{j\frac{10\pi}{9}} & e^{j\frac{14\pi}{9}} & 1 & e^{-j\frac{14\pi}{9}} & e^{-j\frac{10\pi}{9}} & e^{-j\frac{6\pi}{9}} & e^{-j\frac{2\pi}{9}} \\ e^{j\frac{10\pi}{9}} & e^{j\frac{12\pi}{9}} & e^{j\frac{14\pi}{9}} & e^{j\frac{16\pi}{9}} & 1 & e^{-j\frac{16\pi}{9}} & e^{-j\frac{14\pi}{9}} & e^{-j\frac{12\pi}{9}} & e^{-j\frac{10\pi}{9}} \end{pmatrix} \quad (2.412)$$

$$\hat{G}_{os} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.302 & 0.050 & -0.102 & 0.041 & -0.041 & 0.102 & -0.050 & 0.302 \\ 0 & 0.087 & -0.198 & 0.032 & 0.103 & -0.103 & -0.032 & 0.198 & -0.087 \\ 0 & -0.285 & 0.001 & 0.074 & 0.063 & -0.063 & -0.074 & -0.001 & 0.285 \\ 0 & 0.078 & 0.140 & -0.089 & 0.092 & -0.092 & 0.089 & -0.140 & -0.078 \\ 0 & -0.078 & -0.140 & 0.089 & -0.092 & 0.092 & -0.089 & 0.140 & 0.078 \\ 0 & 0.285 & -0.001 & -0.074 & -0.063 & 0.063 & 0.074 & 0.001 & -0.285 \\ 0 & -0.087 & 0.198 & -0.032 & -0.103 & 0.103 & 0.032 & -0.198 & 0.087 \\ 0 & 0.302 & -0.050 & 0.102 & -0.041 & 0.041 & -0.102 & 0.050 & -0.302 \end{pmatrix} \quad (2.413)$$

Example 2.75

Compute the $(1, 2)$ element of the odd antisymmetric sine transform of image (2.410) by using formula (2.408). Compare your answer with that of example 2.74.

Applying the formula for $m = 1$, $n = 2$ and $M = N = 4$, we obtain:

$$\hat{g}_{os}(1, 2) = -\frac{4}{81} \sum_{k=0}^3 \sum_{l=0}^3 g(k, l) \sin \frac{2\pi(k+1)}{9} \sin \frac{2\pi 2(l+1)}{9} \quad (2.414)$$

Or:

$$\begin{aligned}\hat{g}_{os}(1, 2) = & -\frac{4}{81} \left\{ g(0, 0) \sin \frac{2\pi}{9} \sin \frac{4\pi}{9} + g(0, 1) \sin \frac{2\pi}{9} \sin \frac{8\pi}{9} \right. \\ & + g(0, 3) \sin \frac{2\pi}{9} \sin \frac{16\pi}{9} + g(1, 0) \sin \frac{4\pi}{9} \sin \frac{4\pi}{9} \\ & + g(2, 2) \sin \frac{6\pi}{9} \sin \frac{12\pi}{9} + g(2, 3) \sin \frac{6\pi}{9} \sin \frac{16\pi}{9} \\ & + g(3, 1) \sin \frac{8\pi}{9} \sin \frac{8\pi}{9} + g(3, 1) \sin \frac{8\pi}{9} \sin \frac{8\pi}{9} \\ & \left. + g(3, 2) \sin \frac{8\pi}{9} \sin \frac{12\pi}{9} \right\} \end{aligned} \quad (2.415)$$

Here we omitted terms for which $g(k, l) = 0$. Substituting the values of $g(k, l)$ in (2.415) and performing the calculation, we deduce that $\hat{g}_{os}(1, 2) = 0.0497$.

This is in agreement with the values of $\hat{G}_{os}(1, 2) = 0.050$ we deduced in example 2.74.

Example B2.76

The odd antisymmetric sine transform of an M -sample long signal $f(k)$, defined for values $k = 1, \dots, M$, is defined as:

$$\hat{f}_{os}(m) \equiv -j \frac{2}{2M+1} \sum_{k=0}^{M-1} f(k) \sin \frac{2\pi m(k+1)}{2M+1} \quad (2.416)$$

Identify the period of $\hat{f}_{os}(m)$.

The period of a function is the smallest number X for which $\hat{f}_{os}(m+X) = \hat{f}_{os}(m)$, for all m .

Using definition (2.416), we have:

$$\begin{aligned}\hat{f}_{os}(m+X) &= -j \frac{2}{2M+1} \sum_{k=0}^{M-1} f(k) \sin \frac{2\pi(m+X)(k+1)}{2M+1} \\ &= -j \frac{2}{2M+1} \sum_{k=0}^{M-1} f(k) \sin \left(\underbrace{\frac{2\pi m(k+1)}{2M+1}}_{\phi} + \frac{2\pi X(k+1)}{2M+1} \right) \end{aligned} \quad (2.417)$$

In order to have $\hat{f}_{os}(m+X) = \hat{f}_{os}(m)$, we must have:

$$\sin \left(\phi + \frac{2\pi X(k+1)}{2M+1} \right) = \sin \phi \quad (2.418)$$

This is only true if $2\pi X(k+1)/(2M+1)$ is an integer multiple of 2π . The first number for which this is guaranteed is for $X = 2M + 1$. So, $\hat{f}_{os}(m)$ is periodic with period $2M + 1$.

Example B2.77

You are given a 5-sample long signal with the following values: $f(0) = 0$, $f(1) = 1$, $f(2) = 2$, $f(3) = 3$ and $f(4) = 4$. Compute its ODST $\hat{f}_{os}(m)$ and plot both the extended signal and its ODST, for 55 consecutive samples.

The extended signal is $-4, -3, -2, -1, 0, 0, 0, 1, 2, 3, 4$. DFT sees this signal repeated ad infinitum with period 11. Since $M = 5$, according to the result of example 2.76, the ODST of the original data is periodic with period $2M + 1 = 11$ as well. The values of $\hat{f}_{os}(m)$ for one period are:

$$(-1.14, 0.9, -0.46, 0.49, -0.4, 0.4, -0.49, 0.46, -0.9, 1.14, 0) \quad (2.419)$$

Figure 2.30 shows the plots of 55 consecutive samples of the extended signal and the ODST of the original data.

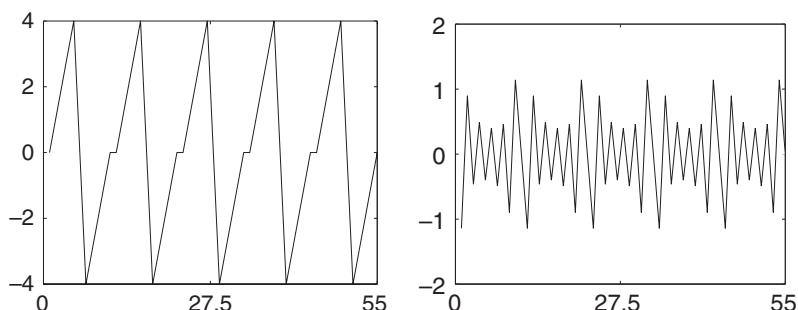


Figure 2.30: On the left, 55 consecutive samples of the extended signal seen by the DFT. On the right, five periods of the ODST of the original 5-sample long signal.

Box 2.12. Derivation of the inverse 1D odd discrete sine transform

To derive the inverse ODST we shall make use of (2.408), where indices \tilde{k} and \tilde{l} refer to the indices of the enlarged image, and they are related to the indices of the original image by being increased by 1 in relation to them.

The 1D version of ODST is then:

$$\hat{f}_{os}(m) \equiv -j \frac{2}{2M+1} \sum_{\tilde{k}=1}^M f(\tilde{k}) \sin \frac{2\pi m \tilde{k}}{2M+1} \quad (2.420)$$

(This definition is equivalent to (2.416), remembering that $\tilde{k} = k + 1$.)

Let us define $f(-\tilde{k}) \equiv -f(\tilde{k})$ for all values of $\tilde{k} = 1, \dots, M$. As $\sin 0 = 0$, the definition of $f(0)$ is immaterial. This means that we may replace definition (2.420) with:

$$\hat{f}_{os}(m) \equiv -j \frac{1}{2(2M+1)} \sum_{\tilde{k}=-M}^M f(\tilde{k}) \sin \frac{2\pi m \tilde{k}}{2M+1} \quad (2.421)$$

To derive the inverse transform we must solve this equation for $f(\tilde{k})$. To achieve this, we multiply both sides of the equation with $j \sin \frac{2\pi mp}{2M+1}$ and sum over m from $-M$ to M :

$$\underbrace{j \sum_{m=-M}^M \hat{f}_{os}(m) \sin \frac{2\pi mp}{2M+1}}_S = \frac{1}{2(2M+1)} \sum_{m=-M}^M \sum_{\tilde{k}=-M}^M f(\tilde{k}) \sin \frac{2\pi m \tilde{k}}{2M+1} \sin \frac{2\pi mp}{2M+1} \quad (2.422)$$

On the right-hand side we replace the trigonometric functions by using formula $\sin \phi \equiv (e^{j\phi} - e^{-j\phi}) / (2j)$, where ϕ is real. We also exchange the order of summations, observing that summation over m applies only to the kernel functions:

$$\begin{aligned} S &= -\frac{1}{8(2M+1)} \sum_{\tilde{k}=-M}^M f(\tilde{k}) \sum_{m=-M}^M \left[e^{j \frac{2\pi m \tilde{k}}{2M+1}} - e^{-j \frac{2\pi m \tilde{k}}{2M+1}} \right] \left[e^{j \frac{2\pi m p}{2M+1}} - e^{-j \frac{2\pi m p}{2M+1}} \right] \\ &= -\frac{1}{8(2M+1)} \sum_{\tilde{k}=-M}^M f(\tilde{k}) \sum_{m=-M}^M \left[e^{j \frac{2\pi m (\tilde{k}+p)}{2M+1}} - e^{j \frac{2\pi m (\tilde{k}-p)}{2M+1}} \right. \\ &\quad \left. - e^{j \frac{2\pi m (-\tilde{k}+p)}{2M+1}} + e^{j \frac{2\pi m (-\tilde{k}-p)}{2M+1}} \right] \end{aligned} \quad (2.423)$$

To compute the sums over m , we apply formula (2.164), on page 95, for $S = 2M + 1$:

$$\begin{aligned}
S &= -\frac{1}{8(2M+1)} \sum_{\tilde{k}=-M}^M f(\tilde{k}) \left[(2M+1)\delta(\tilde{k}+p) - (2M+1)\delta(\tilde{k}-p) \right. \\
&\quad \left. - (2M+1)\delta(-\tilde{k}+p) + (2M+1)\delta(-\tilde{k}-p) \right] \\
&= -\frac{1}{8} \sum_{\tilde{k}=-M}^M f(\tilde{k}) \left[\delta(\tilde{k}+p) - \delta(\tilde{k}-p) - \delta(-\tilde{k}+p) + \delta(-\tilde{k}-p) \right] \\
&= -\frac{1}{8} \sum_{\tilde{k}=-M}^M f(\tilde{k}) [2\delta(\tilde{k}+p) - 2\delta(\tilde{k}-p)] \\
&= -\frac{1}{4} \sum_{\tilde{k}=-M}^M f(\tilde{k}) [\delta(\tilde{k}+p) - \delta(\tilde{k}-p)]
\end{aligned} \tag{2.424}$$

We used here the property of the delta function that $\delta(x) = \delta(-x)$. We note that, from all the terms in the sum, only two will survive, namely the one for $\tilde{k} = -p$ and the one for $\tilde{k} = p$. Given that we defined $f(-\tilde{k}) = -f(\tilde{k})$, both these terms will be equal, ie $f(-p) = -f(p)$, and so we shall obtain $S = f(p)$. This allows us to write the 1D inverse ODST as:

$$f(p) = j2 \sum_{m=-M}^M \hat{f}_{os}(m) \sin \frac{2\pi mp}{2M+1} \quad \text{for } p = 1, 2, \dots, M \tag{2.425}$$

As the sine function is antisymmetric with respect to m and the $\hat{f}_{os}(m)$ can also be seen to be antisymmetric from its definition (2.416), we may conclude that their product is symmetric, and so we may write:

$$f(p) = j4 \sum_{m=1}^M \hat{f}_{os}(m) \sin \frac{2\pi mp}{2M+1} \quad \text{for } p = 1, 2, \dots, M \tag{2.426}$$

To go back to the original indices, we remember that p refers to the indices of the enlarged image, and so it is shifted by 1 in relation to the original data. So, in terms of the original indices, the inverse ODST is:

$$f(k) = j4 \sum_{m=1}^M \hat{f}_{os}(m) \sin \frac{2\pi m(k+1)}{2M+1} \quad \text{for } k = 0, 1, \dots, M-1 \tag{2.427}$$

What is the inverse 2D odd sine transform?

The inverse of equation (2.408) is:

$$f(k, l) = -16 \sum_{m=1}^M \sum_{n=1}^N \hat{f}_{os}(m, n) \sin \frac{2\pi m(k+1)}{2M+1} \sin \frac{2\pi n(l+1)}{2N+1} \tag{2.428}$$

What are the basis images in terms of which the odd sine transform expands an image?

In equation (2.428), we may view function

$$W_m(k) \equiv j4 \sin \frac{2\pi m(k+1)}{2M+1} \quad (2.429)$$

as a function of k with parameter m . Then the basis functions, in terms of which an $M \times N$ image is expanded, are the vector outer products of vector functions $W_m(k)W_n^T(l)$, where $k = 0, \dots, M-1$ and $l = 0, \dots, N-1$. For fixed (m, n) such a vector outer product creates an elementary image of size $M \times N$. Coefficient $\hat{f}_{os}(m, n)$ in (2.428) tells us the degree to which this elementary image is present in the original image $f(k, l)$.

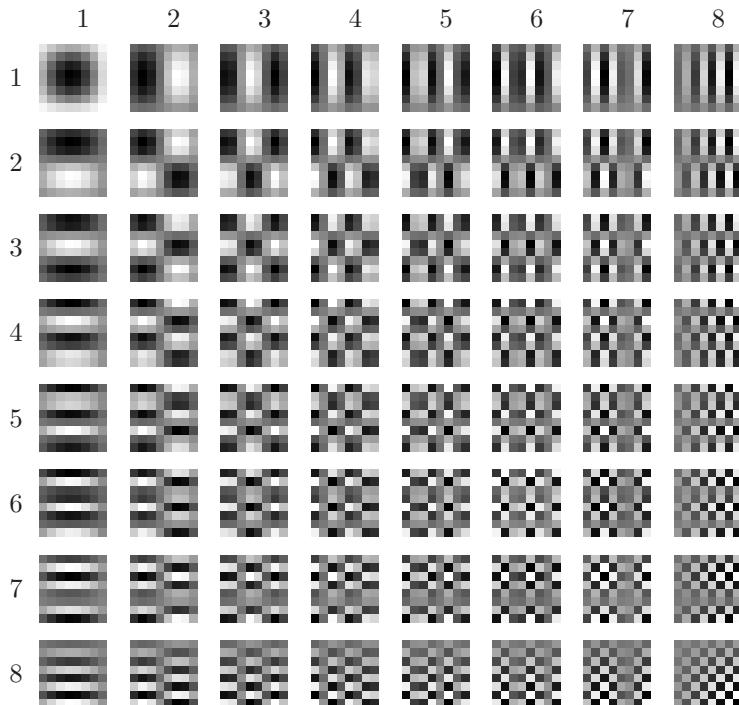


Figure 2.31: Basis images created as vector outer products of functions $W_m(k)$, defined by (2.429). The indices m and n of the functions, that are multiplied, $W_m W_n^T$, to form each image, are given on the left and at the top, respectively.

Figure 2.31 shows the elementary images in terms of which any 8×8 image is expanded by the ODST. These images have been produced by setting $M = 8$ in (2.429) and allowing parameter m to take values $1, \dots, 8$. For every value of m we have a different function $W_m(k)$. Each one of these functions is then sampled at values of $k = 0, \dots, 7$ to form an 8×1 vector. The plots of these eight functions are shown in figure 2.32.

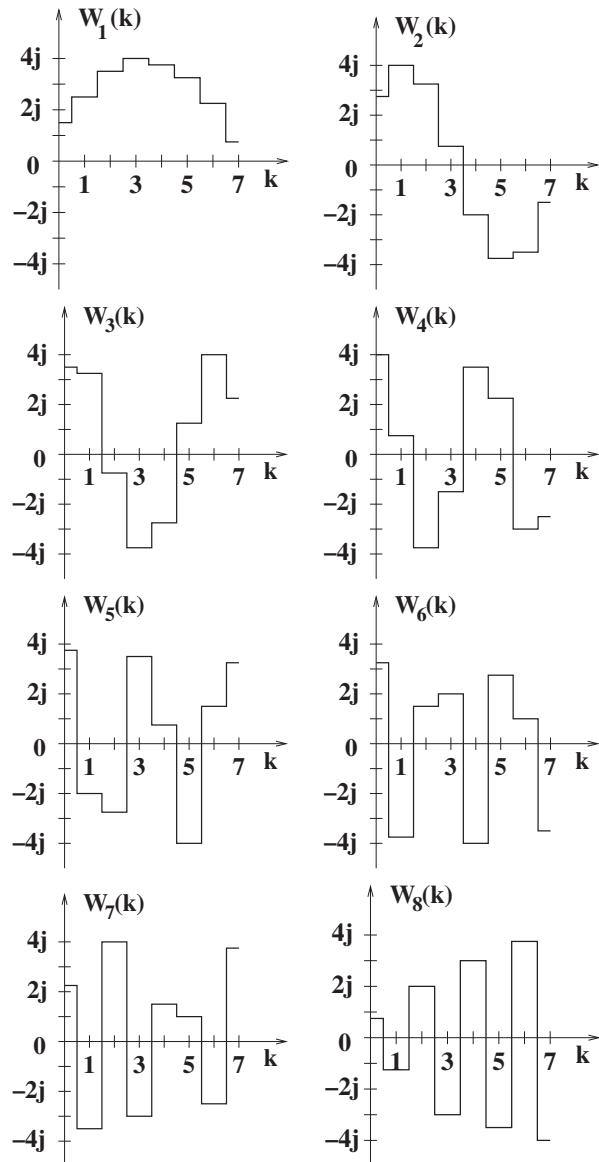


Figure 2.32: The imaginary functions $W_m(k)$ defined by (2.429), used to construct the basis images of size 8×8 shown in figure 2.31.

Figure 2.31 shows along the left and at the top which function $W_m(k)$ (identified by index m) was multiplied with which other function to create the corresponding elementary image. Each one of these elementary images is then scaled individually, so that its grey values range from 1 to 255.

Example 2.78

Take the ODST transform of image (2.103), on page 69, and show the various approximations of it, when for the reconstruction the basis images are created from the first one, two, ..., eight functions $W_m(k)$.

We first remove the mean value of the image from the values of all pixels. Then we perform the transformation and the reconstructions. Before we display the reconstructions, we add the mean value to all pixels. The results are shown in figure 2.33.

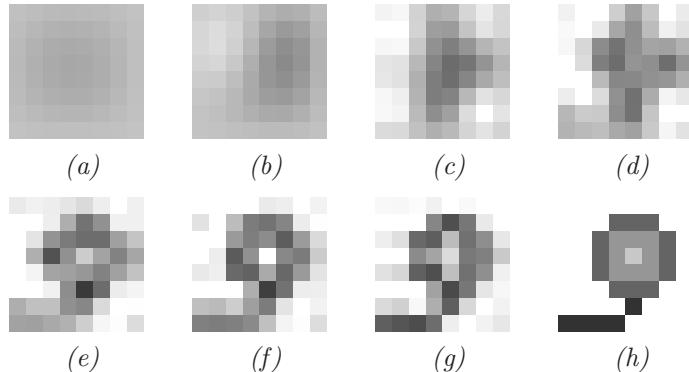


Figure 2.33: Successive approximations of the flower image using the ODST transform. The sum of the squared errors for each reconstructed image is as follows.

Square error for image 2.33a: 350896

Square error for image 2.33b: 326264

Square error for image 2.33c: 254763

Square error for image 2.33d: 205803

Square error for image 2.33e: 159056

Square error for image 2.33f: 109829

Square error for image 2.33g: 67374

Square error for image 2.33h: 0

	0	1	2	3	4	5	6
SVD	230033	118412	46673	11882			
Haar	366394	356192	291740	222550	192518	174625	141100
Walsh	366394	356190	262206	222550	148029	92078	55905
DFT	366394	285895	234539	189508	141481	119612	71908
EDCT	366394	338683	216608	173305	104094	49179	35662
ODCT	368946	342507	221297	175046	96924	55351	39293
EDST	341243	328602	259157	206923	153927	101778	55905
ODST	350896	326264	254763	205803	159056	109829	67374

Table 2.2: The errors of the successive approximations of image (2.103) by the various transforms of this chapter. The numbers at the top indicate the order of the approximation.

What is the “take home” message of this chapter?

This chapter presented the linear, unitary and separable transforms we apply to images. These transforms analyse each image into a linear superposition of elementary basis images. Usually these elementary images are arranged in increasing order of structure (detail). This allows us to represent an image with as much detail as we wish, by using only as many of these basis images as we like, starting from the first one. The optimal way to do that is to use as basis images those that are defined by the image itself, the eigenimages of the image (SVD). This, however, is not very efficient, as our basis images change from one image to the next.

Alternatively, some bases of predefined images may be created with the help of orthonormal sets of functions. These bases try to capture the basic characteristics of all images. Once the basis used has been agreed, images may be communicated between different agents by simply transmitting the weights with which each of the basis images has to be multiplied before all of them are added to create the original image. The first one of these basis images is usually a uniform image, except in the case of the sine transform. The form of the rest of the images of each basis depends on the orthonormal set of functions used to generate them. As these basic images are used to represent a large number of images, more of them are needed to represent a single image than if the eigenimages of the image itself were used for its representation. However, the gain in the number of bits used comes from the fact that the basis images are pre-agreed and they do not need to be stored or transmitted with each image separately.

The bases constructed with the help of orthonormal sets of discrete functions (eg Haar and Walsh) are easy to implement in hardware. However, the basis constructed with the help of the orthonormal set of complex exponential functions is by far the most popular. The representation of an image in terms of it is called discrete Fourier transform. Its popularity stems from the fact that manipulation of the weights with which the basis images are superimposed to form the original image, for the purpose of omitting, for example, certain details in the image, can be achieved by manipulating the image itself with the help of a simple convolution. By-products of the discrete Fourier transform are the sine and cosine transforms, which artificially enlarge the image so its discrete Fourier transform becomes real.

Table 2.2 lists the errors of the reconstructions of image (2.103) we saw in this chapter. EDCT has a remarkably good reconstruction and that is why it is used in JPEG. SVD has the least error, but remember that the basis images have also to be transmitted when this approximation is used. EDST and ODST also require the dc component to be transmitted in addition to the coefficients of the expansion. Finally, DFT requires two numbers to be transmitted per coefficient retained, as it is a complex transform and so the coefficients it produces have real and imaginary parts. So, the *0th* approximation for this particular example, requires 16 real numbers for SVD (two 8×1 real vectors), only 1 number for Haar, Walsh, DFT, EDCT and ODCT, and 2 for EDST and ODST. The *1st* approximation requires 32 real numbers for SVD, only 4 real numbers for Haar, Walsh, EDCT and ODCT, 7 real numbers for DFT (the real valued dc component, plus 3 complex numbers), and 5 real numbers for EDST and ODST (the dc component, plus the 4 coefficients for the first 4 basis images). The *2nd* approximation requires 48 real numbers for SVD, only 9 real numbers for Haar, Walsh, EDCT and ODCT, 17 real numbers for DFT and 10 real numbers for EDST and ODST. (Remember that the *2nd* order approximation uses the 9 basis images at the top left corner of the images shown in figures 2.4, 2.6, 2.11, 2.12, 2.19, 2.22, 2.26 and 2.31.)

Chapter 3

Statistical Description of Images

What is this chapter about?

This chapter provides the necessary background for the statistical description of images from the signal processing point of view. It treats each image as the outcome of some random process, and it shows how we can reason about images using concepts from probability and statistics, in order to express a whole collection of images as composites of some basic images. In some cases it treats an image as the only available version of a large collection of similar (in some sense) images and reasons on the statistical properties of the whole collection.

Why do we need the statistical description of images?

In the previous chapter we saw how we may construct bases of elementary images in terms of which any image of the same size may be expressed as a linear combination of them. Examples of such bases are the Fourier, Walsh and Haar bases of elementary images. These bases are universal but not optimal in any sense, when the expansion of an image in terms of any of them is truncated. They simply allow the user to approximate an image by omitting certain frequencies (Fourier), or certain structural details (Walsh) or even reconstruct preferentially certain parts of the image (Haar). We also saw how to construct basis images that are optimal for a specific image. This led to the Singular Value Decomposition of an image which allows the approximation of an image in the least square error sense. Between these two extremes of universal bases, appropriate for all images, and very specific bases, appropriate for one image only, we may wish to consider bases of elementary images that might be optimal for a specific collection of images. For example, in various applications, we often have to deal with sets of images of a certain type, like X-ray images, traffic scene images, etc. Each image in the set may be different from all the others, but at the same time all images may share certain common characteristics. We need the statistical description of sets of images so that we capture these common characteristics and use them in order to represent an image with fewer bits and reconstruct it with the minimum error “on average”. In such a case, a pixel in an image may be thought of as taking a value randomly selected from the set of values that appear in the same grid location over all images in the set. A pixel, therefore, becomes a **random variable**. As we have many pixels arranged in the spatial coordinates of an image, an image becomes a **random field**.

3.1 Random fields

What is a random field?

A **random field** is a spatial function that assigns a random variable to each spatial position.

What is a random variable?

A **random variable** is the value we assign to the outcome of a random experiment.

What is a random experiment?

It is a process that produces an unpredictable outcome, from a set of possible outcomes. Throwing a die is a random experiment. Drawing the lottery is a random experiment.

How do we perform a random experiment with computers?

We do not. We produce “random variables” which are not truly random, and that is why they are better described as **pseudorandom**. They are produced by applying a sequence of formulae designed to produce different sequences of numbers when initialised by different numbers, called **seeds**. These sequences of produced numbers are usually repeated with a very long cycle. For examples, they may be repeated after 2^{32} numbers have been produced. Usually the number used as seed is the time from the clock. Normally, the user has the option to specify the seed so the user may at a later stage reproduce the pseudorandom sequence of numbers in order to debug or investigate an algorithm that depends on them.

How do we describe random variables?

Random variables are described in terms of their **distribution functions** which in turn are defined in terms of the **probability** of an **event** happening. An event is a collection of outcomes of a random experiment.

Example 3.1

Consider the cube shown in figure 3.1. Consider that you perform the following random experiment: you throw it in the air and let it land on the ground. The outcome of this random experiment is the particular side of the cube that faces up when the cube rests on the ground. We agree to associate with each possible outcome the following values of variable x :

- | | | |
|----------|-------------------|----------|
| Outcome: | Side ABCH face up | $x = 28$ |
| Outcome: | Side BCDG face up | $x = 23$ |
| Outcome: | Side GDEF face up | $x = 18$ |
| Outcome: | Side EFAH face up | $x = 25$ |
| Outcome: | Side EDCH face up | $x = 14$ |
| Outcome: | Side FGBA face up | $x = 18$ |

Variable x is random because it takes values according to the outcome of a random experiment. What is the probability of x taking values in the set $\{14, 18\}$?

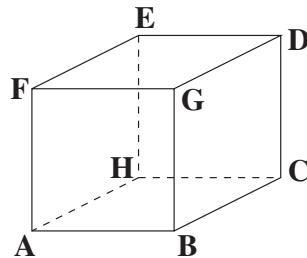


Figure 3.1: A solid cube made up from uniformly dense material.

Assuming that the cube is made from material with uniform density, all sides are equally likely to end up being face up. This means that each one of them has one in six chances to end up in that position. Since number 18 is associated with two faces and number 14 with one, the chances to get an 18 or a 14 are three in six, ie the probability of x getting value either 14 or 18 is 0.5.

What is the probability of an event?

The **probability** of an event happening is a *non-negative* number which has the following properties:

- (i) the probability of the event, which includes all possible outcomes of the experiment, is 1;
- (ii) the probability of two events which do not have any common outcomes is the sum of the probabilities of the two events separately.

Example 3.2

In the random experiment of example 3.1, what is the event that includes all possible outcomes?

In terms of which side lands face up, the event that includes all possible outcomes is the set $\{ABCH, BCDG, GDEF, EFAH, EDCH, FGBA\}$. In terms of values of random variable x , it is $\{28, 23, 18, 25, 14\}$.

Example 3.3

In the random experiment of example 3.1, what is the probability of events $\{14, 18\}$ and $\{23\}$?

In example 3.1 we worked out that the probability of event $\{14, 18\}$ is 0.5. The probability of event $\{23\}$ is one in six. Since these two events do not have any common outcomes, the probability of either one or the other happening is the sum of the probabilities of each one happening individually. So, it is $1/2 + 1/6 = 4/6 = 2/3$.

What is the distribution function of a random variable?

The distribution function of a random variable f is a function which tells us how likely it is for f to be less than the argument of the function:

$$\underbrace{P_f(z)}_{\substack{\text{Distribution} \\ \text{function of } f}} = \underbrace{\mathcal{P}}_{\substack{\text{probability}}} \{ \underbrace{f}_{\substack{\text{random} \\ \text{variable}}} < \underbrace{z}_{\substack{\text{a number}}} \} \quad (3.1)$$

Clearly, $P_f(-\infty) = 0$ and $P_f(+\infty) = 1$.

Example 3.4

If $z_1 \leq z_2$, show that $P_f(z_1) \leq P_f(z_2)$.

Assume that A is the event (ie the set of outcomes) which makes $f < z_1$ and B is the event which makes $f < z_2$. Since $z_1 \leq z_2$, $A \subseteq B \Rightarrow B = (B - A) \cup A$. Events $(B - A)$ and A do not have common outcomes (see figure 3.2).

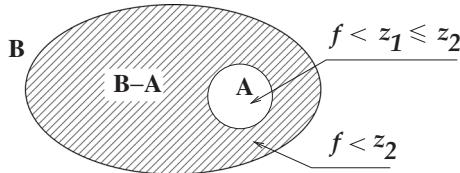


Figure 3.2: The representation of events A and B as sets.

Then by property (ii) in the definition of the probability of an event:

$$\begin{aligned} \mathcal{P}(B) &= \mathcal{P}(B - A) + \mathcal{P}(A) \Rightarrow \\ P_f(z_2) &= \underbrace{\mathcal{P}(B - A)}_{\substack{\text{a non-negative number}}} + P_f(z_1) \Rightarrow P_f(z_2) \geq P_f(z_1) \end{aligned} \quad (3.2)$$

Example 3.5

Show that:

$$\mathcal{P}(z_1 \leq f < z_2) = P_f(z_2) - P_f(z_1) \quad (3.3)$$

According to the notation of example 3.4, $z_1 \leq f < z_2$ when the outcome of the random experiment belongs to $B - A$ (the shaded area in figure 3.2); ie $\mathcal{P}(z_1 \leq f < z_2) = P_f(B - A)$. Since $B = (B - A) \cup A$, $P_f(B - A) = P_f(B) - P_f(A)$ and (3.3) follows.

What is the probability of a random variable taking a specific value?

If the random variable takes values from the set of real numbers, it has zero probability of taking a specific value. (This can be seen if in (3.3) we set $z_1 = z_2$.) However, it may have nonzero probability of taking a value within an infinitesimally small range of values. This is expressed by its **probability density function**.

What is the probability density function of a random variable?

The derivative of the distribution function of a random variable is called the **probability density function** of the random variable:

$$p_f(z) \equiv \frac{dP_f(z)}{dz} \quad (3.4)$$

The **expected or mean value** of the random variable f is defined as

$$\mu_f \equiv E\{f\} \equiv \int_{-\infty}^{+\infty} z p_f(z) dz \quad (3.5)$$

and the **variance** as:

$$\sigma_f^2 \equiv E\{(f - \mu_f)^2\} \equiv \int_{-\infty}^{+\infty} (z - \mu_f)^2 p_f(z) dz \quad (3.6)$$

The **standard deviation** is the positive square root of the variance, ie σ_f .

Example 3.6

Starting from definition 3.4 and using the properties of $P_f(z)$, prove that $\int_{-\infty}^{+\infty} p_f(z) dz = 1$

$$\int_{-\infty}^{+\infty} p_f(z) dz = \int_{-\infty}^{+\infty} \frac{dP_f(z)}{dz} dz = P_f(z)|_{-\infty}^{+\infty} = P_f(+\infty) - P_f(-\infty) = 1 - 0 = 1 \quad (3.7)$$

Example 3.7

The distribution function of a random variable f is given by

$$P_f(z) = \frac{1}{1 + e^{-z}} \quad (3.8)$$

Compute the corresponding probability density function and plot both functions as functions of z .

The probability density function $p_f(z)$ of z is given by the first derivative of its distribution function:

$$\begin{aligned} p_f(z) \equiv \frac{dP_f(z)}{dz} &= \frac{e^{-z}}{(1 + e^{-z})^2} \\ &= \frac{e^{-z}}{(1 + e^{-z})(1 + e^{-z})} \\ &= \frac{1}{(1 + e^z)(1 + e^{-z})} \\ &= \frac{1}{1 + e^z + e^{-z} + 1} \\ &= \frac{1}{2 + 2 \cosh z} \\ &= \frac{1}{2(1 + \cosh z)} \end{aligned} \quad (3.9)$$

Here we made use of the definition of the hyperbolic cosine $\cosh z \equiv (e^z + e^{-z})/2$. The plots of functions (3.8) and (3.9) are shown in figure 3.3.

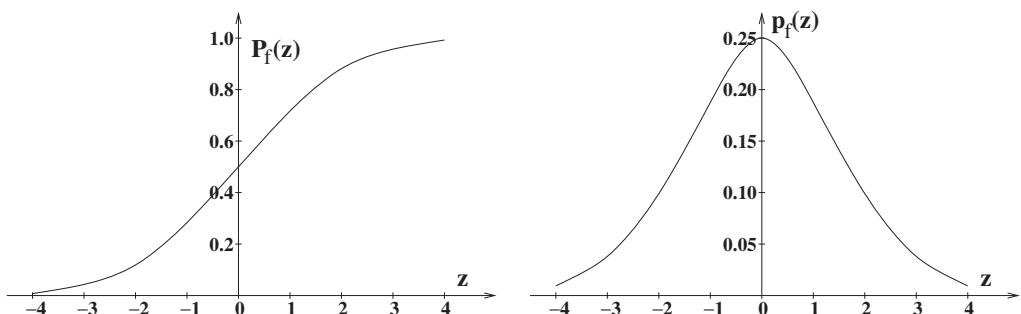


Figure 3.3: A plot of $P_f(z)$ on the left and $p_f(z)$ on the right.

Example B3.8

Compute the mean and variance of random variable z of example 3.7.

According to definition (3.5) and equation (3.9), we have:

$$\begin{aligned}\mu_f &= \int_{-\infty}^{+\infty} z p_f(z) dz \\ &= \int_{-\infty}^{+\infty} z \frac{1}{2(1 + \cosh z)} dz \\ &= 0\end{aligned}\tag{3.10}$$

This is because the integrand is an antisymmetric function and the integration is over a symmetric interval (since $\cosh(z) = \cosh(-z)$).

According to definition (3.6) and equation (3.9), we have:

$$\begin{aligned}\sigma_f^2 &= \int_{-\infty}^{+\infty} (z - \mu_z)^2 p_f(z) dz \\ &= \int_{-\infty}^{+\infty} z^2 \frac{1}{2(1 + \cosh z)} dz \\ &= \int_0^{+\infty} z^2 \frac{1}{1 + \cosh z} dz\end{aligned}\tag{3.11}$$

This is because the integrand now is symmetric and the integration is over a symmetric interval. To compute this integral, we make use of a formula taken from a table of integrals,

$$\int_0^{+\infty} \frac{x^{\mu-1}}{1 + \cosh x} dx = (2 - 2^{3-\mu}) \Gamma(\mu) \zeta(\mu - 1)\tag{3.12}$$

valid for $\mu \neq 2$ with $\zeta(x)$ being Riemann's ζ function. Functions Γ and ζ are well known transcendental functions with values and formulae that can be found in many function books. We apply (3.12) for $\mu = 3$ and obtain:

$$\sigma_f^2 = \Gamma(3) \zeta(2)\tag{3.13}$$

The Γ function for integer arguments is given by $\Gamma(z) = (z - 1)!$. So, $\Gamma(3) = 2! = 2$. Riemann's ζ function, on the other hand, for positive integer arguments, is defined as

$$\zeta(n) \equiv \sum_{k=1}^{+\infty} k^{-n}\tag{3.14}$$

and for $n = 2$ it can be shown to be equal to $\pi^2/6$. We deduce then that $\sigma_f^2 = \pi^2/3$.

How do we describe many random variables?

If we have n random variables we can define their **joint distribution function**:

$$P_{f_1 f_2 \dots f_n}(z_1, z_2, \dots, z_n) \equiv \mathcal{P}\{f_1 < z_1, f_2 < z_2, \dots, f_n < z_n\} \quad (3.15)$$

We can also define their **joint probability density function**:

$$p_{f_1 f_2 \dots f_n}(z_1, z_2, \dots, z_n) \equiv \frac{\partial^n P_{f_1 f_2 \dots f_n}(z_1, z_2, \dots, z_n)}{\partial z_1 \partial z_2 \dots \partial z_n} \quad (3.16)$$

What relationships may n random variables have with each other?

If the distribution of n random variables can be written as

$$P_{f_1 f_2 \dots f_n}(z_1, z_2, \dots, z_n) = P_{f_1}(z_1)P_{f_2}(z_2) \dots P_{f_n}(z_n) \quad (3.17)$$

then these random variables are called **independent**. They are called **uncorrelated** if:

$$E\{f_i f_j\} = E\{f_i\}E\{f_j\}, \forall i, j, i \neq j \quad (3.18)$$

Any two random variables are **orthogonal** to each other if:

$$E\{f_i f_j\} = 0 \quad (3.19)$$

The **covariance** of any two random variables is defined as:

$$c_{ij} \equiv E\{(f_i - \mu_{f_i})(f_j - \mu_{f_j})\} \quad (3.20)$$

Example 3.9

Show that if the covariance c_{ij} of two random variables is zero, the two variables are uncorrelated.

Expanding the right-hand side of the definition of the covariance, we get:

$$\begin{aligned} c_{ij} &= E\{f_i f_j - \mu_{f_i} f_j - \mu_{f_j} f_i + \mu_{f_i} \mu_{f_j}\} \\ &= E\{f_i f_j\} - \mu_{f_i} E\{f_j\} - \mu_{f_j} E\{f_i\} + \mu_{f_i} \mu_{f_j} \\ &= E\{f_i f_j\} - \mu_{f_i} \mu_{f_j} - \mu_{f_j} \mu_{f_i} + \mu_{f_i} \mu_{f_j} \\ &= E\{f_i f_j\} - \mu_{f_i} \mu_{f_j} \end{aligned} \quad (3.21)$$

Notice that the operation of taking the expectation value of a fixed number has no effect on it; ie $E\{\mu_{f_i}\} = \mu_{f_i}$. If $c_{ij} = 0$, we obtain

$$E\{f_i f_j\} = \mu_{f_i} \mu_{f_j} = E\{f_i\}E\{f_j\} \quad (3.22)$$

which shows that f_i and f_j are uncorrelated, according to (3.18).

Example 3.10

Show that if two random variables are independent, their joint probability density function may be written as the product of their individual probability density functions.

According to definition (3.17), when two random variables f_1 and f_2 are independent, their joint distribution function $P_{f_1 f_2}(x, y)$ may be written as the product of their individual distribution functions:

$$P_{f_1 f_2}(x, y) = P_{f_1}(x)P_{f_2}(y) \quad (3.23)$$

According to definition (3.16), their joint probability density function $p_{f_1 f_2}(x, y)$ is

$$\begin{aligned} p_{f_1 f_2}(x, y) &= \frac{\partial^2 P_{f_1 f_2}(x, y)}{\partial x \partial y} = \frac{\partial^2 P_{f_1}(x)P_{f_2}(y)}{\partial x \partial y} \\ &= \frac{dP_{f_1}(x)}{dx} \frac{dP_{f_2}(y)}{dy} = p_{f_1}(x)p_{f_2}(y) \end{aligned} \quad (3.24)$$

where we recognised $dP_{f_1}(x)/dx$ to be the probability density function of f_1 and $dP_{f_2}(y)/dy$ to be the probability density function of f_2 .

Example 3.11

Show that if two random variables f_1 and f_2 are independent, they are uncorrelated.

According to example 3.10, when two random variables f_1 and f_2 are independent, their joint probability density function $p_{f_1 f_2}(x, y)$ is equal to the product of their individual probability density functions $p_{f_1}(x)$ and $p_{f_2}(y)$. To show that they are uncorrelated, we must show that they satisfy equation (3.18). We start by applying a generalised version of definition (3.10) to compute the mean of their product:

$$\begin{aligned} \mu_{f_1 f_2} &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} xyp_{f_1 f_2}(x, y) dx dy \\ &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} xyp_{f_1}(x)p_{f_2}(y) dx dy \\ &= \int_{-\infty}^{+\infty} xp_{f_1}(x) dx \int_{-\infty}^{+\infty} yp_{f_2}(y) dy \\ &= \mu_{f_1} \mu_{f_2} \end{aligned} \quad (3.25)$$

This concludes the proof.

Example B3.12

Variables f_1 and f_2 have a joint probability density function which is uniform inside the square $ABCD$ shown in figure 3.4 and 0 outside it. Work out a formula for $p_{f_1 f_2}(x, y)$.

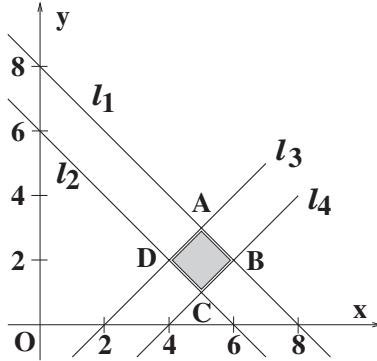


Figure 3.4: Joint probability density function $p(x, y)$ is nonzero and uniform inside square $ABCD$.

First we have to write down equations for lines l_1 , l_2 , l_3 and l_4 :

$$l_1: \quad x + y = 8$$

$$l_2: \quad x + y = 6$$

$$l_3: \quad x - y = 2$$

$$l_4: \quad x - y = 4$$

Next we have to derive the coordinates of the intersection points A , B , C and D . By solving the pairs of the equations that correspond to the intersecting lines, we deduce that:

Point A : Intersection of l_1 and l_3 : Coordinates $(5, 3)$

Point B : Intersection of l_1 and l_4 : Coordinates $(6, 2)$

Point C : Intersection of l_2 and l_4 : Coordinates $(5, 1)$

Point D : Intersection of l_2 and l_3 : Coordinates $(4, 2)$

Since $p_{f_1 f_2}(x, y)$ is uniform inside a square with side $\sqrt{2}$, ie inside an area of 2, and since it has to integrate to 1 over all values of x and y , $p_{f_1 f_2}(x, y) = 1/2$ inside square $ABCD$. So, we may write:

$$p_{f_1 f_2}(x, y) = \begin{cases} \frac{1}{2} & \text{if } 6 \leq x + y \leq 8 \text{ and } 2 \leq x - y \leq 4 \\ 0 & \text{elsewhere} \end{cases} \quad (3.26)$$

Example B3.13

Compute the expectation value of $\mu_{f_1 f_2}$ if the joint probability density function of variables f_1 and f_2 is given by equation (3.26).

By definition, the expectation value $E\{f_1 f_2\}$ is given by:

$$E\{f_1 f_2\} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x y p_{f_1 f_2}(x, y) dx dy \quad (3.27)$$

As the region over which $p_{f_1 f_2}(x, y)$ is nonzero has a boundary made up from linear segments, we must split the integration over x from the x coordinate of point D (ie from 4, see example 3.12) to the x coordinate of point C (ie 5) and from the x coordinate of point C to the x coordinate of point B (ie 6). From figure 3.4 we can see that the limits of integration over y in each one of these ranges of x are from line l_2 to line l_3 for x from 4 to 5, and from line l_4 to line l_1 for x from 5 to 6:

$$\begin{aligned} E\{f_1 f_2\} &= \int_4^5 \int_{y=6-x}^{y=x-2} x y \frac{1}{2} dy dx + \int_5^6 \int_{y=x-4}^{y=8-x} \frac{1}{2} dy dx \\ &= \frac{1}{2} \left\{ \int_4^5 x \left[\frac{y^2}{2} \right]_{6-x}^{x-2} dx + \int_5^6 x \left[\frac{y^2}{2} \right]_{x-4}^{8-x} dx \right\} \\ &= \frac{1}{2} \left\{ \int_4^5 x \left[\frac{x^2 + 4 - 4x}{2} - \frac{36 + x^2 - 12x}{2} \right] dx \right. \\ &\quad \left. + \int_5^6 x \left[\frac{64 + x^2 - 16x}{2} - \frac{x^2 + 16 - 8x}{2} \right] dx \right\} \\ &= \frac{1}{4} \left\{ \int_4^5 x(-32 + 8x) dx + \int_5^6 x(48 - 8x) dx \right\} \\ &= \int_4^5 (2x^2 - 8x) dx + \int_5^6 (12x - 2x^2) dx \\ &= \left[\frac{2x^3}{3} - \frac{8x^2}{2} \right]_4^5 + \left[\frac{12x^2}{2} - \frac{2x^3}{3} \right]_5^6 \\ &= \left[\frac{250}{3} - \frac{200}{2} - \frac{128}{3} + \frac{128}{2} \right] + \left[\frac{432}{2} - \frac{432}{3} - \frac{300}{2} + \frac{250}{3} \right] \\ &= 10 \end{aligned} \quad (3.28)$$

Example B3.14

The joint probability density function of variables f_1 and f_2 is given by equation (3.26). Compute the probability density function of variables f_1 and f_2 . Are variables f_1 and f_2 independent?

We are asked to compute the so called **marginal** probability density functions of variables f_1 and f_2 . The probability density function of f_1 will express the probability of finding a value of f_1 in a particular range of width dx irrespective of the value of f_2 . So, in order to work out this probability we have to eliminate the dependence of $p_{f_1 f_2}(x, y)$ on y . This means that we have to integrate $p_{f_1 f_2}(x, y)$ over all values of y . We have to do this by splitting the range of values of x from 4 to 5 and from 5 to 6 and integrating over the appropriate limits of y in each range:

$$\begin{aligned} p_{f_1}(x) &= \int_{y=6-x}^{y=x-2} \frac{1}{2} dy = \frac{1}{2}(x - 2 - 6 + x) = x - 4 \quad \text{for } 4 \leq x \leq 5 \\ p_{f_1}(x) &= \int_{y=x-4}^{y=8-x} \frac{1}{2} dy = \frac{1}{2}(8 - x - x + 4) = 6 - x \quad \text{for } 5 \leq x \leq 6 \end{aligned} \quad (3.29)$$

Working in a similar way for the probability density function of f_2 , we obtain:

$$\begin{aligned} p_{f_2}(y) &= \int_{x=6-y}^{x=4+y} \frac{1}{2} dx = \frac{1}{2}(4 + y - 6 + y) = y - 1 \quad \text{for } 1 \leq y \leq 2 \\ p_{f_2}(y) &= \int_{x=2+y}^{x=8-y} \frac{1}{2} dx = \frac{1}{2}(8 - y - 2 - y) = 3 - y \quad \text{for } 2 \leq y \leq 3 \end{aligned} \quad (3.30)$$

The two random variables are not independent because we cannot write $p_{f_1 f_2}(x, y) = p_{f_1}(x)p_{f_2}(y)$.

Example B3.15

Compute the mean values of variables f_1 and f_2 of example 3.14. Are these two variables uncorrelated?

We shall make use of the probability density functions of these two variables given by equations (3.29) and (3.30). The mean value of f_1 is given by:

$$\begin{aligned}
E\{f_1\} &= \int_{-\infty}^{+\infty} xp_{f_1}(x)dx \\
&= \int_4^5 x(x-4)dx + \int_5^6 x(6-x)dx \\
&= \int_4^5 (x^2 - 4x)dx + \int_5^6 (6x - x^2)dx \\
&= \left[\frac{x^3}{3} - \frac{4x^2}{2} \right]_4^5 + \left[\frac{6x^2}{2} - \frac{x^3}{3} \right]_5^6 \\
&= \left[\frac{125}{3} - \frac{100}{2} - \frac{64}{3} + \frac{64}{2} \right] + \left[\frac{216}{2} - \frac{216}{3} - \frac{150}{2} + \frac{125}{3} \right] \\
&= 5
\end{aligned} \tag{3.31}$$

In a similar way, we can work out that $E\{f_2\} = 2$. In example 3.13 we worked out that $E\{f_1 f_2\} = 10$. So, $E\{f_1 f_2\} = E\{f_1\}E\{f_2\}$, and the two random variables are uncorrelated. This is an example of random variables that are dependent (because we cannot write $p_{f_1 f_2}(x, y) = p_{f_1}(x)p_{f_2}(y)$) but uncorrelated (because we can write $E\{f_1 f_2\} = E\{f_1\}E\{f_2\}$). In general, uncorrelatedness is a much weaker condition than independence.

How do we define a random field?

If we define a random variable at every point in a 2D space, we say that we have a 2D **random field**. The position of the space where the random variable is defined is like a parameter of the random field: $f(\mathbf{r}; \omega_i)$.

This function for fixed \mathbf{r} is a random variable, but for fixed ω_i (fixed outcome) is a 2D function in the plane, an image, say. As ω_i scans all possible outcomes of the underlying statistical experiment, the random field represents a series of images. On the other hand, for a given outcome, (fixed ω_i), the random field gives the grey level values at the various positions in an image.

Example 3.16

Using an unloaded die, we conducted a series of experiments. Each experiment consisted of throwing the die four times. The outcomes $\{\omega_1, \omega_2, \omega_3, \omega_4\}$ of sixteen experiments are given below:

$\{1, 2, 1, 6\}, \{3, 5, 2, 4\}, \{3, 4, 6, 6\}, \{1, 1, 3, 2\}, \{3, 4, 4, 4\}, \{2, 6, 4, 2\},$

$\{1, 5, 3, 6\}, \{1, 2, 6, 4\}, \{6, 5, 2, 4\}, \{3, 2, 5, 6\}, \{1, 2, 4, 5\}, \{5, 1, 1, 6\},$

$\{2, 5, 3, 1\}, \{3, 1, 5, 6\}, \{1, 2, 1, 5\}, \{3, 2, 5, 4\}$

If \mathbf{r} is a 2D vector taking values

$(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4),$

$(3, 1), (3, 2), (3, 3), (3, 4), (4, 1), (4, 2), (4, 3), (4, 4)$

give the series of images defined by the random field $f(\mathbf{r}; \omega_i)$.

The first image is formed by placing the first outcome of each experiment in the corresponding position, the second by using the second outcome of each experiment, and so on. The ensemble of images we obtain is:

$$\begin{pmatrix} 1 & 3 & 3 & 1 \\ 3 & 2 & 1 & 1 \\ 6 & 3 & 1 & 5 \\ 2 & 3 & 1 & 3 \end{pmatrix} \begin{pmatrix} 2 & 5 & 4 & 1 \\ 4 & 6 & 5 & 2 \\ 5 & 2 & 2 & 1 \\ 5 & 1 & 2 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 & 6 & 3 \\ 4 & 4 & 3 & 6 \\ 2 & 5 & 4 & 1 \\ 3 & 5 & 1 & 5 \end{pmatrix} \begin{pmatrix} 6 & 4 & 6 & 2 \\ 4 & 2 & 6 & 4 \\ 4 & 6 & 5 & 6 \\ 1 & 6 & 5 & 4 \end{pmatrix} \quad (3.32)$$

How can we relate two random variables that appear in the same random field?

For fixed \mathbf{r} , a random field becomes a random variable with an expectation value which depends on \mathbf{r} :

$$\mu_f(\mathbf{r}) = E\{f(\mathbf{r}; \omega_i)\} = \int_{-\infty}^{+\infty} z p_f(z; \mathbf{r}) dz \quad (3.33)$$

Since for different values of \mathbf{r} we have different random variables, $f(\mathbf{r}_1; \omega_i)$ and $f(\mathbf{r}_2; \omega_i)$, we can define their correlation, called **autocorrelation** (we use “auto” because the two variables come from the same random field) as:

$$R_{ff}(\mathbf{r}_1, \mathbf{r}_2) = E\{f(\mathbf{r}_1; \omega_i)f(\mathbf{r}_2; \omega_i)\} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} z_1 z_2 p_f(z_1, z_2; \mathbf{r}_1, \mathbf{r}_2) dz_1 dz_2 \quad (3.34)$$

The **autocovariance** $C(\mathbf{r}_1, \mathbf{r}_2)$ is defined as:

$$C_{ff}(\mathbf{r}_1, \mathbf{r}_2) = E\{[f(\mathbf{r}_1; \omega_i) - \mu_f(\mathbf{r}_1)][f(\mathbf{r}_2; \omega_i) - \mu_f(\mathbf{r}_2)]\} \quad (3.35)$$

Example B3.17

Show that for a random field:

$$C_{ff}(\mathbf{r}_1, \mathbf{r}_2) = R_{ff}(\mathbf{r}_1, \mathbf{r}_2) - \mu_f(\mathbf{r}_1)\mu_f(\mathbf{r}_2) \quad (3.36)$$

Starting from equation (3.35):

$$\begin{aligned} C_{ff}(\mathbf{r}_1, \mathbf{r}_2) &= E\{[f(\mathbf{r}_1; \omega_i) - \mu_f(\mathbf{r}_1)][f(\mathbf{r}_2; \omega_i) - \mu_f(\mathbf{r}_2)]\} \\ &= E\{f(\mathbf{r}_1; \omega_i)f(\mathbf{r}_2; \omega_i) - f(\mathbf{r}_1; \omega_i)\mu_f(\mathbf{r}_2) - \mu_f(\mathbf{r}_1)f(\mathbf{r}_2; \omega_i) \\ &\quad + \mu_f(\mathbf{r}_1)\mu_f(\mathbf{r}_2)\} \\ &= E\{f(\mathbf{r}_1; \omega_i)f(\mathbf{r}_2; \omega_i)\} - E\{f(\mathbf{r}_1; \omega_i)\}\mu_f(\mathbf{r}_2) - \mu_f(\mathbf{r}_1)E\{f(\mathbf{r}_2; \omega_i)\} \\ &\quad + \mu_f(\mathbf{r}_1)\mu_f(\mathbf{r}_2) \\ &= R_{ff}(\mathbf{r}_1, \mathbf{r}_2) - \mu_f(\mathbf{r}_1)\mu_f(\mathbf{r}_2) - \mu_f(\mathbf{r}_1)\mu_f(\mathbf{r}_2) + \mu_f(\mathbf{r}_1)\mu_f(\mathbf{r}_2) \\ &= R_{ff}(\mathbf{r}_1, \mathbf{r}_2) - \mu_f(\mathbf{r}_1)\mu_f(\mathbf{r}_2) \end{aligned} \quad (3.37)$$

Example 3.18

Compute the mean of the ensemble of images (3.32).

The mean of a random field is given by (3.33). However, in this case, instead of having explicitly the probability density function of the random variable associated with each position, $p_f(z; \mathbf{r})$, we have an ensemble of values. These values are assumed to have been drawn according to $p_f(z; \mathbf{r})$. All we have to do then in order to find the mean is simply to average these values. The result is:

$$\mu = \begin{pmatrix} 2.50 & 3.50 & 4.75 & 1.75 \\ 3.75 & 3.50 & 3.75 & 3.25 \\ 4.25 & 4.00 & 3.00 & 3.25 \\ 2.75 & 3.75 & 2.25 & 3.50 \end{pmatrix} \quad (3.38)$$

Example 3.19

Compute the autocorrelation matrix for the ensemble of images (3.32).

The autocorrelation matrix for a random field is given by (3.34). If, however, we do not have an expression for the joint probability density function of the random variables at two positions, namely $p_f(z_1, z_2; \mathbf{r}_1, \mathbf{r}_2)$, we cannot use this formula. If we have instead an ensemble of versions of the random field, all we have to do is to perform the relevant statistics on the ensemble of images we have. This is the case here.

As we have 16 positions, ie 16 random variables, we may have $16^2 = 256$ combinations of positions. We shall work out here just a couple of the values of the autocorrelation function:

$$\begin{aligned} R_{ff}((1, 1), (1, 1)) &= \frac{1^2 + 2^2 + 1^2 + 6^2}{4} = 10.5 \\ R_{ff}((1, 1), (1, 2)) &= \frac{1 \times 3 + 2 \times 5 + 1 \times 2 + 6 \times 4}{4} = 9.75 \\ R_{ff}((2, 3), (4, 1)) &= \frac{1 \times 2 + 5 \times 5 + 3 \times 3 + 6 \times 1}{4} = 10.5 \end{aligned} \quad (3.39)$$

Example 3.20

Compute the autocovariance matrix for the ensemble of images (3.32).

The autocovariance matrix for a random field is given by (3.35). As we do not have an explicit formula for the joint probability density function of the random variables at two positions, we compute the autocovariance matrix using ensemble statistics. We only show here a couple of relative positions. In this calculation we make use of the mean value at each position as computed in example 3.18:

$$\begin{aligned} C_{ff}((1, 1), (1, 1)) &= \frac{(1 - 2.5)^2 + (2 - 2.5)^2 + (1 - 2.5)^2 + (6 - 2.5)^2}{4} = 4.25 \\ C_{ff}((1, 1), (1, 2)) &= \frac{1}{4} [(1 - 2.5)(3 - 3.5) + (2 - 2.5)(5 - 3.5) + \\ &\quad (1 - 2.5)(2 - 3.5) + (6 - 2.5)(4 - 3.5)] = 1 \\ C_{ff}((2, 3), (4, 1)) &= \frac{1}{4} [(1 - 3.75)(2 - 2.75) + (5 - 3.75)(5 - 2.75) + \\ &\quad (3 - 3.75)(3 - 2.75) + (6 - 3.75)(1 - 2.75)] = 0.1875 \end{aligned} \quad (3.40)$$

How can we relate two random variables that belong to two different random fields?

If we have two random fields, ie two series of images generated by two different underlying random experiments, represented by f and g , we can define their **cross correlation**

$$R_{fg}(\mathbf{r}_1, \mathbf{r}_2) = E\{f(\mathbf{r}_1; \omega_i)g(\mathbf{r}_2; \omega_j)\} \quad (3.41)$$

and their **cross covariance**:

$$\begin{aligned} C_{fg}(\mathbf{r}_1, \mathbf{r}_2) &= E\{[f(\mathbf{r}_1; \omega_i) - \mu_f(\mathbf{r}_1)][g(\mathbf{r}_2; \omega_j) - \mu_g(\mathbf{r}_2)]\} \\ &= R_{fg}(\mathbf{r}_1, \mathbf{r}_2) - \mu_f(\mathbf{r}_1)\mu_g(\mathbf{r}_2) \end{aligned} \quad (3.42)$$

Two random fields are called **uncorrelated** if for all values \mathbf{r}_1 and \mathbf{r}_2 :

$$C_{fg}(\mathbf{r}_1, \mathbf{r}_2) = 0 \quad (3.43)$$

This is equivalent to:

$$E\{f(\mathbf{r}_1; \omega_i)g(\mathbf{r}_2; \omega_j)\} = E\{f(\mathbf{r}_1; \omega_i)\}E\{g(\mathbf{r}_2; \omega_j)\} \quad (3.44)$$

Example B3.21

Show that for two uncorrelated random fields we have:

$$E\{f(\mathbf{r}_1; \omega_i)g(\mathbf{r}_2; \omega_j)\} = E\{f(\mathbf{r}_1; \omega_i)\}E\{g(\mathbf{r}_2; \omega_j)\} \quad (3.45)$$

This follows trivially from the definition of uncorrelated random fields ($C_{fg}(\mathbf{r}_1, \mathbf{r}_2) = 0$) and the expression

$$C_{fg}(\mathbf{r}_1, \mathbf{r}_2) = E\{f(\mathbf{r}_1; \omega_i)g(\mathbf{r}_2; \omega_j)\} - \mu_f(\mathbf{r}_1)\mu_g(\mathbf{r}_2) \quad (3.46)$$

which can be proven in a similar way as (3.36).

Example 3.22

You are given an ensemble of versions of a random field:

$$\begin{pmatrix} 5 & 6 & 5 & 7 \\ 7 & 6 & 6 & 8 \\ 6 & 7 & 4 & 5 \\ 6 & 4 & 5 & 3 \end{pmatrix} \begin{pmatrix} 5 & 5 & 4 & 8 \\ 7 & 6 & 5 & 9 \\ 5 & 6 & 7 & 5 \\ 8 & 6 & 4 & 5 \end{pmatrix} \begin{pmatrix} 5 & 4 & 6 & 7 \\ 8 & 4 & 4 & 3 \\ 3 & 5 & 5 & 5 \\ 4 & 4 & 4 & 5 \end{pmatrix} \begin{pmatrix} 7 & 5 & 5 & 4 \\ 3 & 3 & 5 & 6 \\ 4 & 6 & 7 & 5 \\ 3 & 3 & 5 & 6 \end{pmatrix} \quad (3.47)$$

Compute the cross-covariance between this random field and that of the random field represented by ensemble (3.32).

First we have to compute the average at each position in this second random field. This is given by the average of the versions of the field we are given, and found to be:

$$\begin{pmatrix} 5.50 & 5.00 & 5.00 & 6.50 \\ 6.25 & 4.75 & 5.00 & 6.50 \\ 4.50 & 6.00 & 5.75 & 5.00 \\ 5.25 & 4.25 & 4.50 & 4.75 \end{pmatrix} \quad (3.48)$$

The cross-covariance then is computed by using formula (3.42). This formula tells us to consider pairs of positions, and for each member of a pair to subtract the corresponding mean, multiply the value the first member of the pair has in a version of the first random field with the value the second member of the pair has in the corresponding version of the second random field, and average over all versions.

Let us consider two positions $(2, 1)$ and $(3, 3)$. The mean of the first field in position $(2, 1)$ is 3.75 according to (3.38). The mean of the second field in position $(3, 3)$ is 5.75 according to (3.48). The cross-covariance for these two positions between the two fields is:

$$\begin{aligned} C_{fg}((2, 1), (3, 3)) &= \frac{1}{4} [(3 - 3.75)(4 - 5.75) + (4 - 3.75)(7 - 5.75) + \\ &\quad (4 - 3.75)(5 - 5.75) + (4 - 3.75)(7 - 5.75)] = 0.4375 \end{aligned} \quad (3.49)$$

Similarly, for positions $(1, 1)$ and $(4, 4)$, and positions $(1, 1)$ and $(1, 1)$, we find:

$$\begin{aligned} C_{fg}((1, 1), (4, 4)) &= \frac{1}{4} [(1 - 2.5)(3 - 4.75) + (2 - 2.5)(5 - 4.75) + \\ &\quad (1 - 2.5)(5 - 4.75) + (6 - 2.5)(6 - 4.75)] = 1.625 \\ C_{fg}((1, 1), (1, 1)) &= \frac{1}{4} [(1 - 2.5)(5 - 5.5) + (2 - 2.5)(5 - 5.5) + \\ &\quad (1 - 2.5)(5 - 5.5) + (6 - 2.5)(7 - 5.5)] = 1.75 \end{aligned} \quad (3.50)$$

We can work out the full cross-covariance by considering all pairs of positions in the two random fields.

Example 3.23

Show that if the values at any two positions r_1 and r_2 , where $r_1 \neq r_2$, in a random field are uncorrelated, the covariance matrix of the random field is diagonal.

The values at two different positions of a random field are two random variables. According to definition (3.18), two random variables are uncorrelated if the expectation of their product is equal to the product of their expectations. The expectation of the product of the two random variables in this case is the value of the autocorrelation function of the field for the two positions, let us call it $R_{ff}(\mathbf{r}_1, \mathbf{r}_2)$. If we denote by $\mu_f(\mathbf{r}_1)$ and $\mu_f(\mathbf{r}_2)$ the mean of the random field at positions \mathbf{r}_1 and \mathbf{r}_2 , respectively, we may then write

$$R_{ff}(\mathbf{r}_1, \mathbf{r}_2) = \mu_f(\mathbf{r}_1)\mu_f(\mathbf{r}_2) \quad (3.51)$$

since we are told that the values at \mathbf{r}_1 and \mathbf{r}_2 are uncorrelated. According to example 3.17 then, $C_{ff}(\mathbf{r}_1, \mathbf{r}_2) = 0$. Note that this refers only to the case $\mathbf{r}_1 \neq \mathbf{r}_2$. If $\mathbf{r}_1 = \mathbf{r}_2$, according to definition (3.35), $C_{ff}(\mathbf{r}_1, \mathbf{r}_1)$ is the expectation value of a squared number, ie it is the average of non-negative numbers, and as such it cannot be 0. It is actually the variance of the random field at position \mathbf{r}_1 : $C_{ff}(\mathbf{r}_1, \mathbf{r}_1) = \sigma^2(\mathbf{r}_1)$. So, the autocovariance matrix of an uncorrelated random field is diagonal, with all its off-diagonal elements 0 and all its diagonal elements equal to the variance of the field at the corresponding positions.

If we have just one image from an ensemble of images, can we calculate expectation values?

Yes. We make the assumption that the image we have is an instantiation of a random field that is **homogeneous (stationary)** with respect to the mean and the autocorrelation function, and **ergodic** with respect to the mean and the autocorrelation function. This assumption allows us to replace the *ensemble* statistics of the random field (ie the statistics we could compute over a collection of images) with the *spatial* statistics of the single image we have.

When is a random field homogeneous with respect to the mean?

A random field is homogeneous (stationary) with respect to the mean, if the expectation value at all positions is the same, ie if the left-hand side of equation (3.33) does not depend on \mathbf{r} .

When is a random field homogeneous with respect to the autocorrelation function?

If the expectation value of the random field does not depend on \mathbf{r} , and if its autocorrelation function is translation invariant, then the field is called **homogeneous (or stationary)** with respect to the autocorrelation function.

A translation invariant autocorrelation function depends on only one argument, the relative shifting of the positions at which we calculate the values of the random field:

$$R_{ff}(\mathbf{r}_0) = E\{f(\mathbf{r}; \omega_i)f(\mathbf{r} + \mathbf{r}_0; \omega_i)\} \quad (3.52)$$

Example 3.24

Show that the autocorrelation function $R(\mathbf{r}_1, \mathbf{r}_2)$ of a homogeneous (stationary) random field depends only on the difference vector $\mathbf{r}_1 - \mathbf{r}_2$.

The autocorrelation function of a homogeneous random field is translation invariant. Therefore, for any translation vector \mathbf{r}_0 we may write:

$$\begin{aligned} R_{ff}(\mathbf{r}_1, \mathbf{r}_2) &= E\{f(\mathbf{r}_1; \omega_i)f(\mathbf{r}_2; \omega_i)\} = E\{f(\mathbf{r}_1 + \mathbf{r}_0; \omega_i)f(\mathbf{r}_2 + \mathbf{r}_0; \omega_i)\} \\ &= R_{ff}(\mathbf{r}_1 + \mathbf{r}_0, \mathbf{r}_2 + \mathbf{r}_0) \quad \forall \mathbf{r}_0 \end{aligned} \quad (3.53)$$

Choosing $\mathbf{r}_0 = -\mathbf{r}_2$ we see that for a homogeneous random field:

$$R_{ff}(\mathbf{r}_1, \mathbf{r}_2) = R_{ff}(\mathbf{r}_1 - \mathbf{r}_2, \mathbf{0}) = R_{ff}(\mathbf{r}_1 - \mathbf{r}_2) \quad (3.54)$$

How can we calculate the spatial statistics of a random field?

Given a random field we can define its spatial average as

$$\mu(\omega_i) \equiv \lim_{S \rightarrow +\infty} \frac{1}{S} \int_S f(\mathbf{r}; \omega_i) dxdy \quad (3.55)$$

where $\mathbf{r} = (x, y)$ and \int_S is the integral over the whole space S with area S . The result $\mu(\omega_i)$ is clearly a function of the outcome on which f depends; ie $\mu(\omega_i)$ is a random variable.

The spatial autocorrelation function of the random field is defined as:

$$R(\mathbf{r}_0; \omega_i) \equiv \lim_{S \rightarrow +\infty} \frac{1}{S} \int_S f(\mathbf{r}; \omega_i)f(\mathbf{r} + \mathbf{r}_0; \omega_i) dxdy \quad (3.56)$$

This is another random variable.

How do we compute the spatial autocorrelation function of an image in practice?

Let us say that the image is $M \times N$ in size. The relative positions of two pixels may be (i, j) and $(i + k, j + l)$, where k and l may take values from $-M + 1$ to $M - 1$ and from $-N + 1$ to $N - 1$, respectively. The autocorrelation function then will be of size $(2M - 1) \times (2N - 1)$, and its (k, l) element will have value

$$R(k, l) = \frac{1}{NM} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} g(i, j)g(i + k, j + l) \quad (3.57)$$

where $k = -M + 1, -M + 2, \dots, 0, \dots, M - 1$, $l = -N + 1, -N + 2, \dots, 0, \dots, N - 1$ and $g(i, j)$ is the grey value of the image at position (i, j) . In order to have the same number of pairs for all possible relative positions, we assume that the image is repeated ad infinitum in all directions, so all pixels have neighbours at all distances and in all orientations.

Example 3.25

Compute the spatial autocorrelation function of the following image:

$$\begin{pmatrix} 1 & 2 & 1 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \end{pmatrix} \quad (3.58)$$

We apply formula (3.57) for $M = N = 3$. The autocorrelation function will be a 2D discrete function (ie a matrix) of size 5×5 . The result is:

$$R = \frac{1}{9} \begin{pmatrix} 15 & 15 & 18 & 15 & 15 \\ 15 & 15 & 18 & 15 & 15 \\ 15 & 15 & 18 & 15 & 15 \\ 15 & 15 & 18 & 15 & 15 \\ 15 & 15 & 18 & 15 & 15 \end{pmatrix} \quad (3.59)$$

When is a random field ergodic with respect to the mean?

A random field is ergodic with respect to the mean, if it is homogeneous with respect to the mean and its spatial average, defined by (3.55), is independent of the outcome on which f depends, ie it is the same from whichever version of the random field it is computed, and is equal to the ensemble average defined by equation (3.33):

$$E\{f(\mathbf{r}; \omega_i)\} = \lim_{S \rightarrow +\infty} \frac{1}{S} \int_S f(\mathbf{r}; \omega_i) dx dy = \mu = \text{a constant} \quad (3.60)$$

When is a random field ergodic with respect to the autocorrelation function?

A random field is ergodic with respect to the autocorrelation function if it is homogeneous (stationary) with respect to the autocorrelation function and its spatial autocorrelation function, defined by (3.56), is independent of the outcome of the experiment on which f depends, depends only on the displacement \mathbf{r}_0 and is equal to the ensemble autocorrelation function defined by equation (3.52):

$$\underbrace{E\{f(\mathbf{r}; \omega_i)f(\mathbf{r} + \mathbf{r}_0; \omega_i)\}}_{\text{ensemble autocorrelation function}} = \underbrace{\lim_{S \rightarrow +\infty} \frac{1}{S} \int_S f(\mathbf{r}; \omega_i)f(\mathbf{r} + \mathbf{r}_0; \omega_i) dx dy}_{\text{spatial autocorrelation function}} = \underbrace{R(\mathbf{r}_0)}_{\text{independent of } \omega_i} \quad (3.61)$$

Example 3.26

You are given the following ensemble of images:

$$\begin{pmatrix} 5 & 4 & 6 & 2 \\ 5 & 3 & 4 & 3 \\ 6 & 6 & 7 & 1 \\ 5 & 4 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 4 & 2 & 2 & 1 \\ 7 & 2 & 4 & 9 \\ 3 & 5 & 4 & 5 \\ 4 & 6 & 6 & 2 \end{pmatrix}, \begin{pmatrix} 3 & 5 & 2 & 3 \\ 5 & 4 & 4 & 3 \\ 2 & 2 & 6 & 6 \\ 6 & 5 & 4 & 6 \end{pmatrix}, \begin{pmatrix} 6 & 4 & 2 & 8 \\ 3 & 5 & 6 & 4 \\ 4 & 4 & 2 & 2 \\ 5 & 4 & 3 & 4 \end{pmatrix},$$

$$\begin{pmatrix} 4 & 3 & 5 & 4 \\ 6 & 5 & 6 & 2 \\ 4 & 3 & 3 & 4 \\ 3 & 3 & 6 & 5 \end{pmatrix}, \begin{pmatrix} 4 & 5 & 4 & 5 \\ 1 & 6 & 2 & 6 \\ 4 & 8 & 4 & 4 \\ 1 & 3 & 2 & 7 \end{pmatrix}, \begin{pmatrix} 2 & 7 & 6 & 4 \\ 2 & 4 & 2 & 4 \\ 6 & 3 & 4 & 7 \\ 4 & 3 & 6 & 2 \end{pmatrix}, \begin{pmatrix} 5 & 3 & 6 & 6 \\ 4 & 4 & 5 & 2 \\ 4 & 2 & 3 & 4 \\ 5 & 5 & 4 & 4 \end{pmatrix}$$

Is this ensemble of images ergodic with respect to the mean? Is it ergodic with respect to the autocorrelation?

It is ergodic with respect to the mean because the average of each image is 4.125 and the average at each pixel position over all eight images is also 4.125.

It is not ergodic with respect to the autocorrelation function. To prove this, let us calculate one element of the autocorrelation matrix, say element $E\{g_{23}g_{34}\}$ which is the average of product values of all pixels at positions (2,3) and (3,4) over all images:

$$\begin{aligned} E\{g_{23}g_{34}\} &= \frac{4 \times 1 + 4 \times 5 + 4 \times 6 + 6 \times 2 + 6 \times 4 + 2 \times 4 + 2 \times 7 + 5 \times 4}{8} \\ &= \frac{4 + 20 + 24 + 12 + 24 + 8 + 14 + 20}{8} \\ &= \frac{126}{8} = 15.75 \end{aligned} \tag{3.62}$$

This should be equal to the element of the autocorrelation function which expresses the spatial average of pairs of pixels which are diagonal neighbours from top left to bottom right direction, computed from any image. Consider the last image in the ensemble. We have:

$$\begin{aligned} < g_{ij}g_{i+1,j+1} > &= \\ \frac{5 \times 4 + 3 \times 5 + 6 \times 2 + 4 \times 2 + 4 \times 3 + 5 \times 4 + 4 \times 5 + 2 \times 4 + 3 \times 4}{9} &= \\ \frac{20 + 15 + 12 + 8 + 12 + 20 + 20 + 8 + 12}{9} = \frac{127}{9} = 13 &\neq 15.75 \end{aligned} \tag{3.63}$$

The two results are not the same, and therefore the ensemble is not ergodic with respect to the autocorrelation function.

What is the implication of ergodicity?

If an ensemble of images is ergodic, then we can calculate its mean and autocorrelation function by simply calculating spatial averages over *any* image of the ensemble we happen to have (see figure 3.5).

For example, assume that we have a collection of M images of similar type $\{g_1(x, y), g_2(x, y), \dots, g_M(x, y)\}$. The mean and autocorrelation function of this collection can be calculated by taking averages over all images in the collection. On the other hand, if we assume ergodicity, we can pick up only one of these images and calculate the mean and the autocorrelation function from it with the help of spatial averages. This will be correct if the natural variability of all the different images is statistically the same as the natural variability exhibited by the contents of each single image separately.

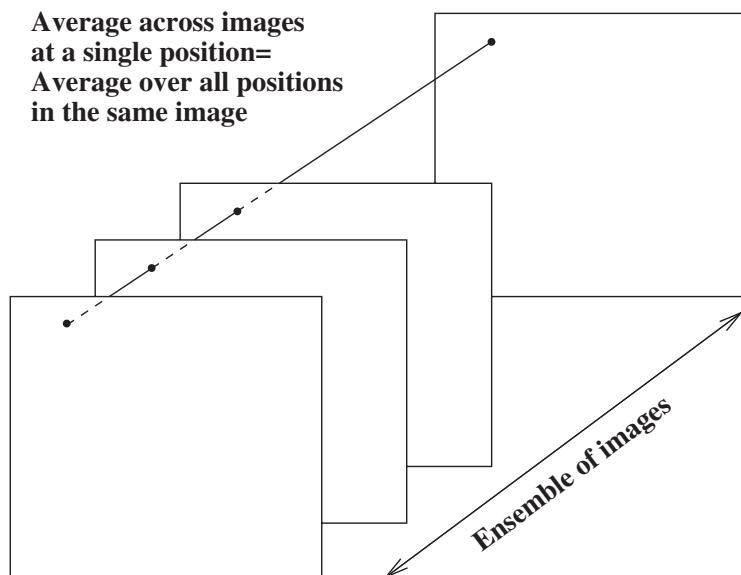


Figure 3.5: Ergodicity in a nutshell.

Example 3.27

You are given the following ensemble of four 3×3 images:

$$\begin{pmatrix} 3 & 2 & 1 \\ 0 & 1 & 2 \\ 3 & 3 & 3 \end{pmatrix} \quad \begin{pmatrix} 2 & 2 & 2 \\ 2 & 3 & 2 \\ 2 & 1 & 2 \end{pmatrix} \quad \begin{pmatrix} 3 & 2 & 3 \\ 3 & 2 & 3 \\ 0 & 2 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 2 & 2 \\ 3 & 2 & 1 \\ 3 & 2 & 3 \end{pmatrix} \quad (3.64)$$

- (i) Is this ensemble ergodic with respect to the mean?
- (ii) Is this ensemble ergodic with respect to the autocorrelation function?

(i) This ensemble is ergodic with respect to the mean, because the average across the four images for each pixel is 2 and the spatial average of every image is 2 too.

(ii) Let us compute the ensemble autocorrelation function for pixels (1, 1) and (2, 1).

$$\frac{3 \times 0 + 2 \times 2 + 3 \times 3 + 0 \times 3}{4} = \frac{13}{4} = 3.25 \quad (3.65)$$

Let us also compute the spatial autocorrelation function of the first image of the ensemble for the same relative position. We have 6 pairs in this relative position:

$$\frac{3 \times 0 + 2 \times 1 + 1 \times 2 + 0 \times 3 + 1 \times 3 + 2 \times 3}{6} = \frac{13}{6} = 2.17 \quad (3.66)$$

The ensemble is not ergodic with respect to the autocorrelation function, because the spatial autocorrelation function computed for two positions, one below the other, gave a different answer from that obtained by taking two positions one under the other and averaging over the ensemble.

Box 3.1. Ergodicity, fuzzy logic and probability theory

Ergodicity is the key that connects probability theory and fuzzy logic. Probability theory performs all its operations assuming that situations, objects, images, or in general the items with which it deals, are the results of some random process which may create an ensemble of versions of each item. It always computes functions over that virtual ensemble, the properties of which are modelled by some parametric function. Fuzzy logic, on the other hand, instead of saying “this item has probability $x\%$ to be red and $y\%$ to be green and $z\%$ to be yellow, etc”, it says “this item consists of a red part making up $x\%$ of it, a green part making up $y\%$ of it, a yellow part making up $z\%$ of it, and so on”. If ergodicity were applicable, *all* items that make up the ensemble used by probability theory would have consisted of parts that reflect the variety of objects in the ensemble in the right proportions. So, if ergodicity were applicable, either we computed functions over the ensemble of items as done by probability theory, or we computed functions over a single item, as done by fuzzy logic, we would have found the same answer, as every item would be expected to contain all variations that may be encountered in the right proportions; every item would be a fair representative of the whole ensemble, and one would not need to have the full ensemble to have a complete picture of the world.

How can we construct a basis of elementary images appropriate for expressing in an optimal way a whole set of images?

We do that by choosing a transformation that diagonalises the ensemble autocovariance matrix of the set of images. Such a transformation is called **Karhunen-Loeve** transform.

3.2 Karhunen-Loeve transform

What is the Karhunen-Loeve transform?

It is the transformation of an image into a basis of elementary images, defined by diagonalising the covariance matrix of a collection of images, which are treated as instantiations of the same random field, and to which collection of images the transformed image is assumed to belong.

Why does diagonalisation of the autocovariance matrix of a set of images define a desirable basis for expressing the images in the set?

Let us consider a space where we have as many coordinate axes as we have pixels in an image, and let us assume that we measure the value of each pixel along one of the axes. Each image then would be represented by a point in this space. The set of all images will make a cluster of such points. The shape of this cluster of points is most simply described in a coordinate system made up from the axes of symmetry of the cluster. For example, in geometry, the equation of a 2D ellipse in an (x, y) coordinate system defined by its axes of symmetry is $x^2/\alpha^2 + y^2/\beta^2 = 1$, where α and β are the semi-major and semi-minor axes of the ellipse. In a general coordinate system, however, the equation of the ellipse is $a\tilde{x}^2 + b\tilde{y}^2 + c\tilde{x}\tilde{y} + d\tilde{x} + e\tilde{y} + f = 0$, where a, b, c, d, e and f are some constants (see figure 3.6). This example demonstrates that every shape implies an intrinsic coordinate system, in terms of which it is described in the simplest possible way. Let us go back now to the cluster of points made up from the images in a set. We would like to represent that cloud of points in the simplest possible way.

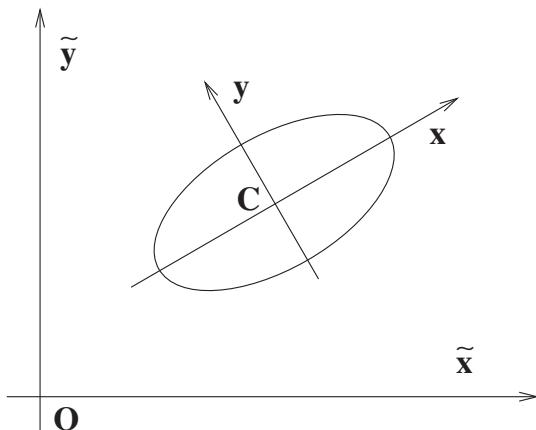


Figure 3.6: The equation of an ellipse is much simpler in coordinate system Cxy , which is intrinsic to the ellipse, than in coordinate system $O\tilde{x}\tilde{y}$.

The first step in identifying an intrinsic coordinate system for it is to shift the origin of the axes to the centre of the cluster (see figure 3.7). It can be shown that if we rotate the axes so that they coincided with the axes of symmetry of the cloud of points, the autocorrelation matrix of the points described in this rotated system would be diagonal (see example 3.29).

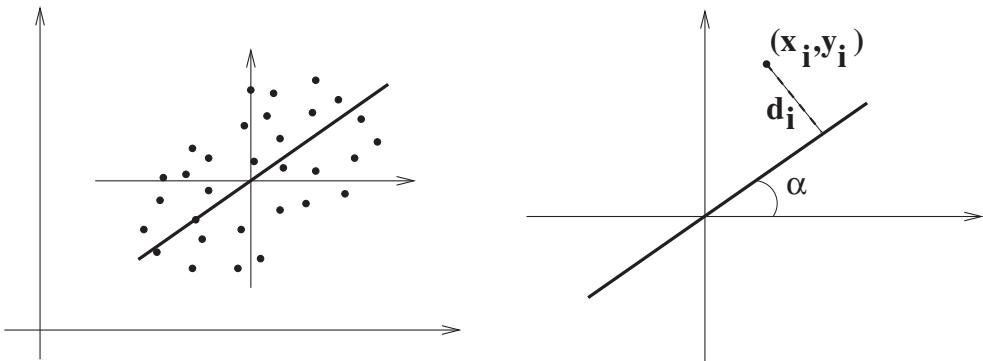


Figure 3.7: To find the axis of symmetry of a cloud of points we first translate the original coordinate system to the centre of the cloud. In practice this means that we remove the average coordinate values from all coordinates of the points of the cloud. Then we postulate that the axis of symmetry is at orientation α with respect to the horizontal axis, and work out what α should be.

Example B3.28

Show that the axis of symmetry of a cloud of points in 2D is the one for which the sum of the squares of the distances of the points from it is minimal.

Let us consider a cloud of points with respect to a coordinate system centred at their average position. Let us call that system (x, y) and let us use subscript i to identify individual points. Let us consider an axis passing through the origin of the coordinate system and having orientation α with respect to the x axis (see figure 3.7). The axis of symmetry of the cloud of points will be such that the sum of the signed distances d_i of all points from the axis will be as close as possible to 0. So, the symmetry axis will have orientation angle α , such that:

$$\left| \sum_i d_i \right| = \text{minimum} \Rightarrow \left(\sum_i d_i \right)^2 = \text{minimum} \Rightarrow \sum_i d_i^2 + 2 \sum_i \sum_{j,j \neq i} d_i d_j = \text{minimum} \quad (3.67)$$

The distance of a point (x_i, y_i) from a line, with directional vector $(\cos \alpha, \sin \alpha)$, is given by:

$$d_i = y_i \cos \alpha - x_i \sin \alpha \quad (3.68)$$

Let us examine the second term in (3.67):

$$\begin{aligned}
\sum_i \sum_{j,j \neq i} d_i d_j &= \sum_i \sum_{j,j \neq i} (y_i \cos \alpha - x_i \sin \alpha)(y_j \cos \alpha - x_j \sin \alpha) \\
&= (\cos \alpha)^2 \sum_i \sum_{j,j \neq i} y_i y_j + (\sin \alpha)^2 \sum_i \sum_{j,j \neq i} x_i x_j \\
&\quad - \cos \alpha \sin \alpha \sum_i \sum_{j,j \neq i} y_i x_j - \sin \alpha \cos \alpha \sum_i \sum_{j,j \neq i} x_i y_j \\
&= (\cos \alpha)^2 \sum_i y_i \sum_{j,j \neq i} y_j + (\sin \alpha)^2 \sum_i x_i \sum_{j,j \neq i} x_j \\
&\quad - \cos \alpha \sin \alpha \sum_i y_i \sum_{j,j \neq i} x_j - \sin \alpha \cos \alpha \sum_i x_i \sum_{j,j \neq i} y_j \\
&= 0
\end{aligned} \tag{3.69}$$

The result is 0 because the axes are centred at the centre of the cloud of points and so $\sum_i y_i = \sum_i x_i = 0$. So, the second term in (3.67) is 0 and the axis of symmetry is defined by the angle α that minimises $\sum_i d_i^2$, ie the sum of the square distances of all points from it.

Example B3.29

Show that the axis of symmetry of a 2D cloud of points is such that the correlation of the values of the points along the two axes is 0.

According to example 3.28, the axis of symmetry is defined by angle α for which $\sum_i d_i^2$ is minimal:

$$\begin{aligned}
\sum_i d_i^2 &= \sum_i (y_i \cos \alpha - x_i \sin \alpha)^2 = \text{minimal} \Rightarrow \\
(\cos \alpha)^2 \sum_i y_i^2 + (\sin \alpha)^2 \sum_i x_i^2 - \sin(2\alpha) \sum_i y_i x_i
\end{aligned} \tag{3.70}$$

Here we made use of $\sin(2\alpha) = 2 \sin \alpha \cos \alpha$. This expression is minimal for the value of α that makes its first derivative with respect to α zero:

$$\begin{aligned}
2 \cos \alpha \sin \alpha \sum_i y_i^2 - 2 \sin \alpha \cos \alpha \sum_i x_i^2 - 2 \cos(2\alpha) \sum_i y_i x_i &= 0 \Rightarrow \\
\sin(2\alpha) \left(\sum_i y_i^2 - \sum_i x_i^2 \right) &= 2 \cos(2\alpha) \sum_i y_i x_i \Rightarrow \\
\tan(2\alpha) &= \frac{2 \sum_i y_i x_i}{\sum_i y_i^2 - \sum_i x_i^2}
\end{aligned} \tag{3.71}$$

Note now that if the correlation between x_i and y_i is zero, ie if $\sum_i y_i x_i = 0$, $\tan(2\alpha) = 0$ and so $\alpha = 0$, ie the symmetry axis coincides with axis x . So, the set of axes defined by the symmetry of the cloud of points is the same set of axes for which the correlation between the values along the different axes is zero. This means that the autocorrelation matrix of points (x_i, y_i) will be diagonal, with elements along the diagonal the variances of the two components. (The autocovariance is the same as the autocorrelation when we are dealing with zero mean random variables. See also example 3.23.)

How can we transform an image so its autocovariance matrix becomes diagonal?

We wish to be able to express the autocovariance matrix of the image in terms of the image itself in a linear way, so that from the transformation of the autocovariance matrix to be able easily to work out the transformation of the image itself. This is not possible if we carry on treating the image as 2D. We saw in Chapter 1 that in the simplest of cases an image has to be manipulated by two matrices of the same size, one from the left and one from the right. Such a manipulation will make the relationship between transformed image and transform of the covariance matrix of the original image far too complicated. On the other hand, if we express the image as a vector, by stacking its columns one under the other, the relationship between the covariance matrix of the elements of this vector and the vector itself is much more straightforward. Another way to see the necessity of using the image in its vector form is to think in terms of the space where the value of each pixel is measured along one axis. The whole image in such a space is represented by a single point, ie by a vector with coordinates the values of its pixels: the spatial arrangement of the pixels in the original image is no longer relevant. All we need to do in order to find the best basis for the cloud of points made up from all images we wish to consider is to find a transformation that makes the correlation matrix of the points, with respect to the new basis system, diagonal, ie the correlation between any pair of pixels zero, irrespective of their spatial positions in the image.

Example 3.30

Write the images of example 3.26 in vector form and compute their ensemble autocovariance matrix.

The images in vector form are:

$$\begin{aligned}\mathbf{g_1}^T &= (5, 5, 6, 5, 4, 3, 6, 4, 6, 4, 7, 2, 2, 3, 1, 3) \\ \mathbf{g_2}^T &= (4, 7, 3, 4, 2, 2, 5, 6, 2, 4, 4, 6, 1, 9, 5, 2) \\ \mathbf{g_3}^T &= (3, 5, 2, 6, 5, 4, 2, 5, 2, 4, 6, 4, 3, 3, 6, 6) \\ \mathbf{g_4}^T &= (6, 3, 4, 5, 4, 5, 4, 4, 2, 6, 2, 3, 8, 4, 2, 4) \\ \mathbf{g_5}^T &= (4, 6, 4, 3, 3, 5, 3, 3, 5, 6, 3, 6, 4, 2, 4, 5)\end{aligned}$$

$$\begin{aligned}
\mathbf{g}_6^T &= (4, 1, 4, 1, 5, 6, 8, 3, 4, 2, 4, 2, 5, 6, 4, 7) \\
\mathbf{g}_7^T &= (2, 2, 6, 4, 7, 4, 3, 3, 6, 2, 4, 6, 4, 4, 7, 2) \\
\mathbf{g}_8^T &= (5, 4, 4, 5, 3, 4, 2, 5, 6, 5, 3, 4, 6, 2, 4, 4)
\end{aligned} \tag{3.72}$$

To compute the autocovariance matrix of the ensemble of these vectors we first remove from them the average vector, calculated to be:

$$\boldsymbol{\mu}_{\mathbf{g}}^T = (4.125, 4.125, 4.125, 4.125, 4.125, 4.125, 4.125, 4.125) \tag{3.73}$$

The new vectors are:

$$\begin{aligned}
\tilde{\mathbf{g}}_1^T &= (0.875, 0.875, 1.875, 0.875, -0.125, -1.125, 1.875, -0.125, 1.875, -0.125, \\
&\quad 2.875, -2.125, -2.125, -1.125, -3.125, -1.125) \\
\tilde{\mathbf{g}}_2^T &= (-0.125, 2.875, -1.125, -0.125, -2.125, -2.125, 0.875, 1.875, -2.125, -0.125, \\
&\quad -0.125, 1.875, -3.125, 4.875, 0.875, -2.125) \\
\tilde{\mathbf{g}}_3^T &= (-1.125, 0.875, -2.125, 1.875, 0.875, -0.125, -2.125, 0.875, -2.125, -0.125, \\
&\quad 1.875, -0.125, -1.125, -1.125, 1.875, 1.875) \\
\tilde{\mathbf{g}}_4^T &= (1.875, -1.125, -0.125, 0.875, -0.125, 0.875, -0.125, -0.125, -2.125, 1.875, \\
&\quad -2.125, -1.125, 3.875, -0.125, -2.125, -0.125) \\
\tilde{\mathbf{g}}_5^T &= (-0.125, 1.875, -0.125, -1.125, -1.125, 0.875, -1.125, -1.125, 0.875, 1.875, \\
&\quad -1.125, 1.875, -0.125, -2.125, -0.125, 0.875) \\
\tilde{\mathbf{g}}_6^T &= (-0.125, -3.125, -0.125, -3.125, 0.875, 1.875, 3.875, -1.125, -0.125, -2.125, \\
&\quad -0.125, -2.125, 0.875, 1.875, -0.125, 2.875) \\
\tilde{\mathbf{g}}_7^T &= (-2.125, -2.125, 1.875, -0.125, 2.875, -0.125, -1.125, -1.125, 1.875, -2.125, \\
&\quad -0.125, 1.875, -0.125, -0.125, 2.875, -2.125) \\
\tilde{\mathbf{g}}_8^T &= (0.875, -0.125, -0.125, 0.875, -1.125, -0.125, -2.125, 0.875, 1.875, 0.875, \\
&\quad -1.125, -0.125, 1.875, -2.125, -0.125, -0.125)
\end{aligned} \tag{3.74}$$

The autocovariance matrix of the set is given by

$$C(k, l) = \frac{1}{16} \sum_{i=1}^8 \tilde{\mathbf{g}}_i(k) \tilde{\mathbf{g}}_i(l) \tag{3.75}$$

where $\tilde{\mathbf{g}}_i(k)$ is the k^{th} element of vector $\tilde{\mathbf{g}}_i$. Since the vectors have 16 elements, k and l take values from 1 to 16 and so the autocovariance matrix of the set is 16×16 .

$$C = \begin{pmatrix} 9.19 & 8.63 & 8.50 & 8.63 & 8.00 & 8.56 & 8.75 & 8.63 & 8.38 & 9.06 & 8.25 \\ 8.63 & 10.31 & 8.06 & 9.06 & 7.50 & 7.75 & 7.94 & 9.13 & 8.13 & 9.25 & 8.81 \\ 8.50 & 8.06 & 9.31 & 8.38 & 8.88 & 8.50 & 8.81 & 8.13 & 9.38 & 8.25 & 8.63 \\ 8.63 & 9.06 & 8.38 & 9.56 & 8.44 & 8.06 & 7.56 & 8.94 & 8.31 & 8.94 & 8.81 \\ 8.00 & 7.50 & 8.88 & 8.44 & 9.56 & 8.81 & 8.50 & 8.06 & 8.81 & 7.81 & 8.75 \\ 8.56 & 7.75 & 8.50 & 8.06 & 8.81 & 9.19 & 8.69 & 8.06 & 8.56 & 8.50 & 8.13 \\ 8.75 & 7.94 & 8.81 & 7.56 & 8.50 & 8.69 & 10.44 & 8.25 & 8.44 & 7.88 & 8.81 \\ 8.63 & 9.13 & 8.13 & 8.94 & 8.06 & 8.06 & 8.25 & 9.06 & 8.06 & 8.69 & 8.63 \\ 8.38 & 8.13 & 9.38 & 8.31 & 8.81 & 8.56 & 8.44 & 8.06 & 10.06 & 8.25 & 8.68 \\ 9.06 & 9.25 & 8.25 & 8.94 & 7.81 & 8.50 & 7.88 & 8.69 & 8.25 & 9.56 & 8.06 \\ 8.25 & 8.81 & 8.63 & 8.81 & 8.75 & 8.13 & 8.81 & 8.63 & 8.69 & 8.06 & 9.69 \\ 8.00 & 9.19 & 8.38 & 8.56 & 8.38 & 8.19 & 7.63 & 8.63 & 8.50 & 8.63 & 8.13 \\ 9.06 & 7.31 & 8.56 & 8.44 & 8.75 & 9.38 & 8.19 & 8.13 & 8.50 & 9.00 & 7.38 \\ 8.38 & 8.69 & 8.19 & 7.94 & 8.19 & 8.06 & 9.69 & 8.94 & 7.50 & 7.88 & 8.44 \\ 7.56 & 8.38 & 8.19 & 8.44 & 9.06 & 8.44 & 7.75 & 8.56 & 8.38 & 7.88 & 8.44 \\ 8.56 & 8.00 & 8.00 & 8.06 & 8.63 & 9.25 & 8.81 & 8.25 & 8.20 & 8.50 & 8.50 \\ 8.00 & 9.06 & 8.38 & 7.56 & 8.56 \\ 9.19 & 7.31 & 8.69 & 8.38 & 8.00 \\ 8.38 & 8.56 & 8.19 & 8.19 & 8.00 \\ 8.56 & 8.44 & 7.94 & 8.44 & 8.06 \\ 8.38 & 8.75 & 8.19 & 9.06 & 8.63 \\ 8.19 & 9.38 & 8.06 & 8.44 & 9.25 \\ 7.63 & 8.19 & 9.69 & 7.75 & 8.81 \\ 8.63 & 8.13 & 8.94 & 8.56 & 8.25 \\ 8.50 & 8.50 & 7.50 & 8.38 & 8.19 \\ 8.62 & 9.00 & 7.88 & 7.88 & 8.50 \\ 8.13 & 7.38 & 8.44 & 8.44 & 8.50 \\ 9.81 & 8.00 & 8.75 & 9.50 & 7.88 \\ 8.00 & 10.69 & 7.63 & 8.06 & 9.06 \\ 8.75 & 7.63 & 10.94 & 8.88 & 8.06 \\ 9.50 & 8.06 & 8.88 & 10.19 & 8.44 \\ 7.88 & 9.06 & 8.06 & 8.44 & 9.94 \end{pmatrix} \quad (3.76)$$

Example 3.31

Compute the spatial autocorrelation function of the vector representation of the first image of the images of example 3.26.

The vector representation of this image is

$$\mathbf{g}_1^T = (5, 5, 6, 5, 4, 3, 6, 4, 6, 4, 7, 2, 2, 3, 1, 3) \quad (3.77)$$

The spatial autocorrelation function of this digital signal now is a function of the relative shift between the samples that make up the pairs of pixels we have to consider. Let us call the function $R(h)$ where h takes values from 0 to maximum 15, as this signal consists of a total of 16 samples and it is not possible to find samples at larger distances than 15 from each other. We compute:

$$\begin{aligned}
 R(0) &= \frac{1}{16} (5^2 + 5^2 + 6^2 + 5^2 + 4^2 + 3^2 + 6^2 + 4^2 + 6^2 + 4^2 + 7^2 + 2^2 + 2^2 + 3^2 \\
 &\quad + 1^2 + 3^2) = 19.75 \\
 R(1) &= \frac{1}{15} (5 \times 5 + 5 \times 6 + 6 \times 5 + 5 \times 4 + 4 \times 3 + 3 \times 6 + 6 \times 4 + 4 \times 6 \\
 &\quad + 6 \times 4 + 4 \times 7 + 7 \times 2 + 2 \times 2 + 2 \times 3 + 3 \times 1 + 1 \times 3) = 17.67 \\
 R(2) &= \frac{1}{14} (5 \times 6 + 5 \times 5 + 6 \times 4 + 5 \times 3 + 4 \times 6 + 3 \times 4 + 6 \times 6 + 4 \times 4 \\
 &\quad + 6 \times 7 + 4 \times 2 + 7 \times 2 + 2 \times 3 + 2 \times 1 + 3 \times 3) = 18.79 \\
 R(3) &= \frac{1}{13} (5 \times 5 + 5 \times 4 + 6 \times 3 + 5 \times 6 + 4 \times 4 + 3 \times 6 + 6 \times 4 + 4 \times 7 + 6 \times 2 \\
 &\quad + 4 \times 2 + 7 \times 3 + 2 \times 1 + 2 \times 3) = 17.54 \\
 R(4) &= \frac{1}{12} (5 \times 4 + 5 \times 3 + 6 \times 6 + 5 \times 4 + 4 \times 6 + 3 \times 4 + 6 \times 7 + 4 \times 2 + 6 \times 2 \\
 &\quad + 4 \times 3 + 7 \times 1 + 2 \times 3) = 17.83 \\
 R(5) &= \frac{1}{11} (5 \times 3 + 5 \times 6 + 6 \times 4 + 5 \times 6 + 4 \times 4 + 3 \times 7 + 6 \times 2 + 4 \times 2 + 6 \times 3 \\
 &\quad + 4 \times 1 + 7 \times 3) = 18.09 \\
 R(6) &= \frac{1}{10} (5 \times 6 + 5 \times 4 + 6 \times 6 + 5 \times 4 + 4 \times 7 + 3 \times 2 + 6 \times 2 + 4 \times 3 + 6 \times 1 \\
 &\quad + 4 \times 3) = 18.2 \\
 R(7) &= \frac{1}{9} (5 \times 4 + 5 \times 6 + 6 \times 4 + 5 \times 7 + 4 \times 2 + 3 \times 2 + 6 \times 3 + 4 \times 1 + 6 \times 3) \\
 &= 18.11 \\
 R(8) &= \frac{1}{8} (5 \times 6 + 5 \times 4 + 6 \times 7 + 5 \times 2 + 4 \times 2 + 3 \times 3 + 6 \times 1 + 4 \times 3) = 17.13 \\
 R(9) &= \frac{1}{7} (5 \times 4 + 5 \times 7 + 6 \times 2 + 5 \times 2 + 4 \times 3 + 3 \times 1 + 6 \times 3) = 15.71 \\
 R(10) &= \frac{1}{6} (5 \times 7 + 5 \times 2 + 6 \times 2 + 5 \times 3 + 4 \times 1 + 3 \times 3) = 14.17 \\
 R(11) &= \frac{1}{5} (5 \times 2 + 5 \times 2 + 6 \times 3 + 5 \times 1 + 4 \times 3) = 11 \\
 R(12) &= \frac{1}{4} (5 \times 2 + 5 \times 3 + 6 \times 1 + 5 \times 3) = 11.5 \\
 R(13) &= \frac{1}{3} (5 \times 3 + 5 \times 1 + 6 \times 3) = 12.67 \\
 R(14) &= \frac{1}{2} (5 \times 1 + 5 \times 3) = 10 \\
 R(15) &= 5 \times 3 = 15
 \end{aligned} \tag{3.78}$$

Example 3.32

Compute the spatial autocorrelation function of the 1D representation of the first image of the images of example 3.26, assuming that the 1D signal is repeated ad infinitum.

The difference with example 3.31 is that now we can have equal number of pairs of samples for all relative shifts. Let us write the augmented signal we shall be using:

$$\mathbf{g}_1^T = (5, 5, 6, 5, 4, 3, 6, 4, 6, 4, 7, 2, 2, 3, 1, 3 | 5, 5, 6, 5, 4, 3, 6, 4, 6, 4, 7, 2, 2, 3, 1, 3) \quad (3.79)$$

The spatial autocorrelation function now takes the following values:

$$\begin{aligned}
 R(0) &= \frac{1}{16} (5^2 + 5^2 + 6^2 + 5^2 + 4^2 + 3^2 + 6^2 + 4^2 + 6^2 + 4^2 + 7^2 + 2^2 + 2^2 + 3^2 \\
 &\quad + 1^2 + 3^2) = 19.75 \\
 R(1) &= \frac{1}{16} (5 \times 5 + 5 \times 6 + 6 \times 5 + 5 \times 4 + 4 \times 3 + 3 \times 6 + 6 \times 4 + 4 \times 6 \\
 &\quad + 6 \times 4 + 4 \times 7 + 7 \times 2 + 2 \times 2 + 2 \times 3 + 3 \times 1 + 1 \times 3 + 3 \times 5) = 17.5 \\
 R(2) &= \frac{1}{16} (5 \times 6 + 5 \times 5 + 6 \times 4 + 5 \times 3 + 4 \times 6 + 3 \times 4 + 6 \times 6 + 4 \times 4 \\
 &\quad + 6 \times 7 + 4 \times 2 + 7 \times 2 + 2 \times 3 + 2 \times 1 + 3 \times 3 + 1 \times 5 + 3 \times 5) = 17.69 \\
 R(3) &= \frac{1}{16} (5 \times 5 + 5 \times 4 + 6 \times 3 + 5 \times 6 + 4 \times 4 + 3 \times 6 + 6 \times 4 + 4 \times 7 + 6 \times 2 \\
 &\quad + 4 \times 2 + 7 \times 3 + 2 \times 1 + 2 \times 3 + 3 \times 5 + 1 \times 5 + 3 \times 6) = 16.63 \\
 R(4) &= \frac{1}{16} (5 \times 4 + 5 \times 3 + 6 \times 6 + 5 \times 4 + 4 \times 6 + 3 \times 4 + 6 \times 7 + 4 \times 2 + 6 \times 2 \\
 &\quad + 4 \times 3 + 7 \times 1 + 2 \times 3 + 2 \times 5 + 3 \times 5 + 1 \times 6 + 3 \times 5) = 16.25 \\
 R(5) &= \frac{1}{16} (5 \times 3 + 5 \times 6 + 6 \times 4 + 5 \times 6 + 4 \times 4 + 3 \times 7 + 6 \times 2 + 4 \times 2 + 6 \times 3 \\
 &\quad + 4 \times 1 + 7 \times 3 + 2 \times 5 + 2 \times 5 + 3 \times 6 + 1 \times 5 + 3 \times 4) = 15.88 \\
 R(6) &= \frac{1}{16} (5 \times 6 + 5 \times 4 + 6 \times 6 + 5 \times 4 + 4 \times 7 + 3 \times 2 + 6 \times 2 + 4 \times 3 + 6 \times 1 \\
 &\quad + 4 \times 3 + 7 \times 5 + 2 \times 5 + 2 \times 6 + 3 \times 5 + 1 \times 4 + 3 \times 3) = 16.69 \\
 R(7) &= \frac{1}{16} (5 \times 4 + 5 \times 6 + 6 \times 4 + 5 \times 7 + 4 \times 2 + 3 \times 2 + 6 \times 3 + 4 \times 1 + 6 \times 3 \\
 &\quad + 4 \times 5 + 7 \times 5 + 2 \times 6 + 2 \times 5 + 3 \times 4 + 1 \times 3 + 3 \times 6) = 17.06 \\
 R(8) &= \frac{1}{16} (5 \times 6 + 5 \times 4 + 6 \times 7 + 5 \times 2 + 4 \times 2 + 3 \times 3 + 6 \times 1 + 4 \times 3 + 6 \times 5 \\
 &\quad + 4 \times 5 + 7 \times 6 + 2 \times 5 + 2 \times 4 + 3 \times 3 + 1 \times 6 + 3 \times 4) = 17.13
 \end{aligned}$$

$$\begin{aligned}
R(9) &= \frac{1}{16}(5 \times 4 + 5 \times 7 + 6 \times 2 + 5 \times 2 + 4 \times 3 + 3 \times 1 + 6 \times 3 + 4 \times 5 + 6 \times 5 \\
&\quad + 4 \times 6 + 7 \times 5 + 2 \times 4 + 2 \times 3 + 3 \times 6 + 1 \times 4 + 3 \times 6) = 17.06 \\
R(10) &= \frac{1}{16}(5 \times 7 + 5 \times 2 + 6 \times 2 + 5 \times 3 + 4 \times 1 + 3 \times 3 + 6 \times 5 + 4 \times 5 + 6 \times 6 \\
&\quad + 4 \times 5 + 7 \times 4 + 2 \times 3 + 2 \times 6 + 3 \times 4 + 1 \times 6 + 3 \times 4) = 16.69 \\
R(11) &= \frac{1}{16}(5 \times 2 + 5 \times 2 + 6 \times 3 + 5 \times 1 + 4 \times 3 + 3 \times 5 + 6 \times 5 + 4 \times 6 + 6 \times 5 \\
&\quad + 4 \times 4 + 7 \times 3 + 2 \times 6 + 2 \times 4 + 3 \times 6 + 1 \times 4 + 3 \times 7) = 15.88 \\
R(12) &= \frac{1}{16}(5 \times 2 + 5 \times 3 + 6 \times 1 + 5 \times 3 + 4 \times 5 + 3 \times 5 + 6 \times 6 + 4 \times 5 + 6 \times 4 \\
&\quad + 4 \times 3 + 7 \times 6 + 2 \times 4 + 2 \times 6 + 3 \times 4 + 1 \times 7 + 3 \times 2) = 16.25 \\
R(13) &= \frac{1}{16}(5 \times 3 + 5 \times 1 + 6 \times 3 + 5 \times 5 + 4 \times 5 + 3 \times 6 + 6 \times 5 + 4 \times 4 + 6 \times 3 \\
&\quad + 4 \times 6 + 7 \times 4 + 2 \times 6 + 2 \times 4 + 3 \times 7 + 1 \times 2 + 3 \times 2) = 16.63 \\
R(14) &= \frac{1}{16}(5 \times 1 + 5 \times 3 + 6 \times 5 + 5 \times 5 + 4 \times 6 + 3 \times 5 + 6 \times 4 + 4 \times 3 + 6 \times 6 \\
&\quad + 4 \times 4 + 7 \times 6 + 2 \times 4 + 2 \times 7 + 3 \times 2 + 1 \times 2 + 3 \times 3) = 17.69 \\
R(15) &= \frac{1}{16}(5 \times 3 + 5 \times 5 + 6 \times 5 + 5 \times 6 + 4 \times 5 + 3 \times 4 + 6 \times 3 + 4 \times 6 + 6 \times 4 \\
&\quad + 4 \times 6 + 7 \times 4 + 2 \times 7 + 2 \times 2 + 3 \times 2 + 1 \times 3 + 3 \times 1) = 17.5
\end{aligned}$$

Note that by assuming repetition of the signal we have introduced some symmetry in the autocorrelation function, as samples that are at a distance h apart from each other can also be thought of as being at a distance $16 - h$ apart. So, $R(h) = R(16 - h)$.

Example 3.33

Show that the spatial autocorrelation function $R(h)$ and the spatial autocovariance function $C(h)$ of an N -sample long signal g with spatial mean \bar{g}^2 are related by:

$$C(h) = R(h) - \bar{g}^2 \quad (3.80)$$

By definition,

$$\begin{aligned}
C(h) &= \frac{1}{N} \sum_i [g(i) - \bar{g}][g(i+h) - \bar{g}] \\
&= \frac{1}{N} \sum_i g(i)g(i+h) - \bar{g} \frac{1}{N} \sum_i g(i) - \bar{g} \frac{1}{N} \sum_i g(i+h) + \bar{g}^2 \frac{1}{N} \sum_i 1 \\
&= R(h) - \bar{g}^2 - \bar{g}^2 + \bar{g}^2
\end{aligned} \quad (3.81)$$

Formula (3.80) then follows. Note that this result could not have been obtained if we had not considered that the signal was repeated, because we could not have replaced $\frac{1}{N} \sum_i g(i + h)$ with \bar{g} .

Example 3.34

Compute the spatial autocovariance function of the 1D representation of the first image of the images of example 3.26, on page 198, assuming that the 1D signal is repeated ad infinitum.

The only difference with example 3.32 is that before performing the calculation of $R(h)$, we should have removed the spatial mean of the samples. The spatial mean is $\bar{g} = 4.125$.

According to example 3.33 all we need do to go from $R(h)$ to $C(h)$ is to remove \bar{g}^2 from the values of $R(h)$. The result is:

$$\begin{aligned} C &= (15.625, 13.375, 13.565, 12.505, 12.125, 11.755, 12.565, 12.935, 13.005, \\ &\quad 12.935, 12.565, 11.755, 12.125, 12.505, 13.565, 13.375) \end{aligned} \quad (3.82)$$

What is the form of the ensemble autocorrelation matrix of a set of images, if the ensemble is stationary with respect to the autocorrelation?

The ensemble being stationary with respect to the autocorrelation means that the value of the autocorrelation will be the same for all pairs of samples that are in the same relative position from each other. We can see that better if we consider a set of 3×3 images. An element of this set is image g^i and it has the form:

$$\begin{pmatrix} g_{11}^i & g_{12}^i & g_{13}^i \\ g_{21}^i & g_{22}^i & g_{23}^i \\ g_{31}^i & g_{32}^i & g_{33}^i \end{pmatrix} \quad (3.83)$$

The autocorrelation function of the set takes a pair of positions and finds the average value of their product over the whole set of images. To visualise this, we create a double entry table and place all possible positions along the rows and the columns of the matrix, such that we have all possible combinations. We represent the average value of each pair of positions with a different letter, using the same letter for positions that are at the same relative position from each other. We obtain:

	g_{11}	g_{21}	g_{31}	g_{12}	g_{22}	g_{32}	g_{13}	g_{23}	g_{33}	
g_{11}	A	B	C	D	E	F	G	H	I	
g_{21}	B	A	B	J	D	E	K	G	H	
g_{31}	C	B	A	L	J	D	M	K	G	
g_{12}	D	J	L	A	B	C	D	E	F	
g_{22}	E	D	J	B	A	B	J	D	E	
g_{32}	F	E	D	C	B	A	L	J	D	
g_{13}	G	K	M	D	J	L	A	B	C	
g_{23}	H	G	K	E	D	J	B	A	B	
g_{33}	I	H	G	F	E	D	C	B	A	

Note that if the ensemble were not stationary, we could have had a different letter (value) at every position in the above table.

Example B3.35

In (3.84) the relative position of two positions has been decided according to 2D. What form would the same matrix have had if we had written all images as vectors, and thus decided the relative positions of two samples from the vector arrangement?

When we write image (3.83) as a vector, we bring next to pixel g_{31}^i , pixel g_{12}^i , and thus these two positions now become next-door neighbours, and in a stationary signal their product is expected to have average value equal to that of positions g_{11}^i and g_{21}^i , for example. So, the autocorrelation matrix now takes the form:

	g_{11}	g_{21}	g_{31}	g_{12}	g_{22}	g_{32}	g_{13}	g_{23}	g_{33}	
g_{11}	A	B	C	D	E	F	G	H	I	
g_{21}	B	A	B	C	D	E	F	G	H	
g_{31}	C	B	A	B	C	D	E	F	G	
g_{12}	D	C	B	A	B	C	D	E	F	
g_{22}	E	D	C	B	A	B	C	D	E	
g_{32}	F	E	D	C	B	A	B	C	D	
g_{13}	G	F	E	D	C	B	A	B	C	
g_{23}	H	G	F	E	D	C	B	A	B	
g_{33}	I	H	G	F	E	D	C	B	A	

How do we go from the 1D autocorrelation function of the vector representation of an image to its 2D autocorrelation matrix?

Assuming ergodicity, the 1D spatial autocorrelation function of the vector representation of the image is treated like the ensemble 2D autocorrelation matrix (see example 3.36).

Example 3.36

You are given only the first image of those in example 3.26 and you are told that it is representative of a whole collection of images that share the same statistical properties. Assuming ergodicity, estimate the autocovariance matrix of the vector representations of the ensemble of these images.

The spatial autocovariance matrix of this image has been computed in example 3.34. We use those values to create the autocovariance matrix of the ensemble, assuming that due to ergodicity, it will have the banded structure shown in (3.85).

$$C = \begin{pmatrix} 15.625 & 13.375 & 13.565 & 12.505 & 12.125 & 11.755 & 12.565 & 12.935 \\ 13.375 & 15.625 & 13.375 & 13.565 & 12.505 & 12.125 & 11.755 & 12.565 \\ 13.565 & 13.375 & 15.625 & 13.375 & 13.565 & 12.505 & 12.125 & 11.755 \\ 12.505 & 13.565 & 13.375 & 15.625 & 13.375 & 13.565 & 12.505 & 12.125 \\ 12.125 & 12.505 & 13.565 & 13.375 & 15.625 & 13.375 & 13.565 & 12.505 \\ 11.755 & 12.125 & 12.505 & 13.565 & 13.375 & 15.625 & 13.375 & 13.565 \\ 12.565 & 11.755 & 12.125 & 12.505 & 13.565 & 13.375 & 15.625 & 13.375 \\ 12.935 & 12.565 & 11.755 & 12.125 & 12.505 & 13.565 & 13.375 & 15.625 \\ 13.005 & 12.935 & 12.565 & 11.755 & 12.125 & 12.505 & 13.565 & 13.375 \\ 12.935 & 13.005 & 12.935 & 12.565 & 11.755 & 12.125 & 12.505 & 13.565 \\ 12.565 & 12.935 & 13.005 & 12.935 & 12.565 & 11.755 & 12.125 & 12.505 \\ 11.755 & 12.565 & 12.935 & 13.005 & 12.935 & 12.565 & 11.755 & 12.125 \\ 12.125 & 11.755 & 12.565 & 12.935 & 13.005 & 12.935 & 12.565 & 11.755 \\ 12.505 & 12.125 & 11.755 & 12.565 & 12.935 & 13.005 & 12.935 & 12.565 \\ 13.565 & 12.505 & 12.125 & 11.755 & 12.565 & 12.935 & 13.005 & 12.935 \\ 13.375 & 13.565 & 12.505 & 12.125 & 11.755 & 12.565 & 12.935 & 13.005 \\ 13.005 & 12.935 & 12.565 & 11.755 & 12.125 & 12.505 & 13.565 & 13.375 \\ 12.935 & 13.005 & 12.935 & 12.565 & 11.755 & 12.125 & 12.505 & 13.565 \\ 12.565 & 12.935 & 13.005 & 12.935 & 12.565 & 11.755 & 12.125 & 12.505 \\ 11.755 & 12.565 & 12.935 & 13.005 & 12.935 & 12.565 & 11.755 & 12.125 \\ 12.125 & 11.755 & 12.565 & 12.935 & 13.005 & 12.935 & 12.565 & 11.755 \\ 12.505 & 12.125 & 11.755 & 12.565 & 12.935 & 13.005 & 12.935 & 12.565 \\ 13.565 & 12.505 & 12.125 & 11.755 & 12.565 & 12.935 & 13.005 & 12.935 \\ 13.375 & 13.565 & 12.505 & 12.125 & 11.755 & 12.565 & 12.935 & 13.005 \\ 15.625 & 13.375 & 13.565 & 12.505 & 12.125 & 11.755 & 12.565 & 12.935 \\ 13.375 & 15.625 & 13.375 & 13.565 & 12.505 & 12.125 & 11.755 & 12.565 \\ 13.565 & 13.375 & 15.625 & 13.375 & 13.565 & 12.505 & 12.125 & 11.755 \\ 12.505 & 13.565 & 13.375 & 15.625 & 13.375 & 13.565 & 12.505 & 12.125 \\ 12.125 & 12.505 & 13.565 & 13.375 & 15.625 & 13.375 & 13.565 & 12.505 \\ 11.755 & 12.125 & 12.505 & 13.565 & 13.375 & 15.625 & 13.375 & 13.565 \\ 12.565 & 11.755 & 12.125 & 12.505 & 13.565 & 13.375 & 15.625 & 13.375 \\ 12.935 & 12.565 & 11.755 & 12.125 & 12.505 & 13.565 & 13.375 & 15.625 \end{pmatrix} \quad (3.86)$$

Example B3.37

What is the structure of the autocorrelation matrix of an ergodic set of 3×3 images, worked out from a single available image, using its 1D representation, and assuming that it is repeated ad infinitum?

When the samples are repeated, the value of the spatial autocorrelation function for shift h is the same as that for shift $9 - h$. Then matrix (3.85) takes the form:

$$\begin{array}{ccccccccc}
 & g_{11} & g_{21} & g_{31} & g_{12} & g_{22} & g_{32} & g_{13} & g_{23} & g_{33} \\
 g_{11} & A & B & C & D & E & E & D & C & B \\
 g_{21} & B & A & B & C & D & E & E & D & C \\
 g_{31} & C & B & A & B & C & D & E & E & D \\
 g_{12} & D & C & B & A & B & C & D & E & E \\
 g_{22} & E & D & C & B & A & B & C & D & E \\
 g_{32} & E & E & D & C & B & A & B & C & D \\
 g_{13} & D & E & E & D & C & B & A & B & C \\
 g_{23} & C & D & E & E & D & C & B & A & B \\
 g_{33} & B & C & D & E & E & D & C & B & A
 \end{array} \tag{3.87}$$

A matrix with the same value along each main diagonal direction is called **Toeplitz**.

How can we transform the image so that its autocorrelation matrix is diagonal?

Let us say that the original image is g of size $N \times N$ and its transformed version is \tilde{g} . We shall use the vector versions of them, \mathbf{g} and $\tilde{\mathbf{g}}$ respectively; ie stack the columns of the two matrices one below the other to create two $N^2 \times 1$ vectors. We assume that the transformation we are seeking has the form

$$\tilde{\mathbf{g}} = A(\mathbf{g} - \mathbf{m}) \tag{3.88}$$

where the transformation matrix A is $N^2 \times N^2$ and the arbitrary vector \mathbf{m} is $N^2 \times 1$. We assume that the image is ergodic. The mean vector of the transformed image is given by

$$\mu_{\tilde{\mathbf{g}}} = E\{\tilde{\mathbf{g}}\} = E\{A(\mathbf{g} - \mathbf{m})\} = AE\{\mathbf{g}\} - A\mathbf{m} = A(\mu_{\mathbf{g}} - \mathbf{m}) \tag{3.89}$$

where we have used the fact that A and \mathbf{m} are nonrandom and, therefore, the expectation operator leaves them unaffected. Notice that although we talk about expectation value and use the same notation as the notation used for ensemble averaging, because of the assumed ergodicity, $E\{\tilde{\mathbf{g}}\}$ means nothing else than finding the average grey value of image \tilde{g} and creating an $N^2 \times 1$ vector with all its elements equal to this average grey value. If ergodicity had not been assumed, $E\{\tilde{\mathbf{g}}\}$ would have meant that the averaging would have to be done over all the versions of image $\tilde{\mathbf{g}}$, and its elements most likely would not have been all equal, unless the ensemble were stationary with respect to the mean.

We can conveniently choose $\mathbf{m} = \boldsymbol{\mu}_g = E\{\mathbf{g}\}$ in (3.89). Then $\boldsymbol{\mu}_{\tilde{\mathbf{g}}} = 0$; ie the transformed image will have zero mean.

The autocorrelation function of $\tilde{\mathbf{g}}$ then is the same as its autocovariance function and is computed as:

$$\begin{aligned} C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} &= E\{\tilde{\mathbf{g}}\tilde{\mathbf{g}}^T\} = E\{A(\mathbf{g} - \boldsymbol{\mu}_g)[A(\mathbf{g} - \boldsymbol{\mu}_g)]^T\} \\ &= E\{A(\mathbf{g} - \boldsymbol{\mu}_g)(\mathbf{g} - \boldsymbol{\mu}_g)^T A^T\} \\ &= A \underbrace{E\left\{(\mathbf{g} - \boldsymbol{\mu}_g)(\mathbf{g} - \boldsymbol{\mu}_g)^T\right\}}_{\text{autocovariance of the untransformed image}} A^T \end{aligned} \quad (3.90)$$

Note that because matrix A is not a random field, it is not affected by the expectation operator. Also note that, due to ergodicity, the ensemble autocovariance function of the untransformed image may be replaced by its spatial autocovariance function.

So: $C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} = AC_{gg}A^T$. Then it is obvious that $C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}}$ is the diagonalised version of the covariance matrix of the untransformed image. Such a diagonalisation is achieved if the transformation matrix A is the matrix formed by the eigenvectors of the autocovariance matrix of the image, used as rows. The diagonal elements of $C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}}$ then are the eigenvalues of matrix C_{gg} . The autocovariance matrix of the image may be calculated from the image itself, since we assumed ergodicity (no large ensemble of similar images is needed). Equation (3.88) then represents the Karhunen-Loeve transform of image g .

How do we compute the K-L transform of an image in practice?

Step 1: Compute the mean of input image G , of size $M \times N$, and remove it from all its elements, to form image \tilde{G} .

Step 2: Write the columns of \tilde{G} one under the other to form a column vector $\tilde{\mathbf{g}}$.

Step 3: Compute the spatial autocorrelation function $C(h) \equiv \sum_i \tilde{g}(i)\tilde{g}(i+h)/(MN)$ and from its elements the autocorrelation matrix C (of size $MN \times MN$), as in example 3.37. (Alternatively, you may use the formula derived in Box 3.2.)

Step 4: Compute the eigenvalues and eigenvectors of C . If C has E nonzero eigenvectors, you will produce E vectors of size $MN \times 1$.

Step 5: Arrange the eigenvectors in decreasing order of the corresponding eigenvalues.

Step 6: Create a matrix A (of size $E \times MN$) made up from the eigenvectors written one under the other as its rows.

Step 7: Multiply matrix A with the image vector $\tilde{\mathbf{g}}$, to produce the transformed vector \mathbf{g}' of size $E \times 1$. This vector is the K-L transform of the input image.

In some cases you may visualise the transform by the following steps:

Step 8: If $E = MN$, you may wrap vector \mathbf{g}' into an image G' of size $M \times N$.

Step 9: Before you display G' you may scale it to the range $[0, 255]$ to avoid the negative values it will contain, and round its values to the nearest integer.

To produce the basis images of the K-L transform, you need Step 10:

Step 10: Wrap every eigenvector you produced in Step 4 to form an image $M \times N$ in size. These will be the E basis images, appropriate for the representation of all images of the same size that have the same autocovariance matrix as image G .

To reproduce the original image as a linear superposition of the basis images, you need Step 11:

Step 11: Multiply each basis image with the corresponding element of \mathbf{g}' and sum the results up.

How do we compute the Karhunen-Loeve (K-L) transform of an ensemble of images?

The algorithm is the same as above. You only need to replace the first three steps, with the following:

Step 1_{ensemble}: Compute the mean image and remove it from all images.

Step 2_{ensemble}: Write each image as a column vector.

Step 3_{ensemble}: Compute the ensemble autocorrelation matrix of all these vectors.

Is the assumption of ergodicity realistic?

The assumption of ergodicity is not realistic. It is unrealistic to expect that a single image will be so large and it will include so much variation in its content that all the diversity represented by a collection of images will be captured by it. Only images consisting of pure random noise satisfy this assumption. So, people often divide an image into small patches, which are expected to be uniform, apart from variation due to noise, and apply the ergodicity assumption to each patch separately.

Box 3.2. How can we calculate the spatial autocorrelation matrix of an image, when it is represented by a vector?

To define a general formula for the spatial autocorrelation matrix of an image G , we must first establish a correspondence between the index of an element of the vector representation \mathbf{g} of the image and the two indices that identify the position of a pixel in the image. Let us assume that the image is of size $N \times N$ and the coordinates of a pixel take values from 0 to $N - 1$. We wish to give an index i to a pixel in the 1D string we shall create, taking values from 0 to $N^2 - 1$. Since the vector representation of an image is created by placing its columns one under the other, pixel (k_i, l_i) will be the i^{th} element of the vector, where:

$$i = l_i N + k_i \quad (3.91)$$

We can solve the above expression for l_i and k_i in terms of i as follows:

$$k_i = i \text{ modulo } N$$

$$l_i = \frac{i - k_i}{N} = \left\lfloor \frac{i}{N} \right\rfloor \quad (3.92)$$

Operator $\lfloor \cdot \rfloor$ is the **floor** operator and it returns the integer part of a number.

Element $C(h)$ of the autocorrelation function may be written as

$$C(h) \equiv \frac{1}{N^2} \sum_{i=0}^{N^2-1} g_i g_{i+h} \quad \text{for } h = 0, \dots, N^2 - 1 \quad (3.93)$$

where we must remember that $g_{i+h} = g_{i+h-N^2}$ if $i + h \geq N^2$.

We observe that:

$$k_{i+h} = (i + h) \text{ modulo } N$$

$$l_{i+h} = \frac{i + h - k_{i+h}}{N} = \left\lfloor \frac{i + h}{N} \right\rfloor \quad (3.94)$$

Since $g_{i+h} = g_{i+h-N^2}$ if $i + h \geq N^2$, instead of just $i + h$, we may write $(i + h) \text{ modulo } N^2$. Then the above equations become:

$$\begin{aligned} k_{i+h} &= [(i + h) \text{ modulo } N^2] \text{ modulo } N = (i + h) \text{ modulo } N \\ l_{i+h} &= \frac{(i + h) \text{ modulo } N^2 - k_{i+h}}{N} = \left\lfloor \frac{(i + h) \text{ modulo } N^2}{N} \right\rfloor \end{aligned} \quad (3.95)$$

Element $C(h)$ of the autocorrelation function of the 1D image representation, \mathbf{g} , may then be computed from its 2D representation, G , directly, using

$$C(h) = \frac{1}{N^2} \sum_{i=0}^{N^2-1} G(k_i, l_i) G(k_{i+h}, l_{i+h}) \quad (3.96)$$

where (k_i, l_i) are given by equations (3.92) and (k_{i+h}, l_{i+h}) are given by equations (3.95).

Elements $C(h)$ may be used to build the 2D autocorrelation matrix of the vector representation of the image (see example 3.36).

Example B3.38

Work out the formula for computing the spatial autocorrelation function of the 1D representation g of an image G of size $M \times N$, when its indices k and l take values from 1 to M and from 1 to N , respectively, and index i has to take values from 1 to MN .

$$\text{M elements in each column} \left\{ \begin{array}{cccc} k=1 & l=1 & l=2 & \dots \\ \cdot & \cdot & \cdot & \cdot \\ k=2 & \cdot & \cdot & \cdot \\ \vdots & \vdots & \vdots & \square \\ \cdot & \cdot & \cdot & \nearrow (k_i, l_i) \\ k=M & \cdot & \cdot & \cdot \end{array} \right.$$

Figure 3.8: The pixel at position (k_i, l_i) has before it $l_i - 1$ columns, with M elements each, and is the k_i^{th} element in its own column.

Consider the image of figure 3.8 where each dot represents a pixel. The (k_i, l_i) dot, representing the i^{th} element of the vector representation of the image, will have index i equal to:

$$i = (l_i - 1)M + k_i \quad (3.97)$$

Then:

$$\begin{aligned} k_i &= i \text{ modulo } M \\ l_i &= 1 + \frac{i - k_i}{M} = \left\lfloor \frac{i}{M} \right\rfloor + 1 \end{aligned} \quad (3.98)$$

Element $C(h)$ of the autocorrelation function may then be computed using

$$C(h) = \frac{1}{NM} \sum_{i=0}^{NM} G(k_i, l_i)G(k_{i+h}, l_{i+h}) \quad (3.99)$$

where (k_i, l_i) are given by equations (3.98) and (k_{i+h}, l_{i+h}) are given by:

$$\begin{aligned} k_{i+h} &= [(i + h) \text{ modulo } MN] \text{ modulo } M \\ l_{i+h} &= 1 + \frac{(i + h) \text{ modulo } MN - k_{i+h}}{M} = \left\lfloor \frac{(i + h) \text{ modulo } MN}{M} \right\rfloor + 1 \end{aligned} \quad (3.100)$$

Example B3.39

Assuming ergodicity, calculate the K-L transform of an ensemble of images, one of which is:

$$\begin{pmatrix} 3 & 5 & 2 & 3 \\ 5 & 4 & 4 & 3 \\ 2 & 2 & 6 & 6 \\ 6 & 5 & 4 & 6 \end{pmatrix} \quad (3.101)$$

The mean value of this image is $66/16 = 4.125$. We subtract this from all the elements of the image and then we compute its spatial autocorrelation function $C(h)$ for $h = 0, 1, \dots, 15$ and use its values to construct the autocorrelation matrix with banded structure similar to that shown in (3.87), but of size 16×16 instead of 9×9 . The elements of $C(h)$ are:

$$\begin{matrix} 2.11 & -0.52 & -0.39 & -0.45 & 0.3 & 0.23 & 0.36 & -0.27 \\ -0.64 & -0.27 & 0.36 & 0.23 & 0.3 & -0.45 & -0.39 & -0.52 \end{matrix} \quad (3.102)$$

Then we compute the eigenvectors of the autocorrelation matrix and sort them so that their corresponding eigenvalues are in decreasing order. Finally, we use them as rows to form the transformation matrix A with which our image can be transformed:

$$A = \begin{pmatrix} 0.35 & 0.14 & -0.25 & -0.33 & 0.00 & 0.33 & 0.25 & -0.14 \\ 0.00 & -0.33 & -0.25 & 0.14 & 0.35 & 0.14 & -0.25 & -0.33 \\ 0.25 & -0.25 & 0.25 & -0.25 & 0.25 & -0.25 & 0.25 & -0.25 \\ -0.35 & 0.13 & 0.25 & -0.33 & 0.00 & 0.33 & -0.25 & -0.14 \\ 0.00 & -0.33 & 0.25 & 0.14 & -0.35 & 0.13 & 0.25 & -0.33 \\ 0.02 & -0.15 & 0.26 & -0.33 & 0.35 & -0.32 & 0.24 & -0.12 \\ 0.35 & -0.32 & 0.24 & -0.12 & -0.02 & 0.15 & -0.26 & 0.33 \\ 0.04 & 0.35 & -0.03 & -0.35 & 0.03 & 0.35 & -0.03 & -0.35 \\ -0.35 & 0.03 & 0.35 & -0.03 & -0.35 & 0.03 & 0.35 & -0.03 \\ 0.10 & -0.31 & 0.34 & -0.17 & -0.10 & 0.31 & -0.34 & 0.17 \\ 0.34 & -0.17 & -0.10 & 0.31 & -0.34 & 0.17 & 0.10 & -0.31 \\ 0.34 & 0.27 & 0.17 & 0.03 & -0.10 & -0.23 & -0.31 & -0.35 \\ 0.10 & 0.23 & 0.31 & 0.35 & 0.34 & 0.27 & 0.17 & 0.03 \\ 0.04 & 0.27 & 0.35 & 0.22 & -0.04 & -0.27 & -0.35 & -0.22 \\ 0.35 & 0.22 & -0.04 & -0.27 & -0.35 & -0.22 & 0.04 & 0.27 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

$$\begin{pmatrix} -0.35 & -0.14 & 0.25 & 0.33 & 0.00 & -0.33 & -0.25 & 0.14 \\ 0.00 & 0.32 & 0.25 & -0.14 & -0.35 & -0.14 & 0.25 & 0.33 \\ 0.25 & -0.25 & 0.25 & -0.25 & 0.25 & -0.25 & 0.25 & -0.25 \\ 0.35 & -0.13 & -0.25 & 0.33 & 0.00 & -0.33 & 0.25 & 0.14 \\ 0.00 & 0.33 & -0.25 & -0.14 & 0.35 & -0.13 & -0.25 & 0.33 \\ -0.02 & 0.15 & -0.26 & 0.33 & -0.35 & 0.32 & -0.24 & 0.12 \\ -0.35 & 0.32 & -0.24 & 0.12 & 0.02 & -0.15 & 0.26 & -0.33 \\ 0.03 & 0.35 & -0.03 & -0.35 & 0.03 & 0.35 & -0.03 & -0.35 \\ -0.35 & 0.03 & 0.35 & -0.03 & -0.35 & 0.03 & 0.35 & -0.03 \\ 0.10 & -0.31 & 0.34 & -0.17 & -0.10 & 0.31 & -0.34 & 0.17 \\ 0.34 & -0.17 & -0.10 & 0.31 & -0.34 & 0.17 & 0.10 & -0.31 \\ -0.34 & -0.27 & -0.17 & -0.03 & 0.10 & 0.23 & 0.31 & 0.35 \\ -0.10 & -0.23 & -0.31 & -0.35 & -0.34 & -0.27 & -0.17 & -0.03 \\ 0.04 & 0.27 & 0.35 & 0.22 & -0.04 & -0.27 & -0.35 & -0.22 \\ 0.35 & 0.22 & -0.04 & -0.27 & -0.35 & -0.22 & 0.04 & 0.27 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix} \quad (3.103)$$

The corresponding eigenvalues are:

$$\begin{array}{cccccccc} 4.89 & 4.89 & 4 & 2.73 & 2.73 & 2.68 & 2.68 & 2.13 \\ & & & & & & & \\ 2.13 & 1.67 & 1.67 & 0.70 & 0.70 & 0.08 & 0.08 & 0 \end{array} \quad (3.104)$$

We stack the columns of the image minus its mean one below the other to form vector $\mathbf{g} - \mu_{\mathbf{g}}$:

$$\begin{aligned} (\mathbf{g} - \mu_{\mathbf{g}})^T = & (-1.125, 0.875, -2.125, 1.875, 0.875, \\ & -0.125, -2.125, 0.875, -2.125, -0.125, \\ & 1.875, -0.125, -1.125, -1.125, 1.875, 1.875) \end{aligned} \quad (3.105)$$

We then multiply this vector with matrix A to derive the Karhunen-Loeve transform of the image. In matrix form this is given by:

$$\tilde{\mathbf{g}} = \begin{pmatrix} 0.30 & -2.30 & 0.89 & 0.03 \\ 3.11 & -2.29 & -0.76 & 0.22 \\ -2.00 & -0.34 & -1.66 & -0.34 \\ -0.43 & -1.86 & 1.18 & 0.00 \end{pmatrix} \quad (3.106)$$

Is the mean of the transformed image expected to be really 0?

No. The choice of vector \mathbf{m} in equation (3.89) is meant to make the average of $\tilde{\mathbf{g}}$ 0. However, that calculation is based on ensemble averages. In practice, the i^{th} element of $\tilde{\mathbf{g}}$ is given by:

$$\tilde{g}_i = \sum_k A_{ik}(g_k - \mu_g) \quad (3.107)$$

To compute the average of the transformed image we sum over all values of i :

$$\sum_i \tilde{g}_i = \sum_i \sum_k A_{ik}(g_k - \mu_g) = \sum_k (g_k - \mu_g) \sum_i A_{ik} \quad (3.108)$$

Obviously, $\sum_k (g_k - \mu_g) = 0$ given that μ_g is the average value of the elements of \mathbf{g} and $\boldsymbol{\mu}_{\mathbf{g}}$ is a vector made up from elements all equal to μ_g . The only way $\mu_{\tilde{\mathbf{g}}}$ is zero is for $\sum_i \tilde{g}_i$ to be 0, and this will happen only if $\sum_i A_{ik}$ is a constant number, independent of k . There is no reason for this to be true, because matrix A is made up from the eigenvectors of the covariance matrix of \mathbf{g} written as rows one under the other. There is no reason to expect the sums of the elements of all columns of A to be the same. So, in general, the average of the transformed image will not be 0 because we compute this average as a spatial average and not as the ensemble average according to the theory.

How can we approximate an image using its K-L transform?

The K-L transform of an image is given by

$$\tilde{\mathbf{g}} = A(\mathbf{g} - \boldsymbol{\mu}_{\mathbf{g}}) \quad (3.109)$$

where $\boldsymbol{\mu}_{\mathbf{g}}$ is an $N^2 \times 1$ vector with elements equal to the average grey value of the image, and A is a matrix made up from the eigenvectors of the autocorrelation matrix of image \mathbf{g} , used as rows and arranged in decreasing order of the corresponding eigenvalues. The inverse transform is:

$$\mathbf{g} = A^T \tilde{\mathbf{g}} + \boldsymbol{\mu}_{\mathbf{g}} \quad (3.110)$$

If we set equal to 0 the last few eigenvalues of the autocorrelation matrix of \mathbf{g} , matrix A will have its corresponding rows replaced by zeros, and so will the transformed image $\tilde{\mathbf{g}}$. The image we shall reconstruct then using (3.110) and the truncated version of A , or the truncated version of $\tilde{\mathbf{g}}$, will be an approximation of the original image.

What is the error with which we approximate an image when we truncate its K-L expansion?

It can be shown (see Box 3.3), that, if we truncate the K-L expansion of an image, the image will *on average* be approximated by a square error that is equal to the sum of the omitted eigenvalues of the autocovariance matrix of the image.

What are the basis images in terms of which the Karhunen-Loeve transform expands an image?

Since $\tilde{\mathbf{g}} = A(\mathbf{g} - \mu_g)$ and A is an orthogonal matrix, the inverse transformation is given by $\mathbf{g} - \mu_g = A^T \tilde{\mathbf{g}}$. We can write this expression explicitly:

$$\begin{aligned} & \begin{pmatrix} g_{11} - \mu_g \\ g_{21} - \mu_g \\ \vdots \\ g_{N1} - \mu_g \\ g_{12} - \mu_g \\ \vdots \\ g_{N2} - \mu_g \\ \vdots \\ g_{1N} - \mu_g \\ \vdots \\ g_{NN} - \mu_g \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{2,1} & \dots & a_{N^2,1} \\ a_{1,2} & a_{2,2} & \dots & a_{N^2,2} \\ \vdots & \vdots & & \vdots \\ a_{1,N} & a_{2,N} & \dots & a_{N^2,N} \\ a_{1,N+1} & a_{2,N+1} & \dots & a_{N^2,N+1} \\ \vdots & \vdots & & \vdots \\ a_{1,2N} & a_{2,2N} & \dots & a_{N^2,2N} \\ \vdots & \vdots & & \vdots \\ a_{1,N^2-N+1} & a_{2,N^2-N+1} & \dots & a_{N^2,N^2-N+1} \\ \vdots & \vdots & & \vdots \\ a_{1,N^2} & a_{2,N^2} & \dots & a_{N^2,N^2} \end{pmatrix} \begin{pmatrix} \tilde{g}_{11} \\ \tilde{g}_{21} \\ \vdots \\ \tilde{g}_{N1} \\ \tilde{g}_{12} \\ \vdots \\ \tilde{g}_{N2} \\ \vdots \\ \tilde{g}_{1N} \\ \vdots \\ \tilde{g}_{NN} \end{pmatrix} \\ & \Rightarrow \begin{cases} g_{11} - \mu_g &= a_{1,1}\tilde{g}_{11} + a_{2,1}\tilde{g}_{21} + \dots + a_{N^2,1}\tilde{g}_{NN} \\ g_{21} - \mu_g &= a_{1,2}\tilde{g}_{11} + a_{2,2}\tilde{g}_{21} + \dots + a_{N^2,2}\tilde{g}_{NN} \\ \dots & \dots \\ g_{N1} - \mu_g &= a_{1,N}\tilde{g}_{11} + a_{2,N}\tilde{g}_{21} + \dots + a_{N^2,N}\tilde{g}_{NN} \\ g_{12} - \mu_g &= a_{1,N+1}\tilde{g}_{11} + a_{2,N+1}\tilde{g}_{21} + \dots + a_{N^2,N+1}\tilde{g}_{NN} \\ \dots & \dots \\ g_{N2} - \mu_g &= a_{1,2N}\tilde{g}_{11} + a_{2,2N}\tilde{g}_{21} + \dots + a_{N^2,2N}\tilde{g}_{NN} \\ \dots & \dots \\ g_{1N} - \mu_g &= a_{1,N^2-N+1}\tilde{g}_{11} + a_{2,N^2-N+1}\tilde{g}_{21} + \dots + a_{N^2,N^2-N+1}\tilde{g}_{NN} \\ \dots & \dots \\ g_{NN} - \mu_g &= a_{1,N^2}\tilde{g}_{11} + a_{2,N^2}\tilde{g}_{21} + \dots + a_{N^2,N^2}\tilde{g}_{NN} \end{cases} \end{aligned}$$

We can rearrange these equations into matrix form:

$$\begin{aligned} & \begin{pmatrix} g_{11} - \mu_g & g_{12} - \mu_g & \dots & g_{N1} - \mu_g \\ g_{21} - \mu_g & g_{22} - \mu_g & \dots & g_{N2} - \mu_g \\ \vdots & \vdots & & \vdots \\ g_{N1} - \mu_g & g_{N2} - \mu_g & \dots & g_{NN} - \mu_g \end{pmatrix} \\ & = \tilde{g}_{11} \begin{pmatrix} a_{1,1} & a_{1,N+1} & \dots & a_{1,N^2-N+1} \\ a_{1,2} & a_{1,N+2} & \dots & a_{1,N^2-N+2} \\ \vdots & \vdots & & \vdots \\ a_{1,N} & a_{1,2N} & \dots & a_{1,N^2} \end{pmatrix} + \tilde{g}_{21} \begin{pmatrix} a_{2,1} & a_{2,N+1} & \dots & a_{2,N^2-N+1} \\ a_{2,2} & a_{2,N+2} & \dots & a_{2,N^2-N+2} \\ \vdots & \vdots & & \vdots \\ a_{2,N} & a_{2,2N} & \dots & a_{2,N^2} \end{pmatrix} + \\ & + \dots + \tilde{g}_{NN} \begin{pmatrix} a_{N^2,1} & a_{N^2,N+1} & \dots & a_{N^2,N^2-N+1} \\ a_{N^2,2} & a_{N^2,N+2} & \dots & a_{N^2,N^2-N+2} \\ \vdots & \vdots & & \vdots \\ a_{N^2,N} & a_{N^2,2N} & \dots & a_{N^2,N^2} \end{pmatrix} \quad (3.111) \end{aligned}$$

This expression makes it obvious that the eigenimages in terms of which the K-L transform expands an image are formed from the eigenvectors of its spatial autocorrelation matrix, by writing them in matrix form; ie by using the first N elements of an eigenvector to form the first column of the corresponding eigenimage, the next N elements to form the next column and so on. The coefficients of this expansion are the elements of the transformed image.

We may understand this more easily by thinking in terms of the multidimensional space where each image is represented by a point (see figure 3.9). The tip of each unit vector along each of the axes, which we specified to be the symmetry axes of the cloud of points, represents a point in this space, ie an image. This is the elementary image that corresponds to that axis and the unit vector is nothing else than an eigenvector of the autocovariance matrix of the set of dots.

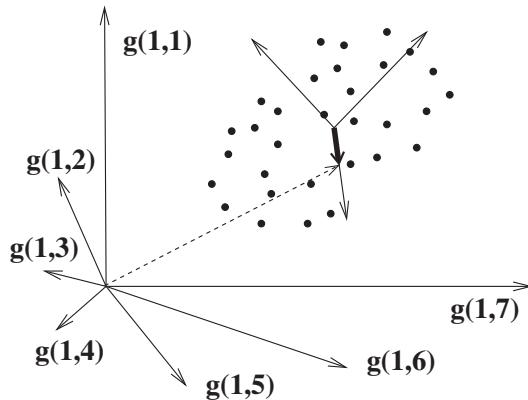


Figure 3.9: An image is a point in a multidimensional space where the value of each pixel is measured along a different axis. The ensemble of images is a cloud of points. The new coordinate system created, centred at the cloud of points, allows each point to be expressed by its coordinates along these new axes, each one of which is defined by a unit vector. These unit vectors are the eigenvectors of the autocovariance matrix of the cloud of points. The tip of the thick unit vector along one of the new axes in this figure represents one of the basis images created to represent the images in the ensemble. The coordinates of this basis image, ie its pixel values, are the components of its position vector in the original coordinate system, represented by the dashed vector.

Example 3.40

Consider a 3×3 image with column representation \mathbf{g} . Write down an expression for the K-L transform of the image in terms of the elements of \mathbf{g} and the elements a_{ij} of the transformation matrix A . Calculate an approximation to the image \mathbf{g} by setting the last six rows of A to zero. Show that the approximation will be a 9×1 vector with the first three elements equal to those of the full transformation of \mathbf{g} and the remaining

six elements zero.

Assume that μ_g is the average grey value of image \mathbf{g} . Then the transformed image will have the form:

$$\begin{aligned}
 \begin{pmatrix} \tilde{g}_{11} \\ \tilde{g}_{21} \\ \tilde{g}_{31} \\ \tilde{g}_{12} \\ \tilde{g}_{22} \\ \tilde{g}_{32} \\ \tilde{g}_{13} \\ \tilde{g}_{23} \\ \tilde{g}_{33} \end{pmatrix} &= \begin{pmatrix} a_{11} & a_{12} & \dots & a_{19} \\ a_{21} & a_{22} & \dots & a_{29} \\ a_{31} & a_{32} & \dots & a_{39} \\ a_{41} & a_{42} & \dots & a_{49} \\ \vdots & \vdots & & \vdots \\ \tilde{g}_{13} & \vdots & \vdots & \vdots \\ \tilde{g}_{23} & \vdots & \vdots & \vdots \\ a_{91} & a_{92} & \dots & a_{99} \end{pmatrix} \begin{pmatrix} g_{11} - \mu_g \\ g_{21} - \mu_g \\ g_{31} - \mu_g \\ g_{12} - \mu_g \\ g_{22} - \mu_g \\ g_{32} - \mu_g \\ g_{13} - \mu_g \\ g_{23} - \mu_g \\ g_{33} - \mu_g \end{pmatrix} \\
 &= \begin{pmatrix} a_{11}(g_{11} - \mu_g) + a_{12}(g_{21} - \mu_g) + \dots + a_{19}(g_{33} - \mu_g) \\ a_{21}(g_{11} - \mu_g) + a_{22}(g_{21} - \mu_g) + \dots + a_{29}(g_{33} - \mu_g) \\ a_{31}(g_{11} - \mu_g) + a_{32}(g_{21} - \mu_g) + \dots + a_{39}(g_{33} - \mu_g) \\ a_{41}(g_{11} - \mu_g) + a_{42}(g_{21} - \mu_g) + \dots + a_{49}(g_{33} - \mu_g) \\ \vdots \\ \vdots \\ a_{91}(g_{11} - \mu_g) + a_{92}(g_{21} - \mu_g) + \dots + a_{99}(g_{33} - \mu_g) \end{pmatrix} \quad (3.112)
 \end{aligned}$$

If we set $a_{41} = a_{42} = \dots = a_{49} = a_{51} = \dots = a_{59} = \dots = a_{99} = 0$, clearly the last six rows of the above vector will be 0 and the truncated transformation of the image will be vector:

$$\tilde{\mathbf{g}}' = (\tilde{g}_{11} \quad \tilde{g}_{21} \quad \tilde{g}_{31} \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0)^T \quad (3.113)$$

According to formula (3.111), the approximation of the image is then:

$$\begin{aligned}
 \begin{pmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{pmatrix} &= \begin{pmatrix} \mu_g & \mu_g & \mu_g \\ \mu_g & \mu_g & \mu_g \\ \mu_g & \mu_g & \mu_g \end{pmatrix} + \tilde{g}_{11} \begin{pmatrix} a_{11} & a_{14} & a_{17} \\ a_{12} & a_{15} & a_{18} \\ a_{13} & a_{16} & a_{19} \end{pmatrix} \\
 &\quad + \tilde{g}_{21} \begin{pmatrix} a_{21} & a_{24} & a_{27} \\ a_{22} & a_{25} & a_{28} \\ a_{23} & a_{26} & a_{29} \end{pmatrix} + \tilde{g}_{31} \begin{pmatrix} a_{31} & a_{34} & a_{37} \\ a_{32} & a_{35} & a_{38} \\ a_{33} & a_{36} & a_{39} \end{pmatrix} \quad (3.114)
 \end{aligned}$$

Example B3.41

Show that if A is an $N^2 \times N^2$ matrix the i^{th} row of which is vector \mathbf{u}_i^T and C_2 an $N^2 \times N^2$ matrix with all its elements zero except the element at position (2, 2), which is equal to c_2 , then:

$$A^T C_2 A = c_2 \mathbf{u}_2 \mathbf{u}_2^T \quad (3.115)$$

Assume that u_{ij} indicates the j^{th} component of vector \mathbf{u}_i . Then:

$$\begin{aligned} A^T C_2 A &= \begin{pmatrix} u_{11} & u_{21} & \dots & u_{N^2 1} \\ u_{12} & u_{22} & \dots & u_{N^2 2} \\ u_{13} & u_{23} & \dots & u_{N^2 3} \\ \vdots & \vdots & & \vdots \\ u_{1N^2} & u_{2N^2} & \dots & u_{N^2 N^2} \end{pmatrix} \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & c_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \\ &\quad \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1N^2} \\ u_{21} & u_{22} & \dots & u_{2N^2} \\ u_{31} & u_{32} & \dots & u_{3N^2} \\ \vdots & \vdots & & \vdots \\ u_{N^2 1} & u_{N^2 2} & \dots & u_{N^2 N^2} \end{pmatrix} \\ &= \begin{pmatrix} u_{11} & u_{21} & \dots & u_{N^2 1} \\ u_{12} & u_{22} & \dots & u_{N^2 2} \\ u_{13} & u_{23} & \dots & u_{N^2 3} \\ \vdots & \vdots & & \vdots \\ u_{1N^2} & u_{2N^2} & \dots & u_{N^2 N^2} \end{pmatrix} \begin{pmatrix} 0 & 0 & \dots & 0 \\ c_2 u_{21} & c_2 u_{22} & \dots & c_2 u_{2N^2} \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \\ &= \begin{pmatrix} c_2 u_{21}^2 & c_2 u_{21} u_{22} & \dots & c_2 u_{21} u_{2N^2} \\ c_2 u_{22} u_{21} & c_2 u_{22}^2 & \dots & c_2 u_{22} u_{2N^2} \\ \vdots & \vdots & & \vdots \\ c_2 u_{2N^2} u_{21} & c_2 u_{2N^2} u_{22} & \dots & c_2 u_{2N^2}^2 \end{pmatrix} \\ &= c_2 \mathbf{u}_2 \mathbf{u}_2^T \end{aligned} \quad (3.116)$$

The last equality follows by observing that c_2 is a common factor of all matrix elements and after it is taken out, what remains is the outer product of vector \mathbf{u}_2 with itself.

Example B3.42

Assuming a 3×3 image, and accepting that we approximate it retaining only the first three eigenvalues of its autocovariance matrix, show that:

$$E\{\tilde{\mathbf{g}}\tilde{\mathbf{g}}'^T\} = C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} \quad (3.117)$$

Using the result of example 3.40 concerning the truncated transform of image $\tilde{\mathbf{g}}'$, we have:

$$\begin{aligned} E\{\tilde{\mathbf{g}}\tilde{\mathbf{g}}'^T\} &= E \left\{ \begin{pmatrix} \tilde{g}_{11} \\ \tilde{g}_{21} \\ \tilde{g}_{31} \\ \tilde{g}_{12} \\ \tilde{g}_{22} \\ \tilde{g}_{32} \\ \tilde{g}_{13} \\ \tilde{g}_{23} \\ \tilde{g}_{33} \end{pmatrix} \begin{pmatrix} \tilde{g}_{11} & \tilde{g}_{21} & \tilde{g}_{31} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \right\} \\ &= E \left\{ \begin{pmatrix} \tilde{g}_{11}^2 & \tilde{g}_{11}\tilde{g}_{21} & \tilde{g}_{11}\tilde{g}_{31} & 0 & 0 & 0 & 0 & 0 & 0 \\ \tilde{g}_{21}\tilde{g}_{11} & \tilde{g}_{21}^2 & \tilde{g}_{21}\tilde{g}_{31} & 0 & 0 & 0 & 0 & 0 & 0 \\ \tilde{g}_{31}\tilde{g}_{11} & \tilde{g}_{31}\tilde{g}_{21} & \tilde{g}_{31}^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & & & & \\ \tilde{g}_{33}\tilde{g}_{11} & \tilde{g}_{33}\tilde{g}_{21} & \tilde{g}_{33}\tilde{g}_{31} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \right\} \\ &= \begin{pmatrix} E\{\tilde{g}_{11}^2\} & E\{\tilde{g}_{11}\tilde{g}_{21}\} & E\{\tilde{g}_{11}\tilde{g}_{31}\} & 0 & 0 & 0 & 0 & 0 & 0 \\ E\{\tilde{g}_{21}\tilde{g}_{11}\} & E\{\tilde{g}_{21}^2\} & E\{\tilde{g}_{21}\tilde{g}_{31}\} & 0 & 0 & 0 & 0 & 0 & 0 \\ E\{\tilde{g}_{31}\tilde{g}_{11}\} & E\{\tilde{g}_{31}\tilde{g}_{21}\} & E\{\tilde{g}_{31}^2\} & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & & & & \\ E\{\tilde{g}_{33}\tilde{g}_{11}\} & E\{\tilde{g}_{33}\tilde{g}_{21}\} & E\{\tilde{g}_{33}\tilde{g}_{31}\} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (3.118) \end{aligned}$$

The transformed image $\tilde{\mathbf{g}}$ is constructed in such a way that it has zero mean and all the off-diagonal elements of its covariance matrix are equal to 0. Therefore, we have:

$$E\{\tilde{\mathbf{g}}\tilde{\mathbf{g}}'^T\} = \begin{pmatrix} E\{\tilde{g}_{11}^2\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & E\{\tilde{g}_{21}^2\} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & E\{\tilde{g}_{31}^2\} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} \quad (3.119)$$

Box 3.3. What is the error of the approximation of an image using the Karhunen-Loeve transform?

We shall show now that the Karhunen-Loeve transform not only expresses an image in terms of uncorrelated data, but also, if truncated after a certain term, it can be used to approximate the image in the least *mean square error* sense.

Assume that the image is of size $N \times N$. The transformation has the form:

$$\tilde{\mathbf{g}} = A\mathbf{g} - A\mu_{\mathbf{g}} \Rightarrow \mathbf{g} = A^T\tilde{\mathbf{g}} + \mu_{\mathbf{g}} \quad (3.120)$$

We assume that we have ordered the eigenvalues of $C_{\mathbf{gg}}$ in decreasing order. Assume that we decide to neglect the last few eigenvalues and, say, we retain the first K most significant ones. $C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}}$ is an $N^2 \times N^2$ matrix and its truncated version, $C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}}$, has the last $N^2 - K$ diagonal elements 0. The transformation matrix A^T was an $N^2 \times N^2$ matrix, the columns of which were the eigenvectors of $C_{\mathbf{gg}}$. Neglecting the $N^2 - K$ eigenvalues is like omitting $N^2 - K$ eigenvectors, so the new transformation matrix A'^T has the last $N^2 - K$ columns 0. The approximated image then is:

$$\mathbf{g}' = A'^T\tilde{\mathbf{g}}' + \mu_{\mathbf{g}} \quad (3.121)$$

The error of the approximation is $\mathbf{g} - \mathbf{g}' = A^T\tilde{\mathbf{g}} - A'^T\tilde{\mathbf{g}}'$. The norm of this matrix is

$$\|\mathbf{g} - \mathbf{g}'\| = \text{trace} \left[(\mathbf{g} - \mathbf{g}')(\mathbf{g} - \mathbf{g}')^T \right] \quad (3.122)$$

where trace means the sum of the diagonal elements of a square matrix. Therefore, the mean square error is:

$$E\{\|\mathbf{g} - \mathbf{g}'\|\} = E \left\{ \text{trace} \left[(\mathbf{g} - \mathbf{g}')(\mathbf{g} - \mathbf{g}')^T \right] \right\} \quad (3.123)$$

We can exchange the order of taking the expectation value and taking the trace:

$$\begin{aligned} E\{\|\mathbf{g} - \mathbf{g}'\|\} &= \text{trace} \left[E \left\{ (\mathbf{g} - \mathbf{g}')(\mathbf{g} - \mathbf{g}')^T \right\} \right] \\ &= \text{trace} \left[E \left\{ (A^T\tilde{\mathbf{g}} - A'^T\tilde{\mathbf{g}}')(A^T\tilde{\mathbf{g}} - A'^T\tilde{\mathbf{g}}')^T \right\} \right] \\ &= \text{trace} \left[E \left\{ (A^T\tilde{\mathbf{g}} - A'^T\tilde{\mathbf{g}}')(\tilde{\mathbf{g}}^T A - \tilde{\mathbf{g}}'^T A') \right\} \right] \\ &= \text{trace} \left[E \left\{ A^T\tilde{\mathbf{g}}\tilde{\mathbf{g}}^T A - A^T\tilde{\mathbf{g}}\tilde{\mathbf{g}}'^T A' - A'^T\tilde{\mathbf{g}}'\tilde{\mathbf{g}}^T A + A'^T\tilde{\mathbf{g}}'\tilde{\mathbf{g}}'^T A' \right\} \right] \end{aligned} \quad (3.124)$$

Matrices A and A' are fixed, so the expectation operator does not affect them. Therefore:

$$\begin{aligned} E\{\|\mathbf{g} - \mathbf{g}'\|\} &= \text{trace} \left[A^T E\{\tilde{\mathbf{g}}\tilde{\mathbf{g}}^T\} A - A^T E\{\tilde{\mathbf{g}}\tilde{\mathbf{g}}'^T\} A' \right. \\ &\quad \left. - A'^T E\{\tilde{\mathbf{g}}'\tilde{\mathbf{g}}^T\} A + A'^T E\{\tilde{\mathbf{g}}'\tilde{\mathbf{g}}'^T\} A' \right] \end{aligned} \quad (3.125)$$

In this expression we recognise $E\{\tilde{\mathbf{g}}\tilde{\mathbf{g}}^T\}$ and $E\{\tilde{\mathbf{g}}'\tilde{\mathbf{g}}'^T\}$ as the correlation matrices of the set of images before and after the transformation: $C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}}$ and $C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}}$.

Matrix $\tilde{\mathbf{g}}\tilde{\mathbf{g}}'^T$ is the product of a vector and its transpose but with the last $N^2 - K$ components of the transpose replaced by 0. The expectation operator will make all the off-diagonal elements of $\tilde{\mathbf{g}}\tilde{\mathbf{g}}'$ zero anyway (since the transformation is such that its autocorrelation matrix has 0 all the off-diagonal elements). The fact that the last $N^2 - K$ elements of $\tilde{\mathbf{g}}'^T$ are 0 too will also make the last $N^2 - K$ diagonal elements 0 (see example 3.42). So, the result is:

$$E\{\tilde{\mathbf{g}}\tilde{\mathbf{g}}'^T\} = C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} \quad (3.126)$$

Similar reasoning leads to:

$$E\{\tilde{\mathbf{g}}'\tilde{\mathbf{g}}^T\} = C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} \quad (3.127)$$

So:

$$E\{\|\mathbf{g} - \mathbf{g}'\|\} = \text{trace} \left[A^T C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A - A^T C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A' - A'^T C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A + A'^T C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A' \right] \quad (3.128)$$

Consider the sum: $-A^T C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A' + A'^T C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A' = -(A - A')^T C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A'$. We can partition A in two sections, a $K \times N^2$ submatrix A_1 and an $(N^2 - K) \times N^2$ submatrix A_2 . A' consists of A_1 and an $(N^2 - K) \times N^2$ submatrix with all its elements zero:

$$\begin{aligned} A &= \begin{pmatrix} A_1 \\ \cdots \\ A_2 \end{pmatrix} & A' &= \begin{pmatrix} A_1 \\ \cdots \\ \mathbf{0} \end{pmatrix} \Rightarrow \\ A - A' &= \begin{pmatrix} \mathbf{0} \\ \cdots \\ A_2 \end{pmatrix} & \text{and} & (A - A')^T = (\underbrace{\mathbf{0}}_{N^2 \times K} \mid \underbrace{A_2^T}_{N^2 \times (N^2 - K)}) \end{aligned} \quad (3.129)$$

Then $(A - A')^T C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A' = (\mathbf{0} \mid A_2^T) C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A'$.

$C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}}$ can be partitioned into four submatrices

$$C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} = \begin{pmatrix} C_1 & \mid & \mathbf{0} \\ \hline \mathbf{0} & \mid & \mathbf{0} \end{pmatrix} \quad (3.130)$$

where C_1 is $K \times K$ diagonal. Then the product is:

$$(\mathbf{0} \mid A_2^T) \begin{pmatrix} C_1 & \mid & \mathbf{0} \\ \hline \mathbf{0} & \mid & \mathbf{0} \end{pmatrix} = (\mathbf{0}) \quad (3.131)$$

Using this result in (3.128), we obtain:

$$E\{\|\mathbf{g} - \mathbf{g}'\|\} = \text{trace} [A^T C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A - A'^T C'_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A] \quad (3.132)$$

Consider the term $A^T C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A$. We may assume that $C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}}$ is the sum of N^2 matrices, each one being $N^2 \times N^2$ and having only one non zero element:

$$C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} + \dots + \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \lambda_{N^2} \end{pmatrix} \quad (3.133)$$

A is made up of rows of eigenvectors while A^T is made up of columns of eigenvectors. Then we may write

$$A^T C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A = \sum_{i=1}^{N^2} (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_{N^2}) C_i \begin{pmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_{N^2}^T \end{pmatrix} \quad (3.134)$$

where C_i is the matrix with its i^{th} diagonal element nonzero and equal to λ_i . Generalising then the result of example 3.41, we have:

$$\begin{aligned} \text{trace}[A^T C_{\tilde{\mathbf{g}}\tilde{\mathbf{g}}} A] &= \text{trace} \left[\sum_{i=1}^{N^2} \lambda_i \mathbf{u}_i \mathbf{u}_i^T \right] \\ &= \text{trace} \left[\sum_{i=1}^{N^2} \lambda_i \begin{pmatrix} u_{i1}^2 & u_{i1}u_{i2} & \dots & u_{i1}u_{iN^2} \\ u_{i2}u_{i1} & u_{i2}^2 & \dots & u_{i2}u_{iN^2} \\ \vdots & \vdots & & \vdots \\ u_{iN^2}u_{i1} & u_{iN^2}u_{i2} & \dots & u_{iN^2}^2 \end{pmatrix} \right] \\ &= \sum_{i=1}^{N^2} \lambda_i \text{trace} \begin{pmatrix} u_{i1}^2 & u_{i1}u_{i2} & \dots & u_{i1}u_{iN^2} \\ u_{i2}u_{i1} & u_{i2}^2 & \dots & u_{i2}u_{iN^2} \\ \vdots & \vdots & & \vdots \\ u_{iN^2}u_{i1} & u_{iN^2}u_{i2} & \dots & u_{iN^2}^2 \end{pmatrix} \\ &= \sum_{i=1}^{N^2} \lambda_i (u_{i1}^2 + u_{i2}^2 + \dots + u_{iN^2}^2) = \sum_{i=1}^{N^2} \lambda_i \end{aligned} \quad (3.135)$$

To obtain this result we made use of the fact that \mathbf{u}_i is an eigenvector, and therefore $u_{i1}^2 + u_{i2}^2 + \dots + u_{iN^2}^2 = 1$.

Applying this to equation (3.132) we eventually get:

$$\text{Mean square error} = \sum_{i=1}^{N^2} \lambda_i - \sum_{i=1}^K \lambda_i = \sum_{i=K+1}^{N^2} \lambda_i \quad (3.136)$$

Note that all eigenvalues of $C_{\mathbf{g}\mathbf{g}}$ are non-negative, as $C_{\mathbf{g}\mathbf{g}}$ is a **Gram** matrix, and, therefore, positive semidefinite.

Thus, when an image is approximated by its truncated Karhunen-Loeve expansion, the mean square error committed is equal to the sum of the omitted eigenvalues of the covariance matrix. Since λ_i are arranged in decreasing order, this shows that the mean square error is the minimum possible.

Example 3.43

The autocovariance matrix of a 2×2 image is given by:

$$C = \begin{pmatrix} 3 & 0 & -1 & 0 \\ 0 & 3 & 0 & -1 \\ -1 & 0 & 3 & 0 \\ 0 & -1 & 0 & 3 \end{pmatrix} \quad (3.137)$$

Calculate the transformation matrix A for the image, which when used for the inverse transform, will approximate the image with mean square error equal to 2.

We must find the eigenvalues of this matrix, by solving the following equation:

$$\begin{aligned} & \left| \begin{array}{cccc} 3-\lambda & 0 & -1 & 0 \\ 0 & 3-\lambda & 0 & -1 \\ -1 & 0 & 3-\lambda & 0 \\ 0 & -1 & 0 & 3-\lambda \end{array} \right| = 0 \Rightarrow \\ & (3-\lambda) \left[(3-\lambda)^3 - (3-\lambda) \right] + (-1) \left[(3-\lambda)^2 - (-1)^2 \right] = 0 \Rightarrow \\ & (3-\lambda)^2 \left[(3-\lambda)^2 - 1 \right] - \left[(3-\lambda)^2 - 1 \right] = 0 \Rightarrow \\ & \left[(3-\lambda)^2 - 1 \right]^2 = 0 \Rightarrow \\ & (3-\lambda-1)^2(3-\lambda+1)^2 = 0 \Rightarrow \\ & (2-\lambda)^2(4-\lambda)^2 = 0 \Rightarrow \\ & \lambda_1 = 4, \lambda_2 = 4, \lambda_3 = 2, \lambda_4 = 2 \end{aligned} \quad (3.138)$$

The corresponding eigenvectors for $\lambda = 4$ are:

$$\begin{pmatrix} 3 & 0 & -1 & 0 \\ 0 & 3 & 0 & -1 \\ -1 & 0 & 3 & 0 \\ 0 & -1 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = 4 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \Rightarrow \begin{cases} 3x_1 - x_3 = 4x_1 \\ 3x_2 - x_4 = 4x_2 \\ -x_1 + 3x_3 = 4x_3 \\ -x_2 + 3x_4 = 4x_4 \end{cases} \Rightarrow \\ x_3 = -x_1, x_4 = -x_2, x_1 = -x_3, x_2 = -x_4 \quad (3.139)$$

Choose: $x_1 = x_3 = 0, x_2 = \frac{1}{\sqrt{2}}, x_4 = -\frac{1}{\sqrt{2}}$

Or choose: $x_1 = \frac{1}{\sqrt{2}}$, $x_3 = -\frac{1}{\sqrt{2}}$, $x_2 = x_4 = 0$.

The first two eigenvectors, therefore, are: $\begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix}$ and $\begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \end{pmatrix}$, which are orthogonal to each other. For $\lambda = 2$ we have:

$$\begin{pmatrix} 3 & 0 & -1 & 0 \\ 0 & 3 & 0 & -1 \\ -1 & 0 & 3 & 0 \\ 0 & -1 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = 2 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \Rightarrow \begin{vmatrix} 3x_1 - x_3 = 2x_1 \\ 3x_2 - x_4 = 2x_2 \\ -x_1 + 3x_3 = 2x_3 \\ -x_2 + 3x_4 = 2x_4 \end{vmatrix} \Rightarrow \\ x_1 = x_3, x_2 = x_4, x_1 = x_3, x_2 = x_4 \quad (3.140)$$

Choose: $x_1 = x_3 = 0$, $x_2 = x_4 = \frac{1}{\sqrt{2}}$.

We do not need to calculate the fourth eigenvector because we are interested in an approximate transformation matrix. By setting some eigenvectors to $(0 \ 0 \ 0 \ 0)$, the mean square error we commit when reconstructing the image is equal to the sum of the corresponding eigenvalues. In this case, if we consider as transformation matrix, matrix \tilde{A} ,

$$\tilde{A} = \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (3.141)$$

the error will be equal to $\lambda_4 = 2$.

Example 3.44

Show the different stages of the Karhunen-Loeve transform of the image in example 2.15, on page 69.

There are 63 nonzero eigenvalues of the spatial autocorrelation matrix of this image. Figure 3.10 shows the corresponding 63 eigenimages.

The eight images shown in figure 3.11 are the reconstructed images when 8, 16, 24, 32, 40, 48, 56 and 63 terms were used for the reconstruction.

The sums of the mean square errors for each reconstructed image are:

$$\text{Square error for image 3.11a: } 196460 \quad \left(\sum_{i=9}^{63} \lambda_i = 197400 \right)$$

<i>Square error for image 3.11b:</i>	136290	$\left(\sum_{i=17}^{63} \lambda_i = 129590 \right)$
<i>Square error for image 3.11c:</i>	82906	$\left(\sum_{i=25}^{63} \lambda_i = 82745 \right)$
<i>Square error for image 3.11d:</i>	55156	$\left(\sum_{i=33}^{63} \lambda_i = 52036 \right)$
<i>Square error for image 3.11e:</i>	28091	$\left(\sum_{i=41}^{63} \lambda_i = 29030 \right)$
<i>Square error for image 3.11f:</i>	13840	$\left(\sum_{i=49}^{63} \lambda_i = 12770 \right)$
<i>Square error for image 3.11g:</i>	257	$\left(\sum_{i=57}^{63} \lambda_i = 295 \right)$
<i>Square error for image 3.11h:</i>	0	

The square errors of the reconstructions do not agree exactly with the sum of the omitted eigenvalues, because each approximation is optimal only in the mean square error sense, over a whole collection of images with the same autocorrelation function.

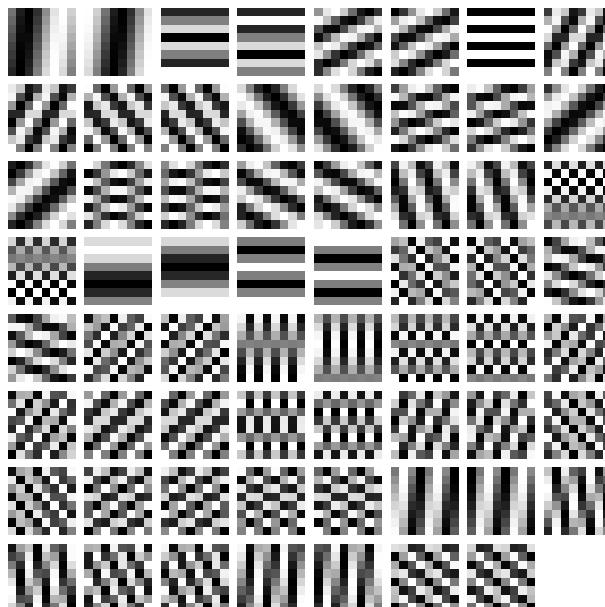


Figure 3.10: The 63 eigenimages, each scaled separately to have values from 0 to 255. They are displayed in lexicographic order, ie from top left to bottom right, sequentially.

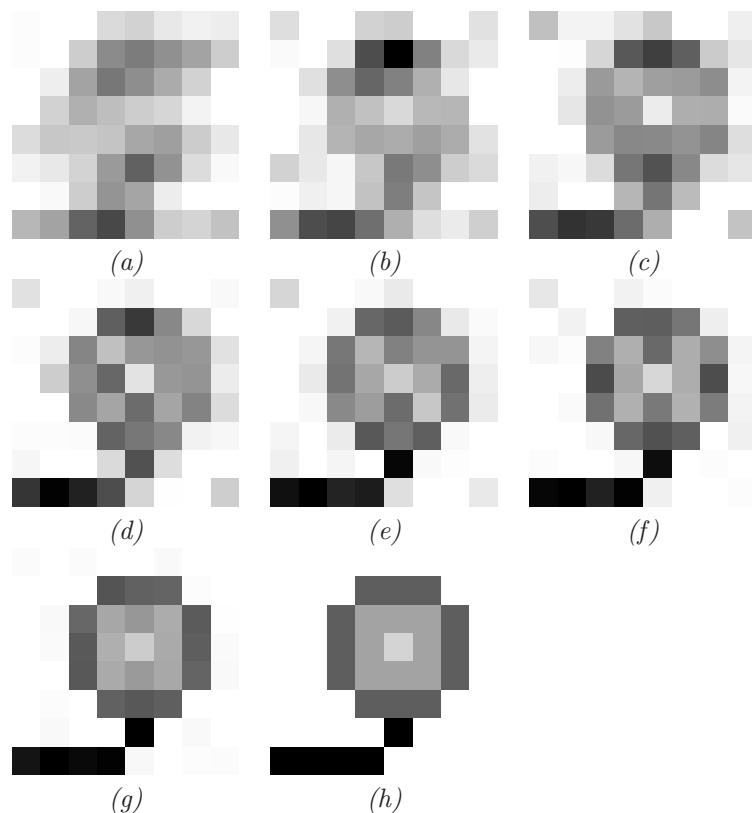


Figure 3.11: Reconstructed image when the first 8, 16, 24, 32, 40, 48, 56 and 63 eigenimages shown in figure 3.10 were used (from top left to bottom right, respectively).

Example 3.45

The autocovariance matrix of a 2×2 image is given by:

$$C = \begin{pmatrix} 4 & 0 & -1 & 0 \\ 0 & 4 & 0 & -1 \\ -1 & 0 & 4 & 0 \\ 0 & -1 & 0 & 4 \end{pmatrix} \quad (3.142)$$

Calculate the transformation matrix A for the image, which, when used for the inverse transform, will approximate the image with mean square error

equal to 6.

We first find the eigenvalues of the autocovariance matrix:

$$\begin{aligned}
 & \left| \begin{array}{cccc} 4-\lambda & 0 & -1 & 0 \\ 0 & 4-\lambda & 0 & -1 \\ -1 & 0 & 4-\lambda & 0 \\ 0 & -1 & 0 & 4-\lambda \end{array} \right| = 0 \Rightarrow \\
 & (4-\lambda) \left| \begin{array}{ccc} 4-\lambda & 0 & -1 \\ 0 & 4-\lambda & 0 \\ -1 & 0 & 4-\lambda \end{array} \right| - 1 \left| \begin{array}{ccc} 0 & 4-\lambda & -1 \\ -1 & 0 & 0 \\ 0 & -1 & 4-\lambda \end{array} \right| = 0 \Rightarrow \\
 & (4-\lambda)[(4-\lambda)^2 - (4-\lambda)] - [(4-\lambda)^2 - 1] = 0 \Rightarrow \\
 & [(4-\lambda)^2 - 1]^2 = 1 \Rightarrow (4-\lambda-1)^2(4-\lambda+1)^2 = 0 \Rightarrow \tag{3.143}
 \end{aligned}$$

$$\lambda_1 = 5$$

$$\lambda_2 = 5$$

$$\lambda_3 = 3$$

$$\lambda_4 = 3$$

Since we allow error of image reconstruction equal to 6, we do not need to calculate the eigenvectors that correspond to $\lambda = 3$.

Eigenvectors for $\lambda = 5$:

$$\left(\begin{array}{cccc} 4 & 0 & -1 & 0 \\ 0 & 4 & 0 & -1 \\ -1 & 0 & 4 & 0 \\ 0 & -1 & 0 & 4 \end{array} \right) \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = 5 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \Rightarrow \begin{array}{l} 4x_1 - x_3 = 5x_1 \Rightarrow x_1 = x_3 \\ 4x_2 - x_4 = 5x_2 \Rightarrow x_2 = -x_4 \\ -x_1 - 4x_3 = 5x_3 \Rightarrow x_1 = -x_3 \\ -x_2 - 4x_4 = 5x_4 \Rightarrow x_2 = -x_4 \end{array} \tag{3.144}$$

Choose $x_1 = x_3 = 0$, $x_2 = \frac{1}{\sqrt{2}}$, $x_4 = -\frac{1}{\sqrt{2}}$. For λ_2 choose an orthogonal eigenvector, eg $x_2 = x_4 = 0$, $x_1 = \frac{1}{\sqrt{2}}$, $x_3 = -\frac{1}{\sqrt{2}}$. Then the transformation matrix which allows reconstruction with mean square error 6 (equal to the sum of the omitted eigenvalues) is:

$$A = \left(\begin{array}{cccc} 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right) \tag{3.145}$$

3.3 Independent component analysis

What is Independent Component Analysis (ICA)?

Independent Component Analysis (ICA) allows one to construct *independent* components from an ensemble of data.

In the previous section, we saw how to define a basis in terms of which we may create *uncorrelated* components from an ensemble of data. Remember that independence is a much stronger requirement than decorrelation (see example 3.15, on page 188). So, identifying independent components is expected to be a much more difficult problem than identifying uncorrelated components. Further, independence implies uncorrelatedness (see example 3.11, on page 185), and when the random variables are of zero-mean, uncorrelatedness implies orthogonality. (This follows trivially from definitions (3.18) and (3.19), on page 184.) This relationship is schematically shown in figure 3.12.

The problem of identification of independent components is best understood in terms of the so called “cocktail party problem”.

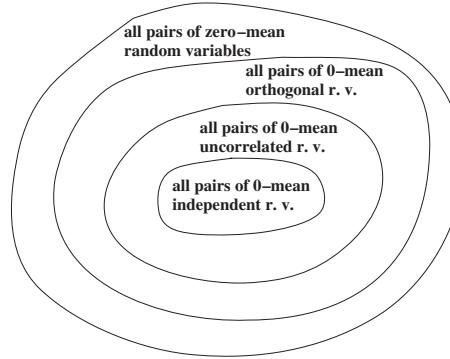


Figure 3.12: The set of all pairs of zero-mean independent random variables is a subset of the set of all zero-mean uncorrelated random variables, which is a subset of the set of all zero-mean orthogonal random variables, which is a subset of the set of all zero-mean random variables. So, when we want to search for independent zero-mean random variables, we may restrict our search among the uncorrelated zero-mean ones.

What is the cocktail party problem?

Imagine that you are in a room where several people are talking. Imagine that there are several microphones recording the conversations. At any instant in time, you have several blended recordings of the same speech signals. Let us say that there are two people talking, producing signals $s_1(t)$ and $s_2(t)$ and there are two microphones recording. The recorded signals $x_1(t)$ and $x_2(t)$ are

$$\begin{aligned} x_1(t) &= a_{11}s_1(t) + a_{12}s_2(t) \\ x_2(t) &= a_{21}s_1(t) + a_{22}s_2(t) \end{aligned} \tag{3.146}$$

where a_{11} , a_{12} , a_{21} and a_{22} are the blending factors, which are unknown. The question is, given that (3.146) constitutes a system of two linear equations with six unknowns (the four blending factors and the two original signals), can we solve it to recover the unknown signals?

How do we solve the cocktail party problem?

Clearly it is impossible to solve system (3.146) in any deterministic way. We solve it by considering the statistical properties that characterise independent signals and by invoking the **central limit theorem**.

What does the central limit theorem say?

According to the central limit theorem, the probability density function of a random variable, that is the sum of n independent random variables, tends to a Gaussian, as n tends to infinity, no matter what the probability density functions of the independent variables are. In other words, in (3.146) the samples of $x_1(t)$ are more Gaussianly distributed than either $s_1(t)$ or $s_2(t)$. So, in order to estimate the values of the independent components, the first thing we need is a way to quantify the non-Gaussianity of a probability density function.

What do we mean by saying that “the samples of $x_1(t)$ are more Gaussianly distributed than either $s_1(t)$ or $s_2(t)$ ” in relation to the cocktail party problem? Are we talking about the *temporal* samples of $x_1(t)$, or are we talking about all possible versions of $x_1(t)$ at a given time?

The answer depends on the application we are interested in, which determines the nature of the implied random experiment. Note that if ergodicity were assumed, it would have made no difference to the outcome either we were using temporal or ensemble statistics, but the nature of the problem is such that ergodicity is not assumed here. Instead, what determines the choice of the random experiment we assume is the application we are interested in. This is different in signal and in image processing.

Example B3.46

It is known that a signal is corrupted by ten different sources of noise, all of which produce random numbers that are added to the true signal value. The random numbers produced by one such source of noise are uniformly distributed in the range $[-\alpha_i, \alpha_i]$, where $i = 1, 2, \dots, 10$ identifies the noise source. We know that the values of α_i are:

$$\alpha_1 = 0.9501, \alpha_2 = 0.2311, \alpha_3 = 0.6068, \alpha_4 = 0.4860, \alpha_5 = 0.8913, \alpha_6 = 0.7621, \alpha_7 = 0.4565, \alpha_8 = 0.0185, \alpha_9 = 0.8214 \text{ and } \alpha_{10} = 0.4447.$$

Work out a model for the probability density function of the composite noise that corrupts this signal.

Let us assume that the signal consists of 3000 samples. The choice of this number is not crucial, as long as the number is large enough to allow us to perform reliable

statistical estimates. Since we know that each source of noise produces random numbers uniformly distributed in the range $[-\alpha_i, \alpha_i]$, let us draw 3000 such random numbers for each of the various values of α_i . Let us call them x_{ij} for $j = 1, \dots, 3000$. From these numbers we may create numbers $z_j \equiv \sum_i x_{ij}$, which could represent the total error added to each true value of the signal. The histograms of the random numbers we drew for $i = 1, 2, \dots, 10$ and the histogram of the 3000 numbers z_j we created from them are shown in figure 3.13. We can see that the z_j numbers have a bell-shaped distribution. We may try to fit it with a Gaussian, by computing their mean μ and standard deviation σ . It turns out that $\mu = 0.0281$ and $\sigma = 1.1645$. In the bottom right panel of figure 3.13, the Gaussian $G(z) \equiv e^{-(z-\mu)^2/(2\sigma^2)} / (\sqrt{2\pi}\sigma)$ is plotted on the same axes as the normalised histogram of the z_j values. To convert the histogram of the eleventh panel into a probability density function that can be compared with the corresponding Gaussian, we first divide the bin entries with the total number of elements we used (ie with 3000) and then we divide each such number with the bin width, in order to make it into a density. The bin width here is 0.3715.

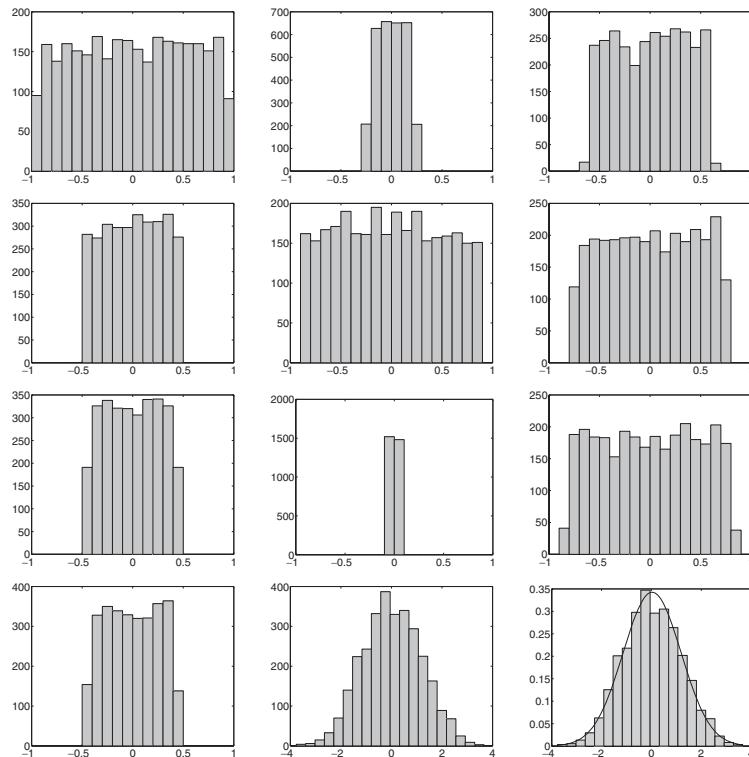


Figure 3.13: The ten histograms of x_{ij} and the histogram of z_j , from top left to bottom middle, respectively. In the bottom right panel, the normalised histogram of z_j and the fitted Gaussian probability density function. The normalised histogram was produced from the histogram in the previous panel, by dividing its values with the total number of samples we used to produce it and with the bin width we used, in order to convert it into a density.

Example B3.47

Confirm that the Gaussian function you used to model the normalised histogram of the random z_j numbers you created in example 3.46 does indeed fit the data, by using the χ^2 -test.

For 3000 points, the Gaussian function predicts a certain number of events per bin for the histogram. Let us say that histogram bin i contains values in the range $(b_{i1}, b_{i2}]$. The probability of finding a value in this range according to the Gaussian probability density function with mean μ and standard deviation σ is:

$$p_i \equiv \frac{1}{\sqrt{2\pi}\sigma} \int_{b_{i1}}^{b_{i2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \quad (3.147)$$

Let us define a new variable of integration $z \equiv \frac{x-\mu}{\sqrt{2}\sigma}$. Then $dx = \sqrt{2}\sigma dz$ and the limits of integration are $z_{i1} \equiv \frac{b_{i1}-\mu}{\sqrt{2}\sigma}$ and $z_{i2} \equiv \frac{b_{i2}-\mu}{\sqrt{2}\sigma}$:

$$\begin{aligned} p_i &= \frac{1}{\sqrt{\pi}} \int_{z_{i1}}^{z_{i2}} e^{-z^2} dz \\ &= \frac{1}{\sqrt{\pi}} \left[\int_0^{z_{i2}} e^{-z^2} dz - \int_0^{z_{i1}} e^{-z^2} dz \right] \end{aligned} \quad (3.148)$$

We may express these integrals in terms of the error function defined as:

$$\text{erf}(z) \equiv \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \quad (3.149)$$

Then:

$$p_i = \frac{1}{2} [\text{erf}(z_{i2}) - \text{erf}(z_{i1})] \quad (3.150)$$

If we multiply p_i with the total number of random numbers we have, we shall have the number of samples we expect to find in bin i . Table 3.1 lists the boundaries of each bin in the bottom middle panel of figure 3.13, the corresponding value p_i computed from (3.150), and the expected value of occupancy of each bin, $E_i \equiv 3000p_i$, since the histogram was created from 3000 numbers. The last column of this table, under O_i , lists the actual number of samples in each bin. Note that the first and the last bin are $-\infty$ and $+\infty$, respectively. When computing then p_i for these bins, we remember that $\text{erf}(-\infty) = 0$ and $\text{erf}(+\infty) = 1$.

<i>bin boundaries</i>	p_i	E_i	O_i
($-\infty, -3.3950]$	0.0016	4.9279	4
($-3.3950, -3.0235]$	0.0027	8.2366	6
($-3.0235, -2.6520]$	0.0063	18.8731	15
($-2.6520, -2.2805]$	0.0130	39.0921	33
($-2.2805, -1.9089]$	0.0244	73.1964	70
($-1.9089, -1.5374]$	0.0413	123.8935	140
($-1.5374, -1.1659]$	0.0632	189.5689	224
($-1.1659, -0.7943]$	0.0874	262.2085	243
($-0.7943, -0.4228]$	0.1093	327.8604	332
($-0.4228, -0.0513]$	0.1235	370.5906	387
($-0.0513, 0.3202]$	0.1262	378.6724	330
($0.3202, 0.6918]$	0.1166	349.7814	340
($0.6918, 1.0633]$	0.0974	292.0743	294
($1.0633, 1.4348]$	0.0735	220.4717	225
($1.4348, 1.8063]$	0.0501	150.4437	163
($1.8063, 2.1779]$	0.0309	92.8017	89
($2.1779, 2.5494]$	0.0172	51.7483	68
($2.5494, 2.9209]$	0.0087	26.0851	25
($2.9209, 3.2925]$	0.0040	11.8862	9
($3.2925, +\infty)$	0.0025	7.5870	3

Table 3.1: The boundaries of the bins used to produce the two bottom right plots in figure 3.13, and the corresponding probabilities of observing an event inside that range of values, p_i , computed from (3.150). The third column is the number of expected events in each interval, produced by multiplying p_i with the number of samples we have, ie with 3000. The last column is the number of observed events in each interval.

The question we have to answer then is: are the numbers in the last column of table 3.1 in agreement with the numbers in the penultimate column, which are the expected numbers according to the normal probability density function? To answer this question, we compute the χ^2 value of these data, as follows:

$$\chi^2 \equiv \sum_{i=1}^N \frac{(O_i - E_i)^2}{E_i} \quad (3.151)$$

Here N is the total number of bins with $E_i \geq 5$. Note that if any of the bins is expected to have fewer than 5 points, that bin is joined with a neighbouring bin, so no bin is expected to have fewer than 5 points when we perform this calculation. This happens for the first bin, according to the entries of table 3.1, which has to be joined with the second bin, to form a single bin with expected value 13.1645 ($= 4.9279 + 8.2366$) and observed value 10 ($= 4 + 6$). Therefore, $N = 19$ in this example. The value of χ^2 has to be compared with a value that we shall read from some statistical tables, identified

by two parameters. The first is the number of **degrees of freedom** of this test: this is equal to N minus the number of parameters we estimated from the data and used for the theoretical prediction. The number of parameters we used was 2, ie the mean and the standard deviation of the Gaussian function used to make the theoretical prediction. So, the degrees of freedom ν are $\nu = N - 2 = 17$. The other parameter, which specifies which statistical table we should use, is the confidence α with which we want to perform the test. This is usually set to $\alpha = 0.95$. This is the probability with which, for ν degrees of freedom, one may find a smaller value of χ^2 than the one given in the table. If the computed χ^2 value is higher than the one given in the table, we may say that “the data are not Gaussianly distributed, with confidence 95%”. If the value of χ^2 we computed is lower than the value given in the table, we may say that “our data are compatible with the hypothesis of a Gaussian distribution at the 95% confidence level”.

The value of χ^2 for this particular example is $\chi^2 = 0.7347$. The threshold value for 17 degrees of freedom and at the 95% confidence level is 8.67. So, we conclude that the hypothesis that the z_j numbers are drawn from a Gaussian probability density function is compatible with the data.

How do we measure non-Gaussianity?

A Gaussian probability density function is fully characterised by its mean and standard deviation. All higher order moments of it are either 0 or they can be expressed in terms of these two parameters. To check for non-Gaussianity, therefore, we check by how much one of the higher order moments differs from the value expected for Gaussianly distributed data. For example, a Gaussian probability density function has zero third moment (also known as **skewness**) (see example 3.48) and its fourth order moment is $3\sigma^4$, where σ^2 is the second order moment (also known as **variance**) (see example 3.50). It is also known that from all probability density functions with fixed standard deviation, the Gaussian has the maximum entropy (see Box 3.4). This leads to another measure of non-Gaussianity, known as **negentropy**.

How are the moments of a random variable computed?

The **moments**¹ of a random variable x with probability density function $p(x)$ are defined as

$$\mu_i \equiv \int_{-\infty}^{+\infty} (x - \mu)^i p(x) dx \quad (3.152)$$

where μ is the mean value of x , ie $\mu \equiv \int_{-\infty}^{+\infty} xp(x) dx$.

In practice, these integrals are replaced by sums over all N available values x_n of the random variable

¹In many books these moments are known as **central moments** because the mean is removed from the random variable before it is integrated with the probability density function. In those books the moments are defined as $\mu_i \equiv \int_{-\infty}^{+\infty} x^i p(x) dx$. For simplicity here we drop the qualification “central” in the terminology we use.

$$\mu_i \equiv \frac{1}{N} \sum_{n=1}^N (x_n - \mu)^i \quad (3.153)$$

where

$$\mu \equiv \frac{1}{N} \sum_{n=1}^N x_n \quad (3.154)$$

In ICA we often use functions of these moments rather than the moments themselves, like, for example, the **kurtosis**.

How is the kurtosis defined?

The **kurtosis proper** or **Pearson kurtosis** is defined as:

$$\beta_2 \equiv \frac{\mu_4}{\mu_2^2} \quad (3.155)$$

The **kurtosis excess** is defined as:

$$\gamma_2 \equiv \frac{\mu_4}{\mu_2^2} - 3 \quad (3.156)$$

The kurtosis excess is also known as **fourth order cumulant**.

When a probability density function is flatter than a Gaussian, it has negative kurtosis excess and is said to be **platykurtic** or **sub-Gaussian**. If it is more peaky, it has positive kurtosis and it is said to be **leptokurtic** or **super-Gaussian** (see figure 3.14).

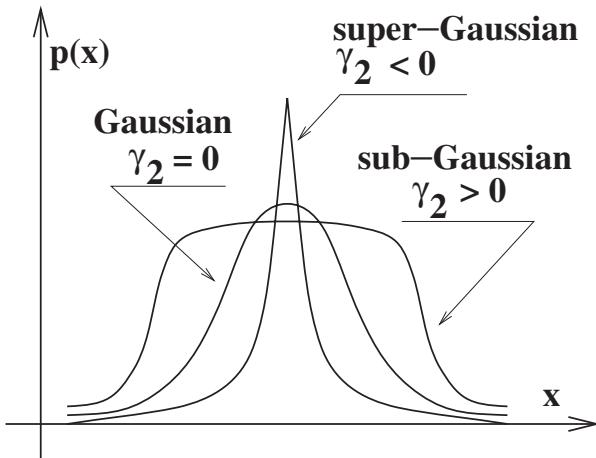


Figure 3.14: The value of the kurtosis excess may be used to characterise a probability density function as being super- or sub-Gaussian.

Usually in ICA when we say “kurtosis” we refer to the kurtosis excess, which has zero value for a Gaussian probability density function. We may, therefore, use the square of the kurtosis excess, or its absolute value as a measure of non-Gaussianity. The higher its value, the more non-Gaussian the probability density function of the data is.

Example B3.48

The third order moment is called skewness and quantifies the asymmetry of a probability density function. Compute the third moment of the Gaussian probability density function.

The third moment of probability density function $g(x) \equiv \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$ is:

$$\mu_3 \equiv \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} (x - \mu)^3 e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \quad (3.157)$$

We change variable of integration to $\tilde{x} \equiv x - \mu \Rightarrow dx = d\tilde{x}$. The limits of integration remain unaffected. Then:

$$\mu_3 = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} \tilde{x}^3 e^{-\frac{\tilde{x}^2}{2\sigma^2}} d\tilde{x} = 0 \quad (3.158)$$

This integral is 0 because it is the integral of an antisymmetric (odd) integrand over a symmetric interval.

Example B3.49

Show that:

$$\int_{-\infty}^{+\infty} e^{-x^2} dx = \sqrt{\pi} \quad (3.159)$$

Consider the integral of $e^{-x^2-y^2}$ over the whole (x, y) 2D plane. It can be computed either using Cartesian or polar coordinates. In Cartesian coordinates:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} e^{-x^2-y^2} dx dy = \int_{-\infty}^{+\infty} e^{-x^2} dx \int_{-\infty}^{+\infty} e^{-y^2} dy = \left(\int_{-\infty}^{+\infty} e^{-x^2} dx \right)^2 \quad (3.160)$$

In polar coordinates (r, θ) , where $r^2 \equiv x^2 + y^2$ and θ is such that $x = r \cos \theta$ and $y = r \sin \theta$, we have:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} e^{-x^2-y^2} dx dy = \int_0^{+\infty} \int_0^{2\pi} e^{-r^2} r dr d\theta = -2\pi \frac{1}{2} e^{-r^2} \Big|_0^{+\infty} = \pi \quad (3.161)$$

By combining the results of equations (3.160) and (3.161), (3.159) follows.

Example B3.50

Compute the fourth moment of the Gaussian probability density function.

The fourth moment of probability density function $g(x) \equiv \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$ is:

$$\mu_4 \equiv \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} (x-\mu)^4 e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \quad (3.162)$$

We change variable of integration to $\tilde{x} \equiv (x-\mu)/(\sqrt{2}\sigma) \Rightarrow dx = \sqrt{2}\sigma d\tilde{x}$. The limits of integration remain unaffected. Then:

$$\begin{aligned} \mu_4 &= \frac{1}{\sqrt{2\pi}\sigma} \left(\sqrt{2}\sigma\right)^5 \int_{-\infty}^{+\infty} \tilde{x}^4 e^{-\tilde{x}^2} d\tilde{x} \\ &= -\frac{4\sigma^4}{\sqrt{\pi}} \int_{-\infty}^{+\infty} \tilde{x}^3 \frac{1}{2} d\left(e^{-\tilde{x}^2}\right) \\ &= -\frac{2\sigma^4}{\sqrt{\pi}} \left\{ \tilde{x}^3 e^{-\tilde{x}^2} \Big|_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty} e^{-\tilde{x}^2} 3\tilde{x}^2 d\tilde{x} \right\} \\ &= -\frac{6\sigma^4}{\sqrt{\pi}} \int_{-\infty}^{+\infty} \tilde{x} \frac{1}{2} d\left(e^{-\tilde{x}^2}\right) \\ &= -\frac{3\sigma^4}{\sqrt{\pi}} \left\{ \tilde{x} e^{-\tilde{x}^2} \Big|_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty} e^{-\tilde{x}^2} d\tilde{x} \right\} \\ &= \frac{3\sigma^4}{\sqrt{\pi}} \int_{-\infty}^{+\infty} e^{-\tilde{x}^2} d\tilde{x} = \frac{3\sigma^4}{\sqrt{\pi}} \sqrt{\pi} = 3\sigma^4 \end{aligned} \quad (3.163)$$

Here we made use of (3.159) and the fact that terms $\tilde{x}^3 e^{-\tilde{x}^2} \Big|_{-\infty}^{+\infty}$ and $\tilde{x} e^{-\tilde{x}^2} \Big|_{-\infty}^{+\infty}$ vanish.

Example B3.51

Compute the kurtosis of the Gaussian probability density function.

Remembering that for the Gaussian probability density function, $\mu_2 \equiv \sigma^2$, and applying formulae (3.155) and (3.156), and the result of example 3.50, we obtain for the kurtosis proper and the kurtosis excess, respectively:

$$\beta_2 = \frac{3\sigma^4}{\sigma^4} = 3 \quad \gamma_2 = 0 \quad (3.164)$$

How is negentropy defined?

The **negentropy** of a random variable y is defined as

$$J(y) \equiv H(\nu) - H(y) \quad (3.165)$$

where $H(y)$ is the **entropy** of y and ν is a Gaussianly distributed random variable with the same covariance matrix as y .

How is entropy defined?

If $P(y = \alpha_i)$ is the probability of the discrete random variable y to take value α_i , the **entropy** of this random variable is given by:

$$H(y) \equiv - \sum_i P(y = \alpha_i) \ln P(y = \alpha_i) \quad (3.166)$$

Example B3.52

For a continuous variable x , with probability density function $g(x)$, the entropy is defined as:

$$H(x) = - \int_{-\infty}^{+\infty} g(x) \ln[g(x)] dx \quad (3.167)$$

Calculate the entropy of the Gaussian probability density function $g(x) \equiv \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.

We remember that since $g(x)$ is a density function, $\int_{-\infty}^{+\infty} g(x) dx = 1$. Substituting for $g(x)$ in (3.167), we obtain:

$$\begin{aligned} H(x) &= - \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \left[-\ln(\sqrt{2\pi}\sigma) - \frac{(x-\mu)^2}{2\sigma^2} \right] dx \\ &= \ln(\sqrt{2\pi}\sigma) \underbrace{\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx}_{=1(\text{density})} + \frac{1}{\sqrt{2\pi}\sigma} \underbrace{\int_{-\infty}^{+\infty} \frac{(x-\mu)^2}{2\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx}_{z \equiv (x-\mu)/(\sqrt{2}\sigma) \Rightarrow dx = \sqrt{2}\sigma dz} \\ &= \ln(\sqrt{2\pi}\sigma) + \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} z^2 e^{-z^2} \sqrt{2}\sigma dz \\ &= \ln(\sqrt{2\pi}\sigma) - \frac{1}{\sqrt{\pi}} \frac{1}{2} \int_{-\infty}^{+\infty} zd(e^{-z^2}) \\ &= \ln(\sqrt{2\pi}\sigma) - \frac{1}{2\sqrt{\pi}} \left\{ ze^{-z^2} \Big|_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty} e^{-z^2} dz \right\} \\ &= \ln(\sqrt{2\pi}\sigma) + \frac{1}{2\sqrt{\pi}} \sqrt{\pi} = \ln(\sqrt{2\pi}\sigma) + \frac{1}{2} \ln e \\ &= \ln(\sqrt{2\pi}\sigma) + \ln \sqrt{e} = \ln(\sqrt{2\pi e}\sigma) = 4.1327\sigma \end{aligned} \quad (3.168)$$

Here we made use of (3.159).

Example B3.53

Calculate the entropy of a uniform probability density function with zero mean and standard deviation σ .

The uniform probability density function with zero mean is defined as

$$p(x) = \begin{cases} \frac{1}{2A} & \text{for } -A \leq x \leq A \\ 0 & \text{otherwise} \end{cases} \quad (3.169)$$

where A is some positive constant. The variance of this probability density function is:

$$\sigma^2 = \frac{1}{2A} \int_{-A}^A x^2 dx = \frac{1}{2A} \left. \frac{x^3}{3} \right|_{-A}^A = \frac{A^2}{3} \Rightarrow A = \sigma\sqrt{3} \quad (3.170)$$

The entropy then is:

$$\begin{aligned} H(x) &= - \int_{-\infty}^{+\infty} p(x) \ln p(x) dx \\ &= - \int_{-\sigma\sqrt{3}}^{\sigma\sqrt{3}} \frac{1}{2\sigma\sqrt{3}} \ln \left(\frac{1}{2\sigma\sqrt{3}} \right) dx \\ &= \ln(2\sigma\sqrt{3}) \end{aligned} \quad (3.171)$$

Example B3.54

Compute the negentropy of the zero mean uniform probability density function with standard deviation σ .

We substitute the results of examples 3.52 and 3.53 in the definition of negentropy given by (3.165):

$$J(y) = \ln(\sqrt{2\pi e}\sigma) - \ln(2\sigma\sqrt{3}) = \ln \left(\frac{\sqrt{2\pi e}}{2\sqrt{3}} \right) = \ln \left(\sqrt{\frac{\pi e}{6}} \right) = 0.176 \quad (3.172)$$

Example B3.55

For z being a real number, prove the inequality:

$$\ln z \leq z - 1 \quad (3.173)$$

Consider function $f(z) \equiv \ln z - z + 1$. Its first and second derivatives are:

$$\frac{df}{dz} = \frac{1}{z} - 1 \quad \frac{d^2f}{dz^2} = -\frac{1}{z^2} < 0 \quad (3.174)$$

Since the second derivative is always negative, the point where $df/dz = 0$ is a maximum for $f(z)$. This maximum is at $z = 1$ and at this point $f(1) = 0$. So, always $f(z) \leq 0$, and (3.173) follows.

Example B3.56

Consider two probability density functions $a(x)$ and $b(x)$. Show that:

$$\int_{-\infty}^{+\infty} a(x) \ln[a(x)] dx \geq \int_{-\infty}^{+\infty} a(x) \ln[b(x)] dx \quad (3.175)$$

Define $z \equiv \frac{b(x)}{a(x)}$. According to (3.173):

$$\ln \left[\frac{b(x)}{a(x)} \right] \leq \frac{b(x)}{a(x)} - 1 \Rightarrow$$

$$\ln [b(x)] - \ln [a(x)] \leq \frac{b(x) - a(x)}{a(x)} \Rightarrow \quad (3.176)$$

$$\int_{-\infty}^{+\infty} \{\ln [b(x)] - \ln [a(x)]\} a(x) dx \leq \int_{-\infty}^{+\infty} b(x) dx - \int_{-\infty}^{+\infty} a(x) dx = 1 - 1 = 0$$

The last equality follows from the fact that $a(x)$ and $b(x)$ are probability density functions, and so each one integrates to 1. Inequality (3.175) then follows trivially.

Box 3.4. From all probability density functions with the same variance, the Gaussian has the maximum entropy

Consider a probability density function $f(x)$ with zero mean. We wish to define $f(x)$ so that $H(x)$ is maximal, subject to the constraint:

$$\int_{-\infty}^{+\infty} x^2 f(x) dx = \sigma^2 \quad (3.177)$$

Assume that $f(x) = Ae^{-\lambda x^2}$, where parameters A and λ should be chosen so that $f(x)$ integrates to 1 and has variance σ^2 . Then the entropy of x is:

$$H(x) = - \int_{-\infty}^{+\infty} f(x) \ln[f(x)] dx = - \int_{-\infty}^{+\infty} f(x) [\ln A - \lambda x^2] dx = - \ln A + \lambda \sigma^2 \quad (3.178)$$

If $\phi(x)$ is another probability density function with the same variance, we must show that the entropy of x now will be less than $\lambda \sigma^2 - \ln A$. From (3.175) we have:

$$\begin{aligned} - \int_{-\infty}^{+\infty} \phi(x) \ln[\phi(x)] dx &\leq - \int_{-\infty}^{+\infty} \phi(x) \ln[f(x)] dx \Rightarrow \\ - \int_{-\infty}^{+\infty} \phi(x) \ln[\phi(x)] dx &\leq - \int_{-\infty}^{+\infty} \phi(x) [\ln A - \lambda x^2] dx = - \ln A + \lambda \sigma^2 \end{aligned} \quad (3.179)$$

This completes the proof.

How is negentropy computed?

Negentropy cannot be computed directly from equations (3.165) and (3.166). However, some approximations have been proposed for its calculation and they are often used in practice. Most of them are valid for probability density functions that are not too different from the Gaussian. These approximations are valid when y and ν are zero-mean and unit variance random variables, with ν being Gaussianly distributed. Some of these approximations are:

$$J_1 \simeq \frac{1}{12} \mu_3^2 + \frac{1}{48} \gamma_2^2 \quad (3.180)$$

$$J_2 \propto \left[E \left\{ \frac{1}{a} \ln[\cosh(ay)] \right\} - E \left\{ \frac{1}{a} \ln[\cosh(a\nu)] \right\} \right]^2 \quad (3.181)$$

$$J_3 \propto \left[E \left\{ e^{-\frac{y^2}{2}} \right\} - \frac{1}{\sqrt{2}} \right]^2 \quad (3.182)$$

$$J_4 \simeq \frac{36}{8\sqrt{3}-9} \left[E \left\{ ye^{-\frac{y^2}{2}} \right\} \right]^2 + \frac{24}{16\sqrt{3}-27} \left[E \left\{ e^{-\frac{y^2}{2}} \right\} - \frac{1}{\sqrt{2}} \right]^2 \quad (3.183)$$

In (3.181) parameter a may take values in the range $[1, 2]$. In practice, often $a = 1$. $E\{\dots\}$ is the expectation operator.

Example B3.57

Compute the negentropy of the zero mean uniform probability density function with standard deviation σ , using approximation (3.180). Compare your answer with the exact answer of example 3.54.

According to example 3.53 the zero mean uniform probability density function with standard deviation σ is defined as:

$$p(x) = \begin{cases} \frac{1}{2\sqrt{3}\sigma} & \text{for } -\sqrt{3}\sigma \leq x \leq \sqrt{3}\sigma \\ 0 & \text{otherwise} \end{cases} \quad (3.184)$$

This is a symmetric function, and so $\mu_3 = 0$. Also, $\mu_2 = \sigma^2$. So, to compute γ_2 from equation (3.156), we require μ_4 :

$$\begin{aligned} \mu_4 &= \frac{1}{2\sqrt{3}\sigma} \int_{-\sqrt{3}\sigma}^{\sqrt{3}\sigma} y^4 dy = \frac{1}{2\sqrt{3}\sigma} \frac{y^5}{5} \Big|_{-\sqrt{3}\sigma}^{\sqrt{3}\sigma} \\ &= \frac{1}{2\sqrt{3}\sigma} \frac{2(\sqrt{3}\sigma)^5}{5} = \frac{9}{5}\sigma^4 \end{aligned} \quad (3.185)$$

Then:

$$\gamma_2 = \frac{\frac{9}{5}\sigma^4}{(\sigma^2)^2} - 3 = \frac{9}{5} - 3 = -\frac{6}{5} \quad (3.186)$$

Upon substitution into (3.180), we obtain:

$$J_1 = \frac{1}{48} \frac{36}{25} = 0.03 \quad (3.187)$$

This result is very different from the exact value computed in example 3.54. This is because approximation (3.180) is valid only for probability density functions that are very near the Gaussian (see Box 3.5, on page 252).

Example 3.58

Four samples of random variable y are: $\{-3, -2, 2, 3\}$. Four samples of a Gaussianly distributed variable ν are: $\{-1, 0, 0, 1\}$. Use (3.181), with $a = 1$, to compute a number proportional to the negentropy of y .

We note that both variables have 0 mean. We must also make sure that they have unit

variance. The variance of y is:

$$\sigma_y^2 = \frac{1}{4} [(-3)^2 + (-2)^2 + 2^2 + 3^2] = 6.5 \quad (3.188)$$

The variance of ν is:

$$\sigma_\nu^2 = \frac{1}{4} [(-1)^2 + 0^2 + 0^2 + 1^2] = 0.5 \quad (3.189)$$

If we divide now all values of y with $\sigma_y = \sqrt{6.5} = 2.55$ and all values of ν with $\sigma_\nu = \sqrt{0.5} = 0.71$, the variables will be of unit variance:

$$\begin{aligned} y : & \{-1.18, -0.78, 0.78, 1.18\} \\ \nu : & \{-1.41, 0.00, 0.00, 1.41\} \end{aligned}$$

We use these values in (3.181) with $a = 1$, to estimate a number proportional to the negentropy of y :

$$\begin{aligned} J_1(y) & \propto \left[\frac{1}{4} \{\ln[\cosh(-1.18)] + \ln[\cosh(-0.78)] + \ln[\cosh(0.78)] + \ln[\cosh(1.18)]\} \right. \\ & \quad \left. - \frac{1}{4} \{\ln[\cosh(-1.41)] + \ln[\cosh(0)] + \ln[\cosh(0)] + \ln[\cosh(1.41)]\} \right]^2 \\ & = 0.0016 \end{aligned} \quad (3.190)$$

Example 3.59

Draw N Gaussianly distributed random numbers with 0 mean and variance 1. Allow N to take values 10, 100, 500, 1000, 2000,..., up to 100,000. For each set of numbers compute $S \equiv E \{\ln[\cosh(a\nu)]\} / a$, for various values of a . What do you observe?

Table 3.2 gives the values of S for $a = 1$ and $a = 1.2$ and for the first few values of N . Figure 3.15 shows how the value of S varies with changing N , for the same two values of a . We notice that at least a few tens of thousands of numbers are needed to somehow stabilise the value of this expression. The average of the values of S obtained for N taking values from 50,000 up to 100,000, in steps of 1000, for various values of a is given in table 3.3. We observe that the value of S strongly depends on the number of samples drawn and the value of a .

N	$a = 1$	$a = 1.2$
10	0.4537	0.3973
100	0.3314	0.4649
500	0.3603	0.3793
1000	0.3629	0.3978

Table 3.2: Number of Gaussianly distributed random samples used and the corresponding value of $S \equiv E \{ \ln[\cosh(a\nu)] \} / a$ for $a = 1$ and $a = 1.2$.

a	average $S = < S >$
1.0	0.3749
1.1	0.3966
1.2	0.4140
1.3	0.4371
1.4	0.4508
1.5	0.4693
2.0	0.5279

Table 3.3: Value of $< S >$ for various values of a , estimated as the average over all values obtained for S , for N taking values from 50,000 to 100,000, in steps of 1,000.

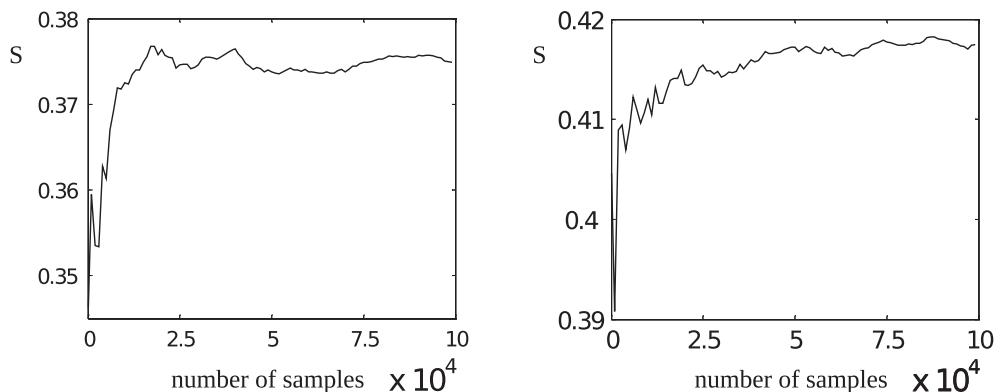


Figure 3.15: The value of $S \equiv E \{ \ln[\cosh(a\nu)] \} / a$ for $a = 1$ and $a = 1.2$ as a function of the number of samples used to produce it.

Example 3.60

Draw 1000 uniformly distributed random numbers with 0 mean and unit variance, in the range $[-1, 1]$. These numbers make up the samples of variable y . Also draw 1000 samples of variable ν from a Gaussian distribution with 0 mean and variance 1. Estimate the negentropy of y using the three formulae (3.181), (3.182) and (3.183). Are the estimates you obtain similar?

After we draw 1000 uniformly distributed random numbers in the range $[-1, 1]$, we calculate their mean m and standard deviation s and normalise them so that they have 0 mean and standard deviation 1, by removing from each number the value of m and dividing the result with s . These normalised numbers make up the values of variable y in formulae (3.181), (3.182) and (3.183). We compute $J_3 = 0.0019$ and $J_4 = 0.0635$.

The value of J_2 depends on the value of a . The values of J_2 for the various values of a are listed in table 3.4.

a	J_2	a	J_2
1.0	0.0010	1.6	0.0027
1.1	0.0013	1.7	0.0029
1.2	0.0016	1.8	0.0032
1.3	0.0018	1.9	0.0034
1.4	0.0021	2.0	0.0036
1.5	0.0024		

Table 3.4: The values of J_2 for various values of parameter a .

The estimates we obtain are not similar and they are not expected to be similar. First of all, J_2 and J_3 are approximate functions proportional to the true value of J , presumably with different constants of proportionality. Second, J_4 is an approximation and not just proportional to the true value of J . It is actually a bad approximation, as the approximation is valid for probability density functions similar to the Gaussian, and the uniform probability density function is not that similar.

We must remember, however, that these approximations are used to maximise the negentropy of the solution, and that in a maximisation or minimisation problem, constants of proportionality do not matter.

Example 3.61

Compute the negentropy of the zero mean uniform probability density function with unit variance, using formulae (3.182) and (3.183). Compare your answer with the values of J_3 and J_4 estimated empirically in example 3.60.

We must compute the expectation value of function $\exp(-x^2/2)$ with respect to the uniform probability density function given by (3.184) for $\sigma = 1$:

$$\begin{aligned}
 E\{e^{-\frac{x^2}{2}}\} &\equiv \frac{1}{2\sqrt{3}} \int_{-\sqrt{3}}^{\sqrt{3}} e^{-\frac{x^2}{2}} dx \\
 &= \frac{1}{2\sqrt{3}} 2 \underbrace{\int_0^{\sqrt{3}} e^{-\left(\frac{x}{\sqrt{2}}\right)^2} d\left(\frac{x}{\sqrt{2}}\right)}_{\text{Set } t \equiv x/\sqrt{2}} \sqrt{2} \\
 &= \sqrt{\frac{2}{3}} \int_0^{\sqrt{3}/\sqrt{2}} e^{-t^2} dt \\
 &= \frac{\sqrt{2}}{\sqrt{3}} \frac{\sqrt{\pi}}{2} \operatorname{erf}(\sqrt{1.5}) \\
 &= \sqrt{\frac{\pi}{6}} \operatorname{erf}(\sqrt{1.5}) \\
 &= 0.6687
 \end{aligned} \tag{3.191}$$

Here we made use of the definition of the error function, equation (3.149), on page 237. Then according to formula (3.182), we have:

$$J \propto \left[0.7132 - \frac{1}{\sqrt{2}} \right]^2 = 0.001478 \tag{3.192}$$

Note that according to formula (3.182) the negentropy is proportional to this value. Formula (3.183), however, is a real approximation. The first term of (3.183) is zero for the uniform probability density function because the uniform probability density function is symmetric and thus the integrand is an odd function integrated over a symmetric interval of integration. The second term of (3.183) is nothing else than the value of (3.182) multiplied with factor $24/(16\sqrt{3} - 27)$. The result is: 0.04977.

This estimate is very different from the exact value computed in example 3.54, which is 0.176 and different from the empirical estimate of J_4 in example 3.60, which is equal to 0.0635. The first discrepancy was expected because we know that approximation (3.183) is a bad approximation for the negentropy of the uniform probability density function. The second discrepancy, however, is unexpected, because in both cases the same formula is used, once theoretically and once experimentally. The empirical result of example 3.60 did not change significantly when 100,000 random numbers were

used to compute it. The first expectation value that appears on the right-hand side of (3.183) remained of the order of 10^{-3} , even when 100,000 random numbers were used to estimate it, never really reaching 0 as it should have done. The second expectation value that appears on the right-hand side of (3.183) was just over 0.66, with changes only appearing in the third and fourth significant digits when the number of random numbers used to estimate it changed. We attribute this to the presence of the exponential, which tends to amplify even small errors when moving from the continuous to the discrete domain. This example shows how careful one should be when using random numbers to perform estimations.

Box 3.5. Derivation of the approximation of negentropy in terms of moments

Assume that the probability density function $f(x)$ we are interested in may be expressed as a perturbation of the Gaussian probability density function $g(x)$,

$$f(x) \simeq g(x)[1 + \epsilon(x)] \quad (3.193)$$

where $\epsilon(x)$ is small for every x . We further assume that both $f(x)$ and $g(x)$ have zero mean and unit variance. Let us compute the entropy of $f(x)$:

$$\begin{aligned} H_f &= - \int_{-\infty}^{+\infty} f(x) \ln[f(x)] dx \\ &\simeq - \int_{-\infty}^{+\infty} g(x)[1 + \epsilon(x)] \ln[g(x)[1 + \epsilon(x)]] dx \\ &= - \int_{-\infty}^{+\infty} [g(x) + g(x)\epsilon(x)] \{\ln[g(x)] + \ln[1 + \epsilon(x)]\} dx \\ &= - \underbrace{\int_{-\infty}^{+\infty} g(x) \ln[g(x)] dx}_{\text{entropy of } g(x)} - \int_{-\infty}^{+\infty} g(x) \ln[1 + \epsilon(x)] dx \\ &\quad - \int_{-\infty}^{+\infty} g(x)\epsilon(x) \ln[g(x)] dx - \int_{-\infty}^{+\infty} g(x)\epsilon(x) \ln[1 + \epsilon(x)] dx \\ &= H_g - \int_{-\infty}^{+\infty} g(x)\epsilon(x) \ln[g(x)] dx - \int_{-\infty}^{+\infty} g(x)[1 + \epsilon(x)] \ln[1 + \epsilon(x)] dx \end{aligned} \quad (3.194)$$

First we note that:

$$g(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \Rightarrow \ln[g(x)] = -\frac{x^2}{2} - \frac{1}{2} \ln(2\pi) \quad (3.195)$$

Then, in the last integral, we use the approximation $(1 + \epsilon) \ln(1 + \epsilon) \simeq \epsilon + \frac{1}{2}\epsilon^2$. So, equation (3.194) takes the form:

$$\begin{aligned} H_f - H_g &\simeq \frac{1}{2} \int_{-\infty}^{+\infty} g(x)\epsilon(x)x^2 dx + \frac{1}{2} \ln(2\pi) \int_{-\infty}^{+\infty} g(x)\epsilon(x)dx \\ &\quad - \int_{-\infty}^{+\infty} g(x)\epsilon(x)dx - \frac{1}{2} \int_{-\infty}^{+\infty} g(x)\epsilon^2(x)dx \end{aligned} \quad (3.196)$$

On the left-hand side of this expression we recognise the negentropy of $f(x)$ with a minus sign in front. From (3.193) we note that

$$\begin{aligned} f(x) &\simeq g(x) + g(x)\epsilon(x) \Rightarrow \\ \int_{-\infty}^{+\infty} f(x)dx &\simeq \int_{-\infty}^{+\infty} g(x)dx + \int_{-\infty}^{+\infty} g(x)\epsilon(x)dx \Rightarrow \\ 1 &\simeq 1 + \int_{-\infty}^{+\infty} g(x)\epsilon(x)dx \Rightarrow \\ \int_{-\infty}^{+\infty} g(x)\epsilon(x)dx &\simeq 0 \end{aligned} \quad (3.197)$$

where we made use of the fact that probability density functions integrate to 1. Similarly,

$$\begin{aligned} \int_{-\infty}^{+\infty} f(x)x^2 dx &\simeq \int_{-\infty}^{+\infty} g(x)x^2 dx + \int_{-\infty}^{+\infty} g(x)\epsilon(x)x^2 dx \Rightarrow \\ 1 &\simeq 1 + \int_{-\infty}^{+\infty} g(x)\epsilon(x)x^2 dx \Rightarrow \\ \int_{-\infty}^{+\infty} g(x)\epsilon(x)x^2 dx &\simeq 0 \end{aligned} \quad (3.198)$$

where we made use of the fact that both probability density functions have zero-mean and unit variance.

Using these results in (3.196) we obtain:

$$J \simeq \frac{1}{2} \int_{-\infty}^{+\infty} g(x)\epsilon^2(x)dx \quad (3.199)$$

It has been shown by statisticians that:

$$\frac{1}{2} \int_{-\infty}^{+\infty} g(x)\epsilon^2(x)dx \simeq \frac{1}{12}\mu_3^2 + \frac{1}{48}\gamma_2^2 \quad (3.200)$$

Then approximation (3.180) follows.

Box 3.6. Approximating the negentropy with nonquadratic functions

Consider all probability density functions $f(x)$ that satisfy the following n constraints

$$E\{G_i(x)\} \equiv \int_{-\infty}^{+\infty} f(x)G_i(x)dx = c_i \quad \text{for } i = 1, 2, \dots, n \quad (3.201)$$

where $G_i(x)$ are some known functions and c_i some known constants. It can be shown that the function with the maximum entropy from among all these functions is the one with the form

$$f_0(x) = Ae^{\sum_i a_i G_i(x)} \quad (3.202)$$

where A and a_i are some functions of c_i . For the special case of $n = 2$, $G_1(x) = x$, $c_1 = 0$, $G_2(x) = x^2$ and $c_2 = 1$, see Box 3.4, on page 246.

Let us assume that $f(x)$ is similar to a normal probability density function $g(x)$ with the same mean (zero) and the same variance (unit). We may express that by saying that $f(x)$, on the top of the n constraints listed above, obeys two more constraints, with:

$$\begin{aligned} G_{n+1}(x) &= x & c_{n+1} &= 0 \\ G_{n+2}(x) &= x^2 & c_{n+2} &= 1 \end{aligned} \quad (3.203)$$

For completeness, we also consider function $G_0 =$ a constant, written as $\ln A$ with $c_0 = \ln A$.

Further, we assume that functions G_i form an orthonormal system of functions, for $i = 1, \dots, n$, with weight $g(x)$

$$\int_{-\infty}^{+\infty} g(x)G_i(x)G_j(x)dx = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad (3.204)$$

and they are orthogonal to functions G_0 , $G_{n+1}(x)$ and $G_{n+2}(x)$:

$$\int_{-\infty}^{+\infty} g(x)G_i(x)x^r dx = 0 \quad \text{for } r = 0, 1, 2 \text{ and } \forall i \quad (3.205)$$

In view of the above, the function with the maximum entropy that obeys the constraints must have the form:

$$\begin{aligned} f_0(x) &= e^{\ln A + a_{n+1}x + a_{n+2}x^2 + \sum_{i=1}^n a_i G_i(x)} \\ &= Ae^{-\frac{x^2}{2} + a_{n+1}x + (a_{n+2} + \frac{1}{2})x^2 + \sum_{i=1}^n a_i G_i(x)} \\ &= Ae^{-\frac{x^2}{2}} e^{a_{n+1}x + (a_{n+2} + \frac{1}{2})x^2 + \sum_{i=1}^n a_i G_i(x)} \end{aligned} \quad (3.206)$$

If $f(x)$ is very near a Gaussian, coefficient a_{n+2} in the above expansion must be the dominant one, with its value near $-1/2$. So, the last factor on the right-hand side of

(3.206) may be considered to be of the form e^ϵ where ϵ is very small. Then this factor may be approximated as $e^\epsilon \simeq 1 + \epsilon$:

$$\begin{aligned} f_0(x) &\simeq Ae^{-\frac{x^2}{2}} \left[1 + a_{n+1}x + \left(a_{n+2} + \frac{1}{2} \right) x^2 + \sum_{i=1}^n a_i G_i(x) \right] \\ &= A\sqrt{2\pi}g(x) \left[1 + a_{n+1}x + \left(a_{n+2} + \frac{1}{2} \right) x^2 + \sum_{i=1}^n a_i G_i(x) \right] \quad (3.207) \end{aligned}$$

We shall take the various moments of this expression in order to work out values for the various parameters that appear in it. First, let us multiply both sides with dx and integrate them from $-\infty$ to $+\infty$:

$$\begin{aligned} \underbrace{\int_{-\infty}^{+\infty} f_0(x)dx}_{=1(\text{integral of a pdf})} &= A\sqrt{2\pi} \underbrace{\int_{-\infty}^{+\infty} g(x)dx}_{=1(\text{integral of a pdf})} + A\sqrt{2\pi}a_{n+1} \underbrace{\int_{-\infty}^{+\infty} g(x)x dx}_{=0(\text{mean})} \\ &\quad + A\sqrt{2\pi} \left(a_{n+2} + \frac{1}{2} \right) \underbrace{\int_{-\infty}^{+\infty} g(x)x^2 dx}_{=1(\text{unit variance})} \\ &\quad + A\sqrt{2\pi} \sum_{i=1}^n a_i \underbrace{\int_{-\infty}^{+\infty} g(x)G_i(x)dx}_{=0(\text{orthogonality of } G_i \text{ with } G_0)} \Rightarrow \\ 1 &= A\sqrt{2\pi} + A\sqrt{2\pi} \left(a_{n+2} + \frac{1}{2} \right) \quad (3.208) \end{aligned}$$

We then multiply both sides of (3.207) with $x dx$ and integrate:

$$\begin{aligned} \underbrace{\int_{-\infty}^{+\infty} f_0(x)x dx}_{=0(\text{mean})} &= A\sqrt{2\pi} \underbrace{\int_{-\infty}^{+\infty} g(x)x dx}_{=0(\text{mean})} + A\sqrt{2\pi}a_{n+1} \underbrace{\int_{-\infty}^{+\infty} g(x)x^2 dx}_{=1(\text{unit variance})} \\ &\quad + A\sqrt{2\pi} \left(a_{n+2} + \frac{1}{2} \right) \underbrace{\int_{-\infty}^{+\infty} g(x)x^3 dx}_{=0(\text{odd integrand})} \\ &\quad + A\sqrt{2\pi} \sum_{i=1}^n a_i \underbrace{\int_{-\infty}^{+\infty} g(x)G_i(x)x dx}_{=0(\text{orthogonality of } G_i \text{ with } G_{n+1})} \Rightarrow \\ 0 &= A\sqrt{2\pi}a_{n+1} \Rightarrow a_{n+1} = 0 \quad (3.209) \end{aligned}$$

Next multiply both sides of (3.207) with $x^2 dx$ and integrate:

$$\begin{aligned}
 \underbrace{\int_{-\infty}^{+\infty} f_0(x)x^2 dx}_{=1(\text{unit variance})} &= A\sqrt{2\pi} \underbrace{\int_{-\infty}^{+\infty} g(x)x^2 dx}_{=1(\text{unit variance})} + A\sqrt{2\pi}a_{n+1} \underbrace{\int_{-\infty}^{+\infty} g(x)x^3 dx}_{=0(\text{odd integrand})} \\
 &\quad + A\sqrt{2\pi} \left(a_{n+2} + \frac{1}{2} \right) \underbrace{\int_{-\infty}^{+\infty} g(x)x^4 dx}_{=3(\text{see example (3.50)})} \\
 &\quad + A\sqrt{2\pi} \sum_{i=1}^n a_i \underbrace{\int_{-\infty}^{+\infty} g(x)G_i(x)x^2 dx}_{=0(\text{orthogonality of } G_i \text{ with } G_{n+2})} \Rightarrow \\
 1 &= A\sqrt{2\pi} + 3A\sqrt{2\pi} \left(a_{n+2} + \frac{1}{2} \right) \tag{3.210}
 \end{aligned}$$

By subtracting (3.208) from (3.210) by parts, we obtain:

$$0 = 2A\sqrt{2\pi} \left(a_{n+2} + \frac{1}{2} \right) \Rightarrow a_{n+2} = -\frac{1}{2} \tag{3.211}$$

From (3.208) and (3.211) we deduce that:

$$1 = A\sqrt{2\pi} \Rightarrow A = \frac{1}{\sqrt{2\pi}} \tag{3.212}$$

Finally, we multiply both sides of (3.207) with $G_j(x)dx$ and integrate. We make also use of the fact that $A\sqrt{2\pi} = 1$:

$$\begin{aligned}
 \underbrace{\int_{-\infty}^{+\infty} f_0(x)G_j(x)dx}_{=c_j} &= \underbrace{\int_{-\infty}^{+\infty} g(x)G_j(x)dx}_{=0(\text{orthogonality of } G_j \text{ with } G_0)} + a_{n+1} \underbrace{\int_{-\infty}^{+\infty} g(x)xG_j(x)dx}_{=0(\text{orthogonality of } G_j \text{ with } G_{n+1})} \\
 &\quad + \left(a_{n+2} + \frac{1}{2} \right) \underbrace{\int_{-\infty}^{+\infty} g(x)x^2 G_j(x)dx}_{=0(\text{orthogonality of } G_j \text{ with } G_{n+2})} \\
 &\quad + \sum_{i=1}^n a_i \underbrace{\int_{-\infty}^{+\infty} g(x)G_i(x)G_j(x)dx}_{=\delta_{ij}} \Rightarrow \\
 c_j &= a_j \Rightarrow a_j = c_j \tag{3.213}
 \end{aligned}$$

Then, equation (3.207) takes the form:

$$f_0(x) \simeq g(x) \left[1 + \sum_{i=1}^n c_i G_i(x) \right] \tag{3.214}$$

This equation has the same form as (3.193) with $\epsilon(x) \equiv \sum_{i=1}^n c_i G_i(x)$. Then we know from Box 3.5, on page 252, that the negentropy of $f_0(x)$ is given by (3.199). That is:

$$\begin{aligned}
J &\simeq \frac{1}{2} \int_{-\infty}^{+\infty} g(x) \left[\sum_{i=1}^n c_i G_i(x) \right]^2 dx \\
&= \frac{1}{2} \int_{-\infty}^{+\infty} g(x) \sum_{i=1}^n c_i G_i(x) \sum_{j=1}^n c_j G_j(x) dx \\
&= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n c_i c_j \int_{-\infty}^{+\infty} g(x) G_i(x) G_j(x) dx \\
&= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n c_i c_j \delta_{ij} \\
&= \frac{1}{2} \sum_{i=1}^n c_i^2 \\
&= \frac{1}{2} \sum_{i=1}^n \left[\int_{-\infty}^{+\infty} f(x) G_i(x) dx \right]^2 \\
&= \frac{1}{2} \sum_{i=1}^n [E\{G_i(x)\}]^2
\end{aligned} \tag{3.215}$$

In practice, we may use only one function in this sum.

Box 3.7. Selecting the nonquadratic functions with which to approximate the negentropy

First we must select functions $G_i(x)$ so that function $f_0(x)$, given by (3.202), is integrable. This will happen only if functions $G_i(x)$ grow at most as fast as x^2 with $|x|$ increasing. The next criterion is to choose these functions so that they satisfy constraints (3.204) and (3.205).

Let us consider two functions $\tilde{G}_1(x)$ and $\tilde{G}_2(x)$ that grow slowly enough with $|x|$, and the first one is odd and the second one is even. We shall show here how to modify them so they satisfy constraints (3.204) and (3.205).

Let us construct from them two functions $\hat{G}_1(x)$ and $\hat{G}_2(x)$

$$\hat{G}_1(x) \equiv \tilde{G}_1(x) + \alpha x \tag{3.216}$$

$$\hat{G}_2(x) \equiv \tilde{G}_2(x) + \beta x^2 + \gamma \tag{3.217}$$

where α , β and γ are some constants, the values of which will be determined so that

constraints (3.205) are satisfied. Note that the orthogonality constraint is automatically satisfied since $g(x)\hat{G}_1(x)\hat{G}_2(x)$ is an odd function. To ensure orthonormality, some scaling of these functions has to take place, but this may be done at the end very easily. Also note that $\hat{G}_1(x)$ is automatically orthogonal to G_0 and $G_4 \equiv x^2$ with weight $g(x)$ since $g(x)\hat{G}_1(x)x^r$ for $r = 0, 2$ is an odd function. For $r = 1$ constraint (3.205) has the form:

$$\begin{aligned} \int_{-\infty}^{+\infty} g(x)\hat{G}_1(x)xdx &= 0 \Rightarrow \\ \int_{-\infty}^{+\infty} g(x)\tilde{G}_1(x)xdx + \alpha \underbrace{\int_{-\infty}^{+\infty} g(x)x^2dx}_{=\text{variance}=1} &= 0 \Rightarrow \\ \alpha &= -\int_{-\infty}^{+\infty} g(x)\tilde{G}_1(x)xdx \end{aligned} \quad (3.218)$$

Next note that as $\hat{G}_2(x)$ is even, it automatically satisfies constraint (3.205) for $r = 1$. To satisfy the same constraints for $r = 0$ and $r = 2$, we must have:

$$\begin{aligned} \int_{-\infty}^{+\infty} g(x)\hat{G}_2(x)dx &= 0 \\ \int_{-\infty}^{+\infty} g(x)\tilde{G}_2(x)x^2dx &= 0 \end{aligned} \quad (3.219)$$

Or:

$$\begin{aligned} \int_{-\infty}^{+\infty} g(x)\tilde{G}_2(x)dx + \beta \underbrace{\int_{-\infty}^{+\infty} g(x)x^2dx}_{=\text{variance}=1} + \gamma \underbrace{\int_{-\infty}^{+\infty} g(x)dx}_{=1} &= 0 \\ \int_{-\infty}^{+\infty} g(x)\tilde{G}_2(x)x^2dx + \beta \underbrace{\int_{-\infty}^{+\infty} g(x)x^4dx}_{=3(\text{example 3.50})} + \gamma \underbrace{\int_{-\infty}^{+\infty} g(x)x^2dx}_{=\text{variance}=1} &= 0 \end{aligned} \quad (3.220)$$

Or:

$$\begin{aligned} \int_{-\infty}^{+\infty} g(x)\tilde{G}_2(x)dx + \beta + \gamma &= 0 \\ \int_{-\infty}^{+\infty} g(x)\tilde{G}_2(x)x^2dx + 3\beta + \gamma &= 0 \end{aligned} \quad (3.221)$$

Subtracting the first from the second equation, we obtain:

$$\beta = \frac{1}{2} \int_{-\infty}^{+\infty} g(x)\tilde{G}_2(x)dx - \frac{1}{2} \int_{-\infty}^{+\infty} g(x)\tilde{G}_2(x)x^2dx \quad (3.222)$$

Using this in the first of (3.221), we obtain:

$$\gamma = \frac{1}{2} \int_{-\infty}^{+\infty} g(x) \tilde{G}_2(x) x^2 dx - \frac{3}{2} \int_{-\infty}^{+\infty} g(x) \tilde{G}_2(x) dx \quad (3.223)$$

So, the two functions we should use to expand the unknown probability density function $f(x)$ could be

$$G_1(x) \equiv \frac{1}{\delta_1} \left[\tilde{G}_1(x) - x \int_{-\infty}^{+\infty} g(z) \tilde{G}_1(z) z dz \right] \quad (3.224)$$

$$\begin{aligned} G_2(x) \equiv & \frac{1}{\delta_2} \left\{ \tilde{G}_2(x) + x^2 \left[\frac{1}{2} \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) dz - \frac{1}{2} \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) z^2 dz \right] \right. \\ & \left. + \frac{1}{2} \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) z^2 dz - \frac{3}{2} \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) dz \right\} \end{aligned} \quad (3.225)$$

where δ_1 and δ_2 ensure orthonormality. Note that here we replaced the dummy variable in the integrals of (3.224) and (3.225) with z to avoid confusion.

We can now compute the values of the corresponding parameters c_i which appear in the approximation of the negentropy by (3.215):

$$\begin{aligned} c_1 & \equiv \int_{-\infty}^{+\infty} f(x) G_1(x) dx \\ & = \frac{1}{\delta_1} \left\{ \int_{-\infty}^{+\infty} f(x) \tilde{G}_1(x) dx - \underbrace{\int_{-\infty}^{+\infty} f(x) x dx}_{=0(\text{mean})} \int_{-\infty}^{+\infty} g(z) \tilde{G}_1(z) z dz \right\} \\ & = \frac{1}{\delta_1} E\{\tilde{G}_1(x)\} \end{aligned} \quad (3.226)$$

$$\begin{aligned} c_2 & \equiv \int_{-\infty}^{+\infty} f(x) G_2(x) dx \\ & = \frac{1}{\delta_2} \left\{ \int_{-\infty}^{+\infty} f(x) \tilde{G}_2(x) dx \right. \\ & \quad \left. + \underbrace{\int_{-\infty}^{+\infty} f(x) x^2 dx}_{=1(\text{unit variance})} \left[\frac{1}{2} \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) dz - \frac{1}{2} \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) z^2 dz \right] \right. \\ & \quad \left. + \int_{-\infty}^{+\infty} f(x) dx \left[\frac{1}{2} \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) z^2 dz - \frac{3}{2} \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) dz \right] \right\} \end{aligned}$$

$$\begin{aligned} & = \frac{1}{\delta_2} \left\{ \int_{-\infty}^{+\infty} f(x) \tilde{G}_2(x) dx - \int_{-\infty}^{+\infty} g(z) \tilde{G}_2(z) dz \right\} \\ & = \frac{1}{\delta_2} [E\{\tilde{G}_2(x)\} - E\{\tilde{G}_2(z)\}] \end{aligned} \quad (3.227)$$

In the last equality it is understood that the expectation of function $\tilde{G}_2(z)$ is computed over a normally distributed variable z .

If we decide to use only one function G_i , and we select to use, say, only an even function, then for $\tilde{G}_2(x) \equiv \ln(\cosh(ax))/a$ we obtain expression (3.181), while for $\tilde{G}_2(x) \equiv \exp(-x^2/2)$, we obtain expression (3.182). Note that these expressions produce numbers approximately proportional to the true value of the negentropy, because we have omitted from them the normalising factors δ_1 and δ_2 .

Note also that none of these “approximations” captures any asymmetric characteristics of probability density function $f(x)$. For example, for a dark image, where no negative numbers are allowed, the assumption that the unknown probability density function $f(x)$ is symmetric must be clearly wrong, as any tail of negative numbers is either truncated or mapped into the positive numbers. This is particularly so for medical images, where the histograms of the grey values are not really Gaussian, but they exhibit strong asymmetries. To capture such asymmetries in the histograms of the data, we must use at least two functions in the expansion of the negentropy, one of them odd and one of them even.

So, if we wish to use also an odd function in the expansion, we may use $\tilde{G}_1(x) \equiv x \exp(-x^2/2)$. The use of this function in conjunction with $\tilde{G}_2(x) \equiv \exp(-x^2/2)$ results in approximation (3.183). The coefficients with which the two terms in the formula are multiplied come from factor 1/2 which has to multiply c_i^2 and from the normalisation constants of the functions used (see examples 3.63 and 3.64), ie these coefficients are $1/(2\delta_1^2)$ and $1/(2\delta_2^2)$, respectively.

Example B3.62

For function $\tilde{G}_2(x) \equiv \exp(-x^2/2)$ compute $E\{\tilde{G}_2(z)\}$ that appears in (3.227).

$$\begin{aligned}
 E\{\tilde{G}_2(z)\} &\equiv \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{z^2}{2}} e^{-\frac{z^2}{2}} dz \\
 &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-z^2} dz \\
 &= \frac{1}{\sqrt{2}}
 \end{aligned} \tag{3.228}$$

Here we made use of (3.159), on page 241. This value appears in approximations (3.182) and (3.183), on page 246, instead of $E\{\tilde{G}_2(z)\}$.

Example B3.63

For functions $\tilde{G}_1(x) \equiv x \exp(-x^2/2)$ and $\tilde{G}_2(x) \equiv \exp(-x^2/2)$ compute the values of parameters α , β and γ using equations (3.218), (3.222) and (3.223).

From (3.218) we have

$$\begin{aligned}
\alpha &= -\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} e^{-\frac{x^2}{2}} x^2 dx \\
&= -\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-x^2} x^2 dx \\
&= -\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} x d(e^{-x^2}) \left(-\frac{1}{2} \right) \\
&= \frac{1}{2\sqrt{2\pi}} x e^{-x^2} \Big|_{-\infty}^{+\infty} - \frac{1}{2\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-x^2} dx \\
&= -\frac{1}{2\sqrt{2\pi}} \sqrt{\pi} \\
&= -\frac{1}{2\sqrt{2}}
\end{aligned} \tag{3.229}$$

where we made use of (3.159), on page 241.

For $\tilde{G}_2(x)$ we need the following integral

$$\begin{aligned}
\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} e^{-\frac{x^2}{2}} dx &= \\
\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-x^2} dx &= \\
\frac{1}{\sqrt{2\pi}} \sqrt{\pi} &= \frac{1}{\sqrt{2}}
\end{aligned} \tag{3.230}$$

where we made use of (3.159). The second term in (3.222) is $\alpha/2$, ie it is $-1/(4\sqrt{2})$. Then from (3.222) we have:

$$\beta = \frac{1}{2} \frac{1}{\sqrt{2}} - \frac{1}{4\sqrt{2}} = \frac{1}{4\sqrt{2}} \tag{3.231}$$

From (3.223) we have:

$$\gamma = \frac{1}{4\sqrt{2}} - \frac{3}{2} \frac{1}{\sqrt{2}} = -\frac{5}{4\sqrt{2}} \tag{3.232}$$

Example B3.64

Calculate the normalisation constants that will make functions $\hat{G}_1(x) \equiv x \exp(-x^2/2) + \alpha x$ and $\hat{G}_2(x) \equiv \exp(-x^2/2) + \beta x^2 + \gamma$ orthonormal with weight the Gaussian kernel $g(x)$ with zero mean and unit variance. The values of α , β and γ have been computed in example 3.63.

These two functions are already orthogonal. Further, each one must integrate to 1 when squared and integrated with kernel $g(x)$. We must work out the values of these integrals.

$$\begin{aligned}
\int_{-\infty}^{+\infty} g(x) \hat{G}_1(x)^2 dx &= \\
\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} \left[e^{-x^2} x^2 + \alpha^2 x^2 + 2\alpha e^{-\frac{x^2}{2}} x^2 \right] dx &= \\
\frac{1}{\sqrt{2\pi}} \left\{ \int_{-\infty}^{+\infty} e^{-\frac{3x^2}{2}} x^2 dx + 2\alpha \int_{-\infty}^{+\infty} e^{-x^2} x^2 dx \right\} + \alpha^2 \underbrace{\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} x^2 dx}_{=1-\text{variance}} &= \\
\frac{1}{\sqrt{2\pi}} \left\{ -\frac{1}{3} \int_{-\infty}^{+\infty} x d(e^{-\frac{3x^2}{2}}) - 2\alpha \frac{1}{2} \int_{-\infty}^{+\infty} x d(e^{-x^2}) \right\} + \alpha^2 &= \\
\frac{1}{\sqrt{2\pi}} \left\{ -\underbrace{\frac{1}{3} x e^{-\frac{3x^2}{2}} \Big|_{-\infty}^{+\infty}}_{=0} + \underbrace{\frac{1}{3} \int_{-\infty}^{+\infty} e^{-\frac{3x^2}{2}} dx}_{\text{Set } z \equiv \sqrt{3}x/\sqrt{2}} \right. &= \\
\left. - \underbrace{\alpha x e^{-x^2} \Big|_{-\infty}^{+\infty}}_{=0} + \alpha \underbrace{\int_{-\infty}^{+\infty} e^{-x^2} dx}_{=\sqrt{\pi} \text{ (eqn(3.159))}} \right\} + \alpha^2 &= \\
\frac{1}{\sqrt{2\pi}} \left\{ \frac{1}{3} \sqrt{\frac{2}{3}} \underbrace{\int_{-\infty}^{+\infty} e^{-z^2} dz}_{=\sqrt{\pi} \text{ (eqn(3.159))}} + \alpha \sqrt{\pi} \right\} + \alpha^2 &= \\
\frac{1}{3\sqrt{3}} + \alpha \frac{1}{\sqrt{2}} + \alpha^2 &= \\
\frac{1}{3\sqrt{3}} - \frac{1}{4} + \frac{1}{8} &= \\
\frac{8\sqrt{3} - 9}{72} &\equiv \delta_1^2
\end{aligned} \tag{3.233}$$

Here we made use of (3.229). So, $\hat{G}_1(x)$ should be normalised by being multiplied with

$\sqrt{72/(8\sqrt{3}-9)}$ to form function $G_1(x)$. When $G_1(x)$ is used in the calculation of the negentropy, this factor squared and divided by 2 will appear as coefficient of the first term of approximation (3.183) (factor $1/(2\delta_1^2)$, see Box 3.7, on page 257). For $\hat{G}_2(x)$ we have:

$$\begin{aligned}
 & \int_{-\infty}^{+\infty} g(x) \hat{G}_2(x)^2 dx = \\
 & \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} e^{-x^2} dx + \beta^2 \underbrace{\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} x^4 dx}_{=3(\text{example (3.50)})} + \gamma^2 \underbrace{\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} dx}_{=1} \\
 & + 2\beta \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} e^{-\frac{x^2}{2}} x^2 dx + 2\gamma \underbrace{\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} e^{-\frac{x^2}{2}} dx}_{=\sqrt{\pi}(\text{eqn (3.159)})} \\
 & + 2\beta\gamma \underbrace{\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2}} x^2 dx}_{=1=\text{variance}} = \\
 & \frac{1}{\sqrt{2\pi}} \underbrace{\int_{-\infty}^{+\infty} e^{-\frac{3x^2}{2}} dx}_{z \equiv \sqrt{3}x/\sqrt{2}} + 3\beta^2 + \gamma^2 + 2\beta \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-x^2} x^2 dx + \gamma\sqrt{2} + 2\beta\gamma = \\
 & \frac{1}{\sqrt{2\pi}} \frac{\sqrt{2}}{\sqrt{3}} \sqrt{\pi} + 3\beta^2 + \gamma^2 + 2\beta \frac{1}{\sqrt{2\pi}} \frac{-1}{2} \int_{-\infty}^{+\infty} x d(e^{-x^2}) + \gamma\sqrt{2} + 2\beta\gamma = \\
 & \frac{1}{\sqrt{3}} + 3\beta^2 + \gamma^2 - \beta \frac{1}{\sqrt{2\pi}} \underbrace{xe^{-x^2} \Big|_{-\infty}^{+\infty}}_{=0} + \beta \frac{1}{\sqrt{2\pi}} \underbrace{\int_{-\infty}^{+\infty} e^{-x^2} dx}_{=\sqrt{\pi}} + \gamma\sqrt{2} + 2\beta\gamma = \\
 & \frac{1}{\sqrt{3}} + 3\beta^2 + \gamma^2 + \beta \frac{1}{\sqrt{2\pi}} \sqrt{\pi} + \gamma\sqrt{2} + 2\beta\gamma = \\
 & \frac{1}{\sqrt{3}} + \frac{3}{32} + \frac{25}{32} + \frac{1}{8} - \frac{5}{4} - \frac{10}{32} = \\
 & \frac{1}{\sqrt{3}} - \frac{18}{32} = \\
 & \frac{16\sqrt{3} - 27}{48} \equiv \delta_2^2 \tag{3.234}
 \end{aligned}$$

Here we made use of the values of β and γ given by (3.231) and (3.232), respectively. So, $\hat{G}_2(x)$ should be normalised by being multiplied with $1/\delta_2 = \sqrt{48}/\sqrt{16\sqrt{3}-27}$ to form function $G_2(x)$. The factor then that should appear in the approximation of negentropy should be $1/(2\delta_2^2) = 24/(16\sqrt{3}-27)$ (see Box 3.7) and that is what appears in equation (3.183), on page 246.

How do we apply the central limit theorem to solve the cocktail party problem?

The solution of a linear problem is also linear. So, we may express the unknown signals $s_1(t)$ and $s_2(t)$ as linear combinations of the known recordings $x_1(t)$ and $x_2(t)$,

$$\begin{aligned} s_1(t) &= w_{11}x_1(t) + w_{12}x_2(t) \\ s_2(t) &= w_{21}x_1(t) + w_{22}x_2(t) \end{aligned} \quad (3.235)$$

where w_{11} , w_{12} , w_{21} and w_{22} are the weights with which we have to combine the recordings to recover the original signals. Matrix W , made up from these weights, is an estimate of the inverse of the unknown matrix A made up from the blending factors. Grossly, the idea then is to hypothesise various combinations of values of the weights and for each hypothesised set of values to compute the degree of non-Gaussianity of the recovered sources, adjusting the weight values until the sources are as non-Gaussian as possible. We shall see later that more elaborate methods, based on some analytical results on negentropy, are used in practice.

How may ICA be used in image processing?

Let us assume that we have a collection of I images of size $M \times N$. Each image is made up of MN pixels. We may consider that the outcome of the assumed underlying random experiment is the value at a specific pixel position and this outcome is different for the different images we have, drawn from some (unknown) distribution. In other words, the random experiment decides the combination of values that make up the “signature” of a pixel across the different images. We may then visualise this distribution by plotting each pixel in a coordinate system with as many axes as we have images, by measuring the value of a pixel in each image along the corresponding axis. This will lead to a plot like the one shown in figure 3.16a.

Alternatively, we may assume that the underlying random experiment decides the combination of values that make up an image. In this case, we may consider a coordinate system with as many axes as we have pixels and measure the value of a pixel in a given image along the corresponding axis. Each image is then represented by a point in such a space. This will lead to a plot like the one shown in figure 3.16b.

In either of the above cases, the independent components may be worked out by computing statistics over the cloud of points we create. Note that, in general, the cloud of points has an irregular shape, possibly elongated along some directions more than along others. According to figure 3.12, on page 234, we should try to search for the independent components among zero-mean orthogonal and uncorrelated random variables.

How do we search for the independent components?

Let us concentrate on the case shown in figure 3.16a. Let us say that we denote by p_{ki} the value of the k th pixel in the i th image. The cloud of points is made up by allowing k to take values from 1 to MN . To make this variable of zero-mean, we must remove from each p_{ki} its mean value:

$$\bar{p}_i \equiv \frac{1}{MN} \sum_{k=1}^{MN} p_{ki} \quad (3.236)$$

Let us call \tilde{p}_{ki} the zero-mean versions of p_{ki} : $\tilde{p}_{ki} \equiv p_{ki} - \bar{p}_i$. Note that \bar{p}_i is the mean value of image i . This is equivalent to describing the cloud of points in 3.16a using a coordinate

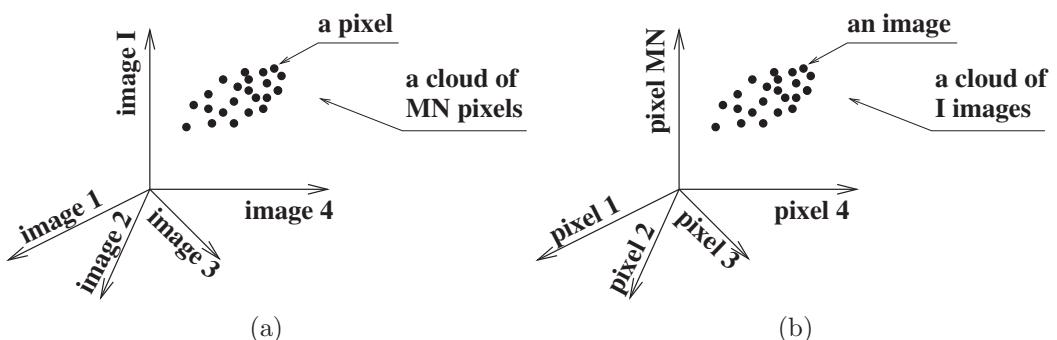


Figure 3.16: (a) If we assume that a random experiment decides which combination of pixel values across all images corresponds to a pixel position, then each pixel is a point in a coordinate system with as many axes as images we have. Assuming that we have I images of size $M \times N$, the cloud of points created this way represents MN outcomes of the random experiment, as each pixel is one such outcome. (b) If we assume that a random experiment decides which combination of pixel values makes up an image, then each image is a point in a coordinate system with as many axes as pixels in the image. Assuming that we have I images of size $M \times N$, the cloud of points created this way represents I outcomes of the random experiment, as each image is one such outcome.

system that is centred at the centre of the cloud and has its axes parallel one by one with those of the original coordinate system (see figure 3.17a).

We may then define a coordinate system in which the components that make up each pixel are uncorrelated. We learnt how to do that in the section on K-L: we have to work out the eigenvectors of the $I \times I$ autocorrelation matrix of the zero-mean data. Such a matrix may have at most I eigenvalues, but in general it will have $E \leq I$. The C_{ij} component of the autocorrelation matrix, corresponding to the correlation between images i and j , is given by:

$$C_{ij} \equiv \frac{1}{MN} \sum_{k=1}^{MN} \tilde{p}_{ki} \tilde{p}_{kj} \quad (3.237)$$

The eigenvectors of this matrix define a very specific coordinate system in which each point is represented by uncorrelated values. The eigenvector that corresponds to the largest eigenvalue coincides with the axis of maximum elongation of the cloud, while the eigenvector that corresponds to the smallest eigenvalue coincides with the axis of minimum elongation of the cloud. This does not allow us any choice of the coordinate system. Imagine, however, if the cloud were perfectly round. All coordinate systems defined with their origins at the centre of the cloud would have been equivalent (see figure 3.17b). We could then choose any one of them to express the data. Such a degeneracy would be expressed by matrix C having only one multiple eigenvalue. The multiplicity of the eigenvalue would be the same as the dimensionality of the cloud of points, ie the same as the number of axes we could define for the coordinate system. Data that are represented by a spherical cloud are called **whitened** data. Such data allow one to choose from among many coordinate systems, one in which the

components of the data are more independent than in any other. So, identifying independent components from our data may be achieved by first whitening the data, in order to have an unlimited number of options of creating uncorrelated components from them, and choosing from among them the most independent ones.

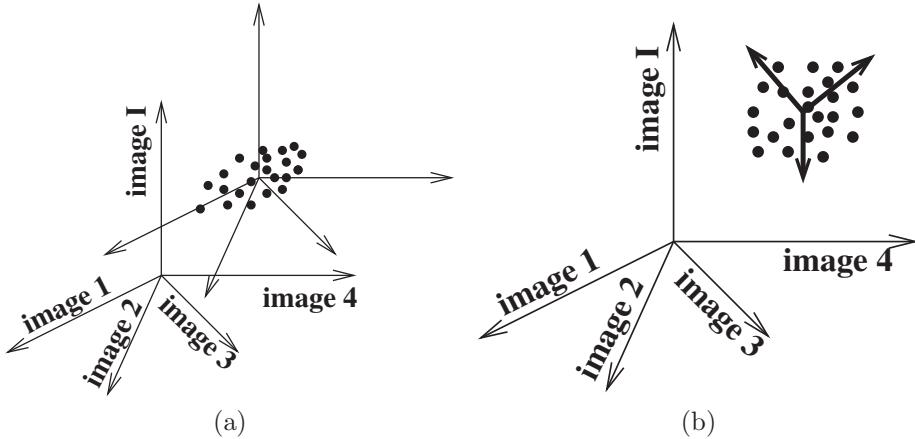


Figure 3.17: (a) Removing the mean of each component is equivalent to shifting the original coordinate system so that its centre coincides with the centre of the cloud of points and each axis remains parallel with itself. (b) When the cloud of points is spherical, all coordinate systems, centred at the centre of the cloud, like the one indicated here by the thick arrows, are equivalent in describing the data.

How can we whiten the data?

Let us consider first how we create the uncorrelated components of p_{ki} by using the eigenvectors of matrix C . Let us call the l th eigenvector of C \mathbf{u}_l and the corresponding eigenvalue λ_l . Let us say that C has E eigenvectors in all, so $l = 1, 2, \dots, E$. Each point in figure 3.16a is represented by a position vector $\tilde{\mathbf{p}}_k \equiv (\tilde{p}_{k1}, \tilde{p}_{k2}, \dots, \tilde{p}_{kI})^T$ in the coordinate system centred at the centre of the cloud. This position vector is projected on each one of the eigenvectors \mathbf{u}_l in turn, to identify the components of vector $\tilde{\mathbf{p}}_k$ in the new coordinate system made up from these eigenvectors. Let us denote these projections by $w_{kl} \equiv \tilde{\mathbf{p}}_k^T \mathbf{u}_l$. The combination of values $(w_{1l}, w_{2l}, \dots, w_{Ml})$ make up the l th uncorrelated component of the original data. For fixed l , the values of w_{kl} for $k = 1, 2, \dots, MN$ have a standard deviation equal to λ_l . If we want the spread of these values to be the same along all axes, defined by \mathbf{u}_l for the different values of l , we must divide them with the corresponding λ_l , ie we must use \tilde{w}_{kl} instead of w_{kl} , given by: $\tilde{w}_{kl} \equiv w_{kl}/\sqrt{\lambda_l}$. This is equivalent to saying

$$\tilde{w}_{kl} \equiv \frac{w_{kl}}{\sqrt{\lambda_l}} = \tilde{\mathbf{p}}_k^T \mathbf{u}_l \frac{1}{\sqrt{\lambda_l}} = \tilde{\mathbf{p}}_k^T \left(\mathbf{u}_l \frac{1}{\sqrt{\lambda_l}} \right) \equiv \tilde{\mathbf{p}}_k^T \tilde{\mathbf{u}}_l \quad (3.238)$$

where we defined the unit vectors of the axes to be $\tilde{\mathbf{u}}_l$, so that the points are equally spread along all axes.

In summary, in order to whiten the data, we use as unit vectors of the coordinate system the eigenvectors of matrix C , divided by the square root of the corresponding eigenvalue.

How can we select the independent components from whitened data?

Once the data are whitened, we must select our first axis so that the projections of all points on this axis are as non-Gaussianly distributed as possible. Then we select a second axis so that it is orthogonal to the first and at the same time the projections of all points on it are as non-Gaussian as possible. The process continues until we select all axes, making sure that each new axis we define is orthogonal to all previously defined axes. In Box 3.9 it is shown that such axes may be defined by iteratively solving an appropriate equation.

The MN E -tuples of values we shall define this way are the independent components of the original MN I -tuples we started with. They are the coefficients of the expansion of each of the MN original I -tuples in terms of the basis vectors defined by the axes we selected. Indeed, the tip of each unit vector we select has a certain position vector in relation to the original coordinate system. The components of this position vector make up the basis I -tuples in terms of which all other I -tuples may be expressed.

So far, our discussion referred to figure 3.16a. This is useful for linear spectral unmixing, a problem we shall discuss in Chapter 7 (see page 695). However, in most other image processing applications, the assumed underlying random experiment is usually the one shown in figure 3.16b.

Example B3.65

Differentiation by a vector is defined as differentiation with respect to each of the elements of the vector. For vectors \mathbf{a} , \mathbf{b} and \mathbf{f} show that:

$$\frac{\partial \mathbf{f}^T \mathbf{a}}{\partial \mathbf{f}} = \mathbf{a} \quad \text{and} \quad \frac{\partial \mathbf{b}^T \mathbf{f}}{\partial \mathbf{f}} = \mathbf{b} \quad (3.239)$$

Assume that vectors \mathbf{a} , \mathbf{b} and \mathbf{f} are $N \times 1$. Then we have:

$$\mathbf{f}^T \mathbf{a} = (f_1 \ f_2 \ \dots \ f_N) \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix} = f_1 a_1 + f_2 a_2 + \dots + f_N a_N \quad (3.240)$$

Use this in:

$$\frac{\partial \mathbf{f}^T \mathbf{a}}{\partial \mathbf{f}} \equiv \begin{pmatrix} \frac{\partial \mathbf{f}^T \mathbf{a}}{\partial f_1} \\ \frac{\partial \mathbf{f}^T \mathbf{a}}{\partial f_2} \\ \vdots \\ \frac{\partial \mathbf{f}^T \mathbf{a}}{\partial f_N} \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix} \Rightarrow \frac{\partial \mathbf{f}^T \mathbf{a}}{\partial \mathbf{f}} = \mathbf{a} \quad (3.241)$$

Similarly:

$$\mathbf{b}^T \mathbf{f} = b_1 f_1 + b_2 f_2 + \dots + b_N f_N \quad (3.242)$$

Then:

$$\frac{\partial \mathbf{b}^T \mathbf{f}}{\partial \mathbf{f}} = \begin{pmatrix} \frac{\partial \mathbf{b}^T \mathbf{f}}{\partial f_1} \\ \frac{\partial \mathbf{b}^T \mathbf{f}}{\partial f_2} \\ \vdots \\ \frac{\partial \mathbf{b}^T \mathbf{f}}{\partial f_N} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix} \Rightarrow \frac{\partial \mathbf{b}^T \mathbf{f}}{\partial \mathbf{f}} = \mathbf{b} \quad (3.243)$$

Box 3.8. How does the method of Lagrange multipliers work?

Assume that we wish to satisfy two equations simultaneously:

$$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \end{aligned} \quad (3.244)$$

Let us assume that in the (x, y) plane the first of these equations is satisfied at point A and the second at point B , so that it is impossible to satisfy both equations, exactly for the same value of (x, y) .

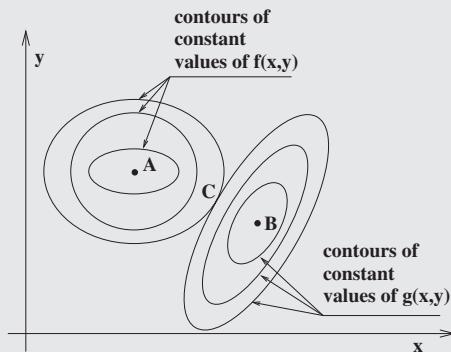


Figure 3.18: Two incompatible constraints are exactly satisfied at points A and B . The point where we make the minimum total violation of the two constraints is the point where two isocontours of the two functions just touch (point C). This is the point identified by the method of Lagrange multipliers.

We wish to find a point C on the plane where we make the least compromise in violating these two equations. The location of this point will depend on how fast the values of $f(x, y)$ and $g(x, y)$ change from 0, as we move away from points A and B , respectively. Let us consider the isocontours of f and g around each of the points A and B , respectively. As the contours grow away from point A , function $|f(x, y)|$ takes larger

and larger values, while as contours grow away from point B , the values function $|g(x, y)|$ takes become larger as well. Point C , where the values of $|f(x, y)|$ and $|g(x, y)|$ are as small as possible (minimum violation of the constraints which demand that $|f(x, y)| = |g(x, y)| = 0$), must be the point where an isocontour around A just touches an isocontour around B , without crossing each other. When two curves just touch each other, their tangents become parallel. The tangent vector to a curve along which $f = \text{constant}$ is ∇f , and the tangent vector to a curve along which $g = \text{constant}$ is ∇g . The two tangent vectors do not need to have the same magnitude for the minimum violation of the constraints. It is enough for them to have the same orientation. Therefore, we say that point C is determined by the solution of equation $\nabla f = \mu \nabla g$ where μ is some constant that takes care of the (possibly) different magnitudes of the two vectors. In other words, the solution to the problem of simultaneous satisfaction of the two incompatible equations (3.244) is the solution of the differential set of equations

$$\nabla f + \lambda \nabla g = 0 \quad (3.245)$$

where λ is the **Lagrange multiplier**, an arbitrary constant.

Box 3.9. How can we choose a direction that maximises the negentropy?

Let us consider that the negentropy we wish to maximise is given by approximation (3.181), on page 246, which we repeat here in a more concise way,

$$J_1(y) \propto [E\{G(y)\} - E\{G(\nu)\}]^2 \quad (3.246)$$

where:

$$G(y) \equiv \frac{1}{a} \ln[\cosh(ay)] \quad (3.247)$$

First of all we observe that the second term in (3.246) is a constant (see example 3.59, on page 248) and so the maximum of $J_1(y)$ will coincide with an extremum of its first term: $E\{G(y)\}$. So, our problem is to select an axis \mathbf{w} , such that the projections y_i on it of all data vectors \mathbf{x}_i , ($y_i \equiv \mathbf{w}^T \mathbf{x}_i$), are distributed as non-Gaussianly as possible. As \mathbf{w} is effectively a directional vector, its magnitude should be 1. So, our problem is phrased as follows: extremise $E\{G(\mathbf{w}^T \mathbf{x}_i)\}$ subject to the constraint $\mathbf{w}^T \mathbf{w} = 1$. According to the method of **Lagrange multipliers**, the solution of such a problem is given by the solution of the following system of equations (see Box 3.8)

$$\frac{\partial}{\partial \mathbf{w}} [E\{G(\mathbf{w}^T \mathbf{x}_i)\} + \lambda \mathbf{w}^T \mathbf{w}] = 0 \quad (3.248)$$

where λ is a parameter called **Lagrange multiplier**. Note that the expectation operator means nothing else than averaging over all \mathbf{x}_i vectors in the ensemble. So, expectation and differentiation may be exchanged as both are linear operators. By using then

the rules of differentiating with respect to a vector (see example 3.65), we equivalently may write

$$\begin{aligned} E \left\{ \frac{\partial}{\partial \mathbf{w}} [G(\mathbf{w}^T \mathbf{x}_i)] \right\} + \lambda \frac{\partial}{\partial \mathbf{w}} [\mathbf{w}^T \mathbf{w}] &= 0 \\ E \left\{ \frac{\partial [G(\mathbf{w}^T \mathbf{x}_i)]}{\partial \mathbf{w}^T \mathbf{x}_i} \frac{\partial \mathbf{w}^T \mathbf{x}_i}{\partial \mathbf{w}} \right\} + \lambda \frac{\partial}{\partial \mathbf{w}} [\mathbf{w}^T \mathbf{w}] &= 0 \\ E \{ G'(\mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i \} + 2\lambda \mathbf{w} &= 0 \end{aligned} \quad (3.249)$$

where $G'(y)$ is the derivative of $G(y)$ with respect to its argument. For $G(y)$ given by (3.247), we have:

$$G'(y) \equiv \frac{dG(y)}{dy} = \frac{1}{a} \frac{1}{\cosh(ay)} \frac{d[\cosh(ay)]}{dy} = \frac{\sinh(ay)}{\cosh(ay)} = \tanh(ay) \quad (3.250)$$

It is convenient to call $2\lambda \equiv -\beta$, so we may say that the solution \mathbf{w} we need is the solution of equation:

$$E \{ G'(\mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i \} - \beta \mathbf{w} = 0 \quad (3.251)$$

This is a system of as many nonlinear equations as components of vectors \mathbf{w} and \mathbf{x}_i . If we denote the left-hand side of (3.251) by \mathbf{F} , we may write:

$$\mathbf{F} \equiv E \{ G'(\mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i \} - \beta \mathbf{w} \quad (3.252)$$

These equations represent a mapping from input vector \mathbf{w} to output vector \mathbf{F} . The **Jacobian** matrix of such a mapping is defined as the matrix of all first order partial derivatives of \mathbf{F} with respect to \mathbf{w} :

$$J_{\mathbf{F}}(\mathbf{w}) \equiv \frac{\partial(F_1, F_2, \dots, F_N)}{\partial(w_1, w_2, \dots, w_N)} \equiv \begin{pmatrix} \frac{\partial F_1}{\partial w_1} & \frac{\partial F_1}{\partial w_2} & \cdots & \frac{\partial F_1}{\partial w_N} \\ \frac{\partial F_2}{\partial w_1} & \frac{\partial F_2}{\partial w_2} & \cdots & \frac{\partial F_2}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial F_N}{\partial w_1} & \frac{\partial F_N}{\partial w_2} & \cdots & \frac{\partial F_N}{\partial w_N} \end{pmatrix} \quad (3.253)$$

Here N is assumed to be the number of components of vectors \mathbf{w} and \mathbf{F} . The Jacobian matrix may be used to expand function $\mathbf{F}(\mathbf{w}^+)$ about a point \mathbf{w} , near \mathbf{w}^+ , using Taylor series, where we keep only the first order terms:

$$\mathbf{F}(\mathbf{w}^+) \simeq \mathbf{F}(\mathbf{w}) + J_{\mathbf{F}}(\mathbf{w})(\mathbf{w}^+ - \mathbf{w}) \quad (3.254)$$

Now, if point \mathbf{w}^+ is where the function becomes 0, ie if $\mathbf{F}(\mathbf{w}^+) = 0$, the above equation may be used to identify point \mathbf{w}^+ , starting from point \mathbf{w} :

$$\mathbf{w}^+ = \mathbf{w} - [J_{\mathbf{F}}(\mathbf{w})]^{-1} \mathbf{F}(\mathbf{w}) \quad (3.255)$$

It can be shown that the Jacobian of system (3.251) is given by

$$J_{\mathbf{F}}(\mathbf{w}) = E \{ G''(\mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i \mathbf{x}_i^T \} - \beta I \quad (3.256)$$

where $G''(y)$ is the first derivative of $G'(y)$ with respect to its argument and I here is the unit matrix (see example 3.66). The inversion of this matrix is difficult, and in order to simplify it, the following approximation is made:

$$E \{G''(\mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i \mathbf{x}_i^T\} \simeq E \{G''(\mathbf{w}^T \mathbf{x}_i)\} E \{\mathbf{x}_i \mathbf{x}_i^T\} = E \{G''(\mathbf{w}^T \mathbf{x}_i)\} I \quad (3.257)$$

The last equality follows because the data represented by vectors \mathbf{x}_i have been centred and whitened. This approximation allows one to write for the Jacobian:

$$\begin{aligned} J_{\mathbf{F}}(\mathbf{w}) &\simeq [E \{G''(\mathbf{w}^T \mathbf{x}_i)\} - \beta] I \\ \Rightarrow [J_{\mathbf{F}}(\mathbf{w})]^{-1} &\simeq \frac{1}{E \{G''(\mathbf{w}^T \mathbf{x}_i)\} - \beta} I \end{aligned} \quad (3.258)$$

If we substitute from (3.258) and (3.252) into (3.255), we deduce that the solution of system (3.251) may be approached by updating the value of an initial good guess of \mathbf{w} , using:

$$\mathbf{w}^+ = \mathbf{w} - \frac{E \{G'(\mathbf{w}^T \mathbf{x}_i)\} \mathbf{x}_i - \beta \mathbf{w}}{E \{G''(\mathbf{w}^T \mathbf{x}_i)\} - \beta} \quad (3.259)$$

This equation may be further simplified to:

$$\mathbf{w}^+ = \frac{\mathbf{w} E \{G''(\mathbf{w}^T \mathbf{x}_i)\} - \mathbf{w} \beta - E \{G'(\mathbf{w}^T \mathbf{x}_i)\} \mathbf{x}_i + \beta \mathbf{w}}{E \{G''(\mathbf{w}^T \mathbf{x}_i)\} - \beta} \quad (3.260)$$

The denominator is a scalar that scales all components of \mathbf{w}^+ equally, so it may be omitted, as long as after every update we scale \mathbf{w}^+ to have unit magnitude. Thus, we deduce the following updating formula:

$$\mathbf{w}^+ = \mathbf{w} E \{G''(\mathbf{w}^T \mathbf{x}_i)\} - E \{G'(\mathbf{w}^T \mathbf{x}_i)\} \mathbf{x}_i \quad (3.261)$$

After every update, we must check for convergence: if vectors \mathbf{w} and \mathbf{w}^+ are almost identical, we stop the process. These two vectors may be deemed to be identical if the absolute value of their dot product is more than, say, 0.999 (it would have to be 1 if we insisted them to be exactly identical).

Example B3.66

Our input data consist of an ensemble of four 3D vectors \mathbf{x}_i , where $i = 1, 2, 3, 4$. In terms of their components, these four vectors are:

$$\begin{aligned} \mathbf{x}_1^T &= (x_{11}, x_{12}, x_{13}) \\ \mathbf{x}_2^T &= (x_{21}, x_{22}, x_{23}) \\ \mathbf{x}_3^T &= (x_{31}, x_{32}, x_{33}) \\ \mathbf{x}_4^T &= (x_{41}, x_{42}, x_{43}) \end{aligned} \quad (3.262)$$

We wish to identify a vector $\mathbf{w}^T \equiv (w_1, w_2, w_3)$ such that the projections y_i , where $i = 1, 2, 3, 4$, of vectors \mathbf{x}_i on this direction extremise $E\{G(y)\}$. Write down the equations that have to be solved to identify \mathbf{w} .

For a start, we write down the expressions for y_i :

$$\begin{aligned} y_1 &\equiv \mathbf{w}^T \mathbf{x}_1 = w_1 x_{11} + w_2 x_{12} + w_3 x_{13} \\ y_2 &\equiv \mathbf{w}^T \mathbf{x}_2 = w_1 x_{21} + w_2 x_{22} + w_3 x_{23} \\ y_3 &\equiv \mathbf{w}^T \mathbf{x}_3 = w_1 x_{31} + w_2 x_{32} + w_3 x_{33} \\ y_4 &\equiv \mathbf{w}^T \mathbf{x}_4 = w_1 x_{41} + w_2 x_{42} + w_3 x_{43} \end{aligned} \quad (3.263)$$

Applying formula (3.251) we can write down the equations we have to solve:

$$\begin{aligned} \frac{1}{4} [G'(y_1)x_{11} + G'(y_2)x_{21} + G'(y_3)x_{31} + G'(y_4)x_{41}] - \beta w_1 &= 0 \\ \frac{1}{4} [G'(y_1)x_{12} + G'(y_2)x_{22} + G'(y_3)x_{32} + G'(y_4)x_{42}] - \beta w_2 &= 0 \\ \frac{1}{4} [G'(y_1)x_{13} + G'(y_2)x_{23} + G'(y_3)x_{33} + G'(y_4)x_{43}] - \beta w_3 &= 0 \end{aligned} \quad (3.264)$$

Here $G'(y) = \tanh(ay)$. Note that the expectation operator that appears in (3.251) was interpreted to mean the average over all vectors \mathbf{x}_i and we wrote down one equation for each component of vector \mathbf{w} .

Example B3.67

Work out the Jacobian matrix of system (3.264).

We start by naming the left-hand side of the equations of the system:

$$\begin{aligned} F_1 &\equiv \frac{1}{4} [G'(y_1)x_{11} + G'(y_2)x_{21} + G'(y_3)x_{31} + G'(y_4)x_{41}] - \beta w_1 \\ F_2 &\equiv \frac{1}{4} [G'(y_1)x_{12} + G'(y_2)x_{22} + G'(y_3)x_{32} + G'(y_4)x_{42}] - \beta w_2 \\ F_3 &\equiv \frac{1}{4} [G'(y_1)x_{13} + G'(y_2)x_{23} + G'(y_3)x_{33} + G'(y_4)x_{43}] - \beta w_3 \end{aligned} \quad (3.265)$$

The Jacobian of this system is defined as:

$$J_{\mathbf{F}}(\mathbf{w}) \equiv \begin{pmatrix} \frac{\partial F_1}{\partial w_1} & \frac{\partial F_1}{\partial w_2} & \frac{\partial F_1}{\partial w_3} \\ \frac{\partial F_2}{\partial w_1} & \frac{\partial F_2}{\partial w_2} & \frac{\partial F_2}{\partial w_3} \\ \frac{\partial F_3}{\partial w_1} & \frac{\partial F_3}{\partial w_2} & \frac{\partial F_3}{\partial w_3} \end{pmatrix} \quad (3.266)$$

The elements of this matrix may be computed with the help of equations (3.265) and (3.263). We compute explicitly only the first one:

$$\begin{aligned} \frac{\partial F_1}{\partial w_1} &= \frac{1}{4} \left[\frac{\partial G'(y_1)}{\partial y_1} \frac{\partial y_1}{\partial w_1} x_{11} + \frac{\partial G'(y_2)}{\partial y_2} \frac{\partial y_2}{\partial w_1} x_{21} + \right. \\ &\quad \left. \frac{\partial G'(y_3)}{\partial y_3} \frac{\partial y_3}{\partial w_1} x_{31} + \frac{\partial G'(y_4)}{\partial y_4} \frac{\partial y_4}{\partial w_1} x_{41} \right] - \frac{\partial \beta w_1}{\partial w_1} \end{aligned} \quad (3.267)$$

We call $G''(y)$ the derivative of $G'(y)$. Then the result for all the elements of (3.266) is:

$$\begin{aligned} \frac{\partial F_1}{\partial w_1} &= \frac{1}{4} [G''(y_1)x_{11}^2 + G''(y_2)x_{21}^2 + G''(y_3)x_{31}^2 + G''(y_4)x_{41}^2] - \beta \\ \frac{\partial F_1}{\partial w_2} &= \frac{1}{4} [G''(y_1)x_{11}x_{12} + G''(y_2)x_{21}x_{22} + G''(y_3)x_{31}x_{32} + G''(y_4)x_{41}x_{42}] \\ \frac{\partial F_1}{\partial w_3} &= \frac{1}{4} [G''(y_1)x_{11}x_{13} + G''(y_2)x_{21}x_{23} + G''(y_3)x_{31}x_{33} + G''(y_4)x_{41}x_{43}] \\ \frac{\partial F_2}{\partial w_1} &= \frac{1}{4} [G''(y_1)x_{12}x_{11} + G''(y_2)x_{22}x_{21} + G''(y_3)x_{32}x_{31} + G''(y_4)x_{42}x_{41}] \\ \frac{\partial F_2}{\partial w_2} &= \frac{1}{4} [G''(y_1)x_{12}^2 + G''(y_2)x_{22}^2 + G''(y_3)x_{32}^2 + G''(y_4)x_{42}^2] - \beta \\ \frac{\partial F_2}{\partial w_3} &= \frac{1}{4} [G''(y_1)x_{12}x_{13} + G''(y_2)x_{22}x_{23} + G''(y_3)x_{32}x_{33} + G''(y_4)x_{42}x_{43}] \\ \frac{\partial F_3}{\partial w_1} &= \frac{1}{4} [G''(y_1)x_{13}x_{11} + G''(y_2)x_{23}x_{21} + G''(y_3)x_{33}x_{31} + G''(y_4)x_{43}x_{41}] \\ \frac{\partial F_3}{\partial w_2} &= \frac{1}{4} [G''(y_1)x_{13}x_{12} + G''(y_2)x_{23}x_{22} + G''(y_3)x_{33}x_{32} + G''(y_4)x_{43}x_{42}] \\ \frac{\partial F_3}{\partial w_3} &= \frac{1}{4} [G''(y_1)x_{13}^2 + G''(y_2)x_{23}^2 + G''(y_3)x_{33}^2 + G''(y_4)x_{43}^2] - \beta \end{aligned} \quad (3.268)$$

Example B3.68

Show that the Jacobian given by equations (3.266) and (3.268) may be written in the form (3.256).

For this case, equation (3.256) takes the form:

$$J_{\mathbf{F}}(\mathbf{w}) = \frac{1}{4} [G''(y_1)\mathbf{x}_1\mathbf{x}_1^T + G''(y_2)\mathbf{x}_2\mathbf{x}_2^T + G''(y_3)\mathbf{x}_3\mathbf{x}_3^T + G''(y_4)\mathbf{x}_4\mathbf{x}_4^T] - \beta I \quad (3.269)$$

We may start by computing the vector outer products that appear on the right-hand side:

$$\begin{aligned} \mathbf{x}_1\mathbf{x}_1^T &= \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} & x_{13} \end{pmatrix} = \begin{pmatrix} x_{11}^2 & x_{11}x_{12} & x_{11}x_{13} \\ x_{12}x_{11} & x_{12}^2 & x_{12}x_{13} \\ x_{13}x_{11} & x_{13}x_{12} & x_{13}^2 \end{pmatrix} \\ \mathbf{x}_2\mathbf{x}_2^T &= \begin{pmatrix} x_{21} \\ x_{22} \\ x_{23} \end{pmatrix} \begin{pmatrix} x_{21} & x_{22} & x_{23} \end{pmatrix} = \begin{pmatrix} x_{21}^2 & x_{21}x_{22} & x_{21}x_{23} \\ x_{22}x_{21} & x_{22}^2 & x_{22}x_{23} \\ x_{23}x_{21} & x_{23}x_{22} & x_{23}^2 \end{pmatrix} \\ \mathbf{x}_3\mathbf{x}_3^T &= \begin{pmatrix} x_{31} \\ x_{32} \\ x_{33} \end{pmatrix} \begin{pmatrix} x_{31} & x_{32} & x_{33} \end{pmatrix} = \begin{pmatrix} x_{31}^2 & x_{31}x_{32} & x_{31}x_{33} \\ x_{32}x_{31} & x_{32}^2 & x_{32}x_{33} \\ x_{33}x_{31} & x_{33}x_{32} & x_{33}^2 \end{pmatrix} \\ \mathbf{x}_4\mathbf{x}_4^T &= \begin{pmatrix} x_{41} \\ x_{42} \\ x_{43} \end{pmatrix} \begin{pmatrix} x_{41} & x_{42} & x_{43} \end{pmatrix} = \begin{pmatrix} x_{41}^2 & x_{41}x_{42} & x_{41}x_{43} \\ x_{42}x_{41} & x_{42}^2 & x_{42}x_{43} \\ x_{43}x_{41} & x_{43}x_{42} & x_{43}^2 \end{pmatrix} \end{aligned} \quad (3.270)$$

If we substitute from (3.270) into (3.269), we shall obtain (3.266).

How do we perform ICA in image processing in practice?

The algorithm that follows is applicable to all choices of random experiment we make. The only thing that has to change from one application to the other is the way we read the data, ie the way we form the input vectors. In order to make the algorithm specific, we show here how it is applied to the case shown in figure 3.16b.

Let us assume that we have I grey images of size $M \times N$.

Step 0: Remove the mean of each image. This step is not necessary, but it is advisable. If the means are not removed, one of the independent components identified may be a flat component. Not particularly interesting, as we are really interested in identifying the modes of image variation.

Step 1: Write the columns of image i one under the other to form an $MN \times 1$ vector \mathbf{p}_i . You will have I such vectors. Plotted in an MN -dimensional coordinate system they will create a cloud of points as shown in figure 3.16b.

You may write these vectors next to each other to form the columns of a matrix P that will be $MN \times I$ in size.

Step 2: Compute the average of all vectors, say vector \mathbf{m} , and remove it from each vector, thus creating I vectors $\tilde{\mathbf{p}}_i$ of size $MN \times 1$.

This operation moves the original coordinate system to the centre of the cloud of points—an analogous operation to the one shown in figure 3.17a.

The new vectors form the $MN \times I$ matrix \tilde{P} , when written next to each other.

Step 3: Compute the autocorrelation matrix of the new vectors. Let us call \tilde{p}_{ki} the k th component of vector $\tilde{\mathbf{p}}_i$. Then the elements of the autocorrelation matrix C are:

$$C_{kj} = \frac{1}{I} \sum_{i=1}^I \tilde{p}_{ki} \tilde{p}_{ji} \quad (3.271)$$

Matrix C is of size $MN \times MN$ and it may also be computed as:

$$C = \frac{1}{I} \tilde{P} \tilde{P}^T \quad (3.272)$$

Step 4: Compute the nonzero eigenvalues of C and arrange them in decreasing order. Let us say that they are E . Let us denote by \mathbf{u}_l the eigenvector that corresponds to eigenvalue λ_l . We may write these eigenvectors next to each other to form matrix U , of size $MN \times E$.

Step 5: Scale the eigenvectors so that the projected components of vectors $\tilde{\mathbf{p}}_i$ will have the same variance along all eigendirections: $\tilde{\mathbf{u}}_l \equiv \mathbf{u}_l / \sqrt{\lambda_l}$.

You may write the scaled eigenvectors next to each other to form matrix \tilde{U} of size $MN \times E$.

Step 6: Project all vectors $\tilde{\mathbf{p}}_i$ on the scaled eigenvectors to produce vectors $\tilde{\mathbf{q}}_i$, where $\tilde{\mathbf{q}}_i$ is an $E \times 1$ vector with components \tilde{q}_{li} given by:

$$\tilde{q}_{li} = \tilde{\mathbf{u}}_l^T \tilde{\mathbf{p}}_i \quad (3.273)$$

This step achieves dimensionality reduction, as usually $E < MN$ and at the same time produces whitened data to work with.

This step may be performed in a concise way as $\tilde{Q} \equiv \tilde{U}^T \tilde{P}$, with vectors $\tilde{\mathbf{q}}_i$ being the columns of matrix \tilde{Q} .

Step 7: Select randomly an $E \times 1$ vector \mathbf{w}_1 , with the values of its components drawn from a uniform distribution in the range $[-1, 1]$. (Any other range will do.)

Step 8: Normalise vector \mathbf{w}_1 so that it has unit norm: if w_{i1} is the i th component of vector \mathbf{w}_1 , define vector $\tilde{\mathbf{w}}_1$, with components:

$$\tilde{w}_{i1} \equiv \frac{w_{i1}}{\sqrt{\sum_j w_{j1}^2}} \quad (3.274)$$

Step 9: Project all data vectors $\tilde{\mathbf{q}}_i$ on $\tilde{\mathbf{w}}_1$, to produce the I different projection components:

$$y_i = \tilde{\mathbf{w}}_1^T \tilde{\mathbf{q}}_i \quad (3.275)$$

These components (the y_1, \dots, y_4 values in example 3.66) will be stored in an $1 \times I$ matrix/row vector which may be produced in one go as $Y \equiv \tilde{\mathbf{w}}_1^T \tilde{Q}$.

Step 10: Update each component of vector $\tilde{\mathbf{w}}_1$ according to

$$w_{k1}^+ = \tilde{w}_{k1} \frac{1}{I} \sum_{i=1}^I G''(y_i) - \frac{1}{I} \sum_{i=1}^I \tilde{q}_{ki} G'(y_i) \quad (3.276)$$

(corresponding to equation (3.261), on page 271).

Note that for $G'(y) = \tanh y$, $G''(y) \equiv dG'(y)/dy = 1 - (\tanh y)^2$.

Step 11: Normalise vector \mathbf{w}_1^+ by dividing each of its elements with the square root of the sum of the squares of its elements, $\sqrt{\sum_j (w_{j1}^+)^2}$, so that it has unit magnitude. Call the normalised version of vector \mathbf{w}_1^+ , vector $\tilde{\mathbf{w}}_1^+$.

Step 12: Check whether vectors $\tilde{\mathbf{w}}_1^+$ and $\tilde{\mathbf{w}}_1$ are sufficiently close. If, say, $|\tilde{\mathbf{w}}_1^{+T} \tilde{\mathbf{w}}_1| > 0.9999$, the two vectors are considered identical and we may adopt the normalised vector $\tilde{\mathbf{w}}_1^+$ as the first axis of the ICA system.

If the two vectors are different, ie if the absolute value of their dot product is less than 0.9999, we set $\tilde{\mathbf{w}}_1 = \tilde{\mathbf{w}}_1^+$ and go to Step 9.

After the first ICA direction has been identified, we proceed to identify the remaining directions. The steps we follow are the same as Steps 7–12, with one extra step inserted: we have to make sure that any new direction we select is orthogonal to the already selected directions. This is achieved by inserting an extra step between Steps 10 and 11, to make sure that we use only the part of vector \mathbf{w}_e^+ (where $e = 2, \dots, E$) which is orthogonal to all previously identified vectors $\tilde{\mathbf{w}}_t^+$ for $t = 1, \dots, e-1$. This extra step is as follows.

Step 10.5: When trying to work out vector \mathbf{w}_e^+ , create a matrix B that contains as columns all $\tilde{\mathbf{w}}_t^+$, $t = 1, \dots, e-1$, vectors worked out so far. Then, in Step 11, instead of using vector \mathbf{w}_e^+ , use vector $\mathbf{w}_e^+ - BB^T\mathbf{w}_e^+$. (See example 3.69.)

To identify the coefficients of the expansion of the input images in terms of the ICA basis, the following steps have to be added to the algorithm.

Step 13: Project all vectors $\tilde{\mathbf{p}}_i$ on the unscaled eigenvectors to produce vectors \mathbf{q}_i , where \mathbf{q}_i is an $E \times 1$ vector with components q_{li} given by:

$$q_{li} = \mathbf{u}_l^T \tilde{\mathbf{p}}_i \quad (3.277)$$

This step may be performed in a concise way as $Q \equiv U^T \tilde{P}$, with vectors \mathbf{q}_i being the columns of matrix Q . Matrix U has been computed in Step 4.

Step 14: Write the identified vectors \mathbf{w}_e^+ next to each other as columns, to form matrix W . Then compute matrix $Z \equiv W^T Q$. The i th column of matrix Z consists of the coefficients of the expansion of the i th pattern in terms of the identified basis.

Each of the $\tilde{\mathbf{w}}_e^+$ vectors is of size $E \times 1$. The components of each such vector are measured along the eigenaxes of matrix C . They may, therefore, be used to express vector $\tilde{\mathbf{w}}_e^+$ in terms of the original coordinate system, via vectors \mathbf{u}_l . So, if we want to view the basis images we identified, the following step may be added to the algorithm.

Step 15: We denote by \mathbf{v}_e the position vector of the tip of vector $\tilde{\mathbf{w}}_e^+$ in the original coordinate system:

$$\mathbf{v}_e = \tilde{w}_{1e}^+ \mathbf{u}_1 + \tilde{w}_{2e}^+ \mathbf{u}_2 + \cdots + \tilde{w}_{Ee}^+ \mathbf{u}_E + \mathbf{m} \quad (3.278)$$

Here \mathbf{m} is the mean vector we removed originally from the cloud of points to move the original coordinate system to the centre of the cloud. All these vectors may be computed simultaneously as columns of matrix V , given by $V = UW + \overline{M}$, where matrix \overline{M} is made up from

vector \mathbf{m} repeated E times to form its columns.

There are E vectors \mathbf{v}_e , and they are of size $MN \times 1$. Each one may be wrapped round to form an $M \times N$ image, by reading its first M elements and placing them as the first column of the image, then the next M elements and placing them as the next image column and so on. These will be the basis images we have created from the original ensemble of images and the coefficients of the expansion of each original image in terms of them are the so called **independent components** of the original image.

If we wish to reconstruct an image, we must add the following steps to the algorithm.

Step 14.5: Construct vectors $\tilde{\mathbf{v}}_e$:

$$\tilde{\mathbf{v}}_e = \tilde{w}_{1e}^+ \mathbf{u}_1 + \tilde{w}_{2e}^+ \mathbf{u}_2 + \cdots + \tilde{w}_{Ee}^+ \mathbf{u}_E \quad (3.279)$$

All these vectors may be computed simultaneously as columns of matrix \tilde{V} , given by $\tilde{V} = UW$.

Step 14.6: To reconstruct the i^{th} pattern, we consider the i^{th} column of matrix Z . The elements of this column are the coefficients with which the columns of \tilde{V} have to be multiplied and added to form the original pattern i . We must remember to add vector \mathbf{m} , ie the mean pattern, in order to have full reconstruction. To visualise the reconstructed pattern we shall have to wrap it into an image.

Example B3.69

Consider two 3×1 vectors \mathbf{w}_1 and \mathbf{w}_2 of unit length and orthogonal to each other. Write them one next to the other to form a 3×2 matrix B . Consider also a 3×1 vector \mathbf{w}_3 . Show that vector $\mathbf{w}_3 - BB^T\mathbf{w}_3$ is the component of \mathbf{w}_3 that is orthogonal to both vectors \mathbf{w}_1 and \mathbf{w}_2 .

Matrix B is $[\mathbf{w}_1, \mathbf{w}_2]$. Matrix B^T is:

$$B^T = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{bmatrix} \quad (3.280)$$

If we multiply B^T with \mathbf{w}_3 we obtain a vector with its first element the dot product of \mathbf{w}_3 with vector \mathbf{w}_1 and its second element the dot product of \mathbf{w}_3 with vector \mathbf{w}_2 :

$$B^T \mathbf{w}_3 = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{bmatrix} \begin{bmatrix} w_{13} \\ w_{23} \\ w_{33} \end{bmatrix} = \begin{bmatrix} w_{11}w_{13} + w_{21}w_{23} + w_{31}w_{33} \\ w_{12}w_{13} + w_{22}w_{23} + w_{32}w_{33} \end{bmatrix} \quad (3.281)$$

As vectors \mathbf{w}_1 and \mathbf{w}_2 are of unit length, the values of these two dot products are the projections of \mathbf{w}_3 on \mathbf{w}_1 and \mathbf{w}_2 , respectively. When multiplied with the corresponding unit vector (\mathbf{w}_1 or \mathbf{w}_2), they become the components of vector \mathbf{w}_3 along the directions

of vectors \mathbf{w}_1 and \mathbf{w}_2 , respectively. By subtracting them from vector \mathbf{w}_3 , we are left with the component of \mathbf{w}_3 orthogonal to both directions:

$$\begin{bmatrix} w_{13} \\ w_{23} \\ w_{33} \end{bmatrix} - (w_{11}w_{13} + w_{21}w_{23} + w_{31}w_{33}) \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix} - (w_{12}w_{13} + w_{22}w_{23} + w_{32}w_{33}) \begin{bmatrix} w_{12} \\ w_{22} \\ w_{32} \end{bmatrix} \quad (3.282)$$

This is the same as $\mathbf{w}_3 - BB^T\mathbf{w}_3 = \mathbf{w}_3 - B(B^T\mathbf{w}_3)$:

$$\begin{aligned} & \begin{bmatrix} w_{13} \\ w_{23} \\ w_{33} \end{bmatrix} - \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} w_{11}w_{13} + w_{21}w_{23} + w_{31}w_{33} \\ w_{12}w_{13} + w_{22}w_{23} + w_{32}w_{33} \end{bmatrix} \quad (3.283) \\ &= \begin{bmatrix} w_{13} - w_{11}(w_{11}w_{13} + w_{21}w_{23} + w_{31}w_{33}) - w_{12}(w_{12}w_{13} + w_{22}w_{23} + w_{32}w_{33}) \\ w_{23} - w_{21}(w_{11}w_{13} + w_{21}w_{23} + w_{31}w_{33}) - w_{22}(w_{12}w_{13} + w_{22}w_{23} + w_{32}w_{33}) \\ w_{33} - w_{31}(w_{11}w_{13} + w_{21}w_{23} + w_{31}w_{33}) - w_{32}(w_{12}w_{13} + w_{22}w_{23} + w_{32}w_{33}) \end{bmatrix} \end{aligned}$$

Example B3.70

For three 2×1 vectors $\mathbf{a}_1 = (a_{11}, a_{21})^T$, $\mathbf{a}_2 = (a_{12}, a_{22})^T$ and $\mathbf{a}_3 = (a_{13}, a_{23})^T$, show that formulae (3.271) and (3.272) give the same answer.

Applying formula (3.271) for $I = 3$, we obtain the four elements of the 2×2 matrix C as follows:

$$\begin{aligned} C_{11} &= \frac{1}{3} \sum_{k=1}^3 a_{1k}^2 = \frac{1}{3} (a_{11}^2 + a_{12}^2 + a_{13}^2) \\ C_{12} &= \frac{1}{3} \sum_{k=1}^3 a_{1k}a_{2k} = \frac{1}{3} (a_{11}a_{21} + a_{12}a_{22} + a_{13}a_{23}) \\ C_{21} &= \frac{1}{3} \sum_{k=1}^3 a_{2k}a_{1k} = \frac{1}{3} (a_{21}a_{11} + a_{22}a_{12} + a_{23}a_{13}) \\ C_{22} &= \frac{1}{3} \sum_{k=1}^3 a_{2k}^2 = \frac{1}{3} (a_{21}^2 + a_{22}^2 + a_{23}^2) \quad (3.284) \end{aligned}$$

To apply formula (3.272), we must first write the three vectors as the columns of a

matrix and then multiply it with its transpose. We obtain:

$$\begin{aligned} C &= \frac{1}{3} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{pmatrix} \\ &= \frac{1}{3} \begin{pmatrix} a_{11}^2 + a_{12}^2 + a_{13}^2 & a_{11}a_{21} + a_{12}a_{22} + a_{13}a_{23} \\ a_{21}a_{11} + a_{22}a_{12} + a_{23}a_{13} & a_{21}^2 + a_{22}^2 + a_{23}^2 \end{pmatrix} \quad (3.285) \end{aligned}$$

The two results are the same.

Example 3.71

Consider the image of figure 3.19. Divide it into blocks of 8×8 and treat each block as an image of a set. Work out the basis images that will allow the identification of the independent components for each image in the set.



Figure 3.19: An original image from which 8×8 tiles are extracted.

From image 3.19, 1000 patches of size 8×8 were extracted at random, allowing overlap. Each patch had its mean removed. Then they were all written as columns of size 64×1 . They were written next to each other to form matrix P of size 64×1000 . The average of all columns was computed, as vector \mathbf{m} . This was removed from all columns of matrix P , to form matrix \tilde{P} . From this, matrix C of size 64×64 was created as $C = \tilde{P}\tilde{P}^T/1000$. The eigenvalues of this matrix were computed and all

those smaller than 0.0002 were set to 0. It is important to use a threshold for neglecting small eigenvalues, because eigenvalues may become arbitrarily small and with negative sign sometimes due to numerical errors. Such erroneous negative values may cause problems when the square root of the eigenvalue is used in the whitening process. The process of eigenvalue thresholding left us with $E = 27$ eigenvalues. The 27 basis images identified by Step 15 of the algorithm are shown in figure 3.20

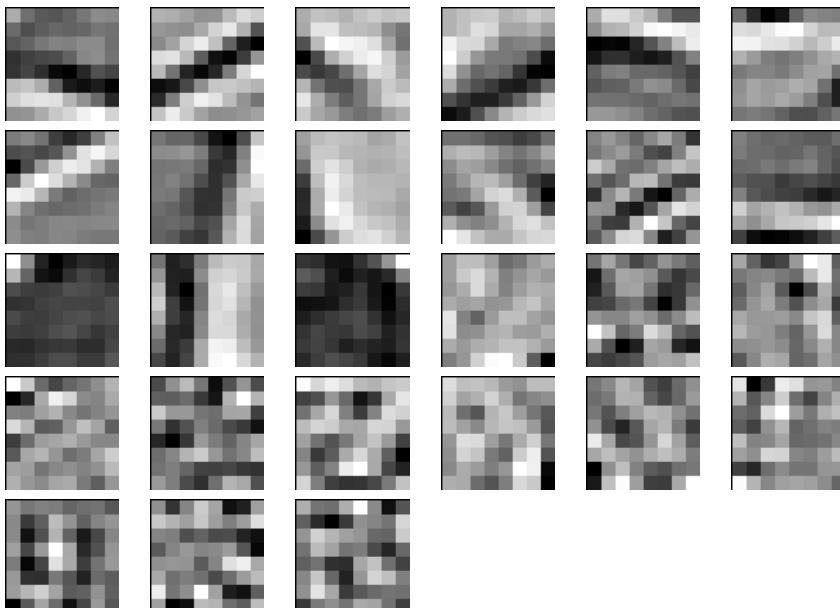


Figure 3.20: The 27 basis images identified by the ICA algorithm for the 1000 patches extracted from image 3.19.

Example 3.72

Consider one of the 8×8 tiles you identified in example 3.71 and reconstruct it using one, two,..., twenty-seven basis images identified by the ICA algorithm.

In order to identify the basis images by the ICA algorithm, the data had to be whitened. Whitening is only necessary for the process of identifying the basis images, and it should be bypassed when we are interested in working out the coefficients of the expansion of a specific image (8×8 tile) in terms of the identified basis. So, to work out the coefficients of the expansion of any patch in terms of the basis images, we first have to know the coefficients of its expansion in terms of the unscaled eigenvectors. This

way we create matrix Q of size 27×1000 , the columns of which are vectors \mathbf{q}_i . The elements of vector \mathbf{q}_i are given by $q_{li} = \mathbf{u}_l^T \tilde{\mathbf{p}}_i$, where vector \mathbf{u}_l is the eigenvector of unit length (Step 13 of the algorithm). We may then form matrix $Z \equiv W^T Q$ (Step 14 of the algorithm). The i th column of matrix Z consists of the coefficients of the expansion of the i th input pattern in terms of the basis images. To form the various approximations of a particular original subimage (tile), we work as follows:

1st order approximation: Use the first element of the i th column of matrix Z and multiply the first column of matrix \tilde{V} constructed by (3.279) in Step 14.5 of the algorithm. Add the mean vector \mathbf{m} , and wrap up the result into an 8×8 image.

2nd order approximation: Multiply the first element of the i th column of matrix Z with the first column of matrix \tilde{V} , add the product of the second element of the i th column of matrix Z with the second column of matrix \tilde{V} , add the mean vector \mathbf{m} , and wrap up the result into an 8×8 image.

3rd order approximation: Multiply the first element of the i th column of matrix Z with the first column of matrix \tilde{V} , add the product of the second element of the i th column of matrix Z with the second column of matrix \tilde{V} , add the product of the third element of the i th column of matrix Z with the third column of matrix \tilde{V} , add the mean vector \mathbf{m} , and wrap up the result into an 8×8 image.

Continue the process until you have incorporated all components in the reconstruction. Figure 3.21 shows the 27 successive reconstructions of the 20th input tile, as well as the original input image.

Figure 3.21: The 27 reconstructions of one of the original images, by incorporating the basis images one at a time. The bottom right panel shows the original image. The final reconstruction is almost perfect. The minor differences between the original image and the full reconstruction are probably due to the omission of some (weak) eigenvalues of matrix C .

Example 3.73

Analyse the flower image in terms of the basis constructed in example 3.71 and reconstruct it using one, two,..., twenty-seven basis images.

The flower image was not one of the input patches used to construct the ICA basis we shall use. To analyse it according to this basis, we first remove from it the mean vector \mathbf{m} and then we project it onto the unnormalised eigenvectors stored in matrix U , to work out its vector $\mathbf{q}_{\text{flower}}$. This vector is then projected onto the ICA vectors in order to construct its coefficients of expansion stored in vector $\mathbf{z}_{\text{flower}} \equiv W^T \mathbf{q}_{\text{flower}}$. These coefficients are then used to multiply the corresponding basis vectors stored in matrix \tilde{V} . Figure 3.22 shows the 27 successive reconstructions of the flower image. The reconstruction is not very good, because this image is totally different from the set of images used to construct the basis.

Figure 3.22: The 27 reconstructions of the flower image, by incorporating the basis images one at a time. The original image is shown at the bottom right.

How do we apply ICA to signal processing?

Let us revisit the cocktail party problem. Let us consider that we have S microphones and each signal we record consists of T samples. There are again two ways to view the problem from the statistical point of view.

(i) We may assume that the underlying random experiment produces combinations of values that are recorded at one time instant by the S microphones. Every such combination produces a point in a coordinate system with as many axes as we have microphones. We shall have T such points, as many as the time instances at which we sampled the recorded signals. This case is depicted in figure 3.23a. First we must remove the mean recording of each *microphone* over time, from each component of the S -tuple, ie we must centre the data. Let us say that after centring the data, x_{ik} represents what the i th microphone recorded at time k . The elements of the correlation matrix we shall have to compute in this case, in order to whiten the data, will be given by:

$$C_{ij} \equiv \frac{1}{T} \sum_{k=1}^T x_{ik} x_{jk} \quad (3.286)$$

The tips of the unit vectors of the axes we shall define by performing ICA will be the basis S -tuple signals, ie combinations of recorded values by the microphones in terms of which all observed combinations of values might be expressed. These signals are of no particular interest. What we are interested in, in this case, is to identify a set of virtual “microphones”, which, if they were used for recording, they would have recorded the independent components (speeches) that make up the mixtures. These virtual “microphones” are represented by the axes we identify by the ICA algorithm, and therefore, the components of the signals when expressed in terms of these axes are what we are looking for. These are the rows of matrix Z , and so the independent components of the mixtures are the signals represented by the rows of matrix Z computed at Step 14.

The method of performing ICA is the same as the algorithm given on page 274, with the only difference that our input vectors \mathbf{p}_i now are T vectors of size $S \times 1$, ie matrix P is of size $S \times T$.

(ii) We may assume that the underlying random experiment produces combinations of values that are recorded by a single microphone over T time instances. Every such combination produces a point in a coordinate system with as many axes as we have time instances. We shall have S such points, as many as microphones. This case is depicted in figure 3.23b. Again we have to centre the input vectors by removing the mean over all microphones from each component of the T -tuple. The elements of the correlation matrix we shall have to compute in this case, in order to whiten the data, will be given by:

$$C_{ij} \equiv \frac{1}{S} \sum_{k=1}^S x_{ik} x_{jk} \quad (3.287)$$

Note that x_{ik} that appears in (3.286) and x_{ik} that appears in (3.287) are not the same. First of all, vector \mathbf{x}_k in (3.286) represents what combination of values were recorded by the various microphones at instant k , while here vector \mathbf{x}_k represents what combination of values microphone k recorded over time. Further, different mean values were removed from the original vectors in order to produce these centred versions of them.

The tips of the unit vectors of the axes we shall define by performing ICA in this case will be the basis T -sample long signals. However, the problem now is that we usually do not have as many microphones as we might wish, ie S is not large enough to allow the calculation of reliable statistics, and so this approach is not usually adopted.

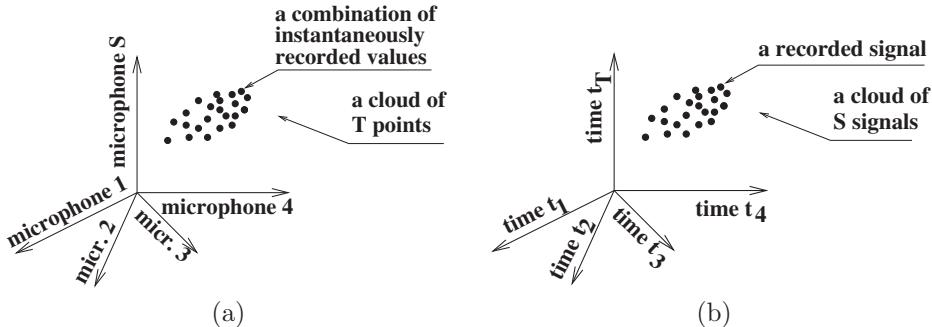


Figure 3.23: (a) If we assume that a random experiment decides which combination of sample values are recorded at each instance by the S microphones, then each such combination is a point in a coordinate system with as many axes as microphones we have. Assuming that each microphone over time recorded T samples, the cloud of points created this way represents T outcomes of the random experiment. (b) If we assume that a random experiment decides which combination of sample values makes up a recorded signal, then each recorded signal is a point in a coordinate system with as many axes as samples in the signal. Assuming that we have S recorded signals, the cloud of points created this way represents S outcomes of the random experiment.

Example 3.74

Figure 3.24 shows three signals which were created using formulae

$$\begin{aligned} f(i) &= \sin \frac{i\pi}{19} \\ g(i) &= \sin \frac{i\pi}{5} + \sin \frac{i\pi}{31} \\ h(i) &= 0.3i \text{ modulo } 5 \end{aligned} \quad (3.288)$$

where i takes integer values from 1 to 1000. Let us consider the three mixtures of them shown in figure 3.25, which were created using:

$$\begin{aligned} m_1(i) &= 0.3f(i) + 0.4g(i) + 0.3h(i) \\ m_2(i) &= 0.5f(i) + 0.2g(i) + 0.3h(i) \\ m_3(i) &= 0.1f(i) + 0.1g(i) + 0.8h(i) \end{aligned} \quad (3.289)$$

Assume that you are only given the mixed signals and of course that you do not know the mixing proportions that appear in (3.289). Use the ICA algorithm to recover the original signals.

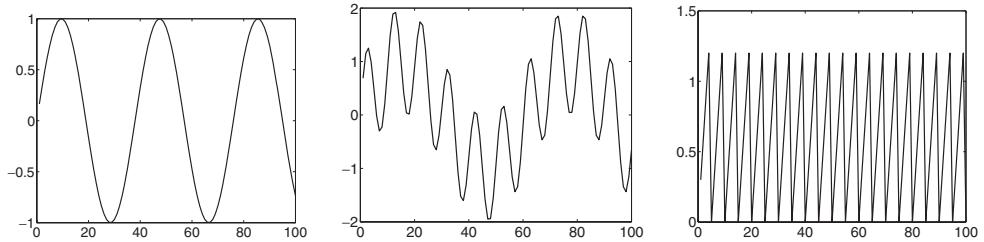


Figure 3.24: The first 100 samples of three original signals.

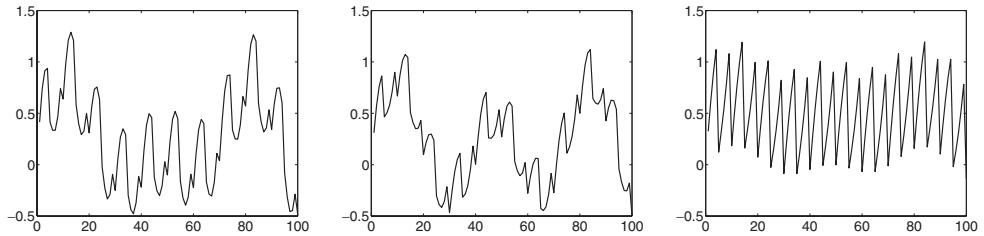


Figure 3.25: The first 100 samples of three mixed signals.

We are going to run the ICA algorithm, assuming that the underlying random experiment produces triplets of numbers recorded by the three sensors over a period of 1000 time instances. This is the case depicted in figure 3.23a. We shall not use Step 0 of the algorithm as there is no point here. We shall apply, however, Steps 1-14. Note that matrix P now is 3×1000 , and matrix C is 3×3 :

$$C = \begin{pmatrix} 0.2219 & 0.1725 & 0.0985 \\ 0.1725 & 0.1825 & 0.0886 \\ 0.0985 & 0.0886 & 0.1303 \end{pmatrix} \quad (3.290)$$

The eigenvalues of this matrix are: $\lambda_1 = 0.4336$, $\lambda_2 = 0.0725$ and $\lambda_3 = 0.0286$. The corresponding eigenvectors, written as columns of matrix U , are:

$$U = \begin{pmatrix} -0.6835 & -0.3063 & -0.6626 \\ -0.6105 & -0.2577 & 0.7489 \\ -0.4002 & 0.9164 & -0.0109 \end{pmatrix} \quad (3.291)$$

After we divide each vector with the square root of the corresponding eigenvalue, we obtain matrix \tilde{U} :

$$\tilde{U} = \begin{pmatrix} -1.0379 & -1.1379 & -3.9184 \\ -0.9271 & -0.9574 & 4.4287 \\ -0.6077 & 3.4042 & -0.0642 \end{pmatrix} \quad (3.292)$$

The initial guess for vector w_1 is $(0.9003, -0.5377, 0.2137)^T$. After the algorithm runs, it produces the following three unit vectors along the directions that will allow the unmixing of the signals, written as columns of matrix W :

$$W = \begin{pmatrix} 0.4585 & -0.6670 & -0.5872 \\ -0.8877 & -0.3125 & -0.3382 \\ -0.0421 & -0.6764 & 0.7354 \end{pmatrix} \quad (3.293)$$

Steps 13 and 14 of the algorithm allow us to recover the original signals, as shown in figure 3.26. Note that these signals are the rows of matrix Z , computed in step 14, as these are the components that would have been recorded by the fictitious “microphones” represented by the three recovered axes of matrix W .

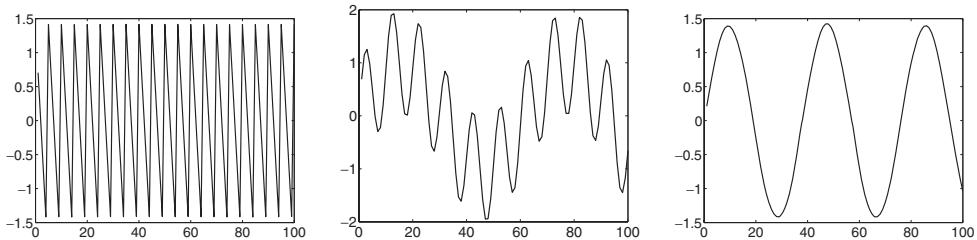


Figure 3.26: The first 100 samples of the three recovered signals by the ICA algorithm.

Example 3.75

Figure 3.27 shows the three mixed signals of example 3.74 with some Gaussian noise added to each one. The added noise values were drawn from a Gaussian probability density function with mean 0 and standard deviation 0.5. Perform again ICA to recover the original signals.

This time matrix C is given by:

$$C = \begin{pmatrix} 0.2326 & 0.1729 & 0.1005 \\ 0.1729 & 0.1905 & 0.0892 \\ 0.1005 & 0.0892 & 0.1392 \end{pmatrix} \quad (3.294)$$

The eigenvalues of this matrix are: $\lambda_1 = 0.4449$, $\lambda_2 = 0.0801$ and $\lambda_3 = 0.0374$. The corresponding eigenvectors, written as columns of matrix U , are:

$$U = \begin{pmatrix} -0.6852 & -0.3039 & -0.6619 \\ -0.6070 & -0.2639 & 0.7496 \\ -0.4025 & 0.9154 & -0.0037 \end{pmatrix} \quad (3.295)$$

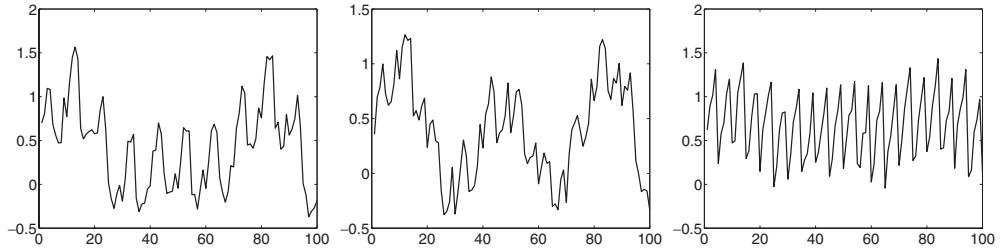


Figure 3.27: The first 100 samples of three mixed signals with noise.

After we divide each vector with the square root of the corresponding eigenvalue, we obtain matrix \tilde{U} :

$$\tilde{U} = \begin{pmatrix} -1.0273 & -1.0736 & -3.4246 \\ -0.9101 & -0.9322 & 3.8781 \\ -0.6034 & 3.2338 & -0.0190 \end{pmatrix} \quad (3.296)$$

The initial guess for vector w_1 is again $(0.9003, -0.5377, 0.2137)^T$. After the algorithm runs, it produces the following three unit vectors along the directions that will allow the unmixing of the signals, written as columns of matrix W :

$$W = \begin{pmatrix} -0.4888 & 0.6579 & -0.5729 \\ 0.8723 & 0.3775 & -0.3107 \\ -0.0118 & 0.6516 & 0.7584 \end{pmatrix} \quad (3.297)$$

The recovered signals are shown in figure 3.28. We note that even a moderate amount of noise deteriorates significantly the quality of the recovered signals. There are two reasons for that: first each mixed signal has its own noise component which means that it is as if we had 6 original signals mixed; second, ICA is designed to recover non-Gaussian signals, and the noise added is Gaussian.

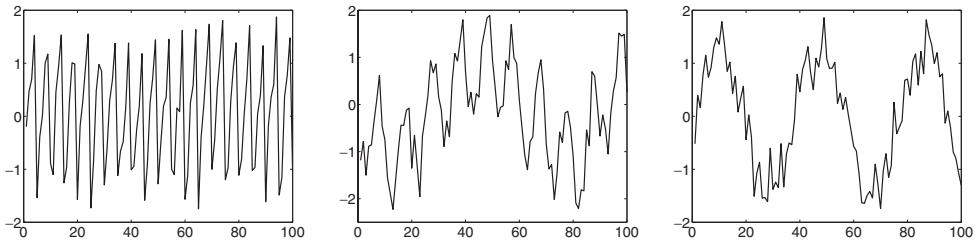


Figure 3.28: The first 100 samples of the three recovered signals by the ICA algorithm, when a Gaussian noise component was present in each mixed signal.

Example 3.76

In a magnetoencephalographic experiment, somebody used 250 channels to record the magnetic field outside a human head, every one millisecond, producing 1000 samples per channel. There are various processes that take place in the human brain, some of which control periodic functions of the human body, like breathing, heart beating, etc. Let us assume that the true signals that are produced by the brain are those shown in figure 3.24. We created 250 mixtures of them by choosing triplets of mixing components at random, the only constraint being that the numbers were in the range $[0, 1]$. They were not normalised to sum to 1, as such normalisation is not realistic in a real situation. Identify the original source signals hidden in the mixtures.

Let us try to solve the problem according to the interpretation of figure 3.23b. This means that our P matrix is $T \times S$ in size, ie its dimensions are 1000×250 , and the sources are given by the columns of matrix V . Figure 3.29 shows the first 100 points of the recovered original signals, identified as the columns of matrix V . We note that the recovery is not so good. The signals are pretty noisy. This is because 250 samples are not really enough to perform the statistics.

Next, let us try to solve the problem according to the interpretation of figure 3.23a. This means that our P matrix is the transpose of the previous one, ie it is $S \times T$, ie its dimensions are 250×1000 . In this case the sources are given by the rows of matrix Z . Figure 3.30 shows the first 100 points of the recovered original signals. The recovery is almost perfect. Note, of course, that in a real experiment, the recovery is never perfect. If nothing else, each recording has its own noisy signal superimposed, and that leads to a recovery of the form shown in figure 3.28.

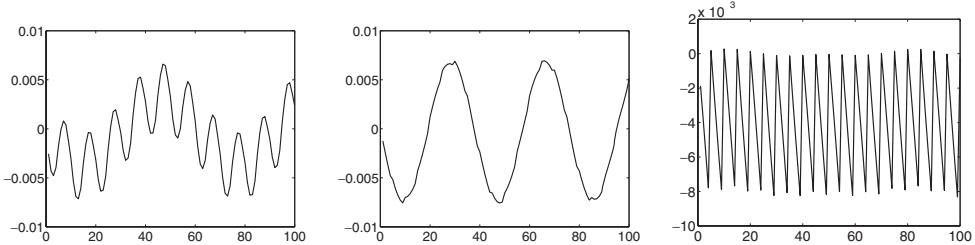


Figure 3.29: The three recovered signals by the ICA algorithm when the P matrix is arranged to be 1000×250 in size, ie $T \times S$.

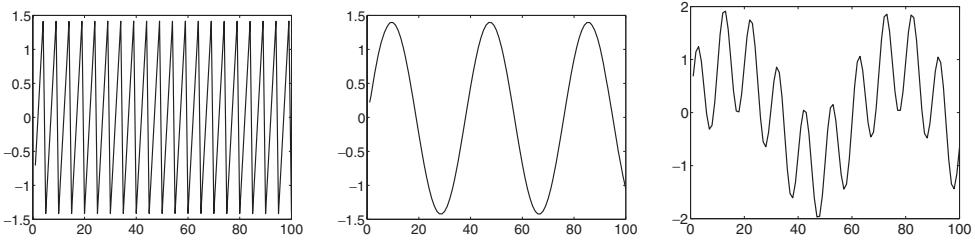


Figure 3.30: The three recovered signals by the ICA algorithm when the P matrix is arranged to be 250×1000 in size, ie $S \times T$.

What are the major characteristics of independent component analysis?

- Independent component analysis extracts the components of a set of blended recordings in an unpredictable order.
- The identified independent components are identified up to a scaling factor.
- Independent component analysis does not produce a basis of elementary signals in terms of which we may express any other signal that fulfils certain constraints, like Karhunen-Loeve transform does. Instead, independent component analysis identifies the possible independent sources that are hidden in mixed recordings. That is why it is part of the family of methods known as **blind source separation**.
- Due to the above characteristic, independent component analysis is always performed over a set of data. It is not meaningful here to talk about a single representative signal of a set of signals.

What is the difference between ICA as applied in image and in signal processing?

There is a confusion in the literature in the terminology used by the different communities of users. Cognitive vision scientists are interested in the actual axes defined by ICA to describe the cloud of points, and in particular at the coordinates of the tips of their unit vectors in terms of the original coordinate system, because these are actually the basis images, linear mixtures of which form all real images we observe. They refer to them as independent components. Image processing people, interested in sparse representations of images, refer to the coefficients of the expansion of the images in terms of these basis images as the independent components. Finally, signal processing people view the axes defined by ICA as “virtual microphones” and for them the independent components are the projections of their original samples along each one of these axes. This point is schematically shown in figure 3.31.

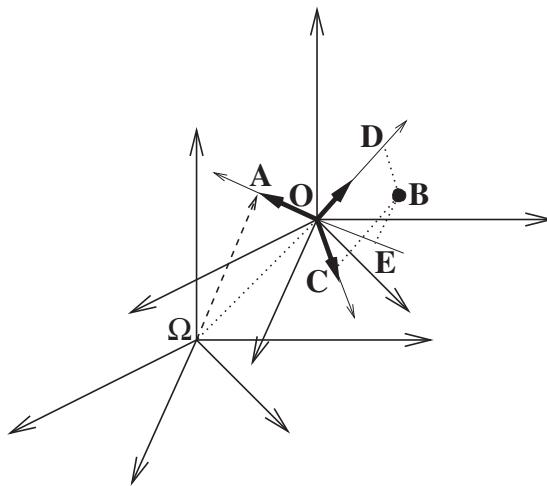


Figure 3.31: Ω is the origin of the original coordinate system in which we are given the data. O is the origin of the same system translated to the centre of the cloud of points when we remove the mean signal from each signal. The thick arrows are the unit vectors of the coordinate system we create by performing ICA. Note that, in general, this coordinate system has fewer axes than the original one. A is the tip of one of these unit vectors. ΩA is the position vector of this point, with respect to the original coordinate system. Vector ΩA corresponds to one column of matrix V constructed at Step 15 of the algorithm. People interested in cognitive vision are interested in the components of vector ΩA . Each such vector defines an elementary image. Such elementary images of size 8×8 or 16×16 correspond well to the receptive fields of some cells in the human visual cortex. Examples are those shown in figure 3.20. B is one of the original signals. If we project it onto the new axes, we shall get its coordinates with respect to the new axes. The signed lengths OC , OD and OE will constitute the ICA components of this point. Image processing people interested in image coding and other similar applications, refer to this set of values as the “source signal”. These values correspond to the columns of matrix Z constructed at Step 14 of the algorithm. Signal processing people consider the projections of all points B along one of the ICA axes and treat them as one of the “source signals” they are looking for. These sources correspond to the rows of matrix Z .

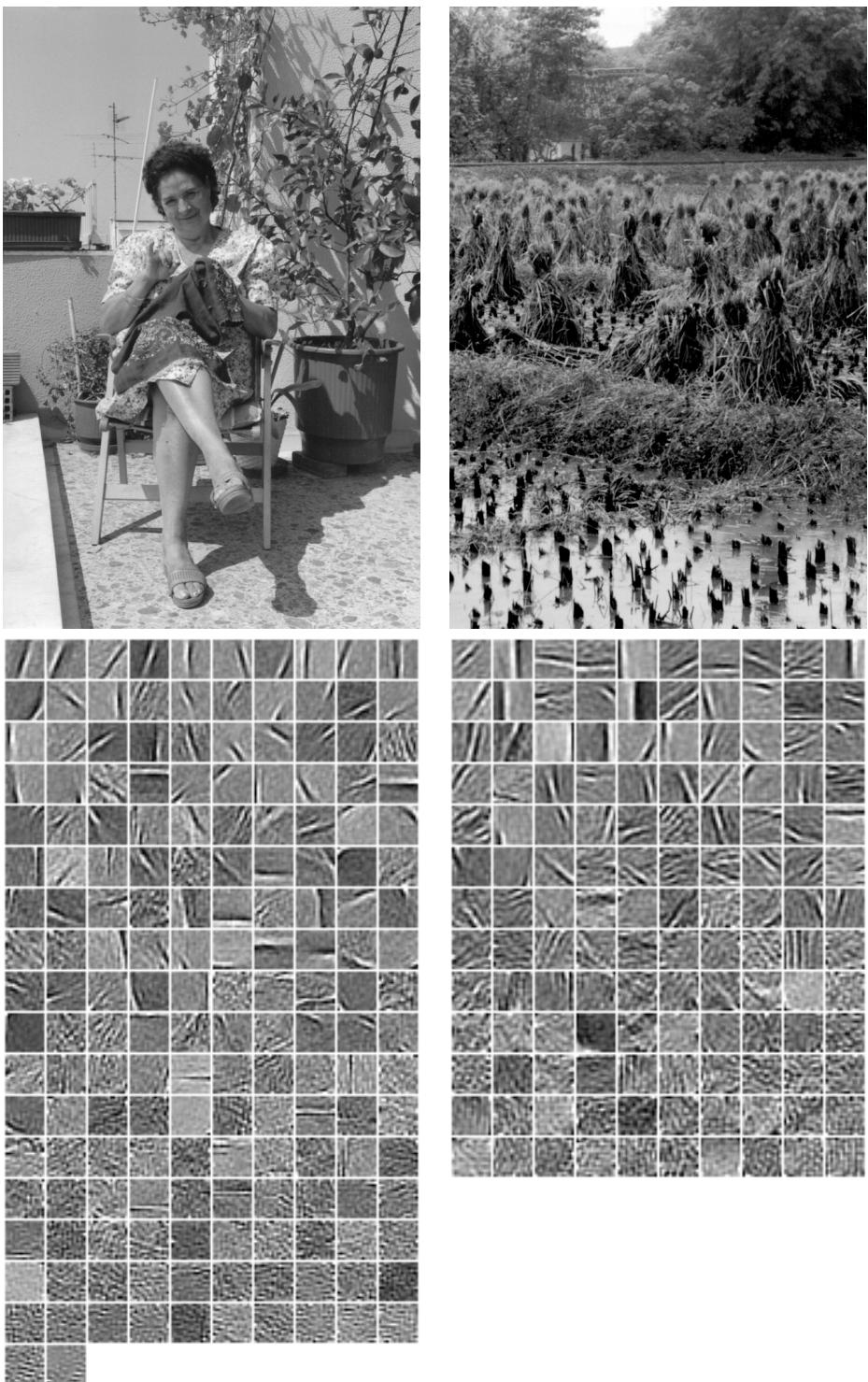


Figure 3.32: “Embroidering” and “Rice field in Chengdu”. From each image 5000 patches of size 16×16 were selected at random. The ICA algorithm identified 172 independent components from the first image and 130 from the second image, shown at the bottom.

What is the “take home” message of this chapter?

If we view an image as an instantiation of a whole lot of images, which are the result of a random process, then we can try to represent it as the linear superposition of some eigenimages which are appropriate for representing the whole ensemble of images. For an $N \times N$ image, there may be as many as N^2 such eigenimages, while in the SVD approach there were only N . The difference is that with these N^2 eigenimages we can represent the whole ensemble of images, while in the case of SVD the N eigenimages were appropriate only for representing the one image. If the set of eigenimages is arranged in decreasing order of the corresponding eigenvalues, truncating the expansion of the image in terms of them approximates any image in the set with the minimum *mean square error*, over the whole ensemble of images. In the SVD case similar truncation led to the minimum square error approximation.

The crux of K-L expansion is the assumption of ergodicity. This assumption states that the spatial statistics of a single image are the same as the ensemble statistics over the whole set of images. If a restricted type of image is considered, this assumption is clearly unrealistic: images are not simply the outcomes of a random process; there is always a deterministic underlying component which makes the assumption invalid. So, in such a case, the K-L transform effectively puts more emphasis on the random component of the image, ie the noise, rather than the component of interest.

However, if many different images are considered, the average grey value over the ensemble, even of the deterministic component, may be the same from pixel to pixel, and the assumption of ergodicity may be nearly valid. Further, if one has available a collection of images representative of the type of image of interest, the assumption of ergodicity is not needed: the K-L transform may be calculated using ensemble statistics and used to define a basis tailor-made for the particular type of image. Such is the case of applications dealing with large databases. The so called **eigenface** method of person identification is nothing more than the use of K-L transform using ensemble statistics as opposed to invoking the ergodicity assumption.

The K-L transform leads to an orthogonal basis of *uncorrelated* elementary images, in terms of which we may express any image that shares the same statistical properties as the image (or images) used to construct the transformation matrix. We may further seek the identification of a basis of *independent* elementary images. Such a basis, however, does not have the same meaning as a K-L basis. It rather serves the purpose of identifying components in the images that act as building blocks of the set considered. They are extracted usually in the hope that they may correspond to semantic components. Indeed, they tend to be elementary image structures like bands and edges in various orientations. Some researchers have identified them with the elementary structures the human vision system has been known to detect with the various processing cells it relies on, known as **ganglion cells**. Figure 3.32 shows the independent components identified from 16×16 patches of two very different images. The two sets of the extracted independent components, however, appear to be very similar, indicating that the building blocks of all images are fundamentally the same.

Chapter 4

Image Enhancement

What is image enhancement?

Image enhancement is the process by which we improve an image so that it looks *subjectively* better. We do not really know what the image should look like, but we can tell whether it has been improved or not, by considering, for example, whether more detail can be seen, or whether unwanted flickering has been removed, or the contrast is better.

How can we enhance an image?

An image is enhanced when we

- remove additive noise and interference;
- remove multiplicative interference;
- increase its contrast;
- decrease its blurring.

Some of the methods we use to achieve the above are

- smoothing and low pass filtering;
- sharpening or high pass filtering;
- histogram manipulation and
- generic deblurring algorithms, or algorithms that remove noise while avoid blurring the image.

Some of the methods in the first two categories are versions of **linear filtering**.

What is linear filtering?

Manipulation of images often entails omitting or enhancing details of certain spatial frequencies. This can be done by multiplying the Fourier transform of the image with a certain function that “kills” or modifies certain frequency components and then taking the inverse Fourier transform. When we do that, we say that we **filter the image**, and the function we use is said to be a **linear filter**.

4.1 Elements of linear filter theory

How do we define a 2D filter?

A 2D filter may be defined in terms of its Fourier transform $\hat{h}(\mu, \nu)$, called the **frequency response function**¹. By taking the inverse Fourier transform of $\hat{h}(\mu, \nu)$ we may calculate the filter in the real domain. This is called the **unit sample** (or **impulse**) **response** of the filter and is denoted by $h(k, l)$. Filters may be defined in the frequency domain, so they have exactly the desirable effect on the signal, or they may be defined in the real domain, so they are easy to implement.

How are the frequency response function and the unit sample response of the filter related?

The frequency response function $\hat{h}(\mu, \nu)$ is defined as a function of continuous frequencies (μ, ν) . The unit sample response $h(k, l)$ is defined as the inverse Fourier transform of $\hat{h}(\mu, \nu)$. However, since it has to be used for the manipulation of a digital image, $h(k, l)$ is defined at discrete points only. Then the equations relating these two functions are:

$$h(k, l) = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \hat{h}(\mu, \nu) e^{j(\mu k + \nu l)} d\mu d\nu \quad (4.1)$$

$$\hat{h}(\mu, \nu) = \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} h(n, m) e^{-j(\mu n + \nu m)} \quad (4.2)$$

If we are interested in real filters only, these equations may be modified as follows:

$$h(k, l) = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \hat{h}(\mu, \nu) \cos(\mu k + \nu l) d\mu d\nu \quad (4.3)$$

$$\hat{h}(\mu, \nu) = \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} h(n, m) \cos(\mu n + \nu m) \quad (4.4)$$

Why are we interested in the filter function in the real domain?

Because we may achieve the enhancement of the image as desired, by simply convolving it with $h(k, l)$ instead of multiplying its Fourier transform with $\hat{h}(\mu, \nu)$. Figure 4.1 shows this schematically for the 1D case. The 2D case is totally analogous.

Are there any conditions which $h(k, l)$ must fulfil so that it can be used as a convolution filter?

Yes, $h(k, l)$ must be zero for $k > K$ and $l > L$, for some finite values K and L ; ie the filter with which we want to convolve the image must be a finite array of numbers. The ideal low

¹Strictly speaking, a filter is defined in terms of its “system transfer function”, which is the Laplace or z-transform of the sample response of the filter. The frequency response function is a special case of the transfer function. Its limitation is that it does not allow one to assess the stability of the filter. However, in image processing applications, we rarely deal with unstable filters, so the issue does not arise.

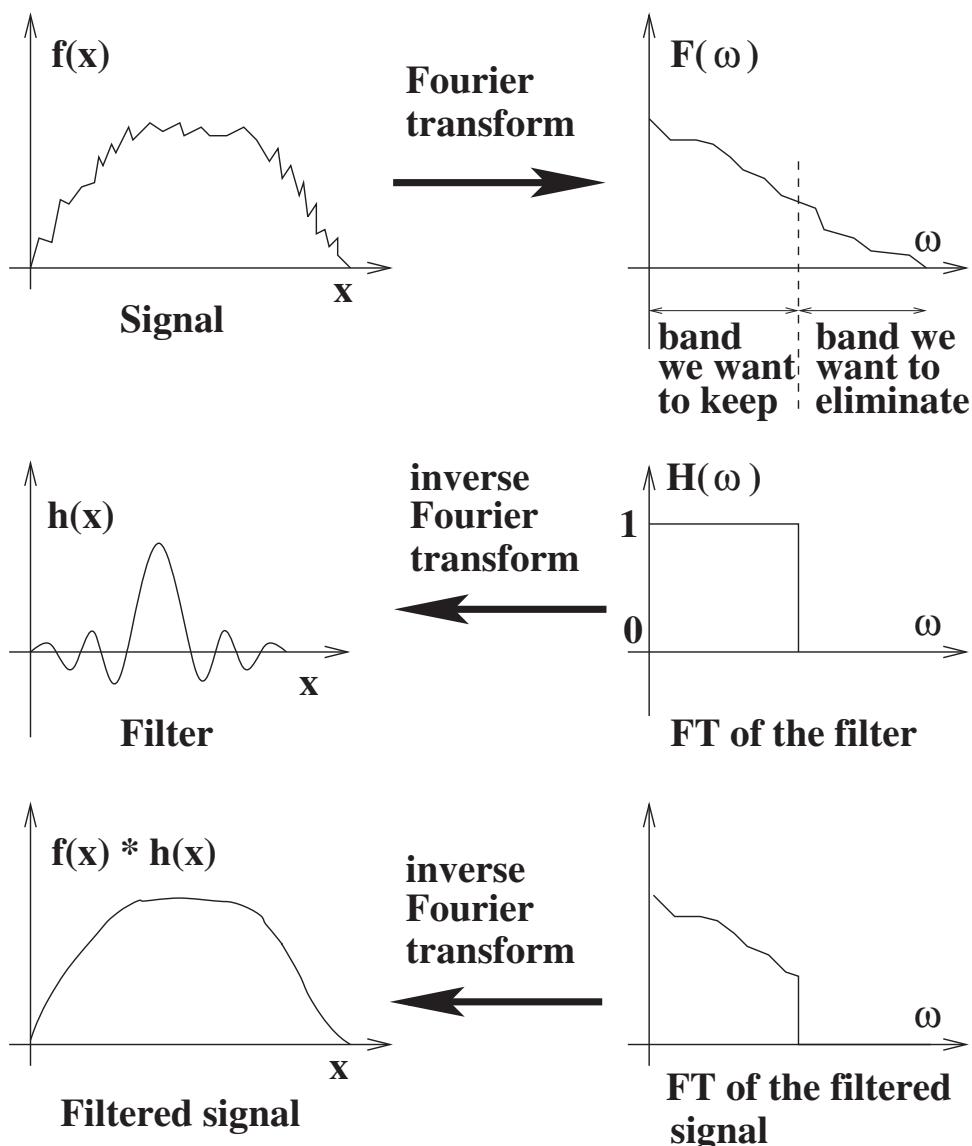


Figure 4.1: The case of a filter defined in the frequency domain for *maximum performance*. The processing route followed in this case is: image \rightarrow FT of the image \rightarrow multiplication with the FT of the filter \rightarrow inverse FT of the product \rightarrow filtered image. Top row: a signal and its Fourier transform. Middle row: on the right the ideal filter we define in the frequency domain and on the left its unit sample response in the real domain, which is not finite and thus it has a rather inconvenient form. Bottom row: on the right the Fourier transform of the filtered signal obtained by multiplying the Fourier transform of the signal at the top, with the Fourier transform of the filter in the middle; on the left the filtered signal that could have been obtained by convolving the signal at the top with the filter in the middle, if filter $h(x)$ were finite.

pass, band pass and high pass filters do not fulfil this condition. That is why they are called **infinite impulse response filters (IIR)**.

Example 4.1

Calculate the impulse response of the ideal 1D low pass filter.

The ideal low pass filter in 1D is defined as

$$\hat{h}(\mu) = \begin{cases} 1 & \text{if } -\mu_0 < \mu < \mu_0 \\ 0 & \text{otherwise} \end{cases} \quad (4.5)$$

where μ is the frequency variable and $0 < \mu_0 < \pi$ is the cutoff frequency parameter. The inverse Fourier transform of this function is:

$$\begin{aligned} h(k) &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{h}(\mu) e^{j\mu k} d\mu \\ &= \frac{1}{2\pi} \int_{-\mu_0}^{\mu_0} e^{j\mu k} d\mu \\ &= \frac{1}{2\pi} \int_{-\mu_0}^{\mu_0} \cos(\mu k) d\mu + j \frac{1}{2\pi} \underbrace{\int_{-\mu_0}^{\mu_0} \sin(\mu k) d\mu}_{=0} \\ &= \frac{1}{2\pi} \left. \frac{\sin(\mu k)}{k} \right|_{-\mu_0}^{\mu_0} = \frac{1}{2\pi k} 2 \sin(\mu_0 k) = \frac{\sin(\mu_0 k)}{\pi k} \end{aligned} \quad (4.6)$$

Box 4.1. What is the unit sample response of the 2D ideal low pass filter?

The 2D ideal low pass filter (see figure 4.2), which sets to zero all frequencies above a certain radial frequency R , is defined as:

$$\hat{h}(\mu, \nu) = \begin{cases} 1 & \text{for } \sqrt{\mu^2 + \nu^2} \leq R \\ 0 & \text{otherwise} \end{cases} \quad (4.7)$$

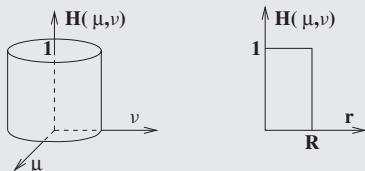


Figure 4.2: The ideal lowpass filter in 2D in the frequency domain. On the right, a cross-section of this filter with cutoff frequency R ($r \equiv \sqrt{\mu^2 + \nu^2}$).

We may use this definition of $\hat{h}(\mu, \nu)$ to calculate the corresponding unit sample response from equation (4.3):

$$h(k, l) = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \cos(\mu k + \nu l) \hat{h}(\mu, \nu) d\mu d\nu \quad (4.8)$$

We introduce polar coordinates (r, θ) in the (μ, ν) frequency space:

$$\begin{aligned} \mu &\equiv r \cos \theta \\ \nu &\equiv r \sin \theta \end{aligned} \left. \right\} \Rightarrow \mu^2 + \nu^2 = r^2 \text{ and } d\mu d\nu = r dr d\theta \quad (4.9)$$

Then:

$$h(k, l) = \frac{1}{(2\pi)^2} \int_0^{2\pi} \int_0^R \cos(rk \cos \theta + rl \sin \theta) r dr d\theta \quad (4.10)$$

We may write

$$\begin{aligned} k \cos \theta + l \sin \theta &= \sqrt{k^2 + l^2} \left[\frac{k}{\sqrt{k^2 + l^2}} \cos \theta + \frac{l}{\sqrt{k^2 + l^2}} \sin \theta \right] \\ &\equiv \sqrt{k^2 + l^2} [\sin \phi \cos \theta + \cos \phi \sin \theta] = \sqrt{k^2 + l^2} \sin(\theta + \phi) \end{aligned} \quad (4.11)$$

where angle ϕ has been defined so that:

$$\sin \phi \equiv \frac{k}{\sqrt{k^2 + l^2}} \text{ and } \cos \phi \equiv \frac{l}{\sqrt{k^2 + l^2}} \quad (4.12)$$

We define a new variable $t \equiv \theta + \phi$. Then equation (4.10) may be written as:

$$\begin{aligned} h(k, l) &= \frac{1}{(2\pi)^2} \int_{\phi}^{2\pi+\phi} \int_0^R \cos(r \sqrt{k^2 + l^2} \sin t) r dr dt \\ &= \frac{1}{(2\pi)^2} \int_{\phi}^{2\pi} \int_0^R \cos(r \sqrt{k^2 + l^2} \sin t) r dr dt \\ &\quad + \frac{1}{(2\pi)^2} \int_{2\pi}^{2\pi+\phi} \int_0^R \cos(r \sqrt{k^2 + l^2} \sin t) r dr dt \end{aligned} \quad (4.13)$$

In the second term we change variable t to $\tilde{t} \equiv t - 2\pi \Rightarrow t = \tilde{t} + 2\pi \Rightarrow \sin t = \sin \tilde{t}$. Therefore, we may write:

$$\begin{aligned} h(k, l) &= \frac{1}{(2\pi)^2} \int_{\phi}^{2\pi} \int_0^R \cos(r \sqrt{k^2 + l^2} \sin t) r dr dt \\ &\quad + \frac{1}{(2\pi)^2} \int_0^{\phi} \int_0^R \cos(r \sqrt{k^2 + l^2} \sin \tilde{t}) r dr d\tilde{t} \\ &= \frac{1}{(2\pi)^2} \int_0^{2\pi} \int_0^R \cos(r \sqrt{k^2 + l^2} \sin t) r dr dt \end{aligned} \quad (4.14)$$

This may be written as:

$$\begin{aligned} h(k, l) &= \frac{1}{(2\pi)^2} \int_0^\pi \int_0^R \cos(r\sqrt{k^2 + l^2} \sin t) r dr dt \\ &\quad + \frac{1}{(2\pi)^2} \int_\pi^{2\pi} \int_0^R \cos(r\sqrt{k^2 + l^2} \sin t) r dr dt \end{aligned} \quad (4.15)$$

In the second term, we define a new variable of integration: $\tilde{t} \equiv t - \pi \Rightarrow t = \tilde{t} + \pi \Rightarrow \sin t = -\sin \tilde{t} \Rightarrow \cos(r\sqrt{k^2 + l^2} \sin t) = \cos(r\sqrt{k^2 + l^2} \sin \tilde{t})$ and $dt = d\tilde{t}$. Then:

$$\begin{aligned} h(k, l) &= \frac{1}{(2\pi)^2} \int_0^\pi \int_0^R \cos(r\sqrt{k^2 + l^2} \sin t) r dr dt \\ &\quad + \frac{1}{(2\pi)^2} \int_0^\pi \int_0^R \cos(r\sqrt{k^2 + l^2} \sin \tilde{t}) r dr d\tilde{t} \\ &= \frac{1}{2\pi^2} \int_0^R \left\{ \int_0^\pi \cos(r\sqrt{k^2 + l^2} \sin t) dt \right\} r dr \end{aligned} \quad (4.16)$$

We know that the Bessel function of the first kind of zero order is defined as:

$$J_0(z) \equiv \frac{1}{\pi} \int_0^\pi \cos(z \sin \theta) d\theta \quad (4.17)$$

If we use definition (4.17) in equation (4.16), we obtain:

$$h(k, l) = \frac{1}{2\pi} \int_0^R r J_0(r\sqrt{k^2 + l^2}) dr \quad (4.18)$$

We define a new variable of integration $x \equiv r\sqrt{k^2 + l^2} \Rightarrow dr = \frac{1}{\sqrt{k^2 + l^2}} dx$. Then:

$$h(k, l) = \frac{1}{2\pi} \frac{1}{k^2 + l^2} \int_0^{R\sqrt{k^2 + l^2}} x J_0(x) dx \quad (4.19)$$

From the theory of Bessel functions, it is known that:

$$\int x^{p+1} J_p(x) dx = x^{p+1} J_{p+1}(x) \quad (4.20)$$

We apply formula (4.20) with $p = 0$ to equation (4.19):

$$\begin{aligned} h(k, l) &= \frac{1}{2\pi} \frac{1}{k^2 + l^2} x J_1(x) \Big|_0^{R\sqrt{k^2 + l^2}} \Rightarrow \\ h(k, l) &= \frac{1}{2\pi} \frac{R}{\sqrt{k^2 + l^2}} J_1(R\sqrt{k^2 + l^2}) \end{aligned} \quad (4.21)$$

This function is a function of infinite extent, defined at each point (k, l) of integer coordinates. It corresponds, therefore, to an array of infinite dimensions. The implication is that this filter cannot be implemented as a linear convolution filter.

Example B4.2

What is the impulse response of the 2D ideal band pass filter?

The ideal band pass filter for band $[R_1, R_2]$ is defined as:

$$\hat{h}(\mu, \nu) = \begin{cases} 1 & \text{for } R_1 \leq \sqrt{\mu^2 + \nu^2} \leq R_2 \\ 0 & \text{otherwise} \end{cases} \quad (4.22)$$

The only difference, therefore, with the ideal lowpass filter, derived in Box 4.1, is in the limits of equation (4.19):

$$\begin{aligned} h(k, l) &= \frac{1}{2\pi} \frac{1}{k^2 + l^2} \int_{R_1 \sqrt{k^2 + l^2}}^{R_2 \sqrt{k^2 + l^2}} x J_0(x) dx \\ &= \frac{1}{2\pi} \frac{1}{k^2 + l^2} x J_1(x) \Big|_{R_1 \sqrt{k^2 + l^2}}^{R_2 \sqrt{k^2 + l^2}} \\ &= \frac{1}{2\pi} \frac{1}{\sqrt{k^2 + l^2}} \left[R_2 J_1(R_2 \sqrt{k^2 + l^2}) - R_1 J_1(R_1 \sqrt{k^2 + l^2}) \right] \end{aligned} \quad (4.23)$$

This is a function defined for all values (k, l) . Therefore, the ideal band pass filter is an infinite impulse response filter.

Example B4.3

What is the impulse response of the 2D ideal high pass filter?

The ideal high pass filter, with cutoff radial frequency R , is defined as:

$$\hat{h}(\mu, \nu) = \begin{cases} 0 & \text{for } \sqrt{\mu^2 + \nu^2} \leq R \\ 1 & \text{otherwise} \end{cases} \quad (4.24)$$

The only difference, therefore, with the ideal lowpass filter, derived in Box 4.1, is in the limits of equation (4.19):

$$\begin{aligned} h(k, l) &= \frac{1}{2\pi} \frac{1}{k^2 + l^2} \int_{R \sqrt{k^2 + l^2}}^{+\infty} x J_0(x) dx \\ &= \frac{1}{2\pi} \frac{1}{k^2 + l^2} x J_1(x) \Big|_{R \sqrt{k^2 + l^2}}^{+\infty} \end{aligned} \quad (4.25)$$

Bessel function $J_1(x)$ tends to 0 for $x \rightarrow +\infty$. However, its asymptotic behaviour is $\lim_{x \rightarrow +\infty} J_1(x) \simeq \frac{1}{\sqrt{x}}$. This means that $J_1(x)$ does not tend to 0 fast enough to compensate for factor x which multiplies it, ie $\lim_{x \rightarrow +\infty} xJ_1(x) \rightarrow +\infty$. Therefore, there is no real domain function that has as Fourier transform the ideal high pass filter. In practice, of course, the highest frequency we may possibly be interested in is $\frac{1}{N}$, where N is the number of samples along one image axis, so the issue of the infinite upper limit in equation (4.25) does not arise and the ideal high pass filter becomes the same as the ideal band pass filter.

What is the relationship between the 1D and the 2D ideal lowpass filters?

For a cutoff frequency $\mu_0 = 1$, the 1D ideal lowpass filter is given by (see example 4.1):

$$h_1(k) = \frac{\sin k}{\pi k} \quad (4.26)$$

For a cutoff radial frequency $R = 1$, the 2D ideal lowpass filter is given by (see Box 4.1)

$$h_2(k, l) = \frac{J_1(\sqrt{k^2 + l^2})}{2\pi\sqrt{k^2 + l^2}} \quad (4.27)$$

where $J_1(x)$ is the first-order Bessel function of the first kind. Figure 4.3 shows the plot of $h_1(k)$ versus k and the plot of $h_2(k, l)$ versus k for $l = 0$. It can be seen that although the two filters look similar, they differ in significant details: their zero crossings are at different places, and the amplitudes of their side-lobes are different.

The plots in this figure were created by observing that $\sin k/k = 1$ for $k = 0$ and that $J_1(k)/k = 1/2$ for $k = 0$. Further, for $k = \pm 1, \pm 2$ and ± 3 , the following approximation formula holds:

$$\begin{aligned} \frac{J_1(k)}{k} &\simeq 0.5 - 0.56249985 \left(\frac{k}{3} \right)^2 + 0.21093573 \left(\frac{k}{3} \right)^4 - 0.03954289 \left(\frac{k}{3} \right)^6 \\ &\quad + 0.00443319 \left(\frac{k}{3} \right)^8 - 0.00031761 \left(\frac{k}{3} \right)^{10} + 0.00001109 \left(\frac{k}{3} \right)^{12} \end{aligned} \quad (4.28)$$

For $k > 3$, the approximation is

$$\begin{aligned} \frac{J_1(k)}{k} &= \frac{1}{k\sqrt{k}} \left[0.79788456 + 0.00000156 \frac{3}{k} + 0.01659667 \left(\frac{3}{k} \right)^2 \right. \\ &\quad + 0.00017105 \left(\frac{3}{k} \right)^3 - 0.00249511 \left(\frac{3}{k} \right)^4 + 0.00113653 \left(\frac{3}{k} \right)^5 \\ &\quad \left. - 0.00020033 \left(\frac{3}{k} \right)^6 \right] \cos \theta_1 \end{aligned} \quad (4.29)$$

where:

$$\begin{aligned}\theta_1 = & k - 2.35619449 + 0.12499612 \frac{3}{k} + 0.00005650 \left(\frac{3}{k}\right)^2 - 0.00637879 \left(\frac{3}{k}\right)^3 \\ & + 0.00074348 \left(\frac{3}{k}\right)^4 + 0.00079824 \left(\frac{3}{k}\right)^5 - 0.00029166 \left(\frac{3}{k}\right)^6\end{aligned}\quad (4.30)$$

The differences in the two filters imply that we cannot take an ideal or optimal (according

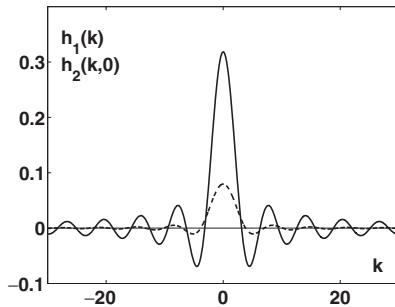


Figure 4.3: The cross-section of the 2D ideal lowpass filter ($h_2(k, l)$) represented here by the continuous line) is similar to but different from the cross-section of the 1D ideal lowpass filter ($h_1(k)$, represented here by the dashed line).

to some criteria) 1D filter, replace its variable by the polar radius (ie replace k by $\sqrt{k^2 + l^2}$ in equation (4.26)) and create the corresponding “ideal” or “optimal” filter in 2D. However, although the 2D filter we shall create this way will not be the ideal or optimal one, according to the corresponding criteria in 2D, it will be a good suboptimal filter with qualitatively the same behaviour as the optimal one.

How can we implement in the real domain a filter that is infinite in extent?

A filter, which is of infinite extent in real space may be implemented in a recursive way, and that is why it is called a **recursive filter**. Filters which are of finite extent in real space are called **nonrecursive** filters. Filters are usually represented and manipulated with the help of their **z -transforms** (see Box 4.2). The z -transforms of infinite in extent filters lead to recursive implementation formulae.

Box 4.2. z -transforms

A filter of finite extent is essentially a finite string of numbers $\{x_l, x_{l+1}, x_{l+2}, \dots, x_m\}$, where l and m are some integers. Sometimes an arrow is used to denote the element of the string that corresponds to the 0^{th} position. The z -transform of such a string is defined as:

$$X(z) \equiv \sum_{k=l}^m x_k z^{-k} \quad (4.31)$$

If the filter is of infinite extent, the sequence of numbers which represents it is of infinite extent too and its z -transform is given by an infinite sum, of the form:

$$X(z) = \sum_{k=-\infty}^{+\infty} x_k z^{-k} \quad \text{or} \quad \sum_{k=0}^{+\infty} x_k z^{-k} \quad \text{or} \quad \sum_{k=-\infty}^0 x_k z^{-k} \quad (4.32)$$

In such a case, we can usually write this sum in closed form as the *ratio* of two polynomials in z , as opposed to writing it as a single polynomial in z (which is the case for the z -transform of the finite filter):

$$H(z) = \frac{\sum_{i=0}^{M_a} a_i z^{-i}}{\sum_{j=0}^{M_b} b_j z^{-j}} \quad (4.33)$$

Here M_a and M_b are some integers. Conventionally we choose $b_0 = 1$.

The reason we use z -transforms is because digital filters can easily be realised in hardware in terms of their z -transforms. The z -transform of a sequence together with its region of convergence uniquely defines the sequence. Further, it obeys the convolution theorem: the z -transform of the convolution of two sequences is the product of the z -transforms of the two sequences.

When we convolve a signal with a digital filter, we essentially multiply the z -transform of the signal with the z -transform of the filter:

$$\underbrace{R(z)}_{\substack{\text{z-transform of} \\ \text{output signal}}} = \underbrace{H(z)}_{\substack{\text{z-transform of} \\ \text{filter}}} \underbrace{D(z)}_{\substack{\text{z-transform of} \\ \text{input signal}}} \quad (4.34)$$

If we substitute from (4.33) into (4.34) and bring the denominator to the left-hand side of the equation, we have:

$$R(z) \sum_{j=0}^{M_b} b_j z^{-j} = \left(\sum_{i=0}^{M_a} a_i z^{-i} \right) D(z) \quad (4.35)$$

In the sum on the left-hand side, we separate the $j = 0$ term and remember that $b_0 = 1$:

$$R(z) + \left(\sum_{j=1}^{M_b} b_j z^{-j} \right) R(z) = \left(\sum_{i=0}^{M_a} a_i z^{-i} \right) D(z) \quad (4.36)$$

Therefore:

$$R(z) = \left(\sum_{i=0}^{M_a} a_i z^{-i} \right) D(z) - \left(\sum_{j=1}^{M_b} b_j z^{-j} \right) R(z) \quad (4.37)$$

Remember that $R(z)$ is a sum in z^{-m} with coefficients, say, r_m . It is clear from the above equation that the value of r_m may be calculated in terms of the previously calculated values of r_m since polynomial $R(z)$ appears on the right-hand side of the equation too. That is why such a filter is called **recursive**. In the case of a finite filter, all b_i 's are zero (except b_0 which is 1) and so coefficients r_m of $R(z)$ are expressed in terms of a_i and the coefficients which appear in $D(z)$ only (ie we have no recursion).

Example B4.4

A good approximation of the ideal low pass filter is the so called Butterworth filter. Butterworth filters constitute a whole family of filters. The z -transform of one of the Butterworth filters is given by:

$$H(z) = \frac{0.58z^{-1} + 0.21z^{-2}}{1 - 0.40z^{-1} + 0.25z^{-2} - 0.044z^{-3}} \quad (4.38)$$

Using equation (4.37) work out how you may use this filter in a recursive way to smooth an image of size 256×256 line by line.

We shall treat each line of the image as a signal of 256 samples. Let us call the samples along this sequence d_0, d_1, \dots, d_{255} . Then its z -transform is:

$$D(z) = \sum_{k=0}^{255} d_k z^{-k} \quad (4.39)$$

Let us denote the samples of the smoothed sequence by r_0, r_1, \dots, r_{255} . Then the z -transform of the output sequence is:

$$R(z) = \sum_{k=0}^{255} r_k z^{-k} \quad (4.40)$$

Our task is to calculate the values of r_k from the known values d_k and the filter parameters.

According to the notation of Box 4.2:

$$\begin{aligned} a_0 &= a_3 = 0, & a_1 &= 0.58, & a_2 &= 0.21 \\ b_1 &= -0.40, & b_2 &= 0.25, & b_3 &= -0.044 \end{aligned} \quad (4.41)$$

We substitute from (4.39), (4.40) and (4.41) into (4.37), in order to work out the values of r_k in terms of d_k (the input pixel values) and the filter coefficients a_1 , a_2 , b_1 , b_2 and b_3 :

$$\begin{aligned}
\sum_{k=0}^{255} r_k z^{-k} &= a_1 z^{-1} \sum_{k=0}^{255} d_k z^{-k} + a_2 z^{-2} \sum_{k=0}^{255} d_k z^{-k} - (b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}) \sum_{k=0}^{255} r_k z^{-k} \\
&= \underbrace{\sum_{k=0}^{255} a_1 d_k z^{-k-1}}_{\text{set } k+1 \equiv \tilde{k} \Rightarrow k = \tilde{k}-1} + \underbrace{\sum_{k=0}^{255} a_2 d_k z^{-k-2}}_{\text{set } k+2 \equiv \tilde{k} \Rightarrow k = \tilde{k}-2} - \underbrace{\sum_{k=0}^{255} b_1 r_k z^{-k-1}}_{\text{set } k+1 \equiv \tilde{k} \Rightarrow k = \tilde{k}-1} \\
&\quad - \underbrace{\sum_{k=0}^{255} b_2 r_k z^{-k-2}}_{\text{set } k+2 \equiv \tilde{k} \Rightarrow k = \tilde{k}-2} - \underbrace{\sum_{k=0}^{255} b_3 r_k z^{-k-3}}_{\text{set } k+3 \equiv \tilde{k} \Rightarrow k = \tilde{k}-3} \\
&= \sum_{\tilde{k}=1}^{256} a_1 d_{\tilde{k}-1} z^{-\tilde{k}} + \sum_{\tilde{k}=2}^{257} a_2 d_{\tilde{k}-2} z^{-\tilde{k}} - \sum_{\tilde{k}=1}^{256} b_1 r_{\tilde{k}-1} z^{-\tilde{k}} \\
&\quad - \sum_{\tilde{k}=2}^{257} b_2 r_{\tilde{k}-2} z^{-\tilde{k}} - \sum_{\tilde{k}=3}^{258} b_3 r_{\tilde{k}-3} z^{-\tilde{k}}
\end{aligned} \tag{4.42}$$

We may drop the tilde from the dummy summation variable \tilde{k} and also we may split the terms in the various sums that are outside the range [3, 255]:

$$\begin{aligned}
r_0 + r_1 z^{-1} + r_2 z^{-2} + \sum_{k=3}^{255} r_k z^{-k} &= \sum_{k=3}^{255} a_1 d_{k-1} z^{-k} + a_1 d_0 z^{-1} + a_1 d_1 z^{-2} + a_1 d_{255} z^{-256} \\
&\quad + \sum_{k=3}^{255} a_2 d_{k-2} z^{-k} + a_2 d_0 z^{-2} + a_2 d_{254} z^{-256} \\
&\quad + a_2 d_{255} z^{-257} - \sum_{k=3}^{255} b_1 r_{k-1} z^{-k} - b_1 r_0 z^{-1} - b_1 r_1 z^{-2} \\
&\quad - b_1 r_{255} z^{-256} - \sum_{k=3}^{255} b_2 r_{k-2} z^{-k} - b_2 r_0 z^{-2} \\
&\quad - b_2 r_{254} z^{-256} - b_2 r_{255} z^{-257} - \sum_{k=3}^{255} b_3 r_{k-3} z^{-k} \\
&\quad - b_3 r_{253} z^{-256} - b_3 r_{254} z^{-257} - b_3 r_{255} z^{-258}
\end{aligned} \tag{4.43}$$

We may then collect together all terms with equal powers of z :

$$\begin{aligned}
& r_0 + (r_1 - a_1 d_0 + b_1 r_0)z^{-1} + (r_2 - a_1 d_1 - a_2 d_0 + b_1 r_1 + b_2 r_0)z^{-2} \\
& + \sum_{k=3}^{255} (r_k - a_1 d_{k-1} - a_2 d_{k-2} + b_1 r_{k-1} + b_2 r_{k-2} + b_3 r_{k-3})z^{-k} \\
& + (-a_1 d_{255} - a_2 d_{254} + b_1 r_{255} + b_2 r_{254} + b_3 r_{253})z^{-256} \\
& + (-a_2 d_{255} + b_2 r_{255} + b_3 r_{254})z^{-257} + b_3 r_{255} z^{-258} = 0
\end{aligned} \tag{4.44}$$

As this equation has to be valid for all values of z , we must set equal to 0 all coefficients of all powers of z :

$$\begin{aligned}
r_0 &= 0 \\
r_1 - a_1 d_0 + b_1 r_0 &= 0 \\
r_2 - a_1 d_1 - a_2 d_0 + b_1 r_1 + b_2 r_0 &= 0 \\
r_k - a_1 d_{k-1} - a_2 d_{k-2} + b_1 r_{k-1} + b_2 r_{k-2} + b_3 r_{k-3} &= 0 \quad \text{for } k = 3, 4, \dots, 255 \\
-a_1 d_{255} - a_2 d_{254} + b_1 r_{255} + b_2 r_{254} + b_3 r_{253} &= 0 \\
-a_2 d_{255} + b_2 r_{255} + b_3 r_{254} &= 0 \\
b_3 r_{255} &= 0
\end{aligned} \tag{4.45}$$

The last three equations may be solved to yield values for r_{253} , r_{254} and r_{255} , which are incompatible with the values for the same unknowns that will be computed from the recursive expression. This problem arises because the sequence is considered finite. If the upper limits in (4.39) and (4.40) were $+\infty$, instead of 255, the last three equations in (4.45) would not have arisen. In practice, we only keep the recursive relation from (4.45), and use it to compute the smoothed values r_k of the line of the image, given the input pixel values d_k and the filter coefficients a_1 , a_2 , b_1 , b_2 and b_3 , as follows:

$$r_k = a_1 d_{k-1} + a_2 d_{k-2} - b_1 r_{k-1} - b_2 r_{k-2} - b_3 r_{k-3} \quad \text{for } k = 0, 1, 2, \dots, 255 \tag{4.46}$$

Note that the first three of equations (4.45) are special cases of the recursive formula, if we consider that values d_{-3} , d_{-2} , d_{-1} , r_{-1} , r_{-2} and r_{-3} are 0.

Alternative options exist to define the variables with the negative indices in the recursive formula. The input image values sometimes are set equal to the last few values of the row of pixels, assuming wrap round boundary conditions (ie signal repetition): $d_{-1} = d_{255}$, $d_{-2} = d_{254}$ and $d_{-3} = d_{253}$. This arbitrariness in the initial conditions of the recursive relationship is the reason some scientists say that the recursive filters have “infinitely long boundary effect”. That is, the choice of boundary conditions we make for the recursive relationship affects all subsequent pixel values, while this is not the case for nonrecursive filters.

Example B4.5

You are given the sequence: 12, 13, 13, 14, 12, 11, 12, 13, 4, 5, 6, 5, 6, 4, 3, 6. Use the filter of example 4.4 to work out a smooth version of it, assuming

- (i) that the sequence is repeated ad infinitum in both directions;
 - (ii) that the values of the samples outside the index range given are all 0.
- Plot the initial and the two smoothed sequences and comment on the result.

We apply equation (4.46) assuming that the values of r_k for negative indices are 0. The reconstructed sequences we obtain are:

(i) 4.1100, 9.8640, 12.9781, 13.1761, 13.3099, 12.5010, 11.1527, 11.1915, 12.2985, 7.6622, 4.2227, 4.8447, 5.3793, 5.6564, 4.7109, 3.2870.

(ii) 0.0000, 6.9600, 12.8440, 13.6676, 13.4123, 12.4131, 11.1136, 11.2023, 12.3087, 7.6619, 4.2205, 4.8443, 5.3797, 5.6565, 4.7108, 3.2869.

Figure 4.4 shows the plots of the original and the smoothed sequences. We observe that in practice the effect of the boundary conditions we choose do not make much difference to the signal after a few samples. We also observe that the smoothed signal is shifted one position to the right of the input sequence. This is expected, as the filter we use has in its numerator as common factor z^{-1} . Every z^{-1} we use to multiply an input sequence with, shifts the sequence by one position to the right.

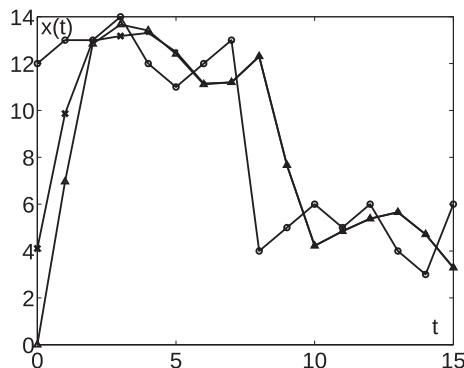


Figure 4.4: The points of the original sequence are denoted by circles. The two smoothed versions of it are denoted by crosses and triangles. After the first few samples, the two sequences become indistinguishable. However, both are shifted by one sample to the right in relation to the original sequence.

Example B4.6

The z -transform of a filter is:

$$H(z) = \frac{0.58 + 0.21z^{-1}}{1 - 0.40z^{-1} + 0.25z^{-2} - 0.044z^{-3}} \quad (4.47)$$

Work out the formulae that will allow you to use this filter to smooth a sequence d_0, d_1, \dots, d_{255} .

According to the notation of Box 4.2:

$$\begin{aligned} a_0 &= 0.58, & a_1 &= 0.21, & a_2 = a_3 &= 0 \\ b_1 &= -0.40, & b_2 &= 0.25, & b_3 &= -0.044 \end{aligned} \quad (4.48)$$

The z -transform of the input sequence is given by (4.39), and that of the output sequence by (4.40). We substitute from (4.39), (4.40) and (4.48) into (4.37), in order to work out the values of r_k in terms of d_k (the input pixel values) and the filter coefficients a_0, a_1, b_1, b_2 and b_3 :

$$\begin{aligned} \sum_{k=0}^{255} r_k z^{-k} &= a_0 \sum_{k=0}^{255} d_k z^{-k} + a_1 z^{-1} \sum_{k=0}^{255} d_k z^{-k} - (b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}) \sum_{k=0}^{255} r_k z^{-k} \\ &= a_0 \sum_{k=0}^{255} d_k z^{-k} + \underbrace{\sum_{k=0}^{255} a_1 d_k z^{-k-1}}_{\text{set } k+1 \equiv \tilde{k} \Rightarrow k=\tilde{k}-1} - \underbrace{\sum_{k=0}^{255} b_1 r_k z^{-k-1}}_{\text{set } k+1 \equiv \tilde{k} \Rightarrow k=\tilde{k}-1} \\ &\quad - \underbrace{\sum_{k=0}^{255} b_2 r_k z^{-k-2}}_{\text{set } k+2 \equiv \tilde{k} \Rightarrow k=\tilde{k}-2} - \underbrace{\sum_{k=0}^{255} b_3 r_k z^{-k-3}}_{\text{set } k+3 \equiv \tilde{k} \Rightarrow k=\tilde{k}-3} \\ &= a_0 \sum_{k=0}^{255} d_k z^{-k} + \sum_{\tilde{k}=1}^{256} a_1 d_{\tilde{k}-1} z^{-\tilde{k}} - \sum_{\tilde{k}=1}^{256} b_1 r_{\tilde{k}-1} z^{-\tilde{k}} \\ &\quad - \sum_{\tilde{k}=2}^{257} b_2 r_{\tilde{k}-2} z^{-\tilde{k}} - \sum_{\tilde{k}=3}^{258} b_3 r_{\tilde{k}-3} z^{-\tilde{k}} \end{aligned} \quad (4.49)$$

We may drop the tilde from the dummy summation variable \tilde{k} and also we may split the terms in the various sums that are outside the range [3, 255]:

$$\begin{aligned}
r_0 + r_1 z^{-1} + r_2 z^{-2} + \sum_{k=3}^{255} r_k z^{-k} &= \sum_{k=3}^{255} a_0 d_k z^{-k} + a_0 d_0 + a_0 d_1 z^{-1} + a_0 d_2 z^{-2} + \\
&\quad \sum_{k=3}^{255} a_1 d_{k-1} z^{-k} + a_1 d_0 z^{-1} + a_1 d_1 z^{-2} + a_1 d_{255} z^{-256} \\
&\quad - \sum_{k=3}^{255} b_1 r_{k-1} z^{-k} - b_1 r_0 z^{-1} - b_1 r_1 z^{-2} \\
&\quad - b_1 r_{255} z^{-256} - \sum_{k=3}^{255} b_2 r_{k-2} z^{-k} - b_2 r_0 z^{-2} \\
&\quad - b_2 r_{254} z^{-256} - b_2 r_{255} z^{-257} - \sum_{k=3}^{255} b_3 r_{k-3} z^{-k} \\
&\quad - b_3 r_{253} z^{-256} - b_3 r_{254} z^{-257} - b_3 r_{255} z^{-258}
\end{aligned} \tag{4.50}$$

We may then collect together all terms with equal powers of z :

$$\begin{aligned}
r_0 - a_0 d_0 + (r_1 - a_0 d_1 - a_1 d_0 + b_1 r_0) z^{-1} + (r_2 - a_0 d_2 - a_1 d_1 + b_1 r_1 + b_2 r_0) z^{-2} \\
+ \sum_{k=3}^{255} (r_k - a_0 d_k - a_1 d_{k-1} + b_1 r_{k-1} + b_2 r_{k-2} + b_3 r_{k-3}) z^{-k} \\
+ (-a_1 d_{255} + b_1 r_{255} + b_2 r_{254} + b_3 r_{253}) z^{-256} \\
+ (b_2 r_{255} + b_3 r_{254}) z^{-257} + b_3 r_{255} z^{-258} = 0
\end{aligned} \tag{4.51}$$

As this equation has to be valid for all values of z , we must set equal to 0 all coefficients of all powers of z :

$$\begin{aligned}
r_0 - a_0 d_0 &= 0 \\
r_1 - a_0 d_1 - a_1 d_0 + b_1 r_0 &= 0 \\
r_2 - a_0 d_2 - a_1 d_1 + b_1 r_1 + b_2 r_0 &= 0 \\
r_k - a_0 d_k - a_1 d_{k-1} + b_1 r_{k-1} + b_2 r_{k-2} + b_3 r_{k-3} &= 0 \quad \text{for } k = 3, 4, \dots, 255 \\
-a_1 d_{255} + b_1 r_{255} + b_2 r_{254} + b_3 r_{253} &= 0 \\
b_2 r_{255} + b_3 r_{254} &= 0 \\
b_3 r_{255} &= 0
\end{aligned} \tag{4.52}$$

The last three equations are ignored in practice. The recursive equation we have to use is:

$$r_k = a_0 d_k + a_1 d_{k-1} - b_1 r_{k-1} - b_2 r_{k-2} - b_3 r_{k-3} \quad \text{for } k = 0, 1, 2, \dots, 255 \tag{4.53}$$

Example B4.7

Smooth the sequence of example 4.5 using the recursive filter of example 4.6, for the cases when

- (i) the sequence is repeated ad infinitum in both directions;
 - (ii) the values of the samples outside the index range given are all 0.
- Compare the results with those of example 4.6.

We apply equation (4.53) assuming that the values of r_k for negative indices are 0. The reconstructed sequences we obtain are:

- (i) 8.2200, 13.3480, 13.5542, 13.2964, 12.4173, 11.1392, 11.2064, 12.3041, 7.6602, 4.2211, 4.8448, 5.3797, 5.6564, 4.7108, 3.2869, 4.4960
- (ii) 6.9600, 12.8440, 13.6676, 13.4123, 12.4131, 11.1136, 11.2023, 12.3087, 7.6619, 4.2205, 4.8443, 5.3797, 5.6565, 4.7108, 3.2869, 4.4959.

We note that in (ii) the results are identical with those of example 4.6, except now they are shifted one position to the left, so the smoothed sequence follows the input sequence more faithfully. In (i) the results are similar to those of example 4.6 but not identical. This is expected, given the different samples of the wrapped round input sequence which affect the first value of the reconstructed sequence.

Can we define a filter directly in the real domain for convenience?

Yes, but we should always keep an eye what the filter does in the frequency domain. For example, if we use a flat averaging filter to convolve the image with, we may create high frequency artifacts in the output image. This is due to the side lobes the Fourier transform of the flat filter has, which may enhance certain high frequencies in the image while suppressing others. This is schematically shown in figure 4.5.

Can we define a filter in the real domain, without side lobes in the frequency domain?

Yes. The Fourier Transform of a Gaussian function is also a Gaussian function. So, if we choose the shape of the filter to be given by a Gaussian, we shall avoid the danger of creating artifacts in the image. However, a Gaussian filter is infinite in extent and in order to use it as a convolution filter, it has to be truncated. If we truncate it, its Fourier transform will no longer be a Gaussian, but it will still be a function with minimal side lobes.

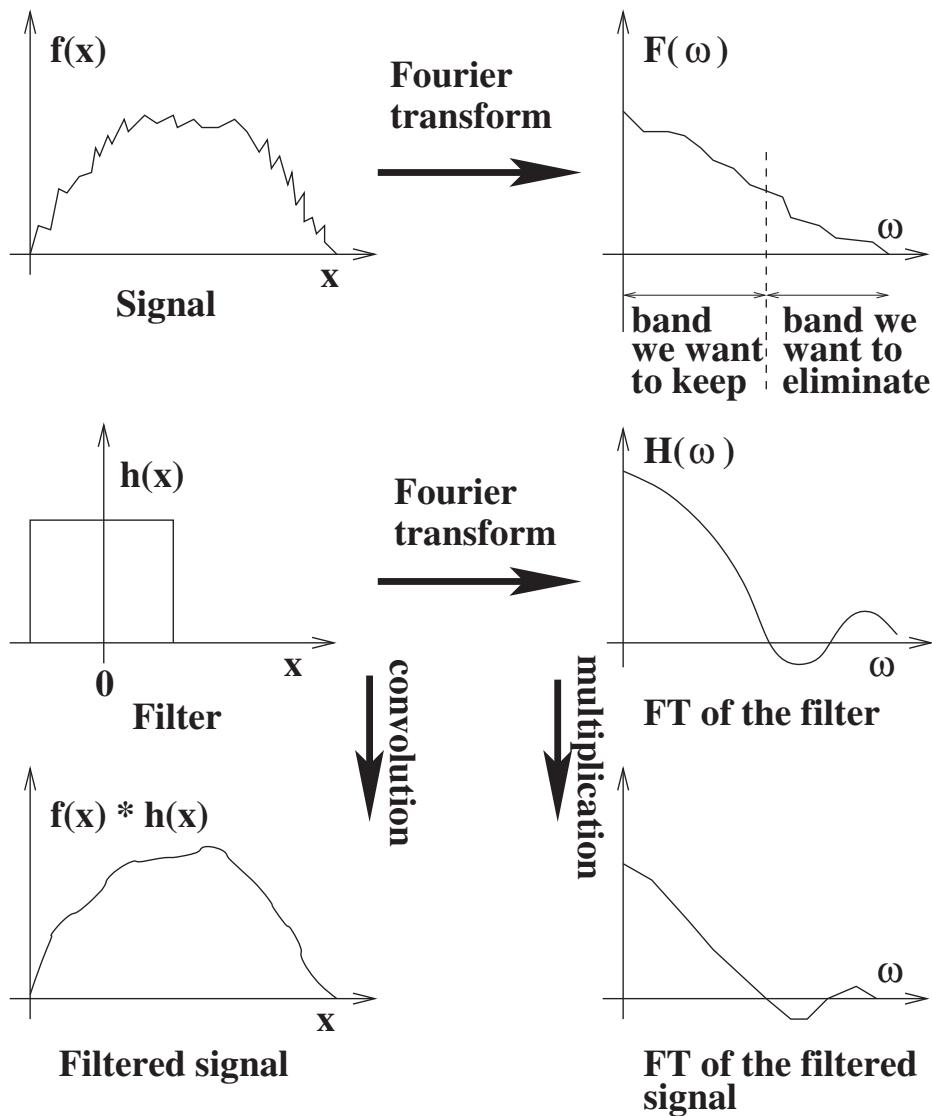


Figure 4.5: The case of a filter defined in the real domain, for *maximum convenience*. The processing route here is: image → convolution with the filter → filtered image. Compare this with the process followed in figure 4.1. Top row: a signal and its Fourier transform. Middle row: a flat filter on the left, and its Fourier transform on the right. Bottom row: on the left, the filtered signal that may be obtained by convolving the signal at the top with the filter in the middle; on the right the Fourier transform of the filtered signal obtained by multiplying the Fourier transform of the signal at the top, with the Fourier transform of the filter in the middle. Note that this filter is very convenient to implement in the real domain, but its side lobes in the frequency domain may cause artifacts in the signal, by keeping some high frequencies while killing others.

4.2 Reducing high frequency noise

What are the types of noise present in an image?

Noise in images is often assumed to be either **impulse noise** or **Gaussian noise**. Image noise is often assumed to be additive, zero-mean, unbiased, independent, uncorrelated, homogeneous, white, Gaussian and iid. For special cases, where high accuracy is required, it is advisable to work out specifically the noise model, as some or all of these assumptions may be violated.

What is impulse noise?

Impulse noise, also known as **shot noise** or **spec noise**, alters at random the values of some pixels. In a binary image this means that some black pixels become white and some white pixels become black. This is why this noise is also called **salt and pepper noise**. It is assumed to be Poisson distributed. A Poisson distribution has the form

$$p(k) = \frac{e^{-\lambda} \lambda^k}{k!} \quad (4.54)$$

where $p(k)$ is the probability of having k pixels affected by the noise in a window of a certain size, and λ is the average number of affected pixels in a window of the same fixed size. The variance of the Poisson distribution is also λ .

What is Gaussian noise?

Gaussian noise is the type of noise in which, at each pixel position (i, j) , the random noise value, that affects the true pixel value, is drawn from a Gaussian probability density function with mean $\mu(i, j)$ and standard deviation $\sigma(i, j)$. Unlike shot noise, which influences a few pixels only, this type of noise affects all pixel values.

What is additive noise?

If the random number of the noise field is added to the true value of the pixel, the noise is **additive**.

What is multiplicative noise?

If the random number of the noise field is multiplied with the true value of the pixel, the noise is **multiplicative**.

What is homogeneous noise?

If the noise parameters are the same for all pixels, the noise is **homogeneous**. For example, in the case of Gaussian noise, if $\mu(i, j)$ and $\sigma(i, j)$ are the same for all pixels (i, j) and equal, say, to μ and σ , respectively, the noise is homogeneous.

What is zero-mean noise?

If the mean value of the noise is zero ($\mu = 0$), the noise is **zero-mean**. Another term for zero-mean noise is **unbiased noise**.

What is biased noise?

If $\mu(i, j) \neq 0$ for at least some pixels, the noise is called **biased**. This is also known as **fixed pattern noise**. Such a noise can be easily converted to zero-mean by removing $\mu(i, j)$ from the value of pixel (i, j) .

What is independent noise?

As the noise value that affects each pixel is random, we may think of the noise process as a random field, the same size as the image, which, point by point, is added to (or multiplied with) the field that represents the image. We may say then, that the value of the noise at each pixel position is the outcome of a random experiment. If the result of the random experiment, which is assumed to be performed at a pixel position, is not affected by the outcome of the random experiment at other pixel positions, the noise is **independent**.

What is uncorrelated noise?

If the *average value of the product* of the noise values at any combination of n pixel positions, (averaged over all such n -tuples of positions in the image) is equal to the *product of the average noise values* at the corresponding positions, the noise is **uncorrelated**:

$$\text{Average_of}\{\text{product}\} = \text{Product_of}\{\text{averages}\} \quad (4.55)$$

For zero-mean noise this is the same as saying that if we consider any n -tuple of pixel positions and multiply their values and average these values over all such n -tuples we find in the image, the answer will be always 0. In practice, we consider only the autocorrelation function of the noise field, ie we consider only pairs of pixels in order to decide whether the noise is correlated or uncorrelated.

Let us perform the following thought experiment. Let us consider the noise field. We consider a pair of samples at a certain relative position from each other and multiply their values. If the noise is zero-mean, sometimes these two noise values will be both positive, sometimes both negative, and sometimes one will be positive and the other negative. This means that sometimes their product will be positive and sometimes negative. If the noise is assumed to be uncorrelated, we expect to have about equal number of positive and negative products if we consider all pairs of samples at the same relative position. So, the average value of the product of the noise values at a pair of positions, over all similar pairs of positions in the noise field, is expected to be 0. We shall get 0 for all relative positions of sample pairs, except when we consider a sample paired with itself, because in that case we average the square of the noise value over all samples. In that case it is not possible to get 0 because we would be averaging non-negative numbers.

What we are calculating with this thought experiment is the spatial autocorrelation function of the random field of noise. The average of the squared noise value is nothing other

than the variance σ^2 of the homogeneous noise field, and the result indicates that the auto-correlation function of this field is a delta function with value σ^2 at zero shift and 0 for all other shifts. So,

$$C(h) = \sigma^2 \delta(h) \quad (4.56)$$

where h is the distance between the pixels we pair together in order to compute the autocorrelation function $C(h)$.

What is white noise?

It is noise that has the same power at all frequencies (flat power spectrum). The term comes from the white light, which is supposed to have equal power at all frequencies of the electromagnetic spectrum. If the spectrum of the noise were not flat, but it had more power in some preferred frequencies, the noise would have been called **coloured noise**. For example, if the noise had more power in the high frequencies, which in the electromagnetic spectrum correspond to the blue light, we might have characterised the noise as **blue noise**. Note that the analogy with the colour spectrum is only a metaphor: in the case of noise we are talking about spatial frequencies, while in the case of light we are talking about electromagnetic frequencies.

What is the relationship between zero-mean uncorrelated and white noise?

The two terms effectively mean the same thing. The autocorrelation function of zero-mean uncorrelated noise is a delta function (see equation (4.56)). The Fourier transform of the auto-correlation function is the power spectrum of the field, according to the **Wiener-Khinchine theorem** (see Box 4.5, on page 325). The Fourier transform of a delta function is a function with equal amplitude at all frequencies (see Box 4.4, on page 325). So, the power spectrum of the uncorrelated zero-mean noise is a flat spectrum, with equal power at all frequencies. Therefore, for zero-mean noise, the terms “uncorrelated” and “white” are interchangeable.

What is iid noise?

This means **independent, identically distributed** noise. The term “independent” means that the joint probability density function of the combination of the noise values may be written as the product of the probability density functions of the individual noise components at the different pixels. The term “identically distributed” means that the noise components at all pixel positions come from identical probability density functions. For example, if the noise value at every pixel is drawn from the same Gaussian probability density function, but with no regard as to what values have been drawn in other pixel positions, the noise is described as iid.

If the noise component n_{ij} at pixel (i, j) is drawn from a Gaussian probability density function with mean μ and standard deviation σ , we may write for the joint probability density function $p(n_{11}, n_{12}, \dots, n_{NM})$ of all noise components

$$p(n_{11}, n_{12}, \dots, n_{NM}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(n_{11}-\mu)^2}{2\sigma^2}} \times \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(n_{12}-\mu)^2}{2\sigma^2}} \times \cdots \times \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(n_{NM}-\mu)^2}{2\sigma^2}} \quad (4.57)$$

where the size of the image has been assumed to be $N \times M$.

Example B4.8

Show that zero-mean iid noise is white.

Independent random variables are also uncorrelated random variables:

$$\begin{aligned}
 \text{Independent} \Rightarrow p(n_{11}, n_{12}, \dots, n_{NM}) &= \prod_{(i,j)=(1,1)}^{(N,M)} p(n_{ij}) \\
 \Rightarrow \text{Mean}(n_{11}, n_{12}, \dots, n_{NM}) &= \prod_{(i,j)=(1,1)}^{(N,M)} \text{Mean}(n_{ij}) \\
 \Rightarrow \text{Mean}(n_{ij}, n_{i+k,j+l}) &= \text{Mean}(n_{ij})\text{Mean}(n_{i+k,j+l}) = 0 \quad \forall(i,j),(k,l) \\
 \Rightarrow \text{Autocorrelation Function} &= \text{Delta Function} \\
 \Rightarrow \text{White spectrum} &
 \end{aligned} \tag{4.58}$$

Example B4.9

Show that biased iid noise is coloured.

Consider two pixels (i, j) and $(i + k, j + l)$. Each one has a noise component that consists of a constant value b and a zero-mean part ϵ_{ij} and $\epsilon_{i+k,j+l}$, respectively, so that $n_{ij} \equiv b + \epsilon_{ij}$ and $n_{i+k,j+l} = b + \epsilon_{i+k,j+l}$. Consider the autocorrelation function for these two pixel positions:

$$\begin{aligned}
 \langle n_{ij} n_{i+k,j+l} \rangle &= \langle (b + \epsilon_{ij})(b + \epsilon_{i+k,j+l}) \rangle \Rightarrow \\
 &= b^2 + b \underbrace{\langle \epsilon_{i+k,j+l} \rangle}_{=0} + b \underbrace{\langle \epsilon_{ij} \rangle}_{=0} + \underbrace{\langle \epsilon_{ij} \epsilon_{i+k,j+l} \rangle}_{=0} \Rightarrow \\
 &= b^2 = \text{a constant}
 \end{aligned} \tag{4.59}$$

The Fourier transform of a constant is a delta function: $2\pi b^2 \delta(\omega)$. Such a spectrum is clearly an impulse, ie not white.

Is it possible to have white noise that is not iid?

Yes. The white spectrum means that the autocorrelation function of the noise is a delta function. For zero-mean noise this implies uncorrelatedness, but not independence (see examples 4.10, 4.11 and 4.12).

Example B4.10

Consider an iid 1D zero-mean uniform noise signal $x(i)$ in the range $[-3, 3]$. From this construct a noise signal $y(j)$ as follows:

$$\begin{aligned} y(2i) &= x(i) \\ y(2i+1) &= k[x(i)^2 - 3] \end{aligned} \quad (4.60)$$

Select a value for k so that the variance of $y(j)$ is also 3 and show that $y(j)$ is a zero-mean noise.

From example 3.53, on page 244, we know that the variance σ^2 of $x(i)$ is 3 ($A = 3$ in (3.170)). The average (expectation value) of the even samples of $y(j)$ is clearly zero, as the average of $x(i)$ is zero. The average of the odd samples of $y(j)$ is

$$\langle y(2i+1) \rangle = \langle k[x(i)^2 - 3] \rangle = k[\langle x(i)^2 \rangle - 3] = k[3 - 3] = 0 \quad (4.61)$$

since the variance, $\langle x(i)^2 \rangle$, of $x(i)$ is 3. So $y(j)$ has mean 0. Then its variance is the same as the average of the squares of its samples. Obviously, the average of its even samples is $\langle x(i)^2 \rangle = 3$. The average of its odd samples is:

$$\begin{aligned} \langle y(2i+1)^2 \rangle &= \langle k^2 x(i)^4 - 6k^2 x(i)^2 + 9k^2 \rangle \\ &= k^2 \langle x(i)^4 \rangle - 6k^2 \langle x(i)^2 \rangle + 9k^2 \\ &= k^2 \frac{9}{5} \times 3^2 - 6k^2 \times 3 + 9k^2 \\ &= k^2 \frac{36}{5} \end{aligned} \quad (4.62)$$

Here we made use of (3.185), on page 247, for the value of $\langle x(i)^4 \rangle \equiv \mu_4$. So, the variance of the odd samples of $y(j)$ will be 3 if:

$$k = \sqrt{\frac{15}{36}} = \frac{\sqrt{15}}{6} = 0.6455 \quad (4.63)$$

Example B4.11

Show that the noise signal $y(j)$ you constructed in example 4.10 is white but not independent.

To show that the signal is white, we must show that it has 0 mean and its autocorrelation function is a delta function. We have already shown in example 4.10 that it has 0 mean. Its autocorrelation function for 0 shift ($h = 0$) is its variance, which was shown to be 3 in example 4.10. The autocorrelation function for shift $h \geq 2$ is expected to be 0, because the pairs of values that will be averaged will be from independently drawn values according to signal $x(i)$. We have only to worry about shift $h = 1$, because clearly, in this case, the second member of a pair of such values depends on the value of the first member of the pair, according to (4.60). Let us consider the average of the product of such a pair of values:

$$\begin{aligned} \langle y(2i)y(2i+1) \rangle &= \langle x(i)k[x(i)^2 - 3] \rangle \\ &= k\langle x(i)^3 \rangle - 3k\langle x(i) \rangle \\ &= 0 \end{aligned} \tag{4.64}$$

Here we made use of the fact that $x(i)$ are uniformly distributed numbers with 0 mean, and so their third moment must be 0. So, the autocorrelation function of $y(j)$ is 0 for all shifts except shift 0. This makes it a delta function and its Fourier transform flat, ie noise $y(j)$ is white.

Noise $y(j)$, however, is clearly not independent by construction. Figure 4.6a shows the first 100 samples of a noise sequence $x(i)$ we created. Figure 4.6b shows the first 100 samples of the corresponding noise sequence $y(j)$. In total we created a sequence of 1000 samples long. The mean of $x(i)$ was computed to be -0.0182 , and its variance 2.8652 . The mean of $y(j)$ was computed to be -0.0544 , and its variance 2.9303 . Figure 4.6c shows its autocorrelation function as a function of the shift h computed using

$$C(h) \equiv \frac{1}{N_h} \sum_{j=1}^{N-h} y(j)y(j+h) \tag{4.65}$$

where N is the total number of samples in the sequence and N_h is the number of pairs of samples at shift h .

Figure 4.6d shows all pairs of two successive samples of the sequence, plotted against each other. We can clearly see that the samples are not independent.

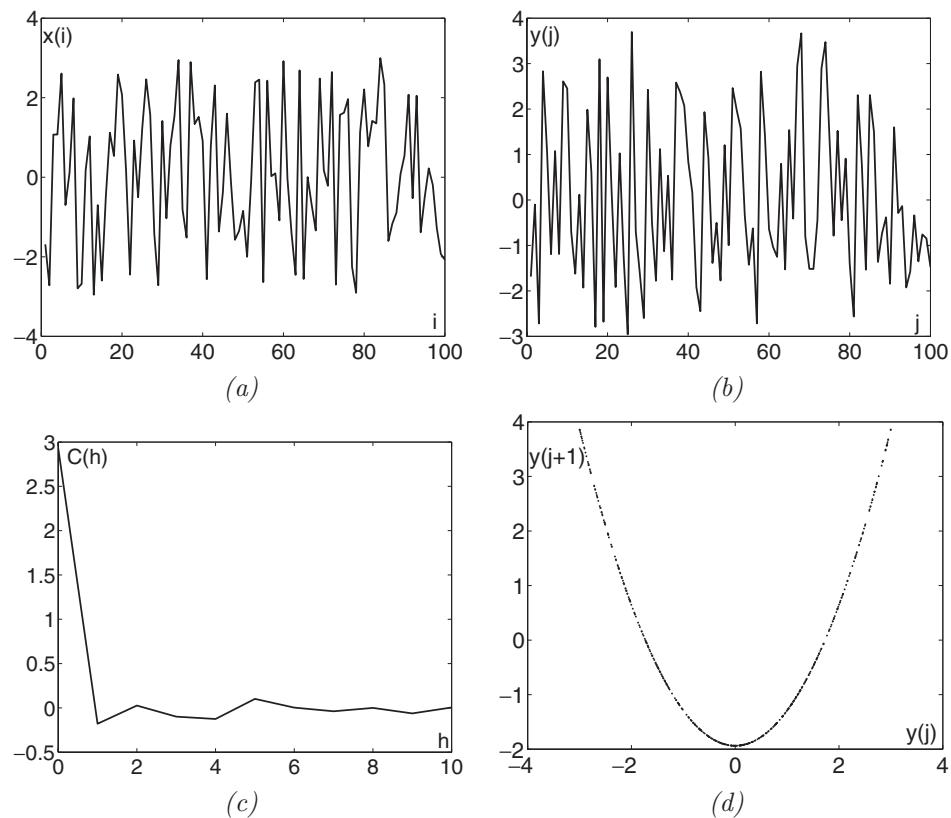


Figure 4.6: (a) The first 100 samples of the noise sequence $x(i)$. (b) The first 100 samples of the noise sequence $y(j)$. (c) The autocorrelation function of $y(j)$ for the first 11 possible shifts ($h = 0, 1, 2, \dots, 10$). It is clearly a delta function indicating uncorrelated noise. (d) $y(j + 1)$ plotted versus $y(j)$.

Example B4.12

Consider an iid 1D zero-mean Gaussian noise signal $x(i)$ with unit variance. From this construct a noise signal $y(j)$ as follows:

$$\begin{aligned} y(2i) &= x(i) \\ y(2i + 1) &= \frac{x(i)^2 - 1}{\sqrt{2}} \end{aligned} \quad (4.66)$$

Show that $y(j)$ is zero-mean white noise with unit variance, which is not iid.

The noise is zero-mean: the mean value of the even samples is clearly 0 by construction. The mean value of the odd samples is:

$$\begin{aligned}\langle y(2i+1) \rangle &= \frac{1}{\sqrt{2}} (\langle x(i)^2 \rangle - 1) \\ &= \frac{1}{\sqrt{2}} (1 - 1) = 0\end{aligned}\quad (4.67)$$

Next, we must examine whether the signal has unit variance. The variance of the even samples is 1 by construction. The variance of the odd samples is:

$$\begin{aligned}\langle y(2i+1)^2 \rangle &= \frac{1}{2} (\langle x(i)^4 \rangle + 1 - 2\langle x(i)^2 \rangle) \\ &= \frac{1}{2}(3 + 1 - 2) = 1\end{aligned}\quad (4.68)$$

Here we made use of (3.163), on page 242.

Then we must examine whether the signal may be used to represent white noise. To show that the signal has a white power spectrum, we must show that its autocorrelation function is a delta function. Its autocorrelation function for 0 shift ($h = 0$) is its variance, which is 1. The autocorrelation function for shift $h \geq 2$ is expected to be 0, because the pairs of values that will be averaged will be from independently drawn values according to signal $x(i)$. We have only to worry about shift $h = 1$, because clearly, in this case, the second value of each pair of samples depends on the first value of the pair, according to (4.66). Let us consider the average of the product of such a pair of values:

$$\begin{aligned}\langle y(2i)y(2i+1) \rangle &= \frac{1}{\sqrt{2}} \langle x(i) [x(i)^2 - 1] \rangle \\ &= \frac{1}{\sqrt{2}} (\langle x(i)^3 \rangle - \langle x(i) \rangle) \\ &= 0\end{aligned}\quad (4.69)$$

Here we made use of the fact that $x(i)$ are Gaussianly distributed numbers with 0 mean, and, therefore, their third moment must be 0 too. So, the autocorrelation function of $y(j)$ is 0 for all shifts except shift 0. This makes it a delta function and its Fourier transform flat, ie noise $y(j)$ is white.

Noise $y(j)$, however, is clearly not independent by construction. Its even samples are clearly Gaussianly distributed, while its odd samples are not (see example 4.13).

Figure 4.7a shows the first 100 samples of a noise sequence $x(i)$ we created. Figure 4.7b shows the first 100 samples of the corresponding noise sequence $y(j)$. In total we created a sequence of 1000 samples long. The mean of $x(i)$ was computed to be

0.0264, and its variance 1.0973. The mean of $y(j)$ was computed to be 0.0471, and its variance 1.1230. Figure 4.7c shows its autocorrelation function as a function of the shift h computed using

$$C(h) \equiv \frac{1}{N_h} \sum_{j=1}^{N-h} y(j)y(j+h) \quad (4.70)$$

where N is the total number of samples in the sequence and N_h is the number of pairs of samples at shift h .

Figure 4.7d shows all pairs of two successive samples of the sequence, plotted against each other. We can clearly see that the samples are not independent.

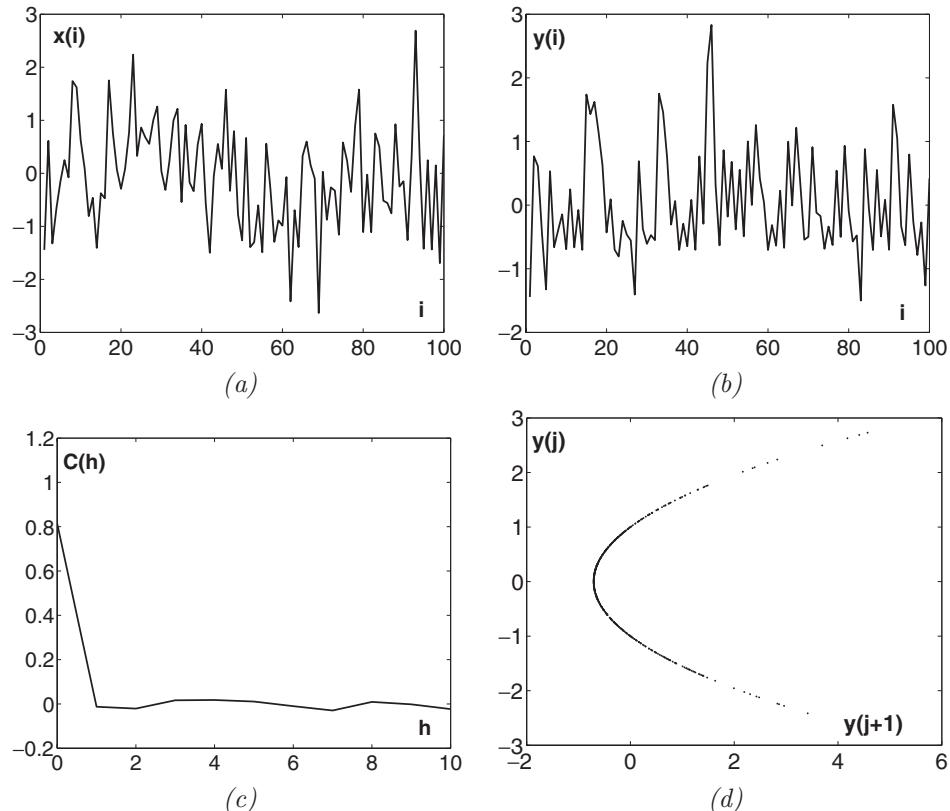


Figure 4.7: (a) The first 100 samples of the noise sequence $x(i)$. (b) The first 100 samples of the noise sequence $y(j)$. (c) The autocorrelation function of $y(j)$ for the first 11 possible shifts ($h = 0, 1, 2, \dots, 10$). It is clearly a delta function indicating uncorrelated noise. (d) $y(j)$ plotted versus $y(j + 1)$.

Box 4.3. The probability density function of a function of a random variable

Assume that variable x is distributed according to probability density function $p_1(x)$. Assume also that $y = g(x)$. Finally, assume that the real roots of this equation are x_1, x_2, \dots, x_n , ie for a specific value of $y = y_0$,

$$x_1 = g^{-1}(y_0), \dots, x_n = g^{-1}(y_0) \quad (4.71)$$

Then the probability density function $p_2(y)$ of y is

$$p_2(y) = \frac{p_1(x_1)}{|g'(x_1)|} + \frac{p_1(x_2)}{|g'(x_2)|} + \dots + \frac{p_1(x_n)}{|g'(x_n)|} \quad (4.72)$$

where $g'(x) = \frac{dg(x)}{dx}$.

This formula is intuitively obvious: $p_2(y_0)$ expresses the number of values of y that fall inside interval dy , around y_0 . If we know that n different values of x give rise to the same value of $y = y_0$, we must consider all of them. Inside an interval dx around the first of these roots, there are $p_1(x_1)dx$ values of x that will give rise to values of y about the y_0 value, inside an interval dy related to dx by $dy/dx|_{x=x_1} = g'(x_1) \Rightarrow dy = g'(x_1)dx$. Inside an interval dx around the second of these roots, there are $p_1(x_2)dx$ values of x that will give rise to values of y about the y_0 value, inside an interval dy related to dx by $dy = g'(x_2)dx$, and so on. To obtain the density we need, we must sum up all these contributing densities. Each number of contributed values has first to be divided by the width in which it falls, in order to become a density. The width of the interval is given by the absolute value of the interval, eg the first of these widths is $|g'(x_1)|dx$. The dx of the denominator in each term cancels the dx of the numerator, thus, formula (4.72) follows.

Example B4.13

Work out the probability density function according to which the odd samples of sequence $y(j)$ in example 4.12 are distributed.

We shall use (4.72) with

$$\begin{aligned} p_1(x) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \\ g(x) &= \frac{1}{\sqrt{2}}(x^2 - 1) \\ g'(x) &= \frac{1}{\sqrt{2}}2x = \sqrt{2}x \end{aligned} \quad (4.73)$$

The roots of $g(x)$ are:

$$\begin{aligned} y &= \frac{1}{\sqrt{2}}(x^2 - 1) \Rightarrow \\ \sqrt{2}y + 1 &= x^2 \Rightarrow \\ x &= \pm\sqrt{\sqrt{2}y + 1} \Rightarrow \begin{cases} x_1 = +\sqrt{\sqrt{2}y + 1} \\ x_2 = -\sqrt{\sqrt{2}y + 1} \end{cases} \end{aligned} \quad (4.74)$$

Then the probability density function $p_2(y)$ of the odd samples of sequence $y(j)$ in example 4.12 is:

$$\begin{aligned} p_2(y) &= \frac{1}{|\sqrt{2}\sqrt{\sqrt{2}y + 1}|} \frac{1}{\sqrt{2\pi}} e^{-\frac{\sqrt{2}y+1}{2}} + \frac{1}{|-\sqrt{2}\sqrt{\sqrt{2}y + 1}|} \frac{1}{\sqrt{2\pi}} e^{-\frac{\sqrt{2}y+1}{2}} \\ &= \frac{1}{|\sqrt{\sqrt{2}y + 1}| \sqrt{\pi}} e^{-\frac{\sqrt{2}y+1}{2}} \end{aligned} \quad (4.75)$$

Figure 4.8 shows a plot of this function. This function is clearly not a Gaussian. So, signal $y(j)$, of example 4.12, has a different probability density function for its even and its odd samples, and, therefore, it is not iid.

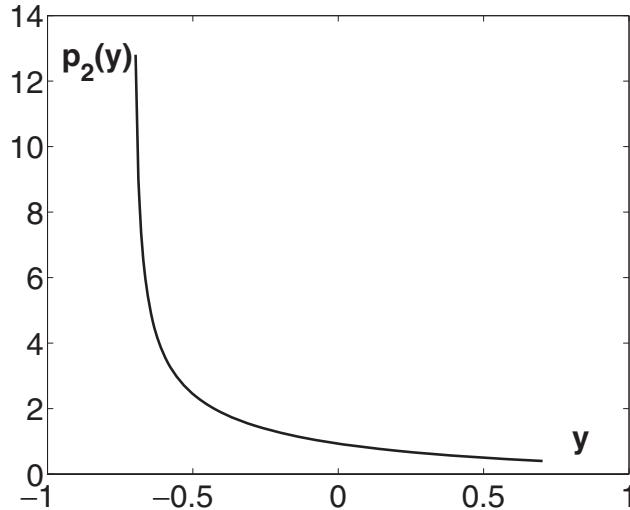


Figure 4.8: The probability density function of the odd samples of sequence $y(j)$ defined by equation (4.66).

Example B4.14

Work out the probability density function according to which the odd samples of sequence $y(j)$ in example 4.10 are distributed.

We shall use (4.72) with

$$\begin{aligned} p_1(x) &= \begin{cases} \frac{1}{6} & \text{for } -3 \leq x \leq 3 \\ 0 & \text{elsewhere} \end{cases} \\ g(x) &= \frac{\sqrt{15}}{6}(x^2 - 3) \\ g'(x) &= \frac{\sqrt{15}}{6}2x = \frac{\sqrt{15}}{3}x \end{aligned} \quad (4.76)$$

The roots of $g(x)$ are:

$$\begin{aligned} y &= \frac{\sqrt{15}}{6}(x^2 - 3) \Rightarrow \\ \frac{6}{\sqrt{15}}y + 3 &= x^2 \Rightarrow \\ x &= \pm \sqrt{\frac{6}{\sqrt{15}}y + 3} \Rightarrow \begin{cases} x_1 &= +\sqrt{\frac{6}{\sqrt{15}}y + 3} \\ x_2 &= -\sqrt{\frac{6}{\sqrt{15}}y + 3} \end{cases} \end{aligned} \quad (4.77)$$

Then the probability density function $p_2(y)$ of the odd samples of sequence $y(j)$ in example 4.11 is:

$$p_2(y) = \begin{cases} \frac{1}{3}\sqrt{\frac{6}{\sqrt{15}}y + 3}^{\frac{1}{3}} & \text{for } -3 \leq \sqrt{\frac{6}{\sqrt{15}}y + 3} \leq 3 \text{ and } -3 \leq -\sqrt{\frac{6}{\sqrt{15}}y + 3} \leq 3 \\ \frac{1}{3}\sqrt{\frac{6}{\sqrt{15}}y + 3}^{\frac{1}{6}} & \text{for } -3 \leq \sqrt{\frac{6}{\sqrt{15}}y + 3} \leq 3 \text{ and } -\sqrt{\frac{6}{\sqrt{15}}y + 3} \notin [-3, 3] \\ \frac{1}{3}\sqrt{\frac{6}{\sqrt{15}}y + 3}^{\frac{1}{6}} & \text{for } \sqrt{\frac{6}{\sqrt{15}}y + 3} \notin [-3, 3] \text{ and } -3 \leq -\sqrt{\frac{6}{\sqrt{15}}y + 3} \leq 3 \\ 0 & \text{for } \sqrt{\frac{6}{\sqrt{15}}y + 3} \notin [-3, 3] \text{ and } -\sqrt{\frac{6}{\sqrt{15}}y + 3} \notin [-3, 3] \end{cases} \quad (4.78)$$

Let us examine the inequalities that appear as conditions of the above equation. First of all, we observe that if $-3 \leq \sqrt{\frac{6}{\sqrt{15}}y + 3} \leq 3$, it is not possible for $-\sqrt{\frac{6}{\sqrt{15}}y + 3}$ not to be in the same interval, as if z is smaller than 3, $-z$ will be bigger than -3 . So, the second and third branches of the equation are impossible. Next, we note that

$\frac{6}{\sqrt{15}}y + 3$ should always be positive, otherwise we shall not have a real root. With this understanding, we can work out the range of values of y :

$$\begin{aligned} -3 \leq \sqrt{\frac{6}{\sqrt{15}}}y + 3 \leq 3 &\Leftrightarrow \\ 0 \leq \frac{6}{\sqrt{15}}y + 3 \leq 9 &\Leftrightarrow \\ -3 \leq \frac{6}{\sqrt{15}}y \leq 6 &\Leftrightarrow \\ -\frac{\sqrt{15}}{2} \leq y \leq \sqrt{15} & \end{aligned} \quad (4.79)$$

Then, the probability density function of y is:

$$p_2(y) = \begin{cases} \frac{1}{\sqrt{15}\sqrt{\frac{6}{\sqrt{15}}y+3}} & \text{for } -\frac{\sqrt{15}}{2} \leq y \leq \sqrt{15} \\ 0 & \text{otherwise} \end{cases} \quad (4.80)$$

Figure 4.9 shows a plot of this function. This function clearly does not represent a uniform distribution. So, signal $y(j)$, of example 4.10, has a different probability density function for its even and its odd samples, and, therefore, is not iid.

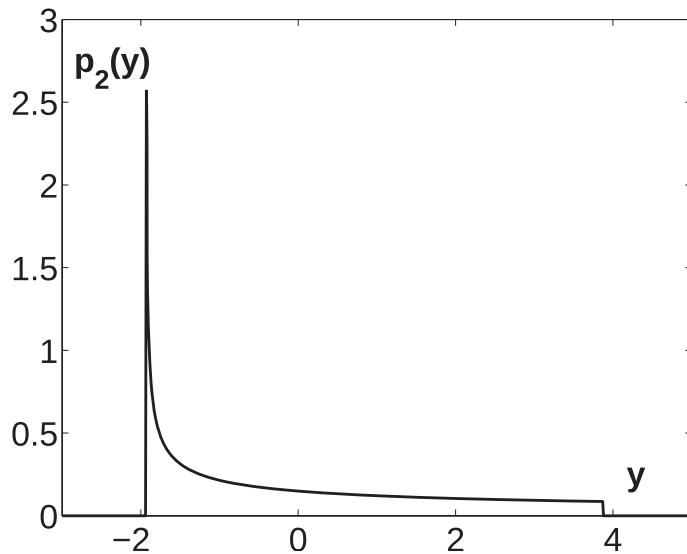


Figure 4.9: The probability density function of the odd samples of sequence $y(j)$ defined by equation (4.60).

Why is noise usually associated with high frequencies?

This is a misconception, particularly when the assumption of white noise is made. White noise affects all frequencies. However, in general, the deterministic component of the image has higher power in the low frequencies. If the noise has the same power in all frequencies, there is a cutoff frequency, beyond which the power spectrum of a noisy image is dominated by the noise spectrum (see figure 4.10 for the case of additive noise). It is this cutoff point that various methods try to identify so that they rid the image from frequencies higher than that in order to remove the noise. Of course, useful high frequencies are also removed at the same time and noise at low frequencies remains, and that is why it is not possible to remove white noise entirely and create a perfect noise-free image.

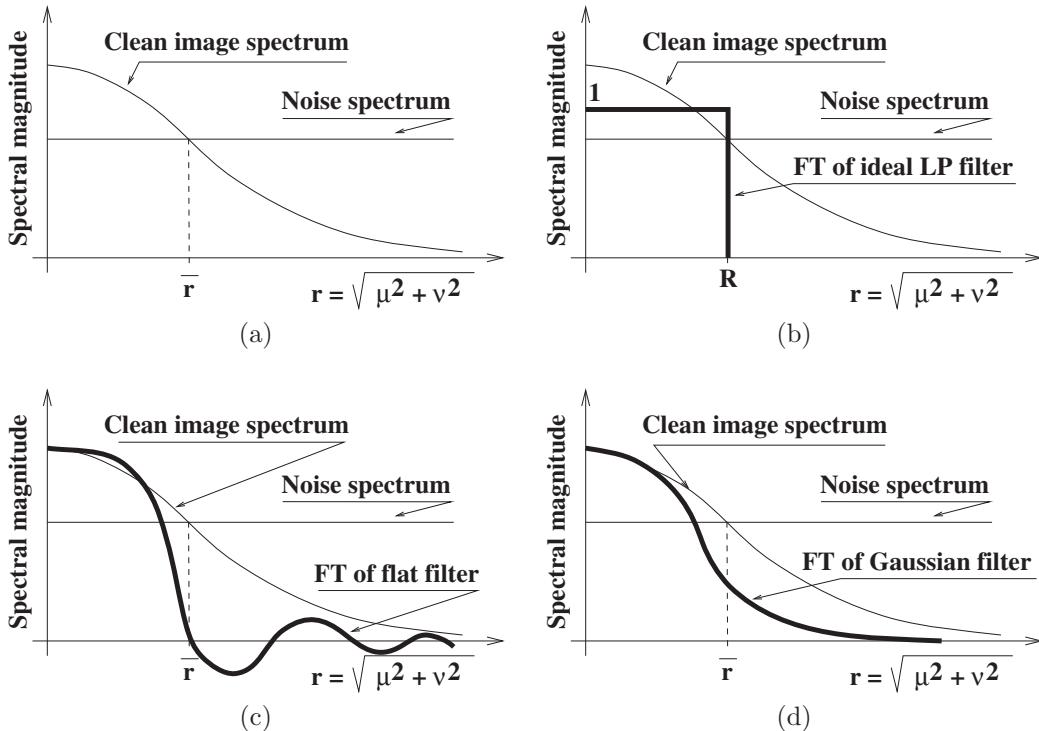


Figure 4.10: The case of additive white noise. (a) Beyond a certain frequency, the spectrum of a noisy image is dominated by the spectrum of the noise. Ideally we should use a filter that will eliminate all frequencies beyond the change over frequency \bar{r} . This is the idea behind low pass filtering a noisy image. The issue is which low pass filter one should use: (b) the ideal low pass filter with cutoff frequency $R = \bar{r}$, which has to be implemented in the frequency domain, or (c) a filter conveniently defined in the real domain with imperfect frequency response, which may enhance some of the frequencies we wish to kill. (d) A compromise is a Gaussian filter that goes smoothly to zero in the real as well as in the frequency domain. However, the Gaussian filter will not treat all frequencies the same: some desirable frequencies will be subdued and some undesirable frequencies will be suppressed less than others.

How do we deal with multiplicative noise?

Multiplicative noise may easily be converted to additive noise by taking the logarithm of the noisy signal. If $s(i)$ is a signal that is affected by multiplicative noise $n(i)$, the result will be a noisy signal $t(i) = s(i)n(i)$. To remove $n(i)$ from $t(i)$, we first take the logarithm of $t(i)$: $\tilde{t}(i) \equiv \log t(i) = \log s(i) + \log n(i)$. We may call $\log s(i) \equiv \tilde{s}(i)$ and $\log n(i) \equiv \tilde{n}(i)$. We then have a noisy signal $\tilde{t}(i) = \tilde{s}(i) + \tilde{n}(i)$, from which we have to remove the additive noise $\tilde{n}(i)$. To perform this task we may apply any method appropriate for dealing with additive noise.

A case of multiplicative noise/interference is discussed in the next section.

Box 4.4. The Fourier transform of the delta function

We insert the delta function into the definition formula of the Fourier transform:

$$\Delta(\omega) = \int_{-\infty}^{+\infty} \delta(t)e^{-j\omega t} dt = e^{-j\omega t} \Big|_{t=0} = 1 \quad (4.81)$$

Here we made use of the following property of the delta function: when the delta function is multiplied with another function and integrated from $-\infty$ to $+\infty$, it picks up the value of the other function at the point where the argument of the delta function is zero. In this particular case, the argument of the delta function is t and it is zero at $t = 0$.

Box 4.5. Wiener-Khinchine theorem

We shall show that the Fourier transform of the spatial autocorrelation function of a real-valued random field $f(x, y)$ is equal to the spectral power density $|\hat{F}(u, v)|^2$ of the field.

The spatial autocorrelation function of $f(x, y)$ is defined as

$$R_{ff}(\tilde{x}, \tilde{y}) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x + \tilde{x}, y + \tilde{y}) f(x, y) dx dy \quad (4.82)$$

We multiply both sides of equation (4.82) with the kernel of the Fourier transform $e^{-j(\tilde{x}u + \tilde{y}v)}$, where u and v are the frequencies along the x and y axis, respectively, and integrate over all spatial coordinates of $R_{ff}(\tilde{x}, \tilde{y})$, to obtain the Fourier transform, $\hat{R}_{ff}(u, v)$, of $R_{ff}(\tilde{x}, \tilde{y})$:

$$\begin{aligned} \hat{R}_{ff}(u, v) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} R_{ff}(\tilde{x}, \tilde{y}) e^{-j(\tilde{x}u + \tilde{y}v)} d\tilde{x} d\tilde{y} \\ &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x + \tilde{x}, y + \tilde{y}) f(x, y) e^{-j(\tilde{x}u + \tilde{y}v)} dx dy d\tilde{x} d\tilde{y} \end{aligned} \quad (4.83)$$

We define new variables of integration $s_1 \equiv x + \tilde{x}$ and $s_2 \equiv y + \tilde{y}$ to replace the integrals over \tilde{x} and \tilde{y} . We have $\tilde{x} = s_1 - x$, $\tilde{y} = s_2 - y$, $d\tilde{x}d\tilde{y} = ds_1ds_2$ and no change in the limits of integration:

$$\hat{R}_{ff}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(s_1, s_2) f(x, y) e^{-j((s_1-x)u+(s_2-y)v)} dx dy ds_1 ds_2$$

The two double integrals on the right-hand side are separable, so we may write:

$$\hat{R}_{ff}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(s_1, s_2) e^{-j(s_1 u + s_2 v)} ds_1 ds_2 \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) e^{j(x u + y v)} dx dy$$

We recognise the first of the double integrals on the right-hand side of this equation to be the Fourier transform $\hat{F}(u, v)$ of $f(s_1, s_2)$ and the second double integral its complex conjugate $\hat{F}^*(u, v)$. Therefore:

$$\hat{R}_{ff}(u, v) = \hat{F}(u, v) \hat{F}^*(u, v) = |\hat{F}(u, v)|^2$$

Is the assumption of Gaussian noise in an image justified?

According to the central limit theorem we discussed in Chapter 3, page 235, when several random numbers are added, the sum tends to be Gaussianly distributed. There are many sources of noise in an image, like instrument noise, quantisation noise, etc. We may, therefore, assume that all these noise components combined may be modelled as Gaussian noise, and that is why it is very common to assume that the noise in an image is Gaussian. The shot noise appears in special cases: in **synthetic aperture radar images (SAR)** due to the special imaging conditions, or in ordinary images due to degradations caused by specific sources, like, for example, damage of old photographs by insects or sprayed chemicals.

How do we remove shot noise?

We use various statistical filters, like **rank order filtering** or **mode filtering**.

What is a rank order filter?

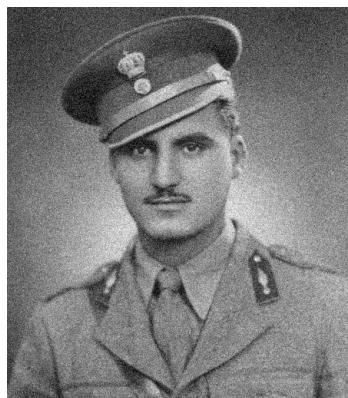
A rank order filter is a filter the output value of which depends on the ranking of the pixels according to their grey values inside the filter window. The most common rank order filter is the **median filter**.

What is median filtering?

The median is the value which divides a distribution in two equally numbered populations. For example, if we use a 5×5 window, we have 25 grey values which we order in an increasing sequence. Then the median is the thirteenth value. Median filtering has the effect of forcing



(a) Image with impulse noise



(b) Image with additive Gaussian noise



(c) Median filtering of (a)



(d) Median filtering of (b)



(e) Smoothing of (a) by averaging



(f) Smoothing of (b) by averaging

Figure 4.11: We must use the right type of filtering for each type of noise: on the left, the image “Officer” damaged by impulse noise, and below it, attempts to remove this noise by median filtering and by spatial averaging; on the right, the same image damaged by zero-mean, additive, white, Gaussian noise, and below it, attempts to remove it by median filtering and by spatial averaging. Note that spatial averaging does not really work for impulse noise, and median filtering is not very effective for Gaussian noise.

points with distinct intensities to be more like their neighbours, thus eliminating intensity spikes which appear isolated.

Figure 4.11c shows image 4.11a processed with a median filter and with a window of size 5×5 , while figure 4.11d shows image 4.11b (which contains Gaussian noise) having been processed in the same way. It is clear that the median filter removes the impulse noise almost completely.

What is mode filtering?

Mode filtering involves assigning to the central pixel the most common value inside the local window around the pixel (the mode of the histogram of the local values).

How do we reduce Gaussian noise?

We can remove Gaussian noise by **smoothing** the image. For example, we may replace the value of each pixel by the average value inside a small window around the pixel. Figures 4.11e and 4.11f show the result of applying this process to images 4.11a and 4.11b, respectively. The size of the window used is the same as for the median filtering of the same images, ie 5×5 . We note that this type of filtering is much more effective for the Gaussian noise, but produces bad results in the case of impulse noise. This is a simple form of **lowpass filtering** of the image. A better way to low pass filter the image is to use a Gaussian filter of size $(2M + 1) \times (2M + 1)$, rather than using a flat filter. Note that all low pass filters (like the Gaussian filter) are effectively averaging filters, computing a weighted average of the values inside the local window.

Example 4.15

We process an image by using windows of size 5×5 . The grey pixel values inside a 5×5 subimage are:

15, 17, 15, 17, 16, 10, 8, 9, 18, 15, 16, 12, 14, 11, 15, 14, 15, 18, 100, 15, 14, 13, 12, 12, 17.

Which value would

- (i) a local averaging,
- (ii) a median and
- (iii) a mode

filter assign to the central pixel of this subimage?

(i) The local averaging filter would assign to the central pixel of the subimage the rounded to the nearest integer average value of the pixels inside the 5×5 window:

$$\begin{aligned} \text{Average} &= \frac{1}{25}(15 + 17 + 15 + 17 + 16 + 10 + 8 + 9 + 18 + 15 + 16 + 12 + 14 + 11 \\ &\quad + 15 + 14 + 15 + 18 + 100 + 15 + 14 + 13 + 12 + 17) = 17.52 \end{aligned}$$

So, the assigned value will be 18.

(ii) The median filter will assign to the central pixel the median value of the grey values inside the window. To identify the median value, we first rank all the grey values we are given:

8, 9, 10, 11, 12, 12, 12, 13, 14, 14, 14, 15, 15, 15, 15, 15, 15, 16, 16, 17, 17, 17, 18, 18, 100

The 13th number in this sequence is the median, which is 15 and this is the value assigned to the central pixel by the median filter.

(iii) The mode in the above list of numbers is also 15, because this is the most frequent number. So, the mode filter will also assign value 15 to the central pixel.

We note that the outlier value 100, which most likely is the result of impulse noise, severely affected the output of the mean filter, but it did not affect the value either of the mode or the median filter. That is why an averaging filter (and in general a low pass convolution filter) is not appropriate for removing impulse noise.

Example 4.16

Work out the weights of Gaussian smoothing filters of size $(2M+1) \times (2M+1)$, for $M = 2, 3$ and 4 .

The values of these filters will be computed according to function

$$g(r) = e^{-\frac{r^2}{2\sigma^2}} \quad (4.84)$$

where σ is a filter parameter, that has to be specified, and $r^2 \equiv x^2 + y^2$. Since the filter will be $(2M+1) \times (2M+1)$, the value of the filter at the truncation point will be given by $g(M) = e^{-M^2/(2\sigma^2)}$ (see figure 4.12). We wish the filter to go smoothly to 0, so this final value of the filter should be small. Let us call it ϵ . Parameter σ should be chosen so that:

$$\begin{aligned} \epsilon &= e^{-\frac{M^2}{2\sigma^2}} \Rightarrow \\ \ln \epsilon &= -\frac{M^2}{2\sigma^2} \Rightarrow \\ \sigma &= \frac{M}{\sqrt{-2 \ln \epsilon}} \end{aligned} \quad (4.85)$$

For $\epsilon = 0.01$ we work out that σ should be 0.659 for $M = 2$, 0.989 for $M = 3$ and 1.318 for $M = 4$.

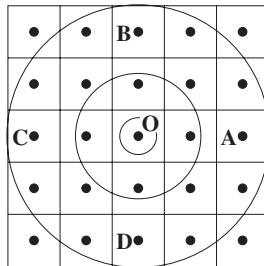


Figure 4.12: The circles represent the isocontours of the Gaussian function we use to compute the values of the $(2M + 1) \times (2M + 1)$ smoothing filter at the black dots. In this case, $M = 2$. Point O is at position $(0, 0)$ and function (4.84) has value 1 there. The value of the function is set to 0 outside the 5×5 square. The places where this truncation creates the maximum step are points A , B , C and D . At those points, function (4.84) has value $e^{-2^2/(2\sigma^2)}$, as those points are at coordinates $(\pm M, 0)$ and $(0, \pm M)$.

Finally, we compute the values of the filter using (4.84) for all positions in a $(2M + 1) \times (2M + 1)$ grid, assuming that the $(0, 0)$ point is the central cell of the grid. The values we find for $M = 2, 3$ and 4 , respectively, are:

0.0001	0.0032	0.0100	0.0032	0.0001
0.0032	0.1000	0.3162	0.1000	0.0032
0.0100	0.3162	1.0000	0.3162	0.0100
0.0032	0.1000	0.3162	0.1000	0.0032
0.0001	0.0032	0.0100	0.0032	0.0001

0.0001	0.0013	0.0060	0.0100	0.0060	0.0013	0.0001
0.0013	0.0167	0.0774	0.1292	0.0774	0.0167	0.0013
0.0060	0.0774	0.3594	0.5995	0.3594	0.0774	0.0060
0.0100	0.1292	0.5995	1.0000	0.5995	0.1292	0.0100
0.0060	0.0774	0.3594	0.5995	0.3594	0.0774	0.0060
0.0013	0.0167	0.0774	0.1292	0.0774	0.0167	0.0013
0.0001	0.0013	0.0060	0.0100	0.0060	0.0013	0.0001

0.0001	0.0007	0.0032	0.0075	0.0100	0.0075	0.0032	0.0007	0.0001
0.0007	0.0056	0.0237	0.0562	0.0750	0.0562	0.0237	0.0056	0.0007
0.0032	0.0237	0.1000	0.2371	0.3162	0.2371	0.1000	0.0237	0.0032
0.0075	0.0562	0.2371	0.5623	0.7499	0.5623	0.2371	0.0562	0.0075
0.0100	0.0750	0.3162	0.7499	1.0000	0.7499	0.3162	0.0750	0.0100
0.0075	0.0562	0.2371	0.5623	0.7499	0.5623	0.2371	0.0562	0.0075
0.0032	0.0237	0.1000	0.2371	0.3162	0.2371	0.1000	0.0237	0.0032
0.0007	0.0056	0.0237	0.0562	0.0750	0.0562	0.0237	0.0056	0.0007
0.0001	0.0007	0.0032	0.0075	0.0100	0.0075	0.0032	0.0007	0.0001

To make sure that the filter will not alter the values of a flat patch, we normalise its values so that they sum up to 1, by dividing each of them with the sum of all. The result is:

0.0000	0.0012	0.0037	0.0012	0.0000
0.0012	0.0366	0.1158	0.0366	0.0012
0.0037	0.1158	0.3662	0.1158	0.0037
0.0012	0.0366	0.1158	0.0366	0.0012
0.0000	0.0012	0.0037	0.0012	0.0000

0.0000	0.0002	0.0010	0.0016	0.0010	0.0002	0.0000
0.0002	0.0027	0.0126	0.0210	0.0126	0.0027	0.0002
0.0010	0.0126	0.0586	0.0977	0.0586	0.0126	0.0010
0.0016	0.0210	0.0977	0.1629	0.0977	0.0210	0.0016
0.0010	0.0126	0.0586	0.0977	0.0586	0.0126	0.0010
0.0002	0.0027	0.0126	0.0210	0.0126	0.0027	0.0002
0.0000	0.0002	0.0010	0.0016	0.0010	0.0002	0.0000

0.0000	0.0001	0.0003	0.0007	0.0009	0.0007	0.0003	0.0001	0.0000
0.0001	0.0005	0.0022	0.0052	0.0069	0.0052	0.0022	0.0005	0.0001
0.0003	0.0022	0.0092	0.0217	0.0290	0.0217	0.0092	0.0022	0.0003
0.0007	0.0052	0.0217	0.0516	0.0688	0.0516	0.0217	0.0052	0.0007
0.0009	0.0069	0.0290	0.0688	0.0917	0.0688	0.0290	0.0069	0.0009
0.0007	0.0052	0.0217	0.0516	0.0688	0.0516	0.0217	0.0052	0.0007
0.0003	0.0022	0.0092	0.0217	0.0290	0.0217	0.0092	0.0022	0.0003
0.0001	0.0005	0.0022	0.0052	0.0069	0.0052	0.0022	0.0005	0.0001
0.0000	0.0001	0.0003	0.0007	0.0009	0.0007	0.0003	0.0001	0.0000

Example 4.17

The image “Fun Fair” of figure 4.13 is corrupted with Gaussian noise. Reduce its noise with the help of the 5×5 filter produced in example 4.16.

First we create an empty grid the same size as the original image. To avoid boundary effects, we do not process a stripe of 2-pixels wide all around the input image. These pixels retain their original value in the output image. For processing all other pixels, we place the filter with its centre coinciding with the pixel, the value of which is to be recalculated, and multiply the filter value with the corresponding pixel value under it, sum up the 25 products and assign the result as the new value of the central pixel in the output image. The window is shifted by one pixel in both directions until all pixels are processed. This process is shown schematically in figure 4.14. In relation to this

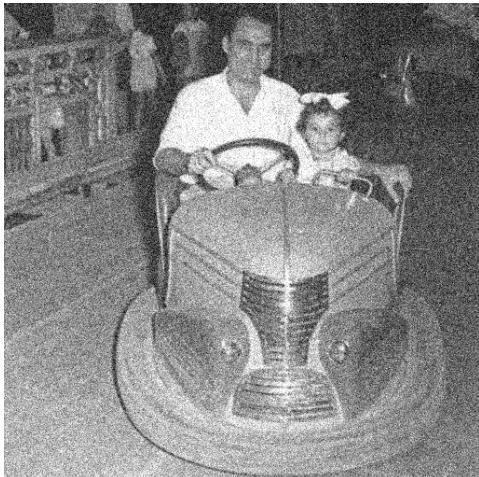
figure, the output values at the central pixels of the two positions of the window shown in (a) are given by:

$$\begin{aligned}
 o_{22} = & g_{00}f_{-2,-2} + g_{01}f_{-2,-1} + g_{02}f_{-2,0} + g_{03}f_{-2,1} + g_{04}f_{-2,2} \\
 & + g_{10}f_{-1,-2} + g_{11}f_{-1,-1} + g_{12}f_{-1,0} + g_{13}f_{-1,1} + g_{14}f_{-1,2} \\
 & + g_{20}f_{0,-2} + g_{21}f_{0,-1} + g_{22}f_{0,0} + g_{23}f_{0,1} + g_{24}f_{0,2} \\
 & + g_{30}f_{1,-2} + g_{31}f_{1,-1} + g_{32}f_{1,0} + g_{33}f_{1,1} + g_{34}f_{1,2} \\
 & + g_{40}f_{2,-2} + g_{41}f_{2,-1} + g_{42}f_{2,0} + g_{43}f_{2,1} + g_{44}f_{2,2}
 \end{aligned} \tag{4.86}$$

And:

$$\begin{aligned}
 o_{43} = & g_{21}f_{-2,-2} + g_{22}f_{-2,-1} + g_{23}f_{-2,0} + g_{24}f_{-2,1} + g_{25}f_{-2,2} \\
 & + g_{31}f_{-1,-2} + g_{32}f_{-1,-1} + g_{33}f_{-1,0} + g_{34}f_{-1,1} + g_{35}f_{-1,2} \\
 & + g_{41}f_{0,-2} + g_{42}f_{0,-1} + g_{43}f_{0,0} + g_{44}f_{0,1} + g_{45}f_{0,2} \\
 & + g_{51}f_{1,-2} + g_{52}f_{1,-1} + g_{53}f_{1,0} + g_{54}f_{1,1} + g_{55}f_{1,2} \\
 & + g_{61}f_{2,-2} + g_{62}f_{2,-1} + g_{63}f_{2,0} + g_{64}f_{2,1} + g_{65}f_{2,2}
 \end{aligned} \tag{4.87}$$

The result of applying this process to the image of figure 4.13a is shown in 4.13b.



(a) Image with Gaussian noise



(b) Image after Gaussian filtering

Figure 4.13: Gaussian filtering applied to remove Gaussian noise from an image.

Figure 4.14 consists of three tables labeled (a), (b), and (c). Table (a) shows a 5x5 grid of input values g_{ij} . Table (b) shows a 5x5 grid of filter values f_{ij} . Table (c) shows the result of applying the filter to the input, with crosses marking unreliable pixels.

g_{00}	g_{01}	g_{02}	g_{03}	g_{04}	g_{05}	g_{06}	g_{07}	g_{08}	
g_{10}	g_{11}	g_{12}	g_{13}	g_{14}	g_{15}	g_{16}	g_{17}	g_{18}	
g_{20}	g_{21}	g_{22}	g_{23}	g_{24}	g_{25}	g_{26}	g_{27}	g_{28}	
g_{30}	g_{31}	g_{32}	g_{33}	g_{34}	g_{35}	g_{36}	g_{37}	g_{38}	
g_{40}	g_{41}	g_{42}	g_{43}	g_{44}	g_{45}	g_{46}	g_{47}	g_{48}	
g_{50}	g_{51}	g_{52}	g_{53}	g_{54}	g_{55}	g_{56}	g_{57}	g_{58}	
g_{60}	g_{61}	g_{62}	g_{63}	g_{64}	g_{65}	g_{66}	g_{67}	g_{68}	
g_{70}	g_{71}	g_{72}	g_{73}	g_{74}	g_{75}	g_{76}	g_{77}	g_{78}	
g_{80}	g_{81}	g_{82}	g_{83}	g_{84}	g_{85}	g_{86}	g_{87}	g_{88}	

$f_{-2,-2}$	$f_{-2,-1}$	$f_{-2,0}$	$f_{-2,1}$	$f_{-2,2}$
$f_{-1,-2}$	$f_{-1,-1}$	$f_{-1,0}$	$f_{-1,1}$	$f_{-1,2}$
$f_{0,-2}$	$f_{0,-1}$	$f_{0,0}$	$f_{0,1}$	$f_{0,2}$
$f_{1,-2}$	$f_{1,-1}$	$f_{1,0}$	$f_{1,1}$	$f_{1,2}$
$f_{2,-2}$	$f_{2,-1}$	$f_{2,0}$	$f_{2,1}$	$f_{2,2}$

(a)

o_{22}	o_{23}	o_{24}	o_{25}	o_{26}	o_{27}	o_{28}	
o_{32}	o_{33}	o_{34}	o_{35}	o_{36}	o_{37}	o_{38}	
o_{42}	o_{43}	o_{44}	o_{45}	o_{46}	o_{47}	o_{48}	
o_{52}	o_{53}	o_{54}	o_{55}	o_{56}	o_{57}	o_{58}	
o_{62}	o_{63}	o_{64}	o_{65}	o_{66}	o_{67}	o_{68}	
o_{72}	o_{73}	o_{74}	o_{75}	o_{76}	o_{77}	o_{78}	
o_{82}	o_{83}	o_{84}	o_{85}	o_{86}	o_{87}	o_{88}	

(b)

(c)

Figure 4.14: (a) The input image with grey values g_{ij} . (b) The 5×5 smoothing filter with values f_{ij} . (c) The result of processing the input image with the filter. The pixels marked with crosses have unreliable values as their values are often chosen arbitrarily. For example, they could be set identical to the input values, or they might be calculated by assuming that those pixels have full neighbourhoods with the missing neighbours having value 0, or value as if the image were repeated in all directions.

Can we have weighted median and mode filters like we have weighted mean filters?

Yes. The weights of a median or a mode indicate how many times the corresponding number should be repeated. Figure 4.15 shows an image with impulse noise added to it, and two versions of improving it by unweighted and by weighted median filtering. The weights used are given in table 4.1.

0	1	1	1	0
1	2	2	2	1
1	2	4	2	1
1	2	2	2	1
0	1	1	1	0

Table 4.1: Weights that might be used in conjunction with a median or a mode filter.



(a) Image with impulse noise



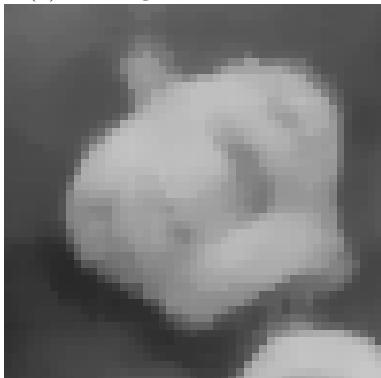
(b) Image detail with no noise



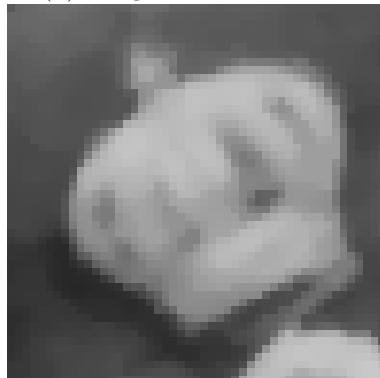
(c) Unweighted median filter



(d) Weighted median filter



(e) Detail of (c)



(f) Detail of (d)

Figure 4.15: Median filtering applied to the “Officer” with impulse noise (where 10% of the pixels are set to grey level 255). The weighted version produces better results, as it may be judged by looking at some image detail and comparing it with the original shown in (b).

Example 4.18

The sequence of 25 numbers of example 4.15 was created by reading sequentially the grey values of an image in a 5×5 window. You are asked to compute the weighted median of that image, using the weights of table 4.1. What value will this filter give for the particular window of this example?

First we write the grey values of example 4.15 in their original spatial arrangement:

15	17	15	17	16
10	8	9	18	15
16	12	14	11	15
14	15	18	100	15
14	13	12	12	17

Then we use the weights to repeat each entry of the above table the corresponding number of times and thus create the sequence of numbers that we shall have to rank: 17, 15, 17, 10, 8, 8, 9, 9, 18, 18, 15, 16, 12, 12, 14, 14, 14, 14, 11, 11, 15, 14, 15, 15, 18, 18, 100, 100, 15, 13, 12, 12.

Ranking these numbers in increasing order yields:

8, 8, 9, 9, 10, 11, 11, 12, 12, 12, 13, 14, 14, 14, 14, 14, 15, 15, 15, 15, 15, 15, 16, 17, 17, 18, 18, 18, 100, 100.

There are 32 numbers in this sequence and the median is between the 16th and the 17th number. Both these numbers are equal to 14, so the median is 14. (If the numbers were different their average rounded to the nearest integer would have been considered.)

The most occurring number is 15, so the output of the mode filter would be 15.

Can we filter an image by using the linear methods we learnt in Chapter 2?

Yes, often low, band and high pass image filtering is done by convolving the image with a suitable filter. This is the reason we prefer the filters to be finite in extent: finite convolution filters may be implemented as matrix operators applied to the image.

Example 4.19

You have a 3×3 image which may be represented by a 9×1 vector. Derive a matrix which, when it operates on this image, smooths its columns by averaging every three successive pixels, giving them weights $\frac{1}{4}, \frac{1}{2}, \frac{1}{4}$, and assigning the result to the central pixel. To deal with the border pixels, assume that the image is repeated periodically in all directions.

Let us say that the original image is

$$\begin{pmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{pmatrix} \quad (4.88)$$

and its smoothed version is:

$$\begin{pmatrix} \tilde{g}_{11} & \tilde{g}_{12} & \tilde{g}_{13} \\ \tilde{g}_{21} & \tilde{g}_{22} & \tilde{g}_{23} \\ \tilde{g}_{31} & \tilde{g}_{32} & \tilde{g}_{33} \end{pmatrix} \quad (4.89)$$

Let us also say that the smoothing matrix we wish to identify is A , with elements a_{ij} :

$$\begin{pmatrix} \tilde{g}_{11} \\ \tilde{g}_{21} \\ \tilde{g}_{31} \\ \tilde{g}_{12} \\ \tilde{g}_{22} \\ \tilde{g}_{32} \\ \tilde{g}_{13} \\ \tilde{g}_{23} \\ \tilde{g}_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{19} \\ a_{21} & a_{22} & \dots & a_{29} \\ a_{31} & a_{32} & \dots & a_{39} \\ a_{41} & a_{42} & \dots & a_{49} \\ a_{51} & a_{52} & \dots & a_{59} \\ a_{61} & a_{62} & \dots & a_{69} \\ a_{71} & a_{72} & \dots & a_{79} \\ a_{81} & a_{82} & \dots & a_{89} \\ a_{91} & a_{92} & \dots & a_{99} \end{pmatrix} \begin{pmatrix} g_{11} \\ g_{21} \\ g_{31} \\ g_{12} \\ g_{22} \\ g_{32} \\ g_{13} \\ g_{23} \\ g_{33} \end{pmatrix} \quad (4.90)$$

From the above equation we have:

$$\begin{aligned} \tilde{g}_{11} = & a_{11}g_{11} + a_{12}g_{21} + a_{13}g_{31} + a_{14}g_{12} + a_{15}g_{22} \\ & + a_{16}g_{32} + a_{17}g_{13} + a_{18}g_{23} + a_{19}g_{33} \end{aligned} \quad (4.91)$$

From the definition of the smoothing mask, we have:

$$\tilde{g}_{11} = \frac{1}{4}g_{31} + \frac{1}{2}g_{11} + \frac{1}{4}g_{21} \quad (4.92)$$

Comparison of equations (4.91) and (4.92) shows that we must set:

$$a_{11} = \frac{1}{2}, a_{12} = \frac{1}{4}, a_{13} = \frac{1}{4}, a_{14} = a_{15} = \dots = a_{19} = 0 \quad (4.93)$$

Working in a similar way for a few more elements, we can see that the matrix we wish to identify has the form:

$$A = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \end{pmatrix} \quad (4.94)$$

How do we deal with mixed noise in images?

If an image is affected by additive Gaussian as well as impulse noise, then we may use the **α -trimmed filter**: after we rank the grey values inside the smoothing window, we may keep only the $N(1 - \alpha)$ values that are closest to the median value. We may then compute only from them the mean value that we shall assign to the central pixel of the window.

Can we avoid blurring the image when we are smoothing it?

Yes. Some methods we may use are:

- (i) **edge adaptive smoothing**;
- (ii) **mean shift smoothing**;
- (iii) **anisotropic diffusion**.

What is the edge adaptive smoothing?

When we smooth an image, we place a window around a pixel, compute an average value inside the window and assign it to the central pixel. If the window we use happens to span two different regions, the boundary between the two regions will be blurred. In edge preserving smoothing, we place several windows around the pixel, having the pixel in all possible relative positions with respect to the window centre. Inside each window we compute the variance of the pixel values. We select the window with the minimum variance. We compute the average (weighted or not) of the pixels inside that window and assign that value to the pixel under consideration. Figure 4.16 shows the example of an image that has a diagonal edge. In this example, window C is expected to be the most homogeneous of all windows to which the pixel identified with the cross belongs. (Window C will have the least variance.) Then the new value for the marked pixel will be computed from the values inside this window.

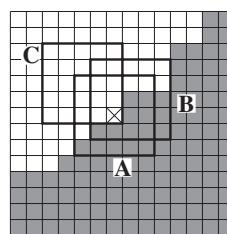


Figure 4.16: This grid represents a 14×14 image with an edge. The cross identifies the pixel for which we have to compute a new value. Let us say that the new value is computed inside a 5×5 window. Conventionally, we would use window A. However, in edge preserving smoothing, we may consider 25 windows of size 5×5 , all of which contain the pixel, but in different locations in relation to the centre of the window. Two of those windows are shown here, identified as windows B and C.

Figure 4.17 shows some noisy images, their smoothed versions with a flat or a Gaussian filter and their smoothed versions with an edge preserving flat or Gaussian filter. Box 4.6 shows how to compute the local variance in an efficient way.

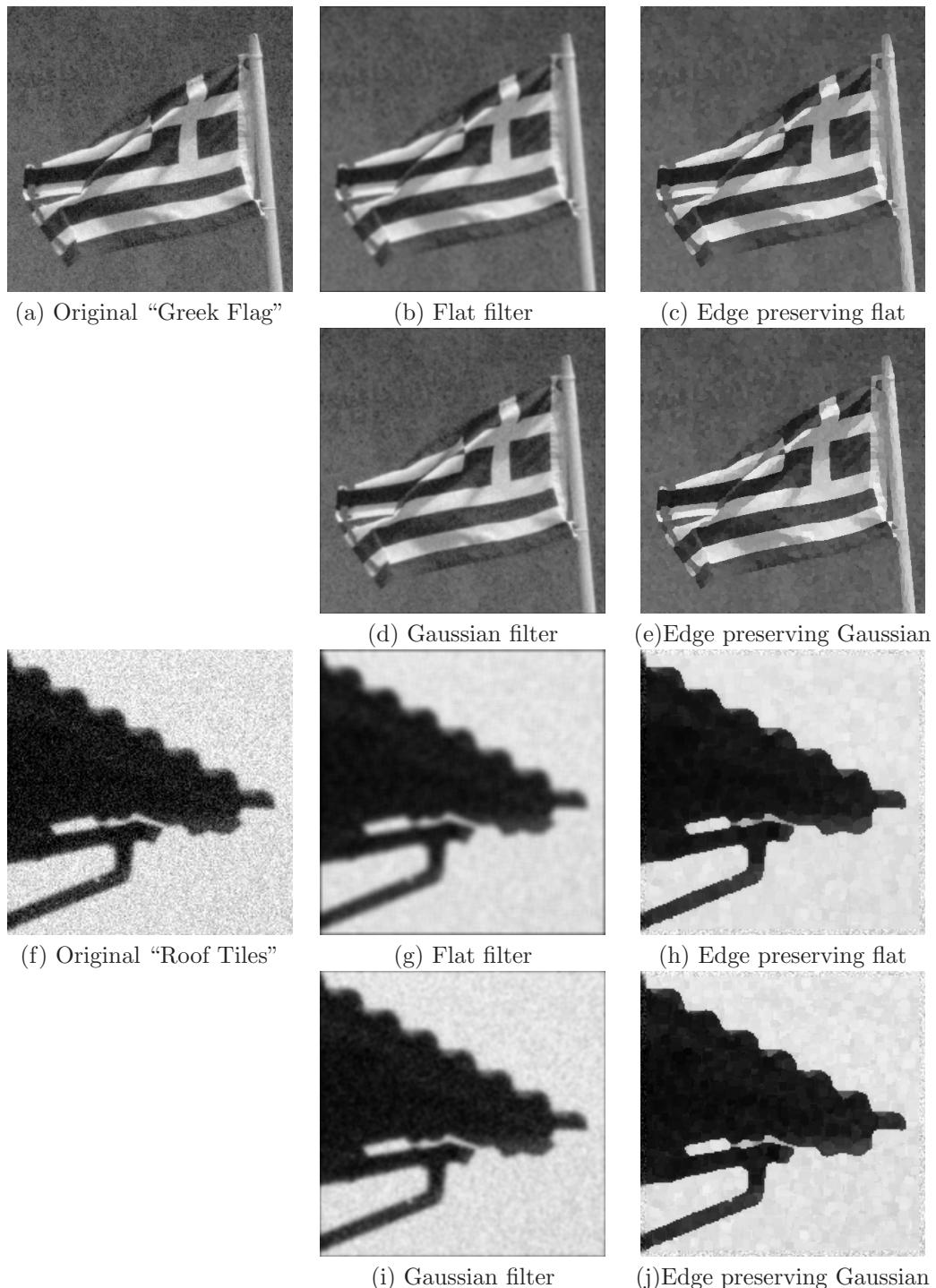


Figure 4.17: Noisy images and their smoothed versions by a 5×5 averaging window placed around each pixel or with its position selected according to the local variance.

Box 4.6. Efficient computation of the local variance

Let us say that we wish to compute the variance of numbers x_i , where $i = 1, 2, \dots, N$. Let us denote by $\langle \rangle$ the averaging operator. The mean of these numbers then is $\mu \equiv \langle x_i \rangle$. The variance is:

$$\begin{aligned}\sigma^2 &\equiv \langle (x_i - \mu)^2 \rangle \\ &= \langle x_i^2 \rangle + \mu^2 - 2\mu \langle x_i \rangle \\ &= \langle x_i^2 \rangle + \mu^2 - 2\mu \langle x_i \rangle \\ &= \langle x_i^2 \rangle + \mu^2 - 2\mu^2 \\ &= \langle x_i^2 \rangle - \langle x_i \rangle^2\end{aligned}\tag{4.95}$$

To select then the window with the least variance that contains each pixel, we use the following algorithm.

Step 1: Convolve the original image I with a flat averaging window of the dimensions you have preselected, say 5×5 . Call the output array A .

Step 2: Square the elements of array A . Call the output array B .

Step 3: Construct an array the same size as the input image where the value of each pixel is squared. Call this array C .

Step 4: Convolve array C with a flat averaging window of the dimensions you have preselected, say 5×5 . Call this array D .

Step 5: Subtract array B from array D . Call this array E .

Step 6: When you want to select a new value for pixel (i, j) , consider all pixels inside a window of the preselected size, say 5×5 , centred at (i, j) , and identify the pixel with the smallest value in array E . Use that pixel as the centre of the window from which you will compute the new value of pixel (i, j) , from the values of the original image I .

How does the mean shift algorithm work?

According to this algorithm, a pixel is represented by a triplet in a 3D space where the two dimensions represent the position of the pixel in the image and the third dimension is used to measure its brightness. A pixel (i, j) then in this space is represented by point (x_{ij}, y_{ij}, g_{ij}) , where to begin with, $x_{ij} = i$, $y_{ij} = j$ and g_{ij} is its grey value. The pixels in this 3D space are allowed to move and create agglomerations. The movement of the pixels happens iteratively, where at each iteration step a new vector (x_{ij}, y_{ij}, g_{ij}) is computed for each pixel (i, j) . At the $m + 1$ iteration, this new vector is given by

$$\begin{aligned}x_{ij}^{m+1} &= \frac{\sum_{(k,l) \in \mathcal{N}_{ij}} x_{kl}^m g\left(\frac{(x_{ij}^m - x_{kl}^m)^2}{h_x^2}\right) g\left(\frac{(y_{ij}^m - y_{kl}^m)^2}{h_y^2}\right) g\left(\frac{(g_{ij}^m - g_{kl}^m)^2}{h_g^2}\right)}{\sum_{(k,l) \in \mathcal{N}_{ij}} g\left(\frac{(x_{ij}^m - x_{kl}^m)^2}{h_x^2}\right) g\left(\frac{(y_{ij}^m - y_{kl}^m)^2}{h_y^2}\right) g\left(\frac{(g_{ij}^m - g_{kl}^m)^2}{h_g^2}\right)} \\ y_{ij}^{m+1} &= \frac{\sum_{(k,l) \in \mathcal{N}_{ij}} y_{kl}^m g\left(\frac{(x_{ij}^m - x_{kl}^m)^2}{h_x^2}\right) g\left(\frac{(y_{ij}^m - y_{kl}^m)^2}{h_y^2}\right) g\left(\frac{(g_{ij}^m - g_{kl}^m)^2}{h_g^2}\right)}{\sum_{(k,l) \in \mathcal{N}_{ij}} g\left(\frac{(x_{ij}^m - x_{kl}^m)^2}{h_x^2}\right) g\left(\frac{(y_{ij}^m - y_{kl}^m)^2}{h_y^2}\right) g\left(\frac{(g_{ij}^m - g_{kl}^m)^2}{h_g^2}\right)}\end{aligned}\tag{4.96}$$

$$g_{ij}^{m+1} = \frac{\sum_{(k,l) \in \mathcal{N}_{ij}} g_{kl}^m g\left(\frac{(x_{ij}^m - x_{kl}^m)^2}{h_x^2}\right) g\left(\frac{(y_{ij}^m - y_{kl}^m)^2}{h_y^2}\right) g\left(\frac{(g_{ij}^m - g_{kl}^m)^2}{h_g^2}\right)}{\sum_{(k,l) \in \mathcal{N}_{ij}} g\left(\frac{(x_{ij}^m - x_{kl}^m)^2}{h_x^2}\right) g\left(\frac{(y_{ij}^m - y_{kl}^m)^2}{h_y^2}\right) g\left(\frac{(g_{ij}^m - g_{kl}^m)^2}{h_g^2}\right)} \quad (4.97)$$

where h_x , h_y and h_g are appropriately chosen scaling constants, \mathcal{N}_{ij} is a neighbourhood of pixel (i,j) , defined as a 3D sphere using the Euclidean metric, and

$$g(x) \equiv e^{-x} \quad \text{or} \quad g(x) \equiv \begin{cases} 1 & \text{for } |x| \leq w \\ 0 & \text{otherwise} \end{cases} \quad (4.98)$$

with w being a parameter specifying the size of the flat kernel. The iterations may be repeated for a prespecified number of times. At the end of the last iteration, pixel (i,j) takes, as grey value, the value $g_{ij}^{m_{final}}$, rounded to the nearest integer.

Figures 4.18 and 4.19 show the results of the mean shift algorithm after it was applied for a few iterations to some noisy images. The algorithm was run with $h_x = h_y = 15$ and $h_g = 25.5$ (that is $h_g = 0.1$ for grey values scaled in the range [0,1]). Neighbourhood \mathcal{N}_{ij} was the full image. This algorithm converges only when all pixels have the same values.

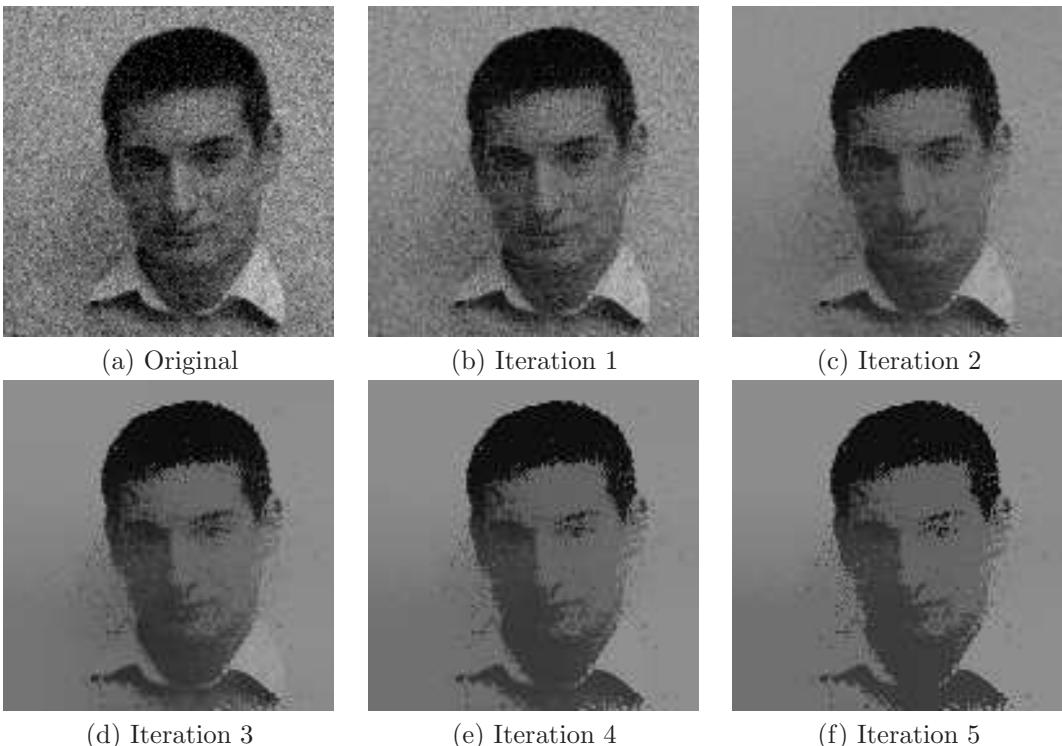


Figure 4.18: Noisy “Leonidas” (128×128 in size) and its smoothed versions by the mean shift algorithm, after a few iterations. As the iterations progress, the noise reduces, but also significant image details are lost, as small regions are incorporated in larger neighbouring regions.

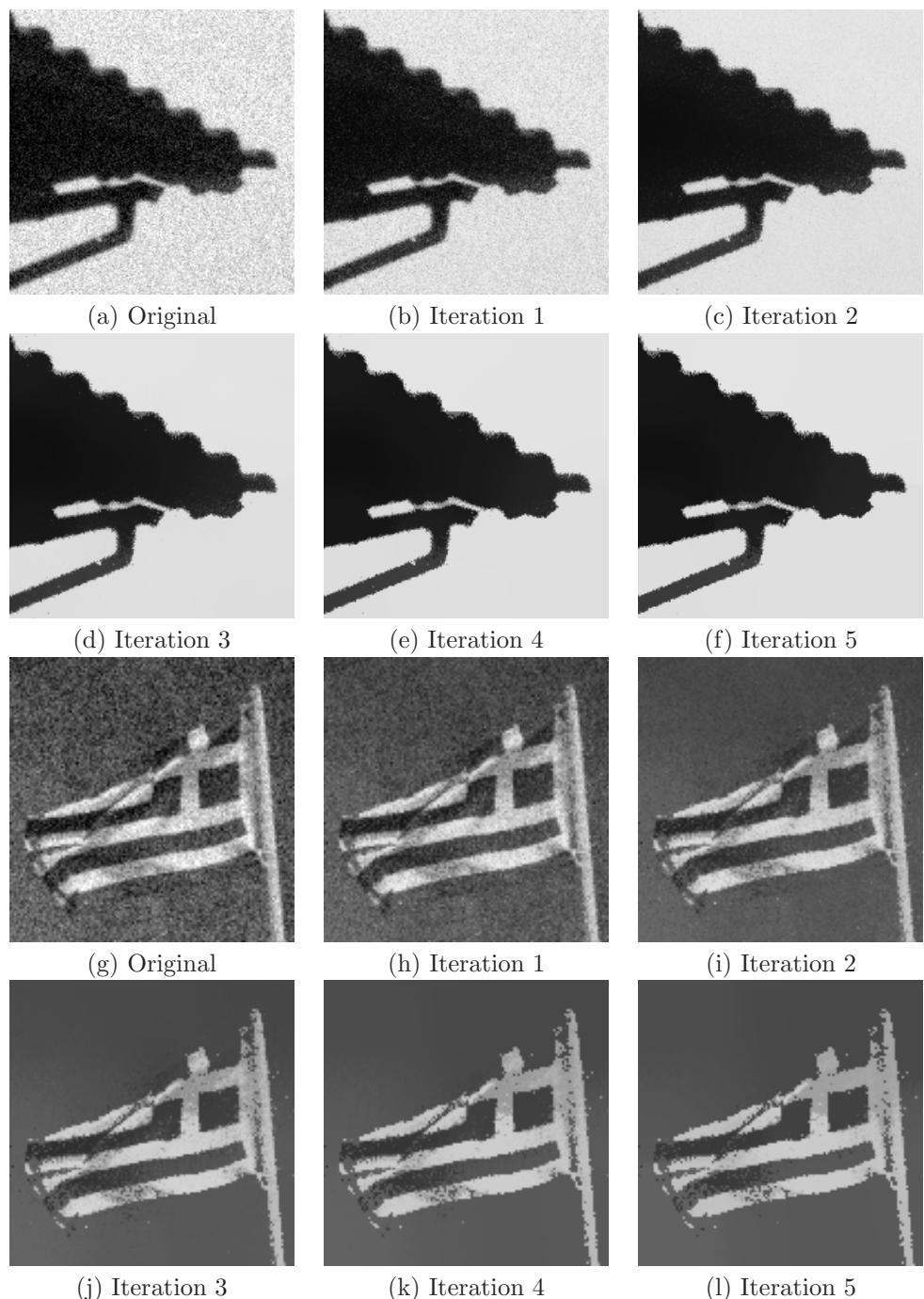


Figure 4.19: Noisy images (128×128 in size) and their smoothed versions by the mean shift algorithm.

What is anisotropic diffusion?

It is an algorithm that generalises Gaussian filtering, used to reduce additive Gaussian noise (see Box 4.7), to make it adaptive to local image gradient (see Box 4.10), so that edges are preserved.

Box 4.7. Scale space and the heat equation

Let us imagine that we smooth an image with low pass Gaussian filters of increasing standard deviation σ . This way we create a stack of images, of progressively lower and lower detail. We may view this stack of images as the image representation in a 3D space, where the two axes are the (x, y) image axes and the third axis is standard deviation σ , which in this context is referred to as “scale”. This 3D space is known as **scale space**. Figure 4.20 shows an example image and some of its smoothed versions with filters of increasing scale. We can see that as the scale increases more and more image features disappear, while the features that survive are the most prominent image features, but appear very blurred. The places, where the borders of the distinct image regions meet, become progressively blurred and the gradient magnitude that measures the contrast of the image in those places gradually diffuses, so only the borders with the strongest contrast survive albeit in a blurred way. Figure 4.21 shows the corresponding gradient magnitude images to those shown in figure 4.20.

This diffusion of image information observed in scale space may be also seen in a cross section of the stack of images created, as shown in figure 4.22. We can see how the grey value from a location diffuses to neighbouring locations as the scale increases. It can be shown (see example 4.21) that this diffusion of grey value from one pixel to other pixels can be modelled by the heat diffusion equation

$$\frac{\partial I(x, y; \sigma)}{\partial \sigma} = \sigma \left[\frac{\partial^2 I(x, y; \sigma)}{\partial x^2} + \frac{\partial^2 I(x, y; \sigma)}{\partial y^2} \right] \quad (4.99)$$

where $I(x, y; \sigma)$ is the image seen as a function defined in the 3D scale space. The bracketed expression on the right-hand side of (4.99) is known as the **Laplacian** of the image, sometimes denoted as ΔI or $\nabla^2 I$. Equation (4.99) may, therefore, be written in all the following equivalent forms (see also Box 4.8):

$$\begin{aligned} \frac{\partial I(x, y; \sigma)}{\partial \sigma} &= \sigma \Delta I(x, y; \sigma) \\ &= \sigma \nabla^2 I(x, y; \sigma) \\ &= \sigma \operatorname{div}(\operatorname{grad} I(x, y; \sigma)) \\ &= \sigma \nabla \cdot \nabla I(x, y; \sigma) \end{aligned} \quad (4.100)$$

In Physics, σ corresponds to time and the image grey value to temperature.



(a) "Father and daughters"

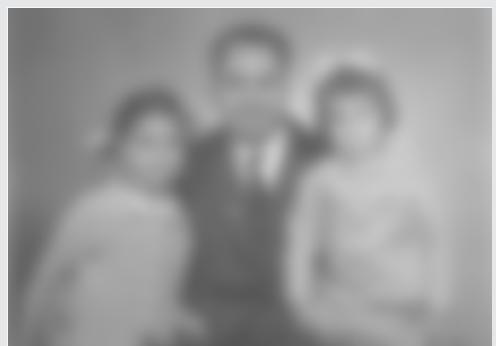
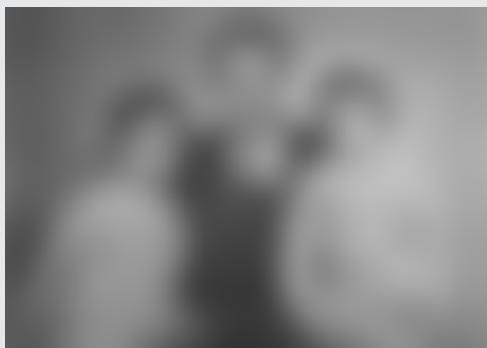
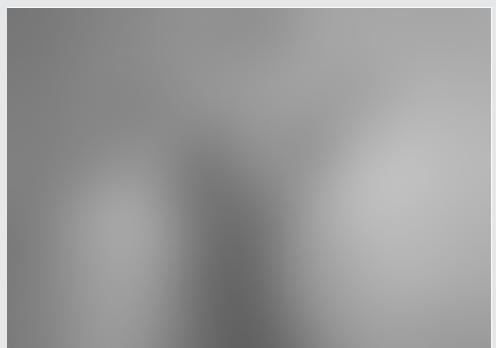
(b) $\sigma = 1.98, M = 6$ (c) $\sigma = 4.28, M = 13$ (d) $\sigma = 8.24, M = 25$ (e) $\sigma = 16.48, M = 50$ (f) $\sigma = 32.95, M = 100$

Figure 4.20: As the scale of the filter with which we smooth an image increases, less and less information survives. The size of the image is 362×512 . The filters used were designed to have size $(2M+1) \times (2M+1)$ with discontinuity $\epsilon = 0.01$, using the method described in example 4.16, on page 329.



(a) "Father and daughters"

(b) $\sigma = 1.98$ (c) $\sigma = 4.28$ (d) $\sigma = 8.24$ (e) $\sigma = 16.48$ (f) $\sigma = 32.95$

Figure 4.21: The gradient magnitude images of figure 4.20, each one individually scaled to the range [0, 255]. As the scale of the filter, with which we smooth the image, increases, the transition between regions becomes less sharp and this manifests itself in very broad stripes of large gradient magnitude.



Figure 4.22: Along the vertical axis we measure scale σ , which increases from $\sigma = 0$ (original image, bottom) to $\sigma = 32.95$ (top), in 101 steps, inclusive. The horizontal axis is the x axis of the image. Left: a cross section of the 3D image representation in scale space. Right: a cross section of the corresponding gradient magnitude images. Note how the strength of the gradient magnitude weakens as the edges diffuse. That is why we had to scale each gradient image individually to be able to visualise it in figure 4.21.

Box 4.8. Gradient, Divergence and Laplacian

The **gradient** of a function $f(x, y)$ is a vector denoted and defined as:

$$\text{grad}(f(x, y)) \equiv \nabla f(x, y) \equiv \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right)^T \quad (4.101)$$

The gradient vector of a function identifies the direction of maximum change of the function.

The **divergence** of a vector $\mathbf{u} \equiv (u_x, u_y)$ is a function, defined and denoted as:

$$\text{div}(\mathbf{u}) \equiv \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \quad (4.102)$$

If the vector is thought of as a velocity vector, its divergence measures the total “flow” away from the point of its definition.

The **Laplacian** of a function $f(x, y)$ is the divergence of its gradient vector:

$$\begin{aligned} \Delta f(x, y) &\equiv \text{div}(\text{grad}(f(x, y))) \\ &= \nabla \cdot \nabla f(x, y) \\ &= \nabla^2 f(x, y) \\ &= \nabla \cdot \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right)^T \\ &= \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2} \end{aligned} \quad (4.103)$$

Thus, the Laplacian of a function is equal to the sum of its second derivatives. These formulae generalise trivially to higher dimensions.

Example B4.20

Show that for function $g(x, y; \sigma) = e^{-(x^2+y^2)/(2\sigma^2)} / (2\pi\sigma^2)$, the following is correct:

$$\frac{1}{\sigma} \frac{\partial g(x, y; \sigma)}{\partial \sigma} = \frac{\partial^2 g(x, y; \sigma)}{\partial x^2} + \frac{\partial^2 g(x, y; \sigma)}{\partial y^2} \quad (4.104)$$

We start by computing the derivative on the left-hand side of (4.104):

$$\begin{aligned} \frac{\partial g(x, y; \sigma)}{\partial \sigma} &= -\frac{2}{2\pi\sigma^3} e^{-\frac{x^2+y^2}{2\sigma^2}} + \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \frac{2x^2+2y^2}{2\sigma^3} \\ &= \frac{1}{2\pi} \left(\frac{x^2+y^2}{\sigma^5} - \frac{2}{\sigma^3} \right) e^{-\frac{x^2+y^2}{2\sigma^2}} \end{aligned} \quad (4.105)$$

Let us then compute the first derivative of $g(x, y; \sigma)$ with respect to x :

$$\begin{aligned} \frac{\partial g(x, y; \sigma)}{\partial x} &= \frac{1}{2\pi\sigma^2} \frac{-2x}{2\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \\ &= -\frac{x}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \end{aligned} \quad (4.106)$$

The second derivative with respect to x is:

$$\frac{\partial^2 g(x, y; \sigma)}{\partial x^2} = -\frac{1}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} + \frac{x^2}{2\pi\sigma^6} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4.107)$$

In a similar way, the second derivative with respect to y is worked out to be:

$$\frac{\partial^2 g(x, y; \sigma)}{\partial y^2} = -\frac{1}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} + \frac{y^2}{2\pi\sigma^6} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4.108)$$

Combining (4.107) and (4.108), we can work out the right-hand side of (4.104):

$$\begin{aligned} \frac{\partial^2 g(x, y; \sigma)}{\partial x^2} + \frac{\partial^2 g(x, y; \sigma)}{\partial y^2} &= -\frac{2}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} + \frac{x^2+y^2}{2\pi\sigma^6} e^{-\frac{x^2+y^2}{2\sigma^2}} \\ &= \frac{1}{2\pi\sigma} \left(\frac{x^2+y^2}{2\pi\sigma^5} - \frac{2}{\sigma^3} \right) e^{-\frac{x^2+y^2}{2\sigma^2}} \end{aligned} \quad (4.109)$$

Upon comparison with (4.105), equation (4.104) follows.

Example B4.21

The embedding of an image into the 3D scale space is achieved by smoothing the image with a Gaussian low pass filter $g(x, y; \sigma)$ with increasing values of σ . Show that the change in the grey value of a pixel as σ changes is expressed by equation (4.99).

We may say that the value of pixel (x, y) , when the image has been smoothed with filter $g(x, y; \sigma)$, is given by the convolution integral:

$$\begin{aligned} I(x, y; \sigma) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(x - u, y - v; \sigma) I(u, v) dudv \\ &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(u, v; \sigma) I(x - u, y - v) dudv \end{aligned} \quad (4.110)$$

We may differentiate this expression with respect to σ to work out the change in the grey value of the pixel as σ changes. We need to apply Leibniz rule (see Box 4.9) twice, once considering the second integral as the integrand that depends on the parameter with respect to which we differentiate, and once in order to differentiate that integrand (which itself is an integral) with respect to the parameter. For the first application, $x = v$, $\lambda = \sigma$, $a(\lambda) = -\infty$, $b(\lambda) = +\infty$, $f(x; \lambda) = \int_{-\infty}^{+\infty} g(u, v; \sigma) I(x - u, y - v) du$. When applying the formula, we have to differentiate $f(x; \lambda)$ with respect to the parameter, which requires the second application of Leibniz rule, with $x = u$, $\lambda = \sigma$, $a(\lambda) = -\infty$, $b(\lambda) = +\infty$ and $f(x; \lambda) = g(u, v; \sigma) I(x - u, y - v)$ this time. The result is:

$$\frac{\partial I(x, y; \sigma)}{\partial \sigma} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \frac{\partial g(u, v; \sigma)}{\partial \sigma} I(x - u, y - v) dudv \quad (4.111)$$

If we make use of (4.104), we obtain:

$$\begin{aligned} \frac{\partial I(x, y; \sigma)}{\partial \sigma} &= \sigma \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \frac{\partial^2 g(u, v; \sigma)}{\partial u^2} I(x - u, y - v) dudv \\ &\quad + \sigma \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \frac{\partial^2 g(u, v; \sigma)}{\partial v^2} I(x - u, y - v) dudv \end{aligned} \quad (4.112)$$

We shall see in Chapter 6 (example 6.49, page 622) that convolution of the image with the second derivative of the Gaussian along the x direction yields an estimate of the second derivative of the image along the x axis, and convolution of the image with the second derivative of the Gaussian along the y direction, yields an estimate of the second derivative of the image along the y axis. Thus, equation (4.99) is valid.

Box 4.9. Differentiation of an integral with respect to a parameter

Assume that the definite integral $I(\lambda)$ depends on a parameter λ , as follows:

$$I(\lambda) = \int_{a(\lambda)}^{b(\lambda)} f(x; \lambda) dx \quad (4.113)$$

Its derivative with respect to λ is given by the following formula, known as the **Leibniz rule**:

$$\frac{dI(\lambda)}{d\lambda} = \frac{db(\lambda)}{d\lambda} f(b(\lambda); \lambda) - \frac{da(\lambda)}{d\lambda} f(a(\lambda); \lambda) + \int_{a(\lambda)}^{b(\lambda)} \frac{\partial f(x; \lambda)}{\partial \lambda} dx \quad (4.114)$$

Box 4.10. From the heat equation to the anisotropic diffusion algorithm

When we smooth an image I with a Gaussian filter of standard deviation σ , the value of a pixel diffuses according to equation (4.100), which may be rewritten in a more general form

$$\frac{\partial I(x, y; \sigma)}{\partial \sigma} = \text{div}(K \text{grad}(I(x, y; \sigma))) \quad (4.115)$$

where K is a constant that controls the rate of the diffusion. The form of this equation means that the grey value of a pixel diffuses isotropically in all directions. If we want to preserve image edges, we must modify this equation so that the grey value diffuses parallel to the edges, rather than orthogonal to them. We can identify the direction of change of the grey value of the image, at any point, by computing its gradient vector at that point (see Box 4.8). We would like the diffusion along that direction to be minimal, while the diffusion orthogonal to that direction to be maximal, as the orthogonal direction is parallel to the image edges. To achieve this, we may replace constant K in (4.115) with a function like $e^{-\frac{|\nabla I(x, y; \sigma)|}{b}}$. Note that when $|\nabla I(x, y; \sigma)| \gg b > 0$, the exponent is large and this function becomes very small. This happens along the direction of maximum change, ie orthogonal to the direction of an image edge. If $|\nabla I(x, y; \sigma)| \ll b$, $e^{-\frac{|\nabla I(x, y; \sigma)|}{b}}$ is large and the diffusion of the grey values along this direction is facilitated. This happens parallel to lines of constant grey value, ie parallel to image edges. Thus, the modified heat equation for anisotropic diffusion becomes:

$$\frac{\partial I(x, y; \sigma)}{\partial \sigma} = \text{div}\left(e^{-\frac{|\nabla I(x, y; \sigma)|}{b}} \text{grad}(I(x, y; \sigma))\right) \quad (4.116)$$

How do we perform anisotropic diffusion in practice?

Assume that $I(i, j)$ is the image we wish to process.

Step 0: Decide upon a value C , which indicates that the $C\%$ weakest gradient magnitude values are assumed to be due to noise and the rest due to genuine image discontinuities. This may be decided by looking at the histogram of the gradient magnitude values.

Step 1: At each image iteration, compute the gradient magnitude of the image pixels by using one of the filters for this purpose (discussed in Chapter 6, pages 596 and 608). Create the histogram of the values of the magnitude of all gradient vectors. Starting from the first bin, accumulate the entries of the successive bins until $C\%$ of pixels have been accounted for. The value of the last bin is noted as threshold B .

Step 2: For each image pixel (i, j) , compute the following quantities:

$$\begin{aligned}\delta_N(i, j) &\equiv I(i-1, j) - I(i, j) \\ \delta_S(i, j) &\equiv I(i+1, j) - I(i, j) \\ \delta_E(i, j) &\equiv I(i, j+1) - I(i, j) \\ \delta_W(i, j) &\equiv I(i, j-1) - I(i, j)\end{aligned}\tag{4.117}$$

Step 3: For each image pixel (i, j) , compute the following quantities

$$\begin{aligned}c_N(i, j) &\equiv g(\delta_N(i, j)) \\ c_S(i, j) &\equiv g(\delta_S(i, j)) \\ c_E(i, j) &\equiv g(\delta_E(i, j)) \\ c_W(i, j) &\equiv g(\delta_W(i, j))\end{aligned}\tag{4.118}$$

where

$$g(x) \equiv e^{-\left(\frac{x}{B}\right)^2}\tag{4.119}$$

Step 4: Update the value of pixel (i, j) using

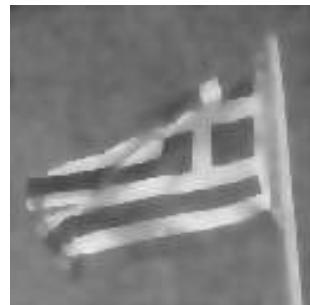
$$\begin{aligned}I(i, j)^{\text{new}} &= I(i, j)^{\text{old}} + \\ &\lambda [c_N(i, j)\delta_N(i, j) + c_S(i, j)\delta_S(i, j) + \\ &c_E(i, j)\delta_E(i, j) + c_W(i, j)\delta_W(i, j)]\end{aligned}\tag{4.120}$$

where $0 < \lambda \leq 0.25$.

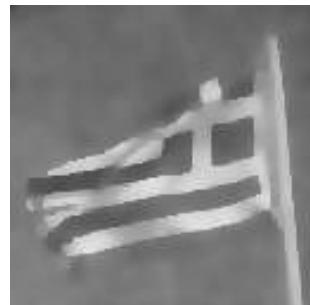
Figure 4.23 shows the results of applying this algorithm to some noisy images. The value of C was chosen to be 50 and $\lambda = 0.25$. Note that this algorithm does not converge, and so one has to run it for several iterations and assess the results as desired.



Three iterations



Seven iterations



Fourteen iterations



Twenty iterations

Figure 4.23: The noisy images of figures 4.18a, 4.19a and 4.19g, after 3, 7, 14 and 20 iterations of anisotropic diffusion. The weakest half of the gradient magnitudes were attributed to fluctuations in the grey value due to noise. So, $C = 50$ was used. Parameter λ in (4.120) was set to 0.25.

4.3 Reducing low frequency interference

When does low frequency interference arise?

Low frequency interference arises when the image has been captured under variable illumination. This is almost always true for indoor scenes, because of the inverse square law of light propagation: the parts of the imaged scene that are furthest away from the illuminating source receive much less light than those near the source. This is not true for outdoor scenes, under natural light, because the sun is so far away, that all points of an imaged scene may be considered at equal distance from it.

Can variable illumination manifest itself in high frequencies?

Yes. Shadows are a form of variable illumination. They may appear in both indoor and outdoor images. Parts of the scene in shadow do not receive light directly from the illuminating source, but indirectly via diffusion of the light by the surrounding objects. This is called **ambient light**. Indoor scenes suffer from both shadows and gradually varying illumination, while outdoor scenes suffer only from shadows. Shadows create sudden changes of brightness, which may be mistaken for real object boundaries. Their effect cannot be corrected by the methods discussed in this section. However, they may be taken care of by the locally adaptive methods discussed later on in this chapter.

In which other cases may we be interested in reducing low frequencies?

It is also possible that we may be interested in the small details of an image, or details that manifest themselves in high frequencies. The process of enhancing the high frequencies of an image is called **sharpening** and it may be achieved by **high pass linear filtering**. Small image details may also be enhanced by using nonlinear filters based on local image statistics.

What is the ideal high pass filter?

The ideal high pass filter is schematically shown in figure 4.24, in the frequency domain.

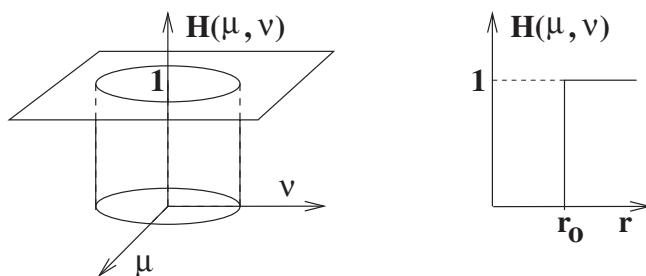


Figure 4.24: The spectrum of the ideal high pass filter is 1 everywhere, except inside a circle of radius r_0 in the frequency domain, where it is 0. On the right, a cross-section of such a filter. Here $r \equiv \sqrt{\mu^2 + \nu^2}$.

Filtering with such a filter in the frequency domain is equivalent to convolving in the real domain with the function that has this filter as its Fourier transform. There is no finite function which corresponds to the ideal high pass filter (see example 4.3, on page 299). So, often, high pass filters are defined in the real domain, for convenience of use rather than optimality in performance, just like we do for low pass filters. Convenient high pass filters, with good properties in the frequency domain, are the various derivatives of the Gaussian function, truncated and discretised (see example 4.17, on page 331). The first derivatives of the Gaussian function (4.84), on page 329, are:

$$g_x(x, y) \equiv xe^{-\frac{x^2+y^2}{2\sigma^2}} \quad g_y(x, y) \equiv ye^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4.121)$$

Note that constant factors in these definitions have been omitted as they are irrelevant, given that the weights of the truncated and discretised filters created from them will be normalised. These filters, used as convolution filters, will enhance the horizontal and vertical transitions of brightness in the image.

The second derivative based Gaussian filter, derived from (4.84), is

$$g_r \equiv \left(1 - \frac{r^2}{\sigma^2}\right) e^{-\frac{r^2}{2\sigma^2}} \quad (4.122)$$

where $r = \sqrt{x^2 + y^2}$. This function may be used to enhance spots and small blobs in the image.

Example 4.22

Apply to the image of figure 4.25 filters (4.121) and (4.122).



Figure 4.25: “A building in Ghent”. Size 256×323 pixels.

Let us consider that the filters we shall use will be $(2M + 1) \times (2M + 1)$ in size. When filter $g_x(x)$ is truncated, its value is $Me^{-\frac{M^2}{2\sigma^2}}$. We wish this value to be equal to ϵ . This way, we may work out the value of σ , given M :

$$\begin{aligned}\epsilon &= Me^{-\frac{M^2}{2\sigma^2}} \Rightarrow \\ \ln \frac{\epsilon}{M} &= -\frac{M^2}{2\sigma^2} \Rightarrow \\ \sigma &= \frac{M}{\sqrt{2(\ln M - \ln \epsilon)}}\end{aligned}\tag{4.123}$$

The values of filter g_x may be computed by allowing x to take values $-M, -M + 1, \dots, -1, 0, 1, 2, \dots, M - 1, M$. The values of g_x should sum up to 0 as this is a high pass filter, so a signal that consists of only a zero frequency component (ie a flat signal), should yield 0 as output. Further, if we wish to have control over the amount by which transitions in brightness are enhanced, we should make sure that all positive weights sum up to 1 and all negative weights sum up to -1, and multiply the whole filter with a factor A that allows us to control the level of enhancement. In general, the weights computed from continuous function $g_x(x)$ may not sum up to 0. If we divide the positive weights by their sum and the negative weights by the absolute value of their own sum, we ensure that both the above conditions are fulfilled. Using this methodology and for $\epsilon = 0.01$, and $M = 2$, $M = 3$ and $M = 4$, we constructed the following filters:

For $M = 2$ the filter is: $-0.01, -0.27, 0.00, 0.27, 0.01$

For $M = 3$ the filter is: $-0.01, -0.16, -0.53, 0.00, 0.53, 0.16, 0.01$

For $M = 4$ the filter is: $-0.01, -0.10, -0.45, -0.69, 0.00, 0.69, 0.45, 0.10, 0.01$

After normalising (so that the positive weights add up to 1, while the negative weights add up to -1), the filters are:

For $M = 2$ the filter is: $-0.04, -0.96, 0.00, 0.96, 0.04$

For $M = 3$ the filter is: $-0.01, -0.23, -0.76, 0.00, 0.76, 0.23, 0.01$

For $M = 4$ the filter is: $-0.01, -0.08, -0.36, -0.55, 0, 0.55, 0.36, 0.08, 0.01$

The above filters may be used on their own as 1D convolution filters, or they may be combined with the 1D version of the smoothing filter developed in example 4.17, applied in the orthogonal direction, to form 2D filters that smooth along one direction while enhancing the brightness transitions along the other. Note that filters g_x and g_y differ only in the direction along which they are applied. The 1D versions of smoothing filter (4.84), on page 329, are:

For $M = 2$: $0.006, 0.191, 0.605, 0.191, 0.006$

For $M = 3$: $0.004, 0.052, 0.242, 0.404, 0.242, 0.052, 0.004$

For $M = 4$: $0.003, 0.022, 0.096, 0.227, 0.303, 0.227, 0.096, 0.022, 0.003$

Figure 4.26 shows the output of applying the 1D version of filter (4.84) along the horizontal direction, followed by filter (4.121) applied along the vertical direction to image 4.25, for various values of M , in order to enhance its horizontal details. Notice that such a filter may create negative outputs, responding with an absolutely large, but negative number, to transitions in brightness from bright to dark and with a large positive number to transitions in brightness from dark to bright. To avoid discriminating

between these two types of transition, the absolute value of the filter output is taken. Then, in order to visualise the results, we use the histogram of the output values in order to select two thresholds: any value below the low threshold t_1 is set to 0, while any value above the high threshold t_2 is set to 255. The values in between are linearly mapped to the range [0, 255]:

$$g_{new} = \begin{cases} 0 & \text{if } g_{old} \leq t_1 \\ 255 & \text{if } g_{old} \geq t_2 \\ \left\lfloor \frac{g_{old}-t_1}{t_2-t_1} 255 + 0.5 \right\rfloor & \text{if } t_1 < g_{old} < t_2 \end{cases} \quad (4.124)$$

This type of stretching allows a much better visualisation of the results than straightforward mapping of the output values to the range [0, 255], as it allows us to remove the effect of outliers.

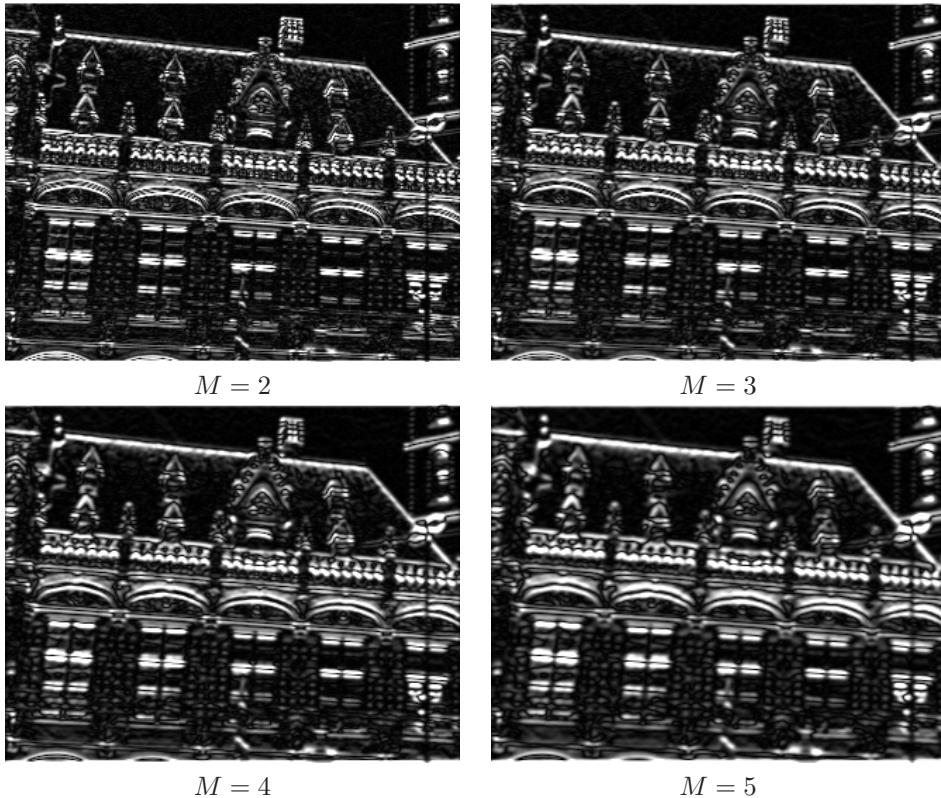


Figure 4.26: Enhancing the horizontal details of the building in Ghent, by low pass filtering along the horizontal direction and high pass filtering along the vertical one.

Figure 4.27 shows the output of applying filter (4.84) along the vertical direction, followed by filter (4.121) applied along the horizontal direction to image 4.25, for various values of M , in order to enhance its vertical details. As above, the absolute value of the filter output is considered and scaled for visualisation.

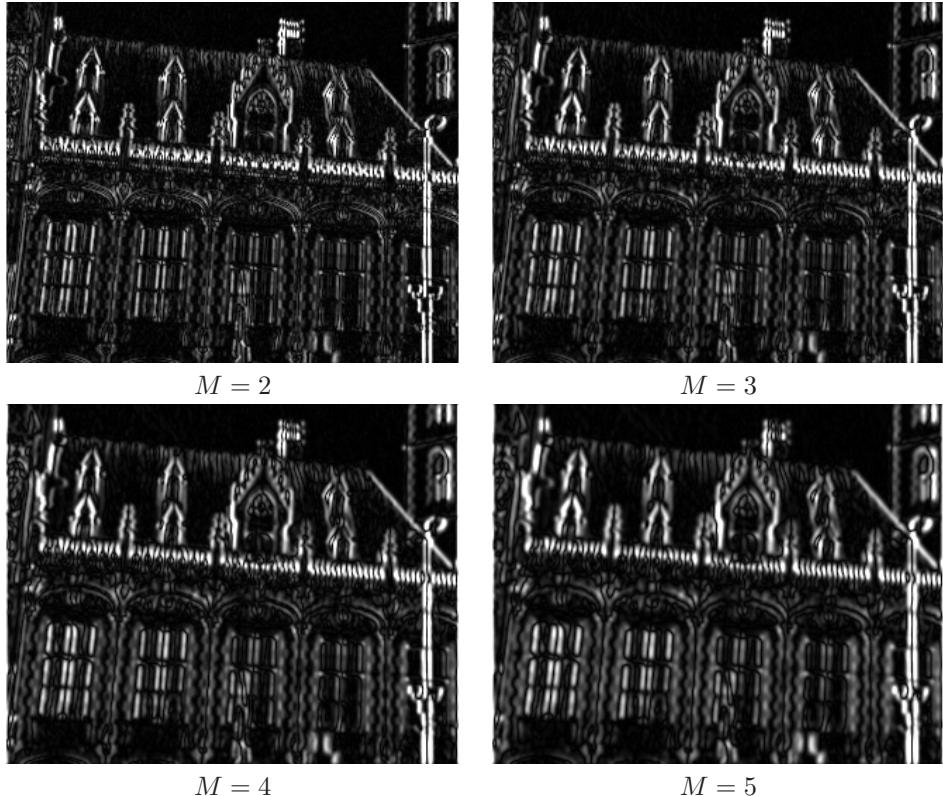


Figure 4.27: Enhancing the vertical details of the building in Ghent, by low pass filtering along the vertical direction and high pass filtering along the horizontal one.

To construct filter g_r we work as follows. Note that the value of σ for this filter determines the radius r at which its values change sign. So, when we select it, we must consider the size of the spots we wish to enhance. The example filters we present next have been computed by selecting $\sigma = M/2$ in (4.122) and allowing x and y to take values $0, \pm 1, \pm 2, \dots, \pm M$. The weights of this filter have to sum up to 0, so after we compute them, we find their sum Σ and we subtract from each weight $\Sigma/(2M+1)^2$. The filters that result in this way, for $M = 2$, $M = 3$ and $M = 4$, are:

0.0913	0.0755	0.0438	0.0755	0.0913
0.0755	-0.2122	-0.3468	-0.2122	0.0755
0.0438	-0.3468	1.0918	-0.3468	0.0438
0.0755	-0.2122	-0.3468	-0.2122	0.0755
0.0913	0.0755	0.0438	0.0755	0.0913

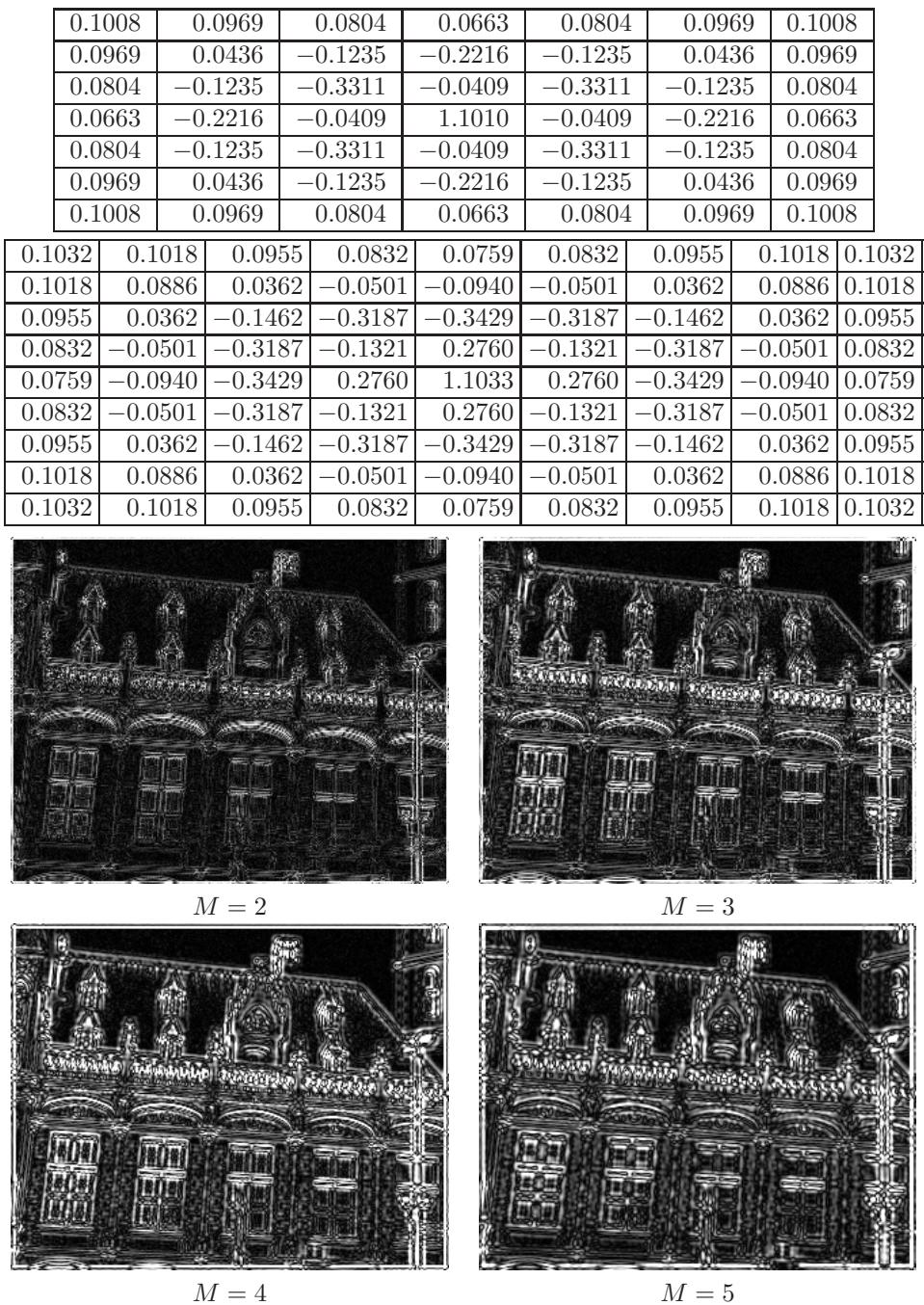


Figure 4.28: Enhancing the blob-like details of the building in Ghent, by high pass filtering with filter (4.122).

Figure 4.28 shows the output of applying filter (4.122) to image 4.25, for various values of M , in order to enhance its blob-like details. To avoid discriminating between dark or bright blob-like details in the image, the absolute value of the filter output is considered before it is scaled in the range [0, 255] for displaying.

How can we enhance small image details using nonlinear filters?

The basic idea of such algorithms is to enhance the local high frequencies. These high frequencies may be identified by considering the level of variation present inside a local window, or by suppressing the low frequencies. This leads to the algorithms of **unsharp masking** and **retinex**, which has been inspired by the human visual system (retinex=retina+cortex). Both algorithms may be used as global ones or as locally adaptive ones.

What is unsharp masking?

This algorithm subtracts from the original image a blurred version of it, so that only high frequency details are left, which are subsequently used to form the enhanced image. The blurred version is usually created by convolving the original image with a Gaussian mask, like one of those defined in example 4.17.

The use of a smoothing filter creates a wide band of pixels around the image, that have either to be left unprocessed or omitted from the final result. As the filters we use here are quite big, such a band around the image would result in neglecting a significant fraction of the image. So, we apply some correction procedure that allows us to use all image pixels: we create an array the same size as the image, with all its elements having value 1; then we convolve this array with the same filter we use for the image and divide the result of the convolution of the image with the result of the convolution of the array of 1s, pixel by pixel. This way the value of a pixel near the border of the image is computed from the available neighbours it has, with weights of the filter that always sum up to 1, even if the neighbourhood used is not complete.

Figure 4.29 shows an original image and its enhanced version by unsharp masking it, using a smoothing Gaussian window of size 121×121 . Figure 4.30 shows another example where either a Gaussian filter was used to create the low pass version of the image, or the mean grey value of the image was considered to be its low pass version. The histograms of the enhanced values are also shown in order to demonstrate how thresholds t_1 and t_2 were selected for applying equation (4.124), on page 354, to produce the displayable result.

How can we apply the unsharp masking algorithm locally?

In the local application of the unsharp masking algorithm, we consider a local window. From the value of a pixel we subtract the value the same pixel has in the low pass filtered version of the image. The difference from the global algorithm is that now the low pass filtered version has been produced by using a small window. The residual is multiplied with an amplifying constant if it is greater than a threshold. The threshold allows one to suppress small high frequency fluctuations which are probably due to noise. The low pass version of the image



(a) Original

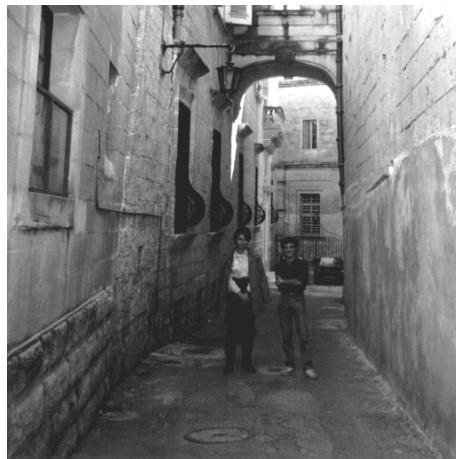
(b) Global, Gaussian 121×121

Figure 4.29: Unsharp masking “A Street in Malta” shown in (a) (size 512×512). (b) Global algorithm, where the low pass version of the image was obtained by convolving it with a 121×121 Gaussian window. Residuals below -50 were set to 0 and above 50 to 255. The in-between values were linearly stretched to the range $[0, 255]$.

may be created by convolving the original image either with a flat averaging window, or with a Gaussian window. Figure 4.31 shows the results of applying this algorithm to image “Leaves”, with threshold 15, a local window of size 21×21 and with an amplifying constant of 2. In these results, any values outside the range $[0, 255]$ were collapsed either to 0 or to 255, accordingly. Figures 4.32a and 4.32b show the enhancement of image 4.29a, using a Gaussian and a flat window, respectively, for the estimation of its low pass version.

How does the locally adaptive unsharp masking work?

In the locally adaptive unsharp masking, the amplification factor is selected according to the local variance of the image.

Let us say that the low pass grey value at (x, y) is $m(x, y)$, the variance of the pixels inside a local window is $\sigma(x, y)$, and the value of pixel (x, y) is $f(x, y)$. We may enhance the variance inside each such window by using a transformation of the form,

$$g(x, y) = A[f(x, y) - m(x, y)] + m(x, y) \quad (4.125)$$

where A is some scalar.

We would like areas which have low variance to have their variance amplified most. So, we choose the amplification factor A inversely proportionally to $\sigma(x, y)$,

$$A = \frac{kM}{\sigma(x, y)} \quad (4.126)$$

where k is a constant, and M is the average grey value of the whole image. The value of the pixel is *not* changed if the difference $f(x, y) - m(x, y)$ is *above* a certain threshold.

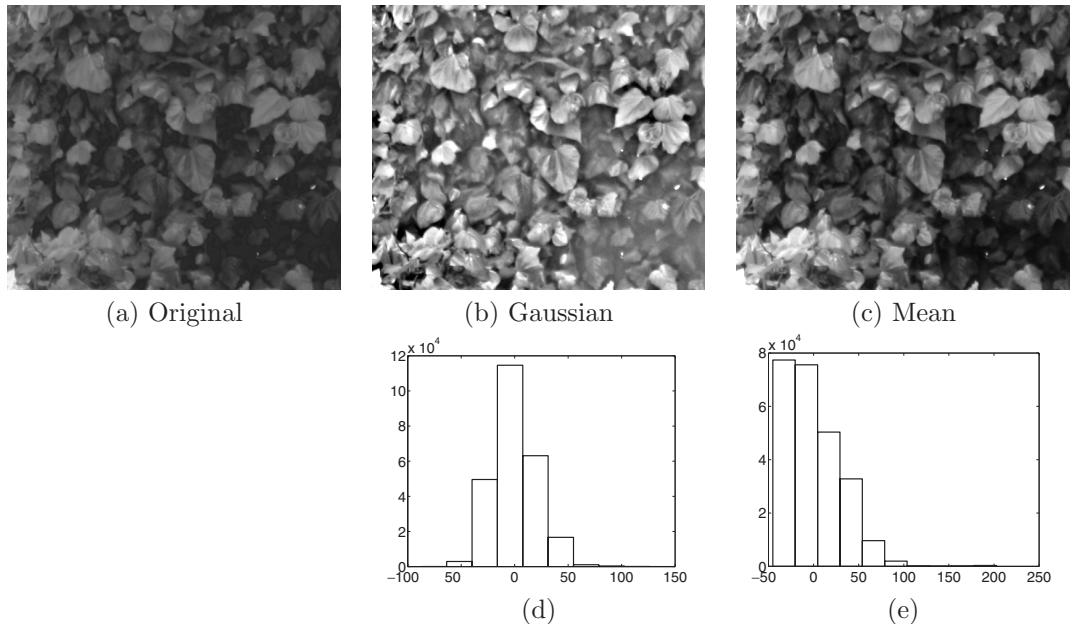


Figure 4.30: (a) The image “Leaves” of size 460×540 . (b) Unsharp masking it by using a Gaussian window of size 121×121 to produce a smoothed version of it which is subtracted from the original image. In (d) the histogram of the residual values. The result was produced by linearly stretching the range of values $[-75, 75]$, while letting values outside this range to become either 0 or 255. (c) Unsharp masking the original image by simply removing from each pixel the mean grey value of the image. In (e) the histogram of the residual values. The result was produced by linearly stretching the range of values $[-50, 100]$, while letting values outside this range to become either 0 or 255.

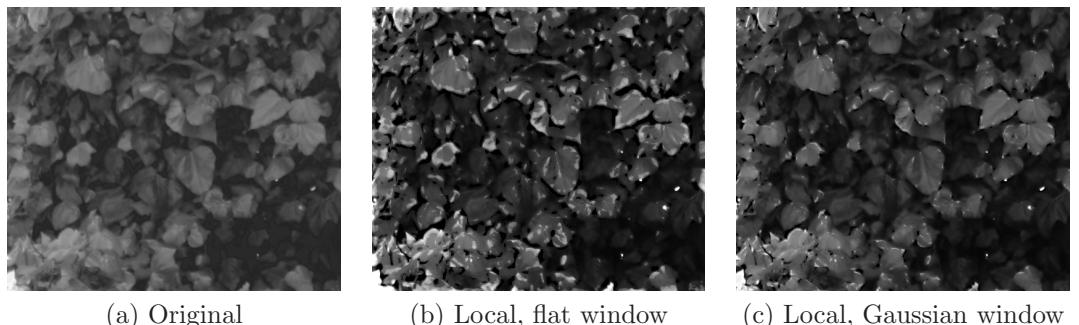


Figure 4.31: Unsharp masking applied locally to image (a). (b) The low pass version of each image patch was created by convolving the original image with an averaging window of size 21×21 . (c) The low pass version of each image patch was created by convolving the original image with a Gaussian window of radius 10. For both results, only differences *larger* than 15 were multiplied with a factor of 2.

Figure 4.32 shows the results of the various versions of the unsharp masking algorithm applied to an image. Figure 4.33 demonstrates the effect of selecting the range of values that will be linearly stretched to the range [0, 255], or simply allowing out of range values to be set to either 0 or 255, without bothering to check the histogram of the resultant values.

How does the retinex algorithm work?

There are many algorithms referred to with the term **retinex**. The simplest one discussed here is also known as **logarithmic transform** or **single scale retinex**.

This algorithm consists of two basic ingredients:

- (i) local grey value normalisation by division with the local mean value;
- (ii) conversion into a logarithmic scale that spreads more the dark grey values and less the bright values (see Box 4.11).

The transformation of the original grey value $f(x, y)$ to a new grey value $g(x, y)$ is expressed as:

$$g(x, y) = \ln(f(x, y) + 1) - \ln \overline{f(x, y)} = \ln \frac{f(x, y) + 1}{\overline{f(x, y)}} \quad (4.127)$$

Note the necessity to add 1 to the image function to avoid having to take the logarithm of 0. (This means that if we wish to scale the image values to be in the range (0, 1], we must divide them by 256.) Function $\overline{f(x, y)}$ is computed by convolving the image with a large Gaussian smoothing filter. The filter is chosen to be large, so that very little detail is left in $\overline{f(x, y)}$. The novelty of the retinex algorithm over unsharp masking is effectively the use of logarithms of the grey image values. Figure 4.34 shows the results of applying this algorithm to the original image of 4.29a.

Box 4.11. Which are the grey values that are stretched most by the retinex algorithm?

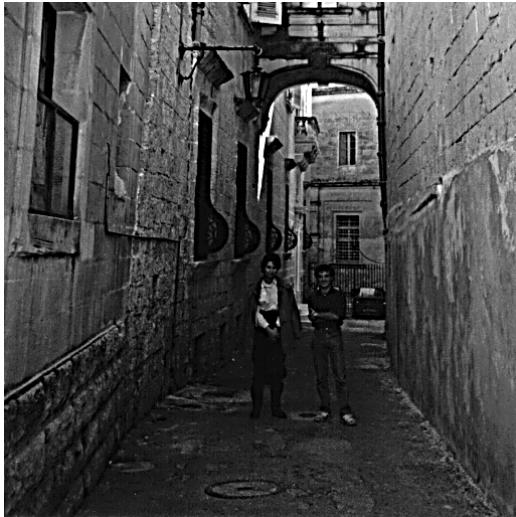
Let us consider a difference Δg in the grey values of the output image, and a corresponding grey level difference Δf in the grey values of the input image. Because of equation (4.127), we may write:

$$\Delta g \sim \frac{\Delta f}{f} \quad (4.128)$$

This relationship indicates that when f is small (dark image patches), a fixed difference in grey values Δf will appear larger in the output image, while when f is large, the same difference in grey values Δf will be reduced. This imitates what the human visual system does, which is known to be more discriminative in dark grey levels than in bright ones. One may easily work this out from the psychophysical **law of Weber-Fechner**. This law says that

$$\frac{\Delta I}{I} \simeq 0.02 \quad (4.129)$$

where ΔI is the minimum grey level difference which may be discriminated by the human eye when the brightness level is I . Since the ratio $\Delta I/I$ is constant, at smaller values of I , (darker greys), we can discriminate smaller differences in I .



(a) Local, Gaussian



(b) Local, flat

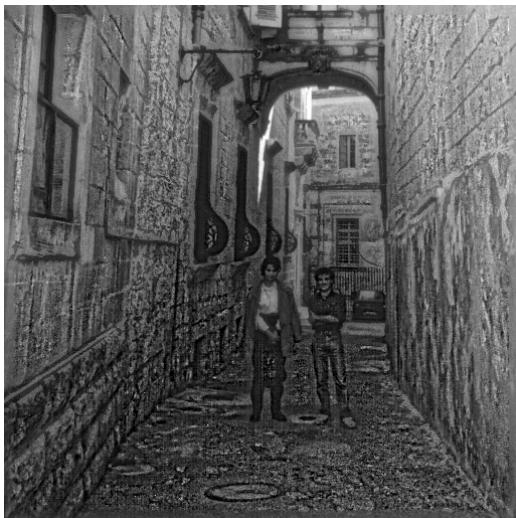
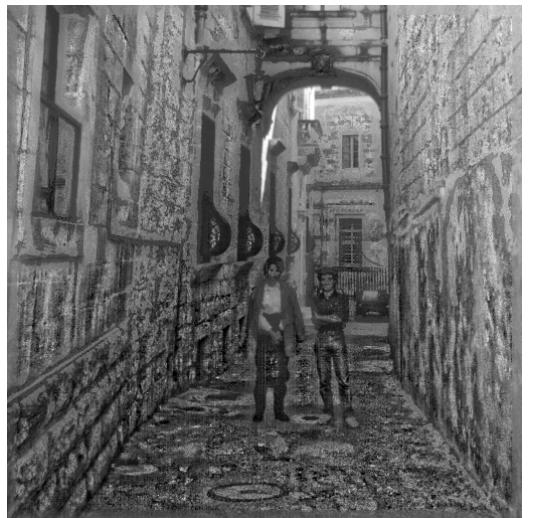
(c) Adaptive, Gaussian, $k = 0.5$ (d) Adaptive, flat, $k = 0.5$

Figure 4.32: Unsharp masking image 4.29a. (a) Local algorithm, where the low pass version of the image was obtained by convolution with a Gaussian mask of size 21×21 . The amplification factor was 2 and differences *below* 3 meant that the pixel value was not changed. (b) As in (a), but a flat 21×21 pixels window was used to obtain the low pass version of the image. (c) The adaptive algorithm where the amplification factor is given by (4.126) with $k = 0.5$. Value $m(x, y)$ used in (4.125) was obtained with a 21×21 Gaussian window. Differences *above* 15 were not enhanced. Finally, only enhanced values in the range $[-75, 275]$ were linearly stretched to the $[0, 255]$ range; those below -75 were set to 0 and those above 275 were set to 255. (d) As in (c), but a 21×21 flat window was used to obtain the value of $m(x, y)$. The range of linearly stretched enhanced values was $[-100, 200]$.

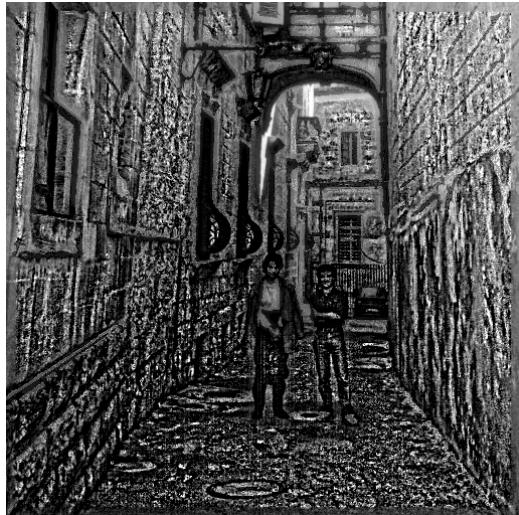
Adaptive, Gaussian, $k = 1.5$ Adaptive, Gaussian, $k = 3$

Figure 4.33: Adaptive unsharp masking. On the left, the enhanced values were simply truncated if they were outside the range [0.255]. On the right, the histogram of the enhanced values was inspected and two thresholds were selected manually. Any value outside the range of the thresholds was either set to 0 or to 255. Values within the two thresholds were linearly stretched to the range [0, 255]. This is very important, particularly for large values of k , which may produce extreme enhanced values. The selected range of values for linear stretching, from top to bottom, respectively, was: $[-50, 250]$ and $[-200, 400]$.

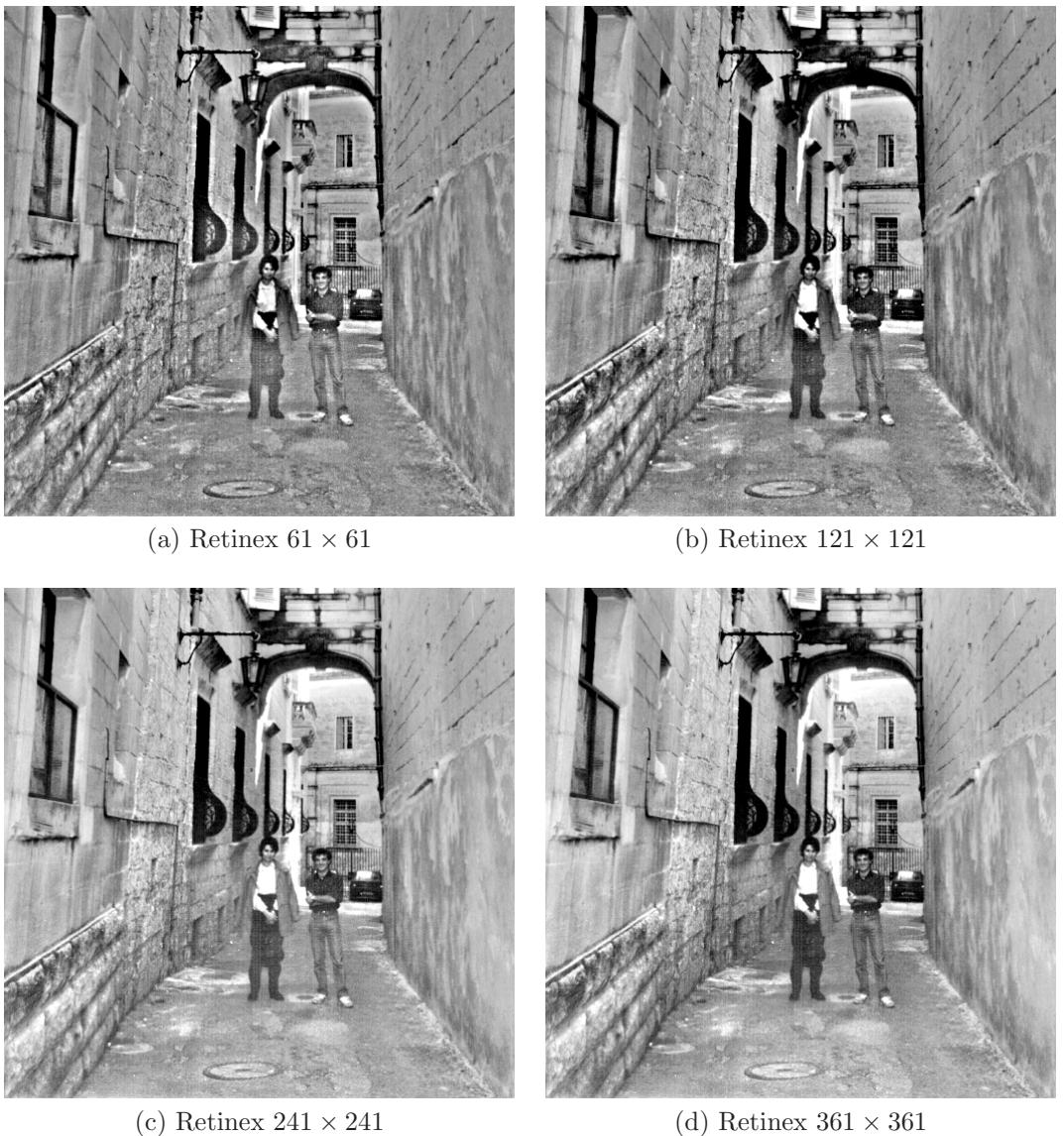


Figure 4.34: Retinex enhancement of the street in Malta. The high frequency details are enhanced by first taking the logarithm of the image and then removing from it its smoothed version, obtained by convolving it with a Gaussian mask of size (a) 61×61 , (b) 121×121 , (c) 241×241 and (d) 361×361 . For the top two panels, equation (4.124) was applied with $t_1 = -200$ and $t_2 = 150$, while for the bottom two panels it was applied with $t_1 = -300$ and $t_2 = 150$. These thresholds were selected by visually inspecting the histograms of the enhanced values.

How can we improve an image which suffers from variable illumination?

The type of illumination variation we are interested in here is due to the inverse square law of the propagation of light. Indeed, according to the laws of physics, the intensity of light reduces according to the inverse of the square of the distance away from the lighting source. This may cause problems in two occasions:

- (i) when the lighting source is very directional and strong, like when we are capturing an image indoors with the light coming from a window somewhere outside the field of view of the camera;
- (ii) when we are interested in performing very accurate measurements using the grey image values. Examples of such applications arise when we use **photometric stereo**, or when we perform industrial inspection that relies on the accurate estimation of the colour of the inspected product.

In both cases (i) and (ii), the problem can be dealt with if we realise that every image function $f(x, y)$ is the product of two factors: an illumination function $i(x, y)$ and a reflectance function $r(x, y)$ that is intrinsic to the imaged surface:

$$f(x, y) = i(x, y)r(x, y) \quad (4.130)$$

To improve the image in the first case, we may use **homomorphic filtering**. To improve the image in the second case, we may apply a procedure called **flatfielding**.

What is homomorphic filtering?

A homomorphic filter enhances the high frequencies and suppresses the low frequencies, so that the variation in the illumination is reduced, while edges (and details) are sharpened.

Illumination is generally of uniform nature and yields low-frequency components in the Fourier transform of the image. Different materials (objects) on the other hand, imaged next to each other, cause sharp changes of the reflectance function, which cause sharp transitions in the intensity of the image. These sharp changes are associated with high-frequency components. We can try to separate these two factors by first taking the logarithm of equation (4.130) so that the two effects are additive rather than multiplicative: $\ln f(x, y) = \ln i(x, y) + \ln r(x, y)$.

The homomorphic filter is applied to this logarithmic image. The cross-section of a homomorphic filter looks like the one shown in figure 4.35.

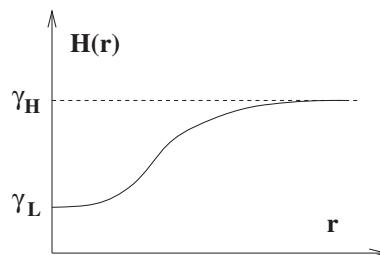
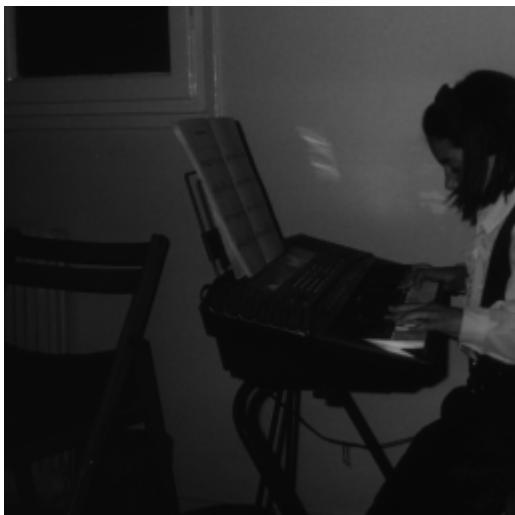


Figure 4.35: A cross-section of a homomorphic filter as a function of polar frequency, $r \equiv \sqrt{\mu^2 + \nu^2}$.

Figure 4.36a shows two images with smoothly varying illumination from left to right. The results after homomorphic filtering, shown in figures 4.36b, constitute clear improvements, with the effect of variable illumination greatly reduced and several details, particularly in the darker parts of the images, made visible.



(a) Original images

(b) After homomorphic filtering

Figure 4.36: These images were captured indoors, with the light of the window coming from the right. The light propagates according to the inverse square law, so its intensity changes gradually as we move to the left of the image.

These results were obtained by applying to the logarithm of the original image, a filter with the following frequency response function:

$$\hat{h}(\mu, \nu) = \frac{1}{1 + e^{-s(\sqrt{\mu^2 + \nu^2} - r_0)}} + A \quad (4.131)$$

with $s = 1$, $r_0 = 128$ and $A = 10$. The parameters of this filter are related as follows to the parameters γ_H and γ_L of figure 4.35:

$$\gamma_L = \frac{1}{1 + e^{sr_0}} + A, \quad \gamma_H = 1 + A \quad (4.132)$$

What is photometric stereo?

In photometric stereo we combine images captured by the same camera, but illuminated by directional light coming from several different directions, in order to work out the orientation of the illuminated surface patch in relation to some coordinate system. The basic point, on which photometric stereo relies, is the observation that the intensity of light received by a surface patch depends on the relative orientation of the surface with respect to the direction of illumination. Exploiting the variation of greyness a pixel exhibits in images captured under different illumination directions, but by the same camera and from the same viewing direction and distance, one can work out the exact orientation of the surface patch depicted by the pixel. The basic assumption is that the variation in greyness, observed for the same pixel, is entirely due to the variation in the relative orientation the corresponding surface patch has with respect to the various illumination sources. In practice, however, part of the variation will also be due to the inverse square law of the propagation of light, and if one ignores that, erroneous estimates of the surface orientation will be made. So, an important first step, before applying such algorithms, is to flatfield the images used.

What does flatfielding mean?

It means to correct an image so that it behaves as if it were captured under illumination of uniform intensity throughout the whole extent of the image.

How is flatfielding performed?

The cases in which flatfielding is required usually arise when the images are captured under controlled conditions, as it happens in systems of visual industrial inspection, or in photometric stereo. In such cases, we have the opportunity to capture also a reference image, by imaging, for example, a uniformly coloured piece of paper, under the same imaging conditions as the image of interest. Then we know that any variation in grey values across this reference image must be due to variation in illumination and noise. The simplest thing to do is to view the reference image as a function $g(x, y)$, where (x, y) are the image coordinates and g is the grey value, and fit this function with a low order polynomial in x and y . This way the high frequency noise is smoothed out, while the low order polynomial captures the variation of illumination across the field of view of the camera. Then the image of interest has to be divided point by point by this low order polynomial function, that models the illumination field, in order to be corrected for the variable illumination. One might divide the image of interest by the raw values of the reference image, point by point, but this may amplify noise.

4.4 Histogram manipulation

What is the histogram of an image?

The histogram of an image is a discrete function that is formed by counting the number of pixels in the image that have a certain grey value. When this function is normalised to sum up to 1 for all the grey values, it can be treated as a probability density function that expresses how probable it is for a certain grey value to be found in the image. Seen this way, the grey value of a pixel becomes a random variable which takes values according to the outcome of an underlying random experiment.

When is it necessary to modify the histogram of an image?

If we cannot see much detail in an image, the reason could be that pixels, which represent different objects or parts of objects, have grey values which are very similar to each other. This is demonstrated with the example histograms shown in figure 4.37. The histogram of the “bad” image is very narrow and it does not occupy the full range of possible grey values, while the histogram of the “good” image is more spread. In order to improve the “bad” image, we might like to modify its histogram so that it looks like that of the “good” image.

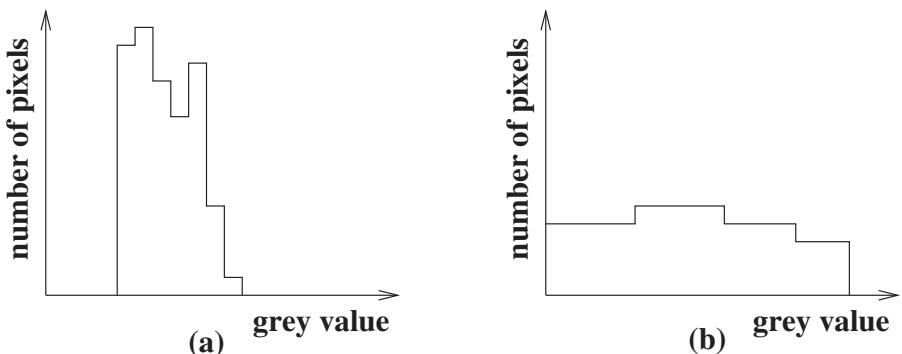


Figure 4.37: (a) The histogram of a “bad” image. (b) The histogram of a “good” image.

How can we modify the histogram of an image?

The simplest way is **histogram stretching**. Let us say that the histogram of the low contrast image ranges from grey value g_{min} to g_{max} . We wish to spread these values over the range $[0, G - 1]$, where $G - 1 > g_{max} - g_{min}$. We may map the grey values to the new range, if the grey value of a pixel g_{old} is replaced with the value g_{new} , given by:

$$g_{new} = \left\lfloor \frac{g_{old} - g_{min}}{g_{max} - g_{min}} G + 0.5 \right\rfloor \quad (4.133)$$

Term 0.5 was added so that the real number $(g_{old} - g_{min})G/(g_{max} - g_{min})$ is rounded by the floor operator $\lfloor \cdot \rfloor$ to its nearest integer as opposed to its integer part. We saw a version of this method on page 354, where equation (4.124) is used instead, designed to trim out extreme values.

Note that all we do by applying equation (4.133) is to spread the grey values, without changing the number of pixels per grey level. There are more sophisticated methods, which as well as stretching the range of grey values, allocate a predefined number of pixels at each grey level. These methods are collectively known as **histogram manipulation**.

What is histogram manipulation?

Histogram manipulation is the change of the grey values of an image, without affecting its semantic information content.

What affects the semantic information content of an image?

The information content of an image is conveyed by the *relative* grey values of its pixels. Usually, the grey values of the pixels do not have meaning in absolute terms, but only in relative terms. If the order (ranking) of pixels in terms of their grey value is destroyed, the information content of the image will be affected. So, an image enhancing method should preserve the relative brightness of pixels.

How can we perform histogram manipulation and at the same time preserve the information content of the image?

Let us assume that the grey values in the original image are represented by variable r and in the new image by variable s . We would like to find a transformation $s = T(r)$ such that the probability density function $p_{old}(r)$, which might look like the one in figure 4.37a, is transformed into a probability density function $p_{new}(s)$, which might look like that in figure 4.37b.

In order to preserve the information content of the image, all pixels that were darker than a pixel with grey value R , say, should remain darker than this pixel even after the transformation, when this pixel gets a new value S , say. So, for *every* grey value R , the number of pixels with lower grey values should be the same as the number of pixels with lower grey values than S , where S is the value to which R is mapped. This may be expressed by saying that the transformation T between the two histograms must preserve the distribution function of the normalised histograms:

$$P_{old}(R) = P_{new}(S)$$

$$\Leftrightarrow \int_0^R p_{old}(r)dr = \int_0^S p_{new}(s)ds \quad (4.134)$$

This equation can be used to define the transformation T that must be applied to the value R of variable r to obtain the corresponding value S of variable s , provided we define function $p_{new}(s)$.

Example 4.23

The histogram of an image may be approximated by the probability density function

$$p_{old}(r) = Ae^{-r} \quad (4.135)$$

where r is the grey level variable taking values between 0 and b , and A is a normalising factor. Calculate the transformation $s = T(r)$, where s is the grey level value in the transformed image, such that the transformed image has probability density function

$$p_{new}(s) = Bse^{-s^2} \quad (4.136)$$

where s takes values between 0 and b , and B is some normalising factor.

Transformation $S = T(R)$ may be calculated using equation (4.134):

$$B \int_0^S se^{-s^2} ds = A \int_0^R e^{-r} dr \quad (4.137)$$

The left-hand side of (4.137) is:

$$\int_0^S se^{-s^2} ds = \frac{1}{2} \int_0^S e^{-s^2} ds^2 = -\frac{1}{2} e^{-s^2} \Big|_0^S = \frac{1 - e^{-S^2}}{2} \quad (4.138)$$

The right-hand side of (4.137) is:

$$\int_0^R e^{-r} dr = -e^{-r} \Big|_0^R = 1 - e^{-R} \quad (4.139)$$

We substitute from (4.138) and (4.139) into (4.137) to obtain:

$$\begin{aligned} \frac{1 - e^{-S^2}}{2} &= \frac{A}{B} (1 - e^{-R}) \Rightarrow \\ e^{-S^2} &= 1 - \frac{2A}{B} (1 - e^{-R}) \Rightarrow \\ -S^2 &= \ln \left[1 - \frac{2A}{B} (1 - e^{-R}) \right] \Rightarrow \\ S &= \sqrt{-\ln \left[1 - \frac{2A}{B} (1 - e^{-R}) \right]} \end{aligned} \quad (4.140)$$

So, each grey value R of the original image should be transformed into grey value S in the enhanced image, according to equation (4.140).

What is histogram equalisation?

Histogram equalisation is the process by which we make all grey values in an image equally probable, ie we set $p_{new}(s) = c$, where c is a constant. Transformation $S = T(R)$ may be calculated from equation (4.134) by substitution of $p_{new}(s)$ and integration. Figures 4.38a-4.38d show an example of applying this transformation to a low contrast image. Notice how narrow the histogram 4.38b of the original image 4.38a is. After histogram equalisation, the histogram in 4.38d is much more spread, but contrary to our expectations, it is not flat, ie it does not look “equalised”.

Why do histogram equalisation programs usually not produce images with flat histograms?

In the above analysis, we tacitly assumed that variables r and s can take continuous values. In reality, of course, the grey level values are discrete. In the continuous domain there is an infinite number of values in any interval $[r, r + dr]$. In digital images, we have only a finite number of pixels in each range. As the range is stretched, and the number of pixels in it is preserved, there is only a finite number of pixels with which the stretched range is populated. The histogram that results is spread over the whole range of grey values, but it is far from flat.

How do we perform histogram equalisation in practice?

In practice, r takes discrete values g , ranging between, say, g_{min} and g_{max} . Also, s takes discrete values t , ranging from 0 to $G - 1$, where typically $G = 256$. Then equation (4.134) becomes:

$$\sum_{t=0}^S p_{new}(t) = \sum_{g=g_{min}}^R p_{old}(g) \quad (4.141)$$

For histogram equalisation, $p_{new}(t) = 1/G$, so that the values of $p_{new}(t)$ over the range $[0, G - 1]$ sum up to 1. Then:

$$\frac{1}{G}(S + 1) = \sum_{g=g_{min}}^R p_{old}(g) \Rightarrow S = G \sum_{g=g_{min}}^R p_{old}(g) - 1 \quad (4.142)$$

For every grey value R , this equation produces a corresponding value S . In general, this S will not be integer, so in order to get an integer value, we round it to an integer by taking its ceiling. This is because, when $R = g_{min}$, the first term on the right-hand side of equation (4.142) may be less than 1 and so S may become negative instead of 0. When $R = g_{max}$, the sum on the right-hand side of (4.142) is 1 and so S becomes $G - 1$, as it should be. So, finally a pixel with grey value g_{old} in the original image should get value g_{new} in the enhanced image, given by

$$g_{new} = \left\lceil G \sum_{g=g_{min}}^{g_{old}} p_{old}(g) - 1 \right\rceil \quad (4.143)$$

where $p_{old}(g)$ is the normalised histogram of the old image.

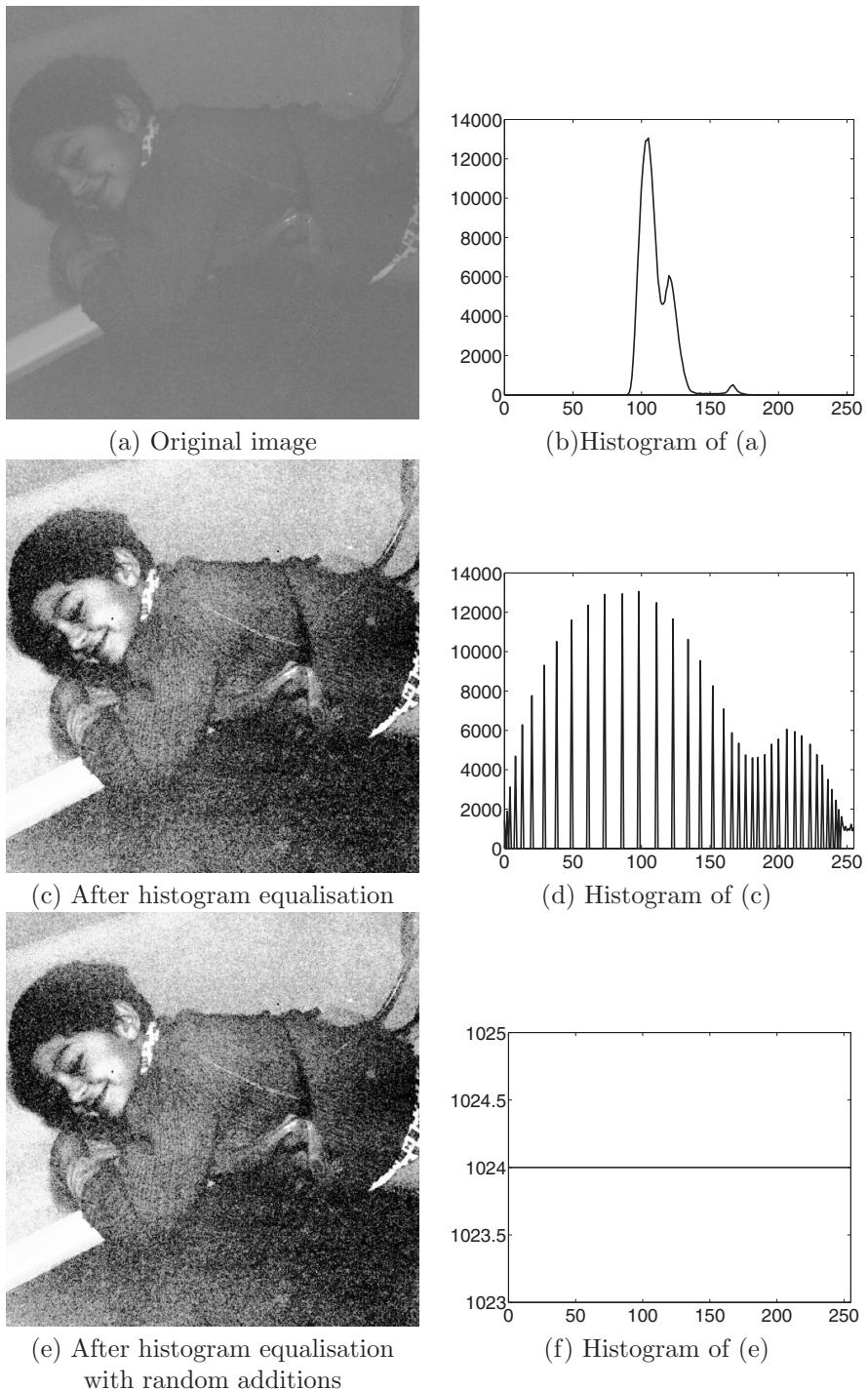
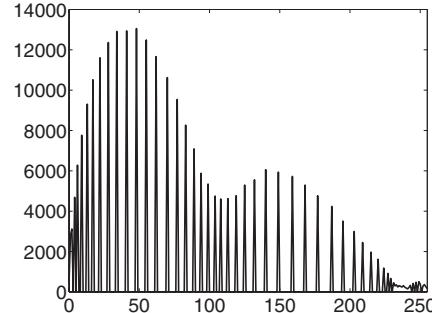


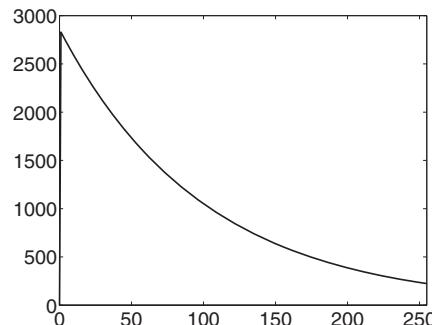
Figure 4.38: Enhancing the image of “The Bathtub Cleaner” by histogram equalisation.



(a) After histogram hyperbolisation



(b) Histogram of (a)

(c) After histogram hyperbolisation
with random additions

(d) Histogram of (c)

Figure 4.39: Histogram hyperbolisation with $\alpha = 0.01$ applied to the image of figure 4.38a.

Can we obtain an image with a perfectly flat histogram?

Yes, if we remove the constraint that the ranking of pixels in terms of their grey values has to be strictly preserved. We may allow, for example, pixels to be moved into neighbouring bins in the histogram, so that all bins have equal number of pixels. This method is known as **histogram equalisation with random additions**. Let us say that the (unnormalised) histogram of the image, after stretching or equalising it, is represented by the 1D array $H(g)$, where $g \in [0, G - 1]$, and that the image has NM pixels. The algorithm of histogram equalisation with random additions should work as follows.

Step 1: To the grey value of each pixel, add a random number drawn from a uniform distribution $[-0.5, 0.5]$.

Step 2: Order the grey values, keeping track which grey value corresponds to which pixel.

Step 3: Change the first $\lfloor \frac{NM}{G} \rfloor$ grey values to 0. Change the next $\lfloor \frac{NM}{G} \rfloor$ grey values to 1, ... etc until the last $\lfloor \frac{NM}{G} \rfloor$ which change to $G - 1$.

The result of applying this algorithm to the image of figure 4.38a is shown in 4.38e.

What if we do not wish to have an image with a flat histogram?

We may define $p_{new}(s)$ in (4.134) to be any function we wish. Once $p_{new}(s)$ is known (the desired histogram), one can solve the integral on the right-hand side to derive a function f_1 of S . Similarly, the integral on the left-hand side may be performed to yield a function f_2 of R , ie

$$f_1(S) = f_2(R) \Rightarrow S = f_1^{-1}f_2(R) \quad (4.144)$$

A special case of this approach is **histogram hyperbolisation**, where $p_{new}(s) = Ae^{-\alpha s}$ with A and α being some positive constants. The effect of this choice is to give more emphasis to low grey values and less to the high ones. This algorithm may also be used in conjunction with random additions, to yield an image with a perfectly hyperbolic histogram (see figure 4.39). In figure 4.39d this can be seen clearly because the method of random additions was used.

How do we do histogram hyperbolisation in practice?

Set $p_{new}(s) = Ae^{-\alpha s}$ in (4.134). First, we work out the value of A , noticing that $p(s)$ has to integrate to 1, from 0 to $G - 1$:

$$\begin{aligned} \int_{s=0}^{G-1} Ae^{-\alpha s} ds &= 1 \Rightarrow A \int_0^{G-1} e^{-\alpha s} d(\alpha s) \frac{1}{\alpha} = 1 \Rightarrow \frac{A}{\alpha} \int_0^{\alpha(G-1)} e^{-t} dt = 1 \Rightarrow \\ &- \frac{A}{\alpha} e^{-\alpha(G-1)} + \frac{A}{\alpha} = 1 \Rightarrow \frac{A}{\alpha} [1 - e^{-\alpha(G-1)}] = 1 \Rightarrow A = \frac{\alpha}{1 - e^{-\alpha(G-1)}} \end{aligned} \quad (4.145)$$

The right-hand side then of (4.134) becomes:

$$\int_0^S Ae^{-\alpha s} ds = A \frac{e^{-\alpha s}}{-\alpha} \Big|_0^S = \frac{A}{\alpha} (1 - e^{-\alpha S}) \quad (4.146)$$

For the discrete normalised histogram of the original image $p_r(g)$, equation (4.144) then takes the form:

$$\frac{A}{\alpha} (1 - e^{-\alpha S}) = \sum_{g=g_{min}}^R p_{old}(g) \Rightarrow S = -\frac{1}{\alpha} \ln \left[1 - \frac{\alpha}{A} \sum_{g=g_{min}}^R p_{old}(g) \right] \quad (4.147)$$

Since S has to take integer values, we add 0.5 and take the floor, so we finally arrive at the transformation:

$$g_{new} = \left\lfloor -\frac{1}{\alpha} \ln \left[1 - \frac{\alpha}{A} \sum_{g=g_{min}}^{g_{old}} p_{old}(g) \right] + 0.5 \right\rfloor \quad (4.148)$$

In figure 4.39, $\alpha = 0.01$ as this gave the most aesthetically pleasing result. It was found with experimentation that $\alpha = 0.05$ and $\alpha = 0.1$ gave images that were too dark, whereas $\alpha = 0.001$ gave an image that was too bright. For this particular image, $G = 256$.

How do we do histogram hyperbolisation with random additions?

The only difference with histogram equalisation with random additions is that now each bin of the desired histogram has to have a different number of pixels. First we have to decide the number of pixels per bin. If t denotes the discrete grey values of the enhanced image, bin $H(t)$ of the desired histogram will have $V(t)$ pixels. So, first we calculate the number of pixels we require per bin. This may be obtained by multiplying the total number of pixels with the integral of the desired probability density function over the width of the bin, ie the integral from t to $t + 1$. For an $N \times M$ image, the total number of pixels is NM . We then have

$$\begin{aligned} V(t) &= NMA \int_t^{t+1} e^{-\alpha t} dt \Rightarrow V(t) = NMA \frac{e^{-\alpha t}}{-\alpha} \Big|_t^{t+1} \\ \Rightarrow V(t) &= -NM \frac{A}{\alpha} \left[e^{-\alpha(t+1)} - e^{-\alpha t} \right] \Rightarrow V(t) = NM \frac{A}{\alpha} \left[e^{-\alpha t} - e^{-\alpha(t+1)} \right] \end{aligned} \quad (4.149)$$

where A is given by (4.145).

The algorithm then of histogram hyperbolisation with random additions is as follows.

Step 1: To the grey value of each pixel add a random number drawn from a uniform distribution $[-0.5, 0.5]$.

Step 2: Order the grey values, keeping track which grey value corresponds to which pixel.

Step 3: Set the first $\lfloor V(0) \rfloor$ pixels to 0.

Step 4: For t from 1 to $G - 1$, assign to the next $\lfloor V(t) + \{V(t - 1) - \lfloor V(t - 1) \rfloor\} \rfloor$ pixels grey value t . Note the correction term $V(t - 1) - \lfloor V(t - 1) \rfloor$ we incorporate in order to account for the left-over part of $V(t - 1)$, which, when added to $V(t)$ may produce a value incremented by 1 when the floor operator is applied.

Why should one wish to perform something other than histogram equalisation?

One may wish to emphasise certain grey values more than others, in order to compensate for a certain effect; for example, to compensate for the way the human eye responds to the different degrees of brightness. This is a reason for doing histogram hyperbolisation: it produces a more pleasing picture.

The human eye can discriminate better darker shades than brighter ones. This is known from psychophysical experiments which have shown that the threshold difference in brightness ΔI , for which the human eye can separate two regions, over the average brightness I is constant and roughly equal to 0.02 (see equation (4.129) of Box 4.11, on page 360). So, the brighter the scene, (higher I) the more different two brightness levels have to be in order for us to be able to discriminate them. In other words, the eye shows more sensitivity to dark shades and this is why histogram hyperbolisation is believed to produce better enhanced images, as it places more pixels in the dark end of the grey spectrum.



Figure 4.40: Enhancing the image of “A Young Train Driver” (of size 512×512).

What if the image has inhomogeneous contrast?

The approach described above is global, ie we modify the histogram which refers to the whole image. However, the image may have variable quality at various parts. In that case, we may apply the above techniques locally: we scan the image with a window inside which we modify the histogram but we alter only the value of the central pixel. Clearly, such a method is costly and various algorithms have been devised to make it more efficient.

Figure 4.40a shows a classical example of an image that requires local enhancement. The picture was taken indoors looking towards windows with plenty of ambient light coming through. All outdoor sections are fine, but in the indoor part the film was under-exposed. The result of global histogram equalisation, shown in figure 4.40b, makes the outdoor parts over-exposed in order to allow us to see the details of the interior. The results of local histogram equalisation, shown in figures 4.40c and 4.40d, are overall much more pleasing.



Figure 4.41: Enhancing the image “At the Karlstejn Castle” (of size 512×512).

The window size used for 4.40c was 81×81 , while for 4.40d it was 241×241 , with the original image being of size 512×512 . Notice that no part of the image gives the impression of being over-exposed or under-exposed. There are parts of the image, however, that look damaged, particularly at the bottom of the image. They correspond to parts of the original film which received too little light to record anything. They correspond to flat black patches, and, by trying to enhance them, we simply enhance the film grain or the instrument noise.

A totally different effect becomes evident in figure 4.41c which shows the local histogram enhancement of a picture taken at Karlstejn castle in the Czech Republic, shown in figure 4.41a. The castle at the back consists of flat grey walls. The process of local histogram equalisation amplifies every small variation of the wall to such a degree that the wall looks like the rough surface of a rock. Further, on the left of the image, we observe again the effect of trying to enhance a totally black area. However, increasing the window size to 241×241 removes most of the undesirable effects.

Can we avoid damaging flat surfaces while increasing the contrast of genuine transitions in brightness?

Yes, there are algorithms that try to do that by taking into consideration pairs of pixel values. Such algorithms may be best understood if we consider the mapping function between input and output grey values. This function has to be one-to-one, but it may be chosen so that it stretches differently different ranges of grey values. This idea is shown schematically in figure 4.42.

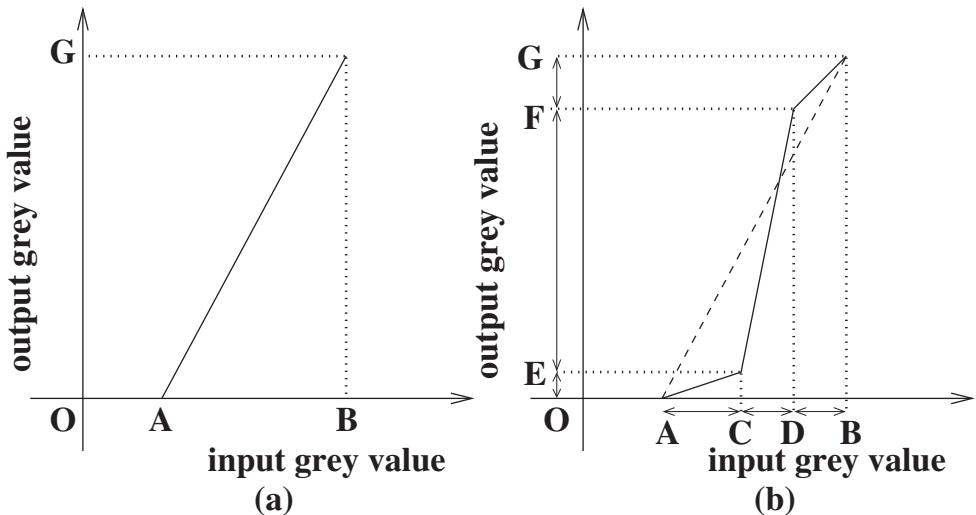


Figure 4.42: (a) A simple stretching (see equation (4.133), on page 367) takes the input range of grey values $[A, B]$ and maps it linearly to the output range $[O, G]$, where $G - O > B - A$. (b) An algorithm that “knows” that grey values in the range $[A, C]$ belong to more or less uniform regions, may suppress the stretching of these values and map them to range $[O, E]$, such that $E - O < C - A$. The same algorithm, “knowing” that values in the range $[C, D]$ often appear in regions of true brightness transitions, may map the grey values in the range $[C, D]$ to the range $[E, F]$, so that $F - E > D - C$. Grey values in the range $[D, B]$ may neither be stretched nor suppressed.

How can we enhance an image by stretching only the grey values that appear in genuine brightness transitions?

Let us consider the image of figure 4.43a. It is a 3-bit 4×4 image. First, let us count how many pairs of grey values of certain type we find next to each other assuming 8-connectivity. We are not interested in ordered pairs, ie (3, 5) is counted the same as (5, 3). The 2D histogram of pairs of values we construct that way occupies only half of the 2D array, as shown at the top of figure 4.43c.

We may select a threshold and say, for example, that pixels that are next to each other and differ by less than 2 grey levels, owe their difference to noise only, and so we do not wish

to stretch, but rather suppress their differences. These pixels correspond to range $[A, C]$ of figure 4.42b. They are the pairs that form the main diagonal of the 2D histogram and the diagonal adjacent to it, as the members of those pairs differ from each other either by 0 or by 1 grey level. The differences of all other pairs are to be stretched. Differences that are to be suppressed should be associated with some negative “force”, that will have to bring the dashed line in figure 4.42b down, while differences that are to be stretched are to be associated with some positive “force”, that will have to bring the dashed line in figure 4.42b up. The more neighbouring pairs a particular grey level participates to, the more likely should be to stretch the mapping curve for its value upwards. So, in the 2D histogram of pairs of values we constructed, we sum the values in each column, ignoring the values along the diagonal strip that represent pairs of values that are to be suppressed. Those are summed separately to form the forces that will have to pull the mapping curve down. The two strings of numbers created that way are shown at the bottom of figure 4.43c as positive and negative forces that try to push the mapping curve upwards or downwards, respectively.

Of course, the mapping function cannot be pushed down and up simultaneously at the same point, so, the two forces have somehow to be combined. We may multiply the negative forces with a constant, say $\alpha = 0.2$, and add the result to the positive forces to form the combined net forces shown in 4.43c. The mapping curve should be stretched so that neighbouring grey values differ proportionally to these numbers. At this point we do not worry about scaling the values to be in the right range. We shall do that at the end.

Next, we work out the cumulative of this string of numbers so that as the grey levels advance, each new grey value differs from its previous one, by as much as the net force we computed for that value. Now, these numbers should be added to the ordinary stretching numbers, ie those represented by the dashed line in 4.42b, which were computed using equation (4.133) of page 367. The ordinary stretching and the calculated cumulative force are added to form the mapping line. The values of this line far exceed the allowed range of 3-bits. So, we scale and round the numbers to be integers between 0 and 7. This is the final mapping curve. The original grey values are mapped to the values in the very bottom line of figure 4.43c. The input values and these values form a look-up table for image enhancement. The mapping curve worked out this way is shown in figure 4.43d. Note that this mapping is not one-to-one. In a real application one may experiment with weight α of the negative forces, to avoid many-to-one mappings.

How do we perform pairwise image enhancement in practice?

The algorithm for such an image enhancement is as follows.

Step 0: If the grey values of the image are in the range $[A, B]$, create a 2D array C of size $(B - A + 1) \times (B - A + 1)$ and initialise all its elements to 0. The elements of this array are identified along both directions with indices from A to B .

Step 1: Accumulate the pairs of grey values that appear next to each other in the image, using 8-connectivity, in the bottom triangle of array C . You need only the bottom triangle because you accumulate in the same cell pairs (g_1, g_2) and (g_2, g_1) . To avoid counting a pair twice, start from the top left corner of the image and proceed from top left to bottom right, by considering for each pixel (i, j) only the pairs it forms with pixels $(i + 1, j)$, $(i + 1, j + 1)$, $(i, j + 1)$ and $(i - 1, j + 1)$, as long as these pixels are within the image boundaries.

Step 2: Decide what the threshold difference d should be, below which you will not enhance

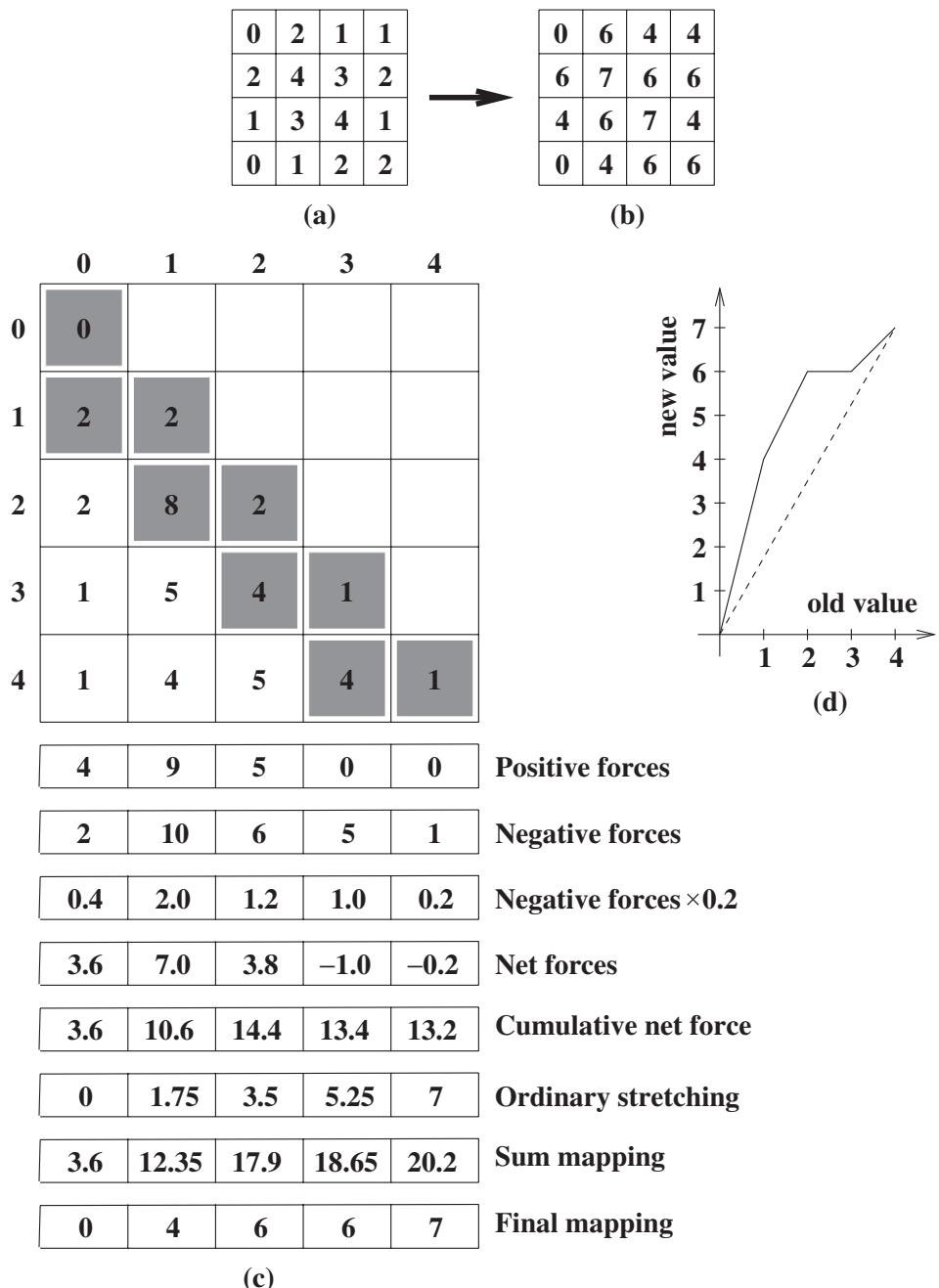


Figure 4.43: An example of image enhancement where pairs of values that appear next to each other and differ by more than a threshold (here equal to 1) are stretched more than other pairs. (a) The original image. (b) The enhanced image. (c) All steps of the algorithm. The grey boxes indicate the cells from which the negative forces are computed. (d) The plot of the final mapping curve.

grey value differences. This determines how wide a strip is along the diagonal of array C , that will be used to form the negative forces.

Step 3: Add the columns of the array you formed in Step 1 that belong to the strip of differences you wish to suppress, to form the string of negative forces:

$$F^-(g) = \begin{cases} C(B, B) & \text{for } g = B \\ C(B-1, B) + C(B-1, B-1) & \text{for } g = B-1 \\ \dots & \\ C(g, g-d+1) + \dots + C(g, g-1) + C(g, g) & \text{for } g > B-d+1 \\ \dots & \\ \sum_{i=g-d}^g C(g, i) & \text{for } g \leq B-d \end{cases} \quad (4.150)$$

Step 4: Add the values of the remaining cells in each column of the 2D array, to form the string of the positive forces:

$$F^+(g) = \begin{cases} 0 & \text{for } g \geq B-d \\ \sum_{i=A}^{g-d-1} C(g, i) & \text{for } g < B-d \end{cases} \quad (4.151)$$

Step 5: Multiply the negative forces with a number α in the range $(0, 1]$ and subtract them point by point from the positive forces, to form the net forces:

$$F^{net}(g) = F^+(g) - \alpha F^-(g) \quad \text{for } A \leq g \leq B \quad (4.152)$$

Step 6: Accumulate the net forces by starting from left to right and creating a running sum, each time adding the next force:

$$S(g) = \sum_{i=A}^g F^{net}(i) \quad \text{for } A \leq g \leq B \quad (4.153)$$

Step 7: Create the mapping from the old to new values, using equation (4.133), $g_{new}(g)$.

Step 8: Add the corresponding values you produced in Steps 6 and 7:

$$\tilde{g}(g) = S(g) + g_{new}(g) \quad \text{for } A \leq g \leq B \quad (4.154)$$

Step 9: Scale and round the resultant values:

$$\tilde{g}_{new}(\tilde{g}(g)) = \left\lfloor \frac{\tilde{g}(g) - \tilde{g}(A)}{\tilde{g}(B) - \tilde{g}(A)} G + 0.5 \right\rfloor \quad (4.155)$$

Step 10: Use the above formula to enhance the image.

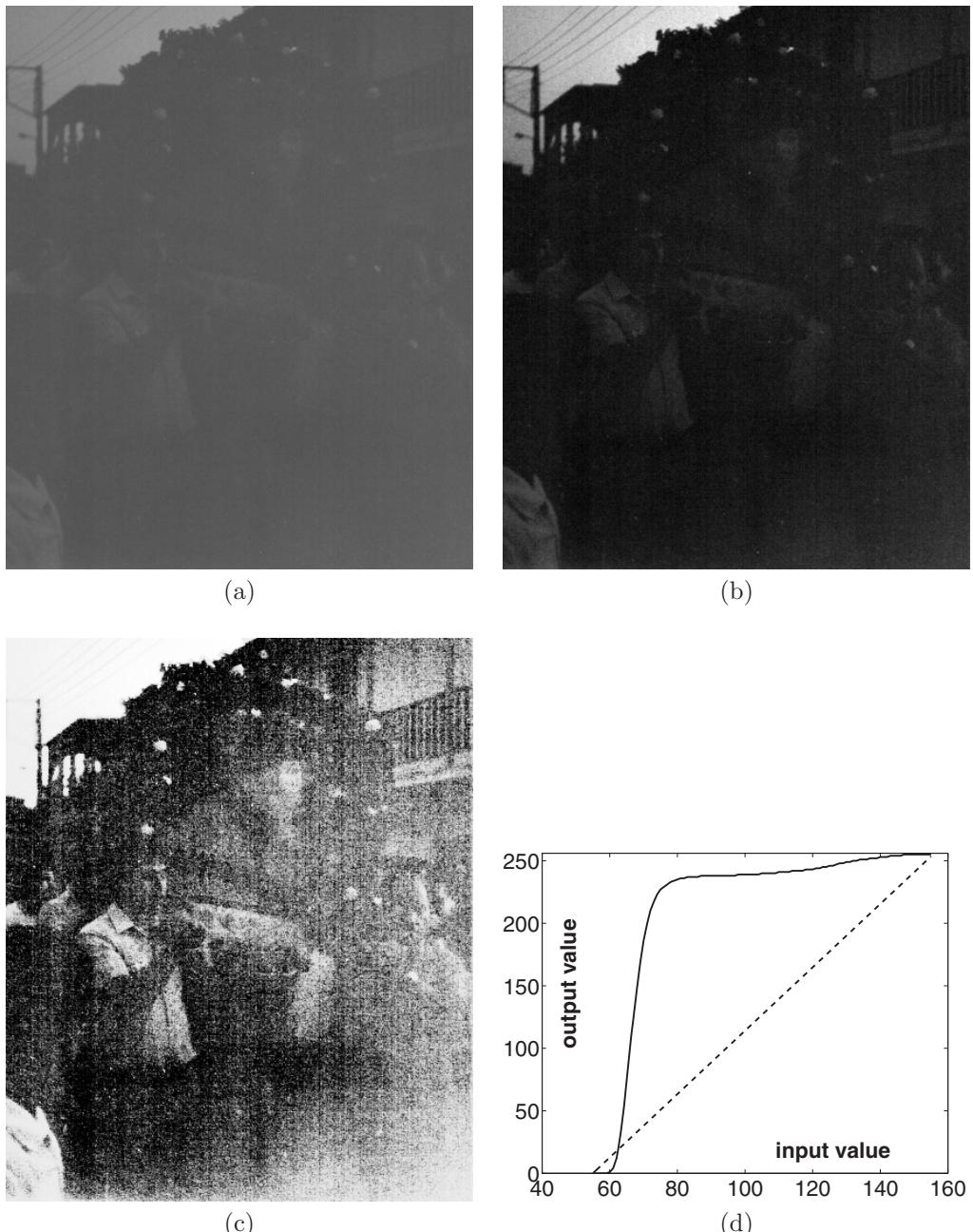


Figure 4.44: “A Catholic Precession” (size 512×425). (b) Enhanced by simple stretching. (c) Enhanced by considering pairs of pixels, using parameters $d = 1$ and $\alpha = 0.5$. (d) The mapping function for stretching (dashed line) and its modification by considering neighbouring pairs of pixels (continuous line).

Note that the above algorithm may be applied globally or locally, inside running windows. When running windows are used, we only change the value of the pixel in the centre of the window and then shift the window by one pixel and repeat the whole process.

Figure 4.44 shows an original image with very low contrast, its enhanced version by simple stretching its range of grey values, and its enhanced version by applying the above algorithm. Figure 4.44d shows the mapping function between the original and the final grey image values. The dashed line is the mapping of simple stretching, while the continuous line is the mapping obtained by the above algorithm. Figure 4.45 shows a bad image and various enhanced versions of it.



(a) Original image



(b) After global histogram equalisation



(c) Local histogram equalisation 81×81



(d) Enhancement with pairwise relations

Figure 4.45: Enhancing the image of “The Hanging Train of Wuppertal”. For the enhancement with the pairwise relations approach, $\alpha = 0.1$ and $d = 3$.

4.5 Generic deblurring algorithms

Proper image deblurring will be discussed in the next chapter, under image restoration. This is because it requires some prior knowledge of the blurring process in order to work correctly. However, one may use some generic methods of deblurring, that may work without any prior knowledge.

If we plot a cross-section of a blurred image, it may look like that of figure 4.46a. The purpose of deblurring is to sharpen the edges, so that they look like those in 4.46b. We shall discuss here some algorithms that may achieve this: mode filtering, mean shift and **toboggan**² contrast enhancement.

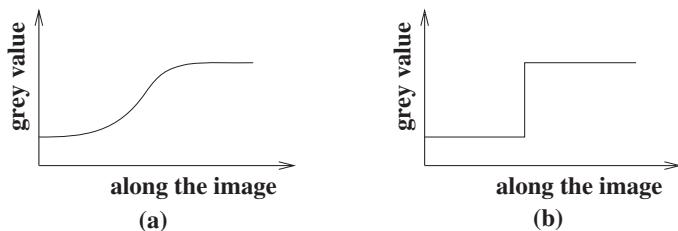


Figure 4.46: A cross section of a blurred image looks like (a). The purpose of deblurring algorithms discussed in this section is to make profiles like (a) become like (b).

How does mode filtering help deblur an image?

It has been shown that repeated application of the mode filter (see page 333) may result in an image made up from patches of uniform grey value with sharp boundaries. The mode may be applied with or without the use of weights. Figure 4.47 shows an image blurred due to shaken camera, and the results of the successive application of mode filtering. The algorithm took 90 iterations to converge. The weights used were

$$\begin{pmatrix} 1 & 3 & 1 \\ 3 & 5 & 3 \\ 1 & 3 & 1 \end{pmatrix} \quad (4.156)$$

These weights were chosen so that the chance of multiple modes was reduced, and the central pixel was given reasonable chance to survive, so that image details might be preserved. What the algorithm does in the case of multiple modes is very critical for the outcome. For these results, when multiple modes were observed, the algorithm worked out the average mode and rounded it to the nearest integer before assigning it to the central pixel. Note that by doing that, we create grey values that might not have been present in the original image. This leads to slow convergence, and ultimately to the creation of artifacts as one can see from figures 4.47e and 4.47f. Figure 4.48a shows the result of applying the mode filtering with the same weights, but leaving the value of the pixel unchanged if multiple modes occurred.

²Toboggan is a type of sledge (originally used by the Canadian Indians) for transportation over snow.

Convergence now was achieved after only 11 iterations. There are no artifacts in the result. Figure 4.48b shows the result of mode filtering with weights:

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \quad (4.157)$$

There is more chance for these weights to create multiple modes than the previous ones. Convergence now was achieved after 12 iterations if a pixel was left unchanged when multiple modes were detected. If the average of multiple modes was used, the output after 12 iterations is shown in 4.48c. After a few more iterations severe artifacts were observed.



Figure 4.47: “Alison” (size 172×113). A blurred image and its deblurred versions by using weighted mode filtering with weights (4.156). If the output of the filter had multiple modes, the average of the modes was used.

The quantisation of the image values used is crucial for this algorithm. Programming environments like Matlab, that convert the image values to real numbers between 0 and 1, have to be used with care: the calculation of the mode requires discrete (preferably integer) values. In general, mode filtering is very slow. The result does not necessarily improve with the number of iterations, and so mode filtering may be applied a small number of times, say for 5 or 6 iterations.

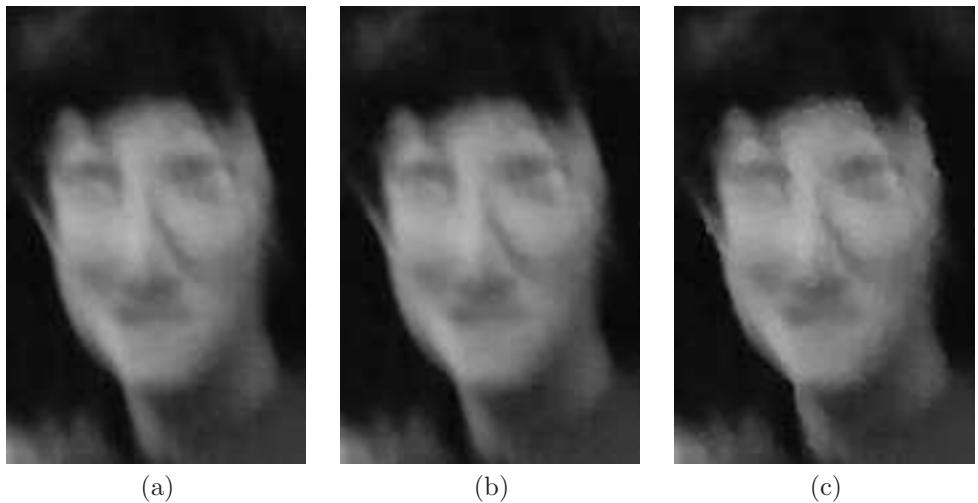


Figure 4.48: (a) Image 4.47a processed with weights (4.156). When the output of the filter had multiple modes, the pixel value was not changed. This is the convergent result after 11 iterations. (b) Image 4.47a processed with weights (4.157). When the output of the filter had multiple modes, the pixel value was not changed. This is the convergent result after 12 iterations. (c) Image 4.47a processed with weights (4.157). When the output of the filter had multiple modes, the average value of these modes was used, rounded to the nearest integer. This is the output after 12 iterations. Further iterations created severe artifacts.

Can we use an edge adaptive window to apply the mode filter?

Yes. The way we use such a window is described on page 337. Once the appropriate window for each pixel has been selected, the mode is computed from the values inside this window. Figure 4.49 shows the results of applying the mode filter to image 4.47a, with a 5×5 edge adaptive window and no weights (top row). In the second row are the results obtained if we use a 3×3 locally adaptive window and weights (4.156).

How can mean shift be used as a generic deblurring algorithm?

The mean shift algorithm, described on page 339, naturally sharpens the edges because it reduces the number of grey values present in the image, and thus, forces intermediate grey values to shift either to one or the other extreme. Figure 4.49 shows the result of applying it to the image of figure 4.47a, with $h_x = 15$, $h_y = 15$ and $h_g = 1$.

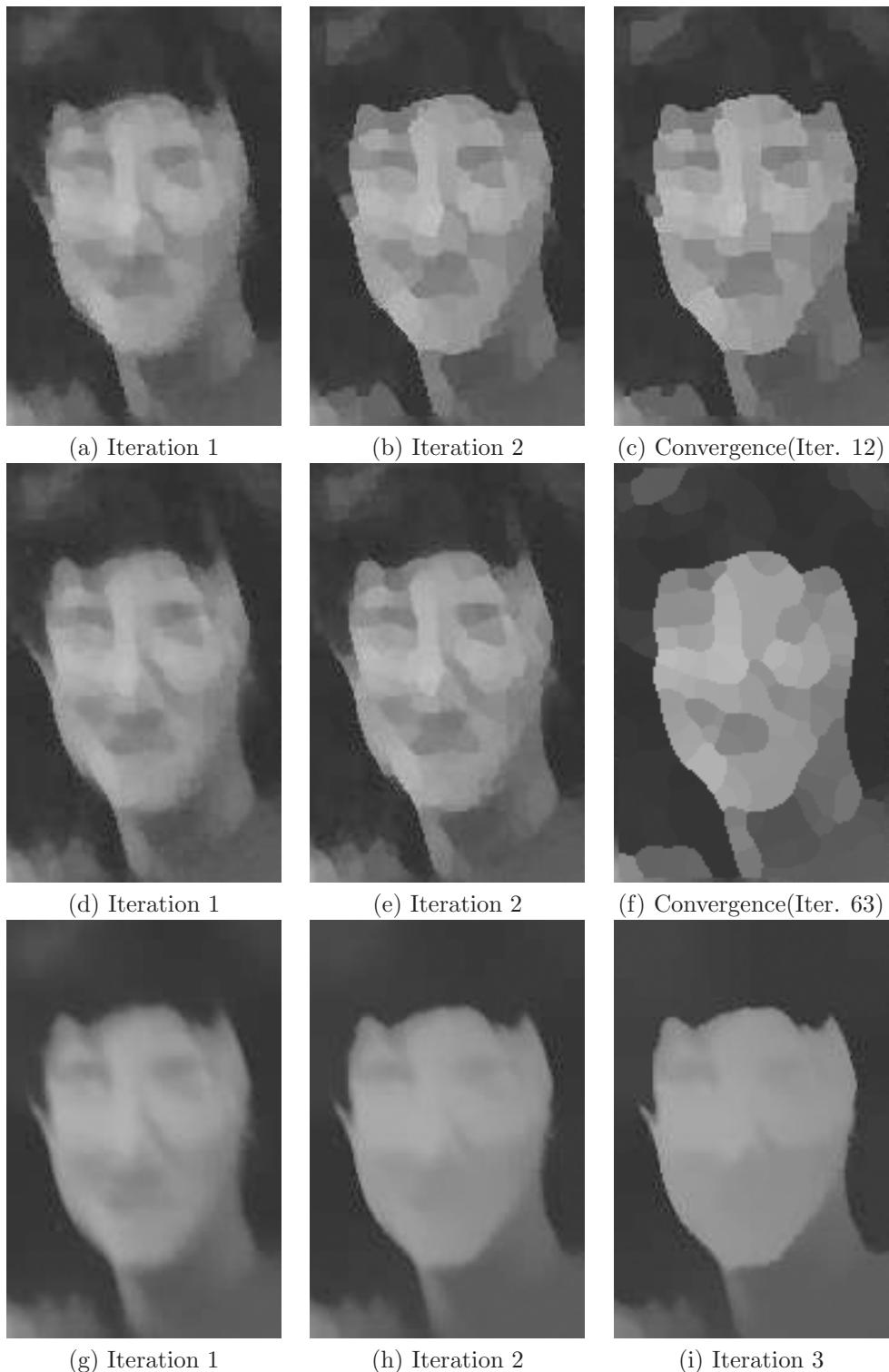


Figure 4.49: Top: edge adaptive 5×5 mode filter. Middle: edge adaptive 3×3 mode filter with weights (4.156). Bottom: mean shift with $h_x = 15$, $h_y = 15$ and $h_g = 1$.

What is toboggan contrast enhancement?

The basic idea of toboggan contrast enhancement is shown in figure 4.50. The pixels “slide” along the arrows shown in 4.50a, so that the blurred profile sharpens. This algorithm consists of three stages.

Stage 1: Work out the magnitude of the gradient vector of each pixel.

Stage 2: Inside a local window around each pixel, identify a pixel with its gradient magnitude being a local minimum.

Stage 3: Assign to the central pixel the value of the pixel with the local minimum gradient magnitude.

How the gradient magnitude of an image may be estimated is covered in Chapter 6 (see pages 596 and 608), so here we are concerned only with Stage 2 of the algorithm.

How do we do toboggan contrast enhancement in practice?

Assuming that the input to the algorithm is a grey image I and an array T of the same size that contains the magnitude of the gradient vector at each pixel position, the following algorithm may be used.

Step 0: Create an array O the same size as the image I and flag all its elements as undefined. The flag may be, for example, a negative number, say -1 for the flag being up. Create also an empty stack where you may temporarily store pixel positions.

Step 1: For each pixel (i, j) in the image: add it to the stack and consider whether its gradient $T(i, j)$ is a local minimum, by comparing it with the values of all its neighbours in its 3×3 neighbourhood.

Step 2: If $T(i, j)$ is a local minimum, set the values of all pixels in the stack equal to the value of the current pixel, empty the stack and go to Step 1.

Step 3: If it is not a local minimum, identify the neighbour with the minimum gradient magnitude.

Step 4: If the flag of the neighbour is down in array O , ie if $O(neighbour) \neq -1$, give to all pixels in the stack the value the neighbour has in the output array, ie set $O(in_stack) = O(neighbour)$. Empty the stack and go to Step 1.

Step 5: If the neighbour is still flagged in O (ie if $O(neighbour) = -1$), and if the gradient magnitude of the neighbour is a local minimum, in array O assign to the neighbour and to all pixels in the stack, the same grey value the neighbour has in image I .

Empty the stack and go to Step 1.

Step 6: If the neighbour is still flagged in O (ie if $O(neighbour) = -1$), and if the gradient magnitude of the neighbour is not a local minimum, add the address of the neighbour to the stack, find the pixel in its 8-neighbourhood with the minimum gradient magnitude and go to Step 2.

Step 7: Exit the algorithm when all pixels in the output array have their flags down, ie all pixels have acquired grey values.

Figure 4.51 shows figure 4.47a deblurred by this algorithm.

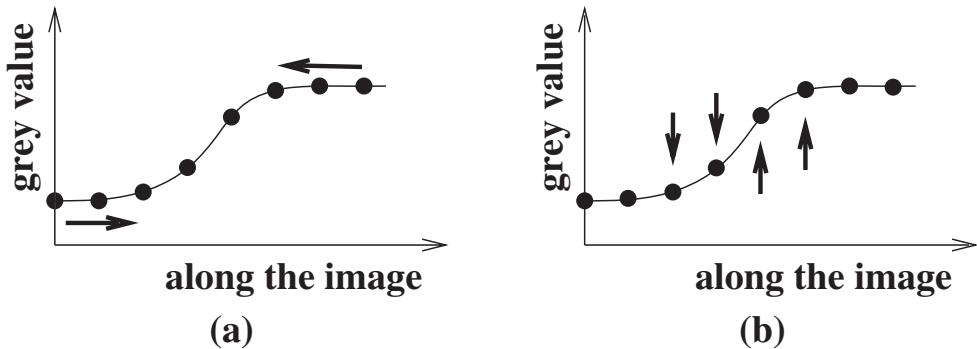


Figure 4.50: The black dots represent pixels. Pixels at the flat parts of the image (extreme left and extreme right in (a)) bequest their grey values to their neighbouring pixels. In other words, pixels in the slanted parts of the cross-section, inherit the values of the pixels with zero gradient. In (a) the arrows show the direction along which information is transferred, while in (b) the arrows show which pixels have their grey values increased or reduced.



Figure 4.51: Toboggan contrast enhancement applied to Alison (figure 4.47a.)

Example 4.24

Apply toboggan contrast enhancement to the image of figure 4.52a. The gradient magnitude for each pixel location is given in 4.52b. Show all intermediate steps.

4	4	3	2
3	7	6	1
2	7	6	2
0	1	0	2

(a)

12	14	23	13
16	11	21	18
21	24	23	12
12	21	20	10

(b)

Figure 4.52: (a) An original image. (b) The value of the gradient magnitude at each pixel position.

All steps of the algorithm are shown in figures 4.53–4.57, where the first array is always the gradient magnitude map T , the second is the input image I and the third is the output image O .

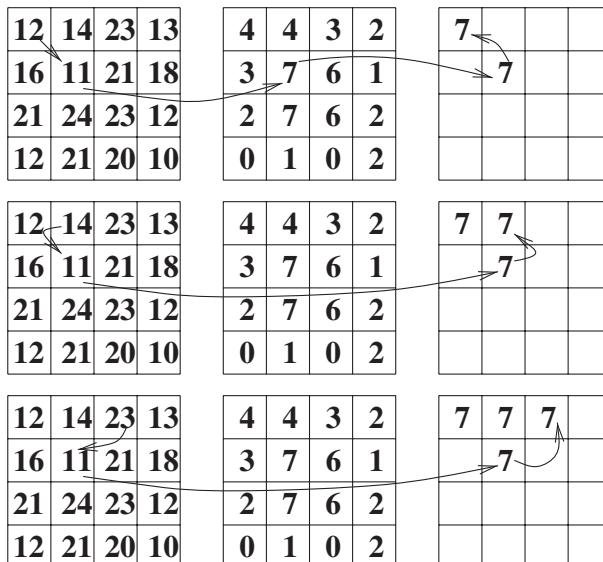


Figure 4.53: Pixel $(0, 0)$: the neighbour with the minimum gradient is pixel $(1, 1)$ ($T(1, 1) = 11$). As $T(1, 1)$ is a local minimum, the value of $I(1, 1)$ is assigned in the output array to both pixels $(0, 0)$ and $(1, 1)$. Pixel $(1, 0)$: the neighbour with the minimum gradient is pixel $(1, 1)$. As this pixel has already a value assigned to it in the output array, pixel $(1, 0)$ inherits that value. The same happens to pixel $(2, 0)$.

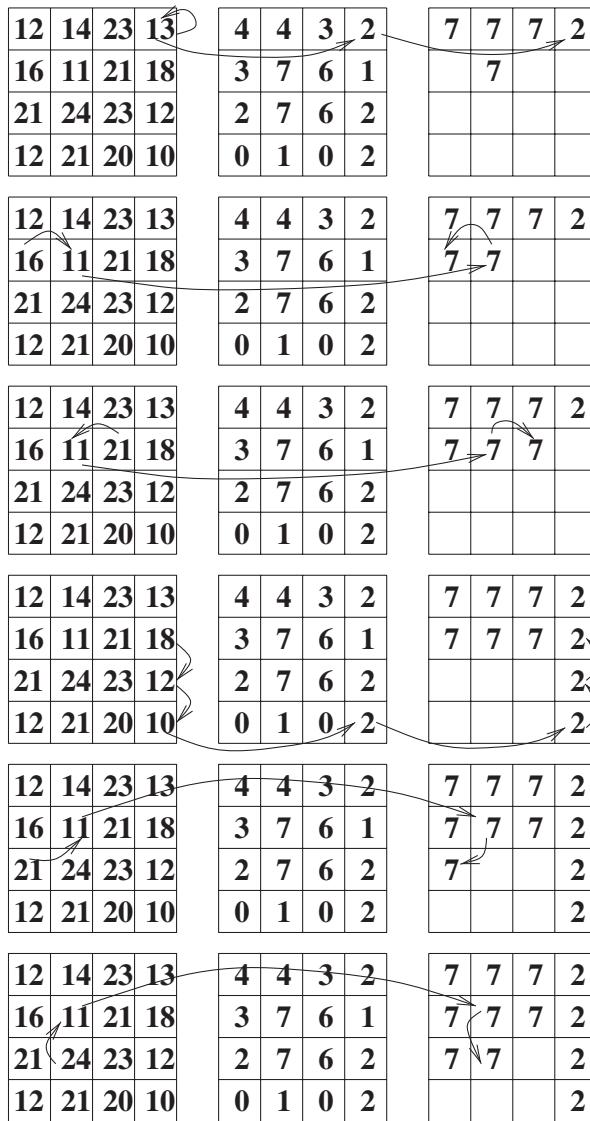


Figure 4.54: Pixel (3, 0) is a local minimum in T , so it gets value 2 in the output array, ie the same value it had in the input array. Pixels (0, 1) and (2, 1) have neighbours with minimum gradient, which already have values assigned to them in the output array, so they inherit that value. The neighbour with the minimum gradient for pixel (3, 1) is pixel (3, 2). $T(3, 2)$ is not a local minimum; its own neighbour with the minimum gradient magnitude is pixel (3, 3). $T(3, 3)$ is a local minimum. Pixels (3, 1), (3, 2) and (3, 3) all take value $I(3, 3)$ in the output array. Pixels (0, 2) and (1, 2) have neighbours with minimum gradient magnitude, which have already assigned values in the output array, so they inherit the values of those neighbours.

12	14	23	13
16	11	21	18
21	24	23	12
12	21	20	10

4	4	3	2
3	7	6	1
2	7	6	2
0	1	0	2

7	7	7	2
7	7	7	2
7	7	2	2
0	0	0	2

Figure 4.55: Pixel (2, 2) has a neighbour with minimum gradient magnitude, which has already an assigned value in the output array, so it inherits the value of that neighbour.

12	14	23	13
16	11	21	18
21	24	23	12
12	21	20	10

4	4	3	2
3	7	6	1
2	7	6	2
0	1	0	2

7	7	7	2
7	7	7	2
7	7	2	2
0	0	0	2

Figure 4.56: Pixel (0, 3) has gradient magnitude that is a local minimum. Its value in the output array is set to be the same as that in the input array: $O(0, 3) = I(0, 3)$.

12	14	23	13
16	11	21	18
21	24	23	12
12	21	20	10

4	4	3	2
3	7	6	1
2	7	6	2
0	1	0	2

7	7	7	2
7	7	7	2
7	7	2	2
0	0	0	2

12	14	23	13
16	11	21	18
21	24	23	12
12	21	20	10

4	4	3	2
3	7	6	1
2	7	6	2
0	1	0	2

7	7	7	2
7	7	7	2
7	7	2	2
0	0	2	2

Figure 4.57: Pixels (1, 3) and (2, 3) have neighbours with minimum gradient magnitude, which have already assigned values in the output array, so they inherit the values of those neighbours.

Example 4.25

Deblur the image of figure 4.58a using toboggan deblurring and mode filtering with weights (4.157) and (4.156).



(a) Original



(b) Toboggan



(c) Mode (iter. 2, weights (4.157))



(d) Mode (iter. 6, weights (4.157))



(e) Mode (iter. 1, weights (4.156))



(f) Mode (iter. 8, weights (4.156))

Figure 4.58: (a) “Faces”, blurred due to shaky camera (size 300 × 358). (b) Using toboggan deblurring. (c)-(f): Mode filtering with different weights for various iterations.

What is the “take home” message of this chapter?

With image enhancement we try to make images look better according to *subjective* criteria.

We may enhance an image in a desirable way, by manipulating its Fourier spectrum: we can preferentially kill frequency bands we do not want, or enhance frequencies we want. This can be achieved with the help of filters defined in the frequency domain, with exactly specified spectra. The use of such filters involves taking the Fourier transform of the image, multiplying it with the Fourier transform of the filter, and then taking the inverse Fourier transform. We can avoid this tedious process by working solely in the real domain, but the filters we shall use then have to be finite (to be implemented using convolution) or infinite but approximate (to be implemented using z -transforms). In either case, these filters are optimal for convenience of use, rather than optimal for their frequency characteristics.

Further, we may enhance an image using nonlinear methods, which manipulate its grey values directly, by mapping them to a broader range of values. When applying such methods, care should be taken so the ranking of pixels is more or less preserved, in order to preserve the semantic content of the image and not create artifacts.

Contrast enhancement of a grey image can be achieved by manipulating the grey values of the pixels so that they become more diverse. This can be done by defining a transformation that converts the distribution of the grey values to a prespecified shape. The choice of this shape may be totally arbitrary.

Finally, in the absence of any information, generic deblurring may be achieved by using mode filtering, mean shift or toboggan enhancement. Figures 4.59 and 4.60 show the profile of the same cross section of the various restored versions of image 4.47a. We can see that the edges have indeed been sharpened, but unless an algorithm that takes into consideration spatial information is used, the edges may be shifted away from their true position and thus lose their lateral continuity. So, these algorithms do not really restore the image into its unblurred version, but they simply sharpen its edges and make it look patchy.

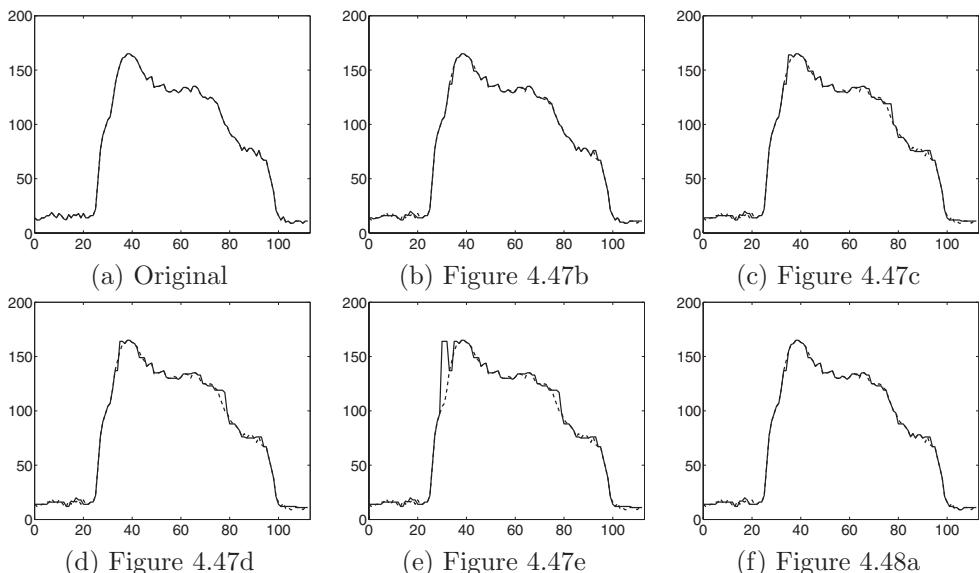


Figure 4.59: Line 124 of Alison, originally and after deblurring. Averaging multiple modes introduces an artifact on the left in (e). The dashed line in each panel is the original profile.

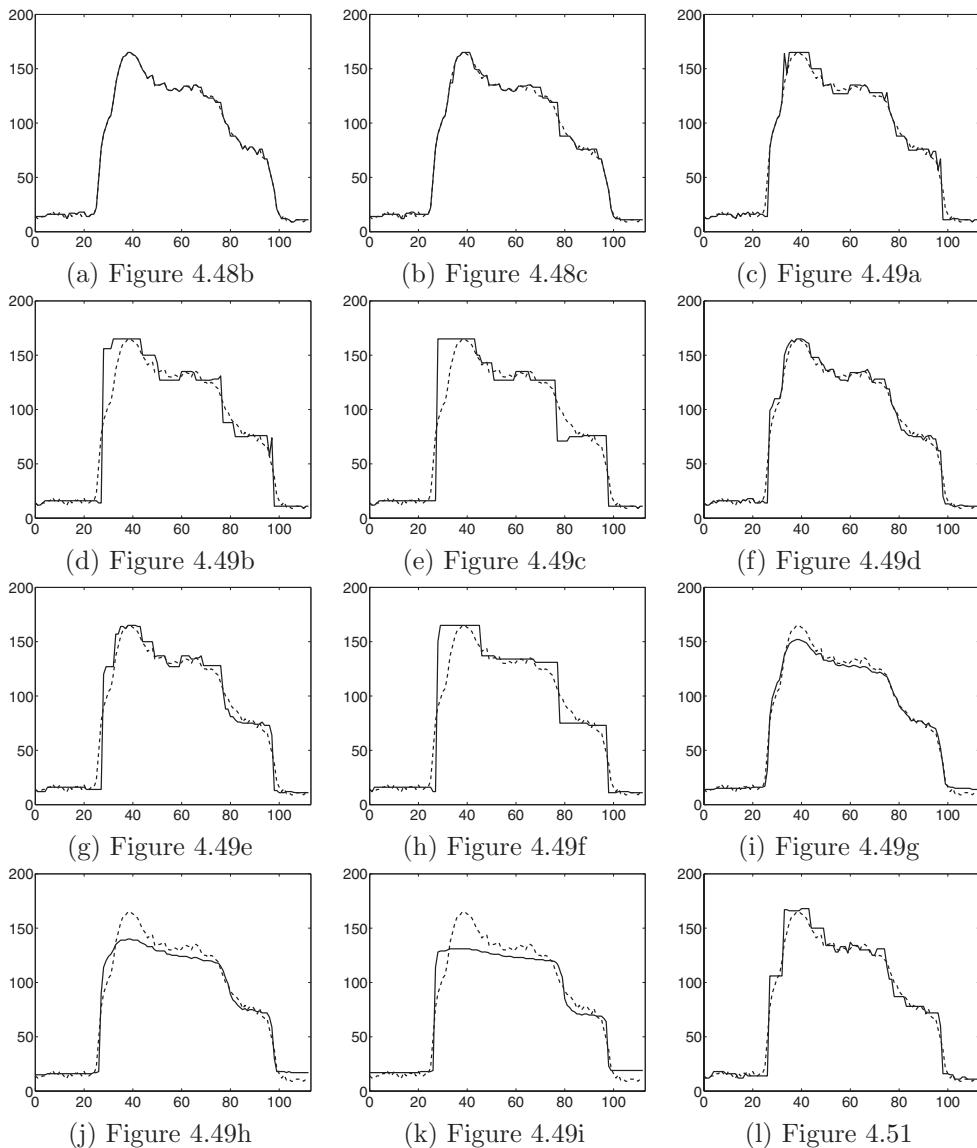


Figure 4.60: The profile of line 124 of image Alison, originally and after applying the various deblurring methods. Note how the mean shift algorithm, (panels (i), (j) and (k)), creates large flat patches in the images. All algorithms make edges sharper and reduce small grey value fluctuations. The original profile is shown as a dashed line superimposed to each resultant profile.

Chapter 5

Image Restoration

What is image restoration?

Image restoration is the improvement of an image using *objective* criteria and prior knowledge as to what the image should look like.

Why may an image require restoration?

An image may be degraded because the grey values of individual pixels may be altered, or it may be distorted because the position of individual pixels may be shifted away from their correct position. The second case is the subject of **geometric restoration**, which is a type of **image registration**.

What is image registration?

Image registration is the establishment of a correspondence between the pixels of two images, depicting the same scene, on the basis that the corresponding pixels are images of the same physical patch of the imaged scene. Image registration is a very broad topic, with applications in medical image processing, remote sensing and multiview vision, and it is beyond the scope of this book.

How is image restoration performed?

Grey value restoration may be modelled as a linear process, in which case it may be solved by a linear method. If the degradation is homogeneous, ie the degradation model is the same for the whole image, then the problem becomes that of defining an appropriate convolution filter with which to process the degraded image in order to remove the degradation. For linear but inhomogeneous degradations, a linear solution may be found, but it cannot be expressed in the form of a simple convolution. For general degradation processes, where linear and nonlinear effects play a role, nonlinear restoration methods should be used.

What is the difference between image enhancement and image restoration?

In image enhancement we try to improve the image using *subjective* criteria, while in image restoration we are trying to reverse a specific damage suffered by the image, using *objective* criteria.

5.1 Homogeneous linear image restoration: inverse filtering

How do we model homogeneous linear image degradation?

Under the assumption that the effect which causes the damage is linear, equation (1.15), on page 13, should be used. Then, in the continuous domain, the output image $g(\alpha, \beta)$ may be written in terms of the input image $f(x, y)$ as

$$g(\alpha, \beta) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) h(x, \alpha, y, \beta) dx dy \quad (5.1)$$

where $h(x, \alpha, y, \beta)$ is the point spread function that expresses the degradation effect. If this effect is the same in the whole image, the point spread function is shift invariant and equation (1.17) applies. We may then model the degraded image as the convolution between the undegraded image $f(x, y)$ and the point spread function of the degradation process:

$$g(\alpha, \beta) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) h(\alpha - x, \beta - y) dx dy \quad (5.2)$$

In terms of Fourier transforms of the functions involved, this may be written as

$$\hat{G}(u, v) = \hat{F}(u, v) \hat{H}(u, v) \quad (5.3)$$

where \hat{G} , \hat{F} and \hat{H} are the Fourier transforms of functions g , f and h , respectively.

How may the problem of image restoration be solved?

The problem of image restoration may be solved if we have prior knowledge of the **point spread function** or its Fourier transform (the **frequency response function**) of the degradation process.

How may we obtain information on the frequency response function $\hat{H}(u, v)$ of the degradation process?

1. From the knowledge of the physical process that caused the degradation. For example, if the degradation is due to diffraction, $\hat{H}(u, v)$ may be calculated. Similarly, if the degradation is due to atmospheric turbulence or motion, the physical process may be modelled and $\hat{H}(u, v)$ calculated.
2. We may try to extract information on $\hat{H}(u, v)$ or $h(\alpha - x, \beta - y)$ from the image itself, ie from the effect the process has on the images of some known objects, ignoring the actual nature of the underlying physical process that takes place.

Example 5.1

When a certain static scene was being recorded, the camera underwent planar motion parallel to the image plane (x, y) . This motion appeared as if the scene moved in the x and y directions by distances, which are functions of time t , $x_0(t)$ and $y_0(t)$, respectively. The shutter of the camera remained open from $t = 0$ to $t = T$ where T is a positive real number. Write down the equation that expresses the intensity recorded at pixel position (x, y) in terms of the scene intensity function $f(x, y)$.

The total exposure time at any point of the recording medium (say the film) will be T and we shall have for the blurred image:

$$g(x, y) = \int_0^T f(x - x_0(t), y - y_0(t)) dt \quad (5.4)$$

This equation says that all points that were at close enough distances from point (x, y) to be shifted passing point (x, y) in time interval T , will have their values recorded and accumulated by the sensor at position (x, y) .

Example 5.2

In example 5.1, derive the frequency response function with which you can model the degradation suffered by the image due to the camera motion, assuming that the degradation was linear with a shift invariant point spread function.

Consider the Fourier transform $\hat{G}(u, v)$ of $g(x, y)$ defined in example 5.1:

$$\hat{G}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(x, y) e^{-2\pi j(ux+vy)} dx dy \quad (5.5)$$

If we substitute (5.4) into (5.5), we have:

$$\hat{G}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_0^T f(x - x_0(t), y - y_0(t)) dt e^{-2\pi j(ux+vy)} dx dy \quad (5.6)$$

We may exchange the order of the integrals:

$$\hat{G}(u, v) = \int_0^T \underbrace{\left\{ \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x - x_0(t), y - y_0(t)) e^{-2\pi j(ux+vy)} dx dy \right\}}_{\text{This is the Fourier transform of a shifted function by } x_0 \text{ and } y_0 \text{ in directions } x \text{ and } y, \text{ respectively}} dt \quad (5.7)$$

This is the Fourier transform of a shifted function by x_0 and y_0 in directions x and y , respectively

We have shown (see equation (2.241), on page 115) that the Fourier transform of a

shifted function and the Fourier transform of the unshifted function are related by:

$$\text{FT of shifted function} = (\text{FT of unshifted function})e^{-2\pi j(ux_0(t)+vy_0(t))} \quad (5.8)$$

Therefore,

$$\hat{G}(u, v) = \int_0^T \hat{F}(u, v)e^{-2\pi j(ux_0(t)+vy_0(t))} dt \quad (5.9)$$

where $\hat{F}(u, v)$ is the Fourier transform of the scene intensity function $f(x, y)$, ie the unblurred image. $\hat{F}(u, v)$ is independent of time, so it may come out of the integral sign:

$$\hat{G}(u, v) = \hat{F}(u, v) \int_0^T e^{-2\pi j(ux_0(t)+vy_0(t))} dt \quad (5.10)$$

Comparing this equation with (5.3), we conclude that:

$$\hat{H}(u, v) = \int_0^T e^{-2\pi j(ux_0(t)+vy_0(t))} dt \quad (5.11)$$

Example 5.3

Assume that the motion in example 5.1 was in the x direction only and with constant speed $\frac{\alpha}{T}$, so that $y_0(t) = 0$, $x_0(t) = \frac{\alpha t}{T}$. Calculate the frequency response function of the process that caused motion blurring.

In equation (5.11), substitute $y_0(t)$ and $x_0(t)$ to obtain:

$$\begin{aligned} \hat{H}(u, v) &= \int_0^T e^{-2\pi ju\frac{\alpha t}{T}} dt = \frac{e^{-2\pi ju\frac{\alpha T}{T}}}{-2\pi ju\frac{\alpha}{T}} \Big|_0^T = -\frac{T}{2\pi ju\alpha} [e^{-2\pi ju\alpha} - 1] \\ &= \frac{T}{2\pi ju\alpha} [1 - e^{-2\pi ju\alpha}] = \frac{Te^{-\pi ju\alpha}}{2\pi ju\alpha} [e^{\pi ju\alpha} - e^{-\pi ju\alpha}] \\ &= \frac{Te^{-\pi ju\alpha} 2j \sin(\pi u\alpha)}{2j\pi u\alpha} = T \frac{\sin(\pi u\alpha)}{\pi u\alpha} e^{-j\pi u\alpha} \end{aligned} \quad (5.12)$$

Example B5.4

It was established that during the time interval T , when the shutter of the camera was open, the camera moved in such a way that it appeared as if the objects in the scene moved along the positive y axis, with constant acceleration 2α and initial velocity s_0 , starting from zero displacement. Derive the frequency response function of the degradation process for this case.

In this case $x_0(t) = 0$ in (5.11) and

$$\frac{d^2y_0}{dt^2} = 2\alpha \Rightarrow \frac{dy_0}{dt} = 2\alpha t + b \Rightarrow y_0(t) = \alpha t^2 + bt + c \quad (5.13)$$

where b and c are some constants of integration, to be specified by the initial conditions of the problem. We have the following initial conditions:

$$\begin{aligned} t = 0 & \quad \text{zero shifting, ie } c = 0 \\ t = 0 & \quad \text{velocity of shifting} = s_0 \Rightarrow b = s_0 \end{aligned} \quad (5.14)$$

Therefore:

$$y_0(t) = \alpha t^2 + s_0 t \quad (5.15)$$

We substitute $x_0(t)$ and $y_0(t)$ in equation (5.11) for $\hat{H}(u, v)$:

$$\begin{aligned} \hat{H}(u, v) &= \int_0^T e^{-2\pi jv(\alpha t^2 + s_0 t)} dt \\ &= \int_0^T \cos [2\pi v \alpha t^2 + 2\pi v s_0 t] dt - j \int_0^T \sin [2\pi v \alpha t^2 + 2\pi v s_0 t] dt \end{aligned} \quad (5.16)$$

We may use formulae

$$\begin{aligned} \int \cos(ax^2 + bx + c) dx &= \sqrt{\frac{\pi}{2a}} \left\{ \cos \frac{ac - b^2}{a} C \left(\frac{ax + b}{\sqrt{a}} \right) - \sin \frac{ac - b^2}{a} S \left(\frac{ax + b}{\sqrt{a}} \right) \right\} \\ \int \sin(ax^2 + bx + c) dx &= \sqrt{\frac{\pi}{2a}} \left\{ \cos \frac{ac - b^2}{a} S \left(\frac{ax + b}{\sqrt{a}} \right) + \sin \frac{ac - b^2}{a} C \left(\frac{ax + b}{\sqrt{a}} \right) \right\} \end{aligned} \quad (5.17)$$

where $S(x)$ and $C(x)$ are

$$\begin{aligned} S(x) &\equiv \sqrt{\frac{2}{\pi}} \int_0^x \sin(t^2) dt \\ C(x) &\equiv \sqrt{\frac{2}{\pi}} \int_0^x \cos(t^2) dt \end{aligned} \quad (5.18)$$

and they are called **Fresnel integrals**.

We shall use the above formulae with

$$\begin{aligned} a &\rightarrow 2\pi v \alpha \\ b &\rightarrow 2\pi v s_0 \\ c &\rightarrow 0 \end{aligned} \quad (5.19)$$

to obtain:

$$\begin{aligned} \hat{H}(u, v) &= \frac{1}{2\sqrt{v\alpha}} \left\{ \cos \frac{2\pi v s_0^2}{\alpha} C\left(\sqrt{2\pi v} \frac{\alpha t + s_0}{\sqrt{\alpha}}\right) \right. \\ &\quad - \sin\left(-\frac{2\pi v s_0^2}{\alpha}\right) S\left(\sqrt{2\pi v} \frac{\alpha t + s_0}{\sqrt{\alpha}}\right) - j \cos \frac{2\pi v s_0^2}{\alpha} S\left(\sqrt{2\pi v} \frac{\alpha t + s_0}{\sqrt{\alpha}}\right) \\ &\quad \left. - j \sin\left(-\frac{2\pi v s_0^2}{\alpha}\right) C\left(\sqrt{2\pi v} \frac{\alpha t + s_0}{\sqrt{\alpha}}\right) \right\} \Big|_0^T \\ &= \frac{1}{2\sqrt{v\alpha}} \left\{ \cos \frac{2\pi v s_0^2}{\alpha} \left[C\left(\sqrt{\frac{2\pi v}{\alpha}}(\alpha T + s_0)\right) - j S\left(\sqrt{\frac{2\pi v}{\alpha}}(\alpha T + s_0)\right) \right] \right. \\ &\quad + \sin \frac{2\pi v s_0^2}{\alpha} \left[S\left(\sqrt{\frac{2\pi v}{\alpha}}(\alpha T + s_0)\right) + j C\left(\sqrt{\frac{2\pi v}{\alpha}}(\alpha T + s_0)\right) \right] \\ &\quad - \cos \frac{2\pi v s_0^2}{\alpha} \left[C\left(\sqrt{\frac{2\pi v}{\alpha}} s_0\right) - j S\left(\sqrt{\frac{2\pi v}{\alpha}} s_0\right) \right] \\ &\quad \left. - \sin \frac{2\pi v s_0^2}{\alpha} \left[S\left(\sqrt{\frac{2\pi v}{\alpha}} s_0\right) + j C\left(\sqrt{\frac{2\pi v}{\alpha}} s_0\right) \right] \right\} \end{aligned} \quad (5.20)$$

Example B5.5

What is the frequency response function for the case of example 5.4, if the shutter remained open for a very long time and the starting velocity of the shifting was negligible?

It is known that functions $S(x)$ and $C(x)$, that appear in (5.20), have the following

asymptotic behaviour:

$$\begin{aligned}\lim_{x \rightarrow +\infty} S(x) &= \frac{1}{2} \\ \lim_{x \rightarrow +\infty} C(x) &= \frac{1}{2} \\ \lim_{x \rightarrow 0} S(x) &= 0 \\ \lim_{x \rightarrow 0} C(x) &= 0\end{aligned}\tag{5.21}$$

Therefore, for $s_0 \simeq 0$ and $T \rightarrow +\infty$, we have:

$$\begin{aligned}C\left(\sqrt{\frac{2\pi v}{\alpha}}(\alpha T + s_0)\right) &\rightarrow \frac{1}{2} & C\left(\sqrt{\frac{2\pi v}{\alpha}}s_0\right) &\rightarrow 0 \\ S\left(\sqrt{\frac{2\pi v}{\alpha}}(\alpha T + s_0)\right) &\rightarrow \frac{1}{2} & S\left(\sqrt{\frac{2\pi v}{\alpha}}s_0\right) &\rightarrow 0 \\ \cos \frac{2\pi vs_0^2}{\alpha} &\rightarrow 1 & \sin \frac{2\pi vs_0^2}{\alpha} &\rightarrow 0\end{aligned}\tag{5.22}$$

Therefore, equation (5.20) becomes:

$$\hat{H}(u, v) \simeq \frac{1}{2\sqrt{v\alpha}} \left[\frac{1}{2} - j \frac{1}{2} \right] = \frac{1-j}{4\sqrt{v\alpha}}\tag{5.23}$$

Example B5.6

It was established that during the time interval T , when the shutter of the camera was open, the camera moved in such a way that it appeared as if the objects in the scene moved along the positive y axis, with constant acceleration 2α and initial speed 0, until time T_1 , from which time onwards they carried on moving with constant speed. Derive the frequency response function of the degradation process for this case.

Following the notation of example 5.4, we have $s_0 = 0$ and so equation (5.15) takes the form: $y_0(t) = \alpha t^2$. The first derivative of $y_0(t)$ is the speed of the motion at time t . The final speed attained at the end of the constant acceleration period is therefore:

$$s_1 \equiv \left. \frac{dy(t)}{dt} \right|_{t=T_1} = 2\alpha T_1\tag{5.24}$$

Then, applying the results of examples 5.2 and 5.4, for $s_0 = 0$, and by using (5.21), the frequency response function of the motion blurring induced is:

$$\begin{aligned}
 \hat{H}(u, v) &= \int_0^{T_1} e^{-2\pi j v \alpha t^2} dt + \int_{T_1}^T e^{-2\pi j v s_1 t} dt \\
 &= \frac{1}{2\sqrt{v\alpha}} \left\{ C\left(\sqrt{2\pi v\alpha} T_1\right) - jS\left(\sqrt{2\pi v\alpha} T_1\right) \right\} - \frac{e^{-2\pi j v s_1 t}}{2\pi j v s_1} \Big|_{T_1}^T \\
 &= \frac{1}{2\sqrt{v\alpha}} \left\{ C\left(\sqrt{2\pi v\alpha} T_1\right) - jS\left(\sqrt{2\pi v\alpha} T_1\right) \right\} \\
 &\quad - \frac{1}{2\pi j v s_1} \{ e^{-2\pi j v s_1 T} - e^{-2\pi j v s_1 T_1} \} \\
 &= \frac{1}{2\sqrt{v\alpha}} \left\{ C\left(\sqrt{2\pi v\alpha} T_1\right) - jS\left(\sqrt{2\pi v\alpha} T_1\right) \right\} \\
 &\quad - \frac{1}{2\pi j v s_1} \{ \cos(2\pi v s_1 T) - j \sin(2\pi v s_1 T) - \cos(2\pi v s_1 T_1) + j \sin(2\pi v s_1 T_1) \} \\
 &= \frac{1}{2\sqrt{v\alpha}} C\left(\sqrt{2\pi v\alpha} T_1\right) + \frac{1}{4\pi v\alpha T_1} [\sin(4\pi v\alpha T_1 T) - \sin(4\pi v\alpha T_1^2)] \\
 &\quad - j \frac{1}{2\sqrt{v\alpha}} S\left(\sqrt{2\pi v\alpha} T_1\right) + j \frac{1}{4\pi v\alpha T_1} [\cos(4\pi v\alpha T_1 T) - \cos(4\pi v\alpha T_1^2)] \quad (5.25)
 \end{aligned}$$

Example B5.7

Explain how you may infer the point spread function of the degradation process from an astronomical image.

We know that by definition the point spread function is the output of the imaging system when the input is a point source. In an astronomical image, a very distant star may be considered as a point source. By measuring then the brightness profile of a star, we immediately have the point spread function of the degradation process this image has been subjected to.

Example B5.8

Assume that we have an ideal bright straight line in the scene parallel to the image axis x . Use this information to derive the point spread function

of the process that degrades the captured image.

Mathematically, the undegraded image of a bright line may be represented as

$$f(x, y) = \delta(y) \quad (5.26)$$

where we assume that the line actually coincides with the x axis. Then the image of this line will be:

$$h_l(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - x', y - y') \delta(y') dy' dx' = \int_{-\infty}^{+\infty} h(x - x', y) dx'$$

We change variable $\tilde{x} \equiv x - x' \Rightarrow dx' = -d\tilde{x}$. The limits of \tilde{x} are from $+\infty$ to $-\infty$. Then:

$$h_l(x, y) = - \int_{+\infty}^{-\infty} h(\tilde{x}, y) d\tilde{x} = \int_{-\infty}^{+\infty} h(\tilde{x}, y) d\tilde{x} \quad (5.27)$$

The right-hand side of this equation does not depend on x and therefore the left-hand side should not depend either. This means that the image of the line will be parallel to the x axis (or rather coincident with it) and its profile will be constant all along it:

$$h_l(x, y) = h_l(y) = \underbrace{\int_{-\infty}^{+\infty} h_l(\tilde{x}, y) d\tilde{x}}_{\tilde{x} \text{ is a dummy variable, independent of } x} \quad (5.28)$$

Take the Fourier transform of $h_l(y)$:

$$\hat{H}_l(v) \equiv \int_{-\infty}^{+\infty} h_l(y) e^{-2\pi j v y} dy \quad (5.29)$$

The Fourier transform of the point spread function is the frequency response function, given by:

$$\hat{H}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x, y) e^{-2\pi j(ux+vy)} dx dy \quad (5.30)$$

If we set $u = 0$ in this expression, we obtain

$$\hat{H}(0, v) = \int_{-\infty}^{+\infty} \underbrace{\left[\int_{-\infty}^{+\infty} h(x, y) dx \right]}_{h_l(y) \text{ from (5.28)}} e^{-2\pi j v y} dy \quad (5.31)$$

By comparing equation (5.29) with (5.31), we get:

$$\hat{H}(0, v) = \hat{H}_l(v) \quad (5.32)$$

This equation is known as the **Fourier slice theorem**. This theorem states that by taking a slice of the Fourier transform of a function $h(x, y)$ (ie by setting $u = 0$ in $\hat{H}(u, v)$), we obtain the Fourier transform (ie $\hat{H}_l(u)$) of the projection of the function along the corresponding direction (ie the y axis in this example). Then by taking the inverse Fourier transform, we can obtain the projection of the function (ie $h_l(y)$) along that direction.

That is, the image of the ideal line gives us the profile of the point spread function along a single direction, ie the direction orthogonal to the line. This is understandable, as the cross-section of a line orthogonal to its length is no different from the cross-section of a point. By definition, the cross-section of the image of a point is the point spread function of the blurring process. If now we have lots of ideal lines in various directions in the image, we are going to have information as to how the frequency response function looks along the directions orthogonal to these lines in the frequency plane. By interpolation then we can calculate $\hat{H}(u, v)$ at any point in the frequency plane.

Example B5.9

It is known that a certain scene contains a sharp edge. How can the image of the edge be used to infer the point spread function of the imaging device?

Let us assume that the ideal edge can be represented by a step function along the x axis, defined as:

$$u(y) = \begin{cases} 1 & \text{for } y > 0 \\ 0 & \text{for } y \leq 0 \end{cases} \quad (5.33)$$

The image of this function will be:

$$h_e(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - x', y - y') u(y') dx' dy' \quad (5.34)$$

We may define new variables $\tilde{x} \equiv x - x'$, $\tilde{y} \equiv y - y'$. Obviously $dx' = -d\tilde{x}$ and $dy' = -d\tilde{y}$. The limits of both \tilde{x} and \tilde{y} are from $+\infty$ to $-\infty$. Then:

$$h_e(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\tilde{x}, \tilde{y}) u(y - \tilde{y}) d\tilde{x} d\tilde{y} \quad (5.35)$$

Let us take the partial derivative of both sides of this equation with respect to y . We can do that by applying Leibniz rule (see Box 4.9, on page 348), with $\lambda = y$, $a(\lambda) = -\infty$ and $b(\lambda) = +\infty$:

$$\frac{\partial h_e(x, y)}{\partial y} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\tilde{x}, \tilde{y}) \frac{\partial u(y - \tilde{y})}{\partial y} d\tilde{x} d\tilde{y} \quad (5.36)$$

It is known that the derivative of a step function with respect to its argument is a delta function:

$$\begin{aligned}\frac{\partial h_e(x, y)}{\partial y} &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\tilde{x}, \tilde{y}) \delta(y - \tilde{y}) d\tilde{x} d\tilde{y} \\ &= \int_{-\infty}^{+\infty} h(\tilde{x}, y) d\tilde{x}\end{aligned}\quad (5.37)$$

If we compare (5.37) with equation (5.27), we see that the derivative of the image of the edge is the image of a line parallel to the edge. Therefore, we can derive information concerning the point spread function of the imaging process by obtaining images of ideal step edges at various orientations. Each such image should be differentiated along a direction orthogonal to the direction of the edge. Each resultant derivative image should be treated as the image of an ideal line and used to yield the profile of the point spread function along the direction orthogonal to the line, as described in example 5.8.

Example B5.10

Use the methodology of example 5.9 to derive the point spread function of the camera of your mobile phone.

Using a ruler and black ink we create the chart shown in figure 5.1a.

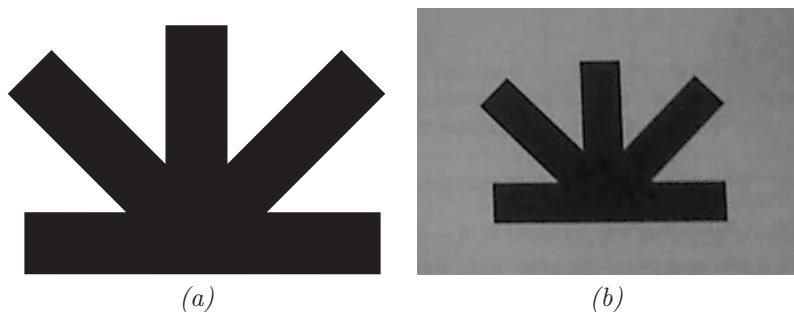


Figure 5.1: (a) A test chart for the derivation of the point spread function of an imaging device. (b) The image of the test chart captured with the camera of a Motorola U9 mobile phone (size 200 × 284).

This chart can be used to measure the point spread function of our imaging system at

orientations 0° , 45° , 90° and 135° . First, the test chart is photographed by the digital camera of the mobile phone. The image is shown in 5.1b. Then, the partial derivatives of the image are computed by convolutions at orientations 0° , 45° , 90° and 135° using the **Robinson operators**. These operators are shown in figure 5.2.

<table border="1"><tr><td>1</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-1</td></tr></table>	1	2	1	0	0	0	-1	-2	-1	<table border="1"><tr><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>-1</td></tr><tr><td>0</td><td>-1</td><td>-2</td></tr></table>	2	1	0	1	0	-1	0	-1	-2	<table border="1"><tr><td>1</td><td>0</td><td>-1</td></tr><tr><td>2</td><td>0</td><td>-2</td></tr><tr><td>1</td><td>0</td><td>-1</td></tr></table>	1	0	-1	2	0	-2	1	0	-1	<table border="1"><tr><td>0</td><td>-1</td><td>-2</td></tr><tr><td>1</td><td>0</td><td>-1</td></tr><tr><td>2</td><td>1</td><td>0</td></tr></table>	0	-1	-2	1	0	-1	2	1	0
1	2	1																																					
0	0	0																																					
-1	-2	-1																																					
2	1	0																																					
1	0	-1																																					
0	-1	-2																																					
1	0	-1																																					
2	0	-2																																					
1	0	-1																																					
0	-1	-2																																					
1	0	-1																																					
2	1	0																																					
(a) M0	(b) M1	(c) M2	(d) M3																																				

Figure 5.2: Filters used to compute the derivative of an image along directions orthogonal to 0° , 45° , 90° and 135° .

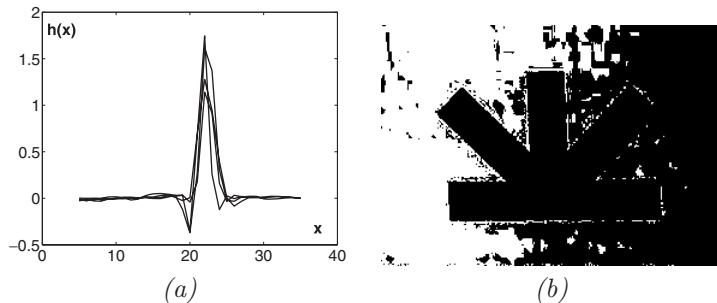


Figure 5.3: (a) The point spread function of the camera when several cross-sections of the convolved image are averaged. (b) The thresholded version of 5.1b, showing the problem of variable illumination of the background. The threshold used was 137.

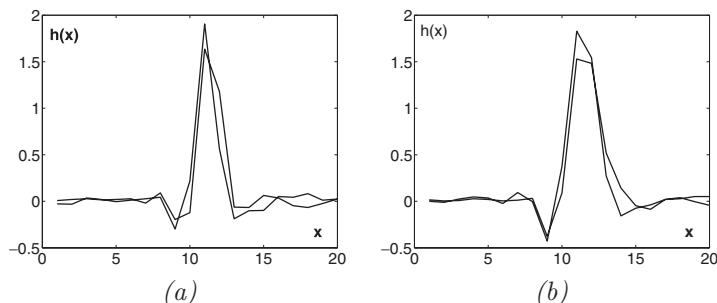


Figure 5.4: The point spread function of the camera when single cross-sections at orientations (a) 45° and 135° and (b) 0° and 90° are considered.

The profiles of the resultant images, along several lines orthogonal to the original edges, are computed and averaged to produce the four profiles for 0° , 45° , 90° and 135° plotted in figure 5.3a. These are the profiles of the point spread function. However, they do not look very satisfactory. For a start, they do not have the same peak value. The reason for this becomes obvious once the original image is thresholded: the illumination of the background is variable, and so the “white” background has different value in the right part of the image from that in the left part of the image. This is a case which would have been benefited if we had applied the process of flatfielding described on page 366. In addition, the edges of the test chart are not perfectly aligned with the axes of the image. This means that averaging several profiles will make the point spread function appear wider than it actually is for the edges that are not perfectly aligned with the selected direction. To avoid these two problems, we select cross-sections from the left part of the convolved image only and also use only single cross-sections to avoid the effect of misalignment. The obtained profiles of the point spread function are shown in figure 5.4. We plot separately the profiles that correspond to orientations 45° and 135° , as the distance of the pixels along these orientations is $\sqrt{2}$ longer than the distance of the pixels along 0° and 90° . Thus, the value of the point spread function that is plotted as being 1 pixel away from the peak, in reality is approximately 1.4 pixels away. Indeed, if we take the ratio of the widths of the two pairs of the profiles, we find the value of ~ 1.4 . In a real practical application, (i) the effects of variable illumination will be avoided by a more careful scene illumination and (ii) several profiles will be constructed, and aligned carefully with sub-pixel accuracy, and averaged, in order to produce the profile of the point spread function, to be used for image restoration.

Note that, apart from the fact that mobile phones do not usually have high quality cameras, the captured images are compressed in a lossy way, and so the point spread function computed reflects all these image imperfections.

In a practical application, if the four profiles are found to be very similar, they are averaged to produce a single cross-section of a circularly symmetric point spread function. The Fourier transform of this 2D function is the frequency response function of the imaging device.

If we know the frequency response function of the degradation process, isn't the solution to the problem of image restoration trivial?

If we know the frequency response function of the degradation and calculate the Fourier transform of the degraded image, it appears from equation (5.3), on page 396, that we can obtain the Fourier transform of the undegraded image very easily:

$$\hat{F}(u, v) = \frac{\hat{G}(u, v)}{\hat{H}(u, v)} \quad (5.38)$$

Then, by taking the inverse Fourier transform of $\hat{F}(u, v)$, we should be able to recover $f(x, y)$, which is what we want. However, this straightforward approach produces unacceptably poor results.

What happens at frequencies where the frequency response function is zero?

$\hat{H}(u, v)$ probably becomes 0 at some points in the (u, v) plane and this means that $\hat{G}(u, v)$ will also be zero at the same points as seen from equation (5.3), on page 396. The ratio $\hat{G}(u, v)/\hat{H}(u, v)$ as it appears in (5.38) will be 0/0, ie undetermined. All this means is that for the particular frequencies (u, v) the frequency content of the original image cannot be recovered. One can overcome this problem by simply omitting the corresponding points in the frequency plane, provided of course they are countable.

Will the zeros of the frequency response function and the image always coincide?

No. If there is the slightest amount of noise in equation (5.3), the zeros of $\hat{H}(u, v)$ will not coincide with the zeros of $\hat{G}(u, v)$. Even if the numerator of (5.38) is extremely small, when the denominator becomes 0, the result is infinitely large. This means that frequencies killed by the imaging process will actually be infinitely amplified. $\hat{G}(u, v)$ will always have noise if nothing else, because of the digitisation process that produces integer valued images.

How can we avoid the amplification of noise?

In many cases, $|\hat{H}(u, v)|$ drops rapidly away from the origin, while $|\hat{N}(u, v)|$ remains more or less constant. To avoid the amplification of noise then when using equation (5.38), we do not use as filter factor $1/\hat{H}(u, v)$, but a windowed version of it, cutting it off at a frequency before $|\hat{H}(u, v)|$ becomes too small or before its first zero. In other words, we use

$$\hat{F}(u, v) = \hat{M}(u, v)\hat{G}(u, v) \quad (5.39)$$

where

$$\hat{M}(u, v) \equiv \begin{cases} \frac{1}{\hat{H}(u, v)} & \text{for } u^2 + v^2 \leq \omega_0^2 \\ 1 & \text{for } u^2 + v^2 > \omega_0^2 \end{cases} \quad (5.40)$$

with ω_0 chosen so that all zeros of $H(u, v)$ are excluded. Of course, one may use other windowing functions instead of a window with rectangular profile, to make $\hat{M}(u, v)$ go smoothly to zero at ω_0 .

Example 5.11

When a certain static scene was being recorded, the camera underwent planar motion parallel to the i axis of the image plane, from right to left. This motion appeared as if the scene moved by the same distance from left to right. The shutter of the camera remained open long enough for the values of i_T consecutive scene patches, that otherwise would have produced i_T consecutive pixels, to be recorded by the same pixel of the produced image. Write down the equation that expresses the intensity recorded at pixel position (i, j) in terms of the unblurred image $f(i, j)$ that might have been produced under ideal conditions.

The blurred image $g(i, j)$ in terms of the ideal image $f(i, j)$ is given by

$$g(i, j) = \frac{1}{i_T} \sum_{k=0}^{i_T-1} f(i-k, j) \quad i = 0, 1, \dots, N-1 \quad (5.41)$$

where i_T is the total number of pixels with their brightness recorded by the same cell of the camera and N is the total number of pixels in a row of the image.

Example 5.12

In example 5.11, derive the frequency response function with which you can model the degradation suffered by the image due to the camera motion, assuming that the degradation is linear with a shift invariant point spread function.

The discrete Fourier transform of the blurred image $g(i, j)$ is given by (see (2.161)):

$$\hat{G}(m, n) = \frac{1}{N^2} \sum_{l=0}^{N-1} \sum_{t=0}^{N-1} g(l, t) e^{-j(\frac{2\pi ml}{N} + \frac{2\pi nt}{N})} \quad (5.42)$$

If we substitute $g(l, t)$ from equation (5.41), we have:

$$\hat{G}(m, n) = \frac{1}{N^2} \frac{1}{i_T} \sum_{l=0}^{N-1} \sum_{t=0}^{N-1} \sum_{k=0}^{i_T-1} f(l-k, t) e^{-j(\frac{2\pi ml}{N} + \frac{2\pi nt}{N})} \quad (5.43)$$

We rearrange the order of the summations to obtain:

$$\hat{G}(m, n) = \frac{1}{i_T} \sum_{k=0}^{i_T-1} \underbrace{\frac{1}{N^2} \sum_{l=0}^{N-1} \sum_{t=0}^{N-1} f(l-k, t) e^{-j(\frac{2\pi ml}{N} + \frac{2\pi nt}{N})}}_{DFT \text{ of shifted } f(l, t)} \quad (5.44)$$

By applying the property of the Fourier transform concerning shifted functions (see (2.241), on page 115), we have

$$\hat{G}(m, n) = \frac{1}{i_T} \sum_{k=0}^{i_T-1} \hat{F}(m, n) e^{-j\frac{2\pi m}{N} k} \quad (5.45)$$

where $\hat{F}(m, n)$ is the Fourier transform of the original image. As $\hat{F}(m, n)$ does not depend on k , it can be taken out of the summation:

$$\hat{G}(m, n) = \hat{F}(m, n) \frac{1}{i_T} \sum_{k=0}^{i_T-1} e^{-j \frac{2\pi m}{N} k} \quad (5.46)$$

We identify then the Fourier transform of the point spread function of the degradation process as:

$$\hat{H}(m, n) = \frac{1}{i_T} \sum_{k=0}^{i_T-1} e^{-j \frac{2\pi m}{N} k} \quad (5.47)$$

The sum on the right-hand side of this equation is a geometric progression with ratio between successive terms:

$$q \equiv e^{-j \frac{2\pi m}{N}} \quad (5.48)$$

We apply then formula (2.165), on page 95, with this q and $S = i_T$, to obtain:

$$\hat{H}(m, n) = \frac{1}{i_T} \frac{e^{-j \frac{2\pi m}{N} i_T} - 1}{e^{-j \frac{2\pi m}{N}} - 1} = \frac{1}{i_T} \frac{e^{-j \frac{\pi m}{N} i_T} (e^{-j \frac{\pi m}{N} i_T} - e^{j \frac{\pi m}{N} i_T})}{e^{-j \frac{\pi m}{N}} (e^{-j \frac{\pi m}{N}} - e^{j \frac{\pi m}{N}})} \quad (5.49)$$

Therefore:

$$\hat{H}(m, n) = \frac{1}{i_T} \frac{\sin(\frac{\pi m}{N} i_T)}{\sin \frac{\pi m}{N}} e^{-j \frac{\pi m}{N} (i_T - 1)} \quad m \neq 0 \quad (5.50)$$

Notice that for $m = 0$ we have $q = 1$ and we cannot apply the formula of the geometric progression. Instead, we have a sum of 1s in (5.47), which is equal to i_T , and so:

$$\hat{H}(0, n) = 1 \quad \text{for } 0 \leq n \leq N - 1 \quad (5.51)$$

It is interesting to compare equation (5.50) with its continuous counterpart, equation (5.12), on page 398. We can see that there is a fundamental difference between the two equations: in the denominator, equation (5.12) has the frequency along the blurring axis, u , appearing on its own, while in the denominator of equation (5.50), we have the sine of this frequency appearing. This is because discrete images are treated by the discrete Fourier transform as periodic signals, repeated ad infinitum in all directions.

How do we apply inverse filtering in practice?

The basic algorithm is as follows.

Step 1: Identify the frequency response function $\hat{H}(u, v)$ of the degradation process.

If this is done analytically, work out the frequency at which $\hat{H}(u, v)$ becomes 0 for the first time after its main maximum.

If this is done numerically, identify the first place after the main maximum where $\hat{H}(u, v)$ changes sign, and work out the frequency at which $\hat{H}(u, v)$ becomes 0.

Call these limiting frequencies u_0 and v_0 , along the two axes.

Step 2: Compute the DFT $\hat{G}(u, v)$ of the degraded image.

Step 3: Set:

$$\hat{F}(u, v) = \begin{cases} \frac{\hat{G}(u, v)}{\hat{H}(u, v)} & \text{if } u < u_0 \text{ and } v < v_0 \\ \hat{G}(u, v) & \text{if } u \geq u_0 \text{ or } v \geq v_0 \end{cases} \quad (5.52)$$

Step 4: Take the inverse DFT of $\hat{F}(u, v)$ to reconstruct the image.

Example 5.13

Consider the 128×128 image of figure 5.5a. To imitate the way this image would look if it were blurred by motion, we take every 10 consecutive pixels along the x axis, find their average value, and assign it to the tenth pixel. This is what would have happened if, when the image was being recorded, the camera had moved 10 pixels to the left: the brightness of a line segment in the scene with length equivalent to 10 pixels would have been recorded by a single pixel. The result would look like figure 5.5b. Restore this image by using inverse filtering omitting division by 0.

The blurred image may be modelled by equation (5.41) with $i_T = 10$ and $N = 128$. Let us denote by $\hat{G}(m, n)$ the Fourier transform of the blurred image. We may analyse it in its real and imaginary parts:

$$\hat{G}(m, n) \equiv G_1(m, n) + jG_2(m, n) \quad (5.53)$$

We may then write it in magnitude-phase form

$$\hat{G}(m, n) = \sqrt{G_1^2(m, n) + G_2^2(m, n)} e^{j\phi(m, n)} \quad (5.54)$$

where

$$\begin{aligned} \cos \phi(m, n) &= \frac{G_1(m, n)}{\sqrt{G_1^2(m, n) + G_2^2(m, n)}} \\ \sin \phi(m, n) &= \frac{G_2(m, n)}{\sqrt{G_1^2(m, n) + G_2^2(m, n)}} \end{aligned} \quad (5.55)$$

To obtain the Fourier transform of the original image, we divide $\hat{G}(m, n)$ with $\hat{H}(m, n)$:

$$\hat{F}(m, n) = \frac{\sqrt{G_1^2(m, n) + G_2^2(m, n)}}{\sin \frac{i_T \pi m}{N}} i_T \sin \frac{\pi m}{N} e^{j(\phi(m, n) + \frac{\pi m}{N}(i_T - 1))} \quad (5.56)$$

Therefore, the real and the imaginary parts of $\hat{F}(m, n)$, $F_1(m, n)$ and $F_2(m, n)$, respectively, are given by:

$$\begin{aligned} F_1(m, n) &= i_T \sin \frac{\pi m}{N} \frac{\sqrt{G_1^2(m, n) + G_2^2(m, n)}}{\sin \frac{i_T \pi m}{N}} \cos \left(\phi(m, n) + \frac{\pi m}{N}(i_T - 1) \right) \\ F_2(m, n) &= i_T \sin \frac{\pi m}{N} \frac{\sqrt{G_1^2(m, n) + G_2^2(m, n)}}{\sin \frac{i_T \pi m}{N}} \sin \left(\phi(m, n) + \frac{\pi m}{N}(i_T - 1) \right) \end{aligned} \quad (5.57)$$

If we use formulae $\cos(a + b) = \cos a \cos b - \sin a \sin b$ and $\sin(a + b) = \cos a \sin b + \sin a \cos b$, and substitute for $\cos \phi(m, n)$ and $\sin \phi(m, n)$ from equations (5.55), we obtain:

$$\begin{aligned} F_1(m, n) &= i_T \sin \frac{\pi m}{N} \frac{G_1(m, n) \cos \frac{\pi m(i_T-1)}{N} - G_2(m, n) \sin \frac{\pi m(i_T-1)}{N}}{\sin \frac{i_T \pi m}{N}} \\ F_2(m, n) &= i_T \sin \frac{\pi m}{N} \frac{G_1(m, n) \sin \frac{\pi m(i_T-1)}{N} + G_2(m, n) \cos \frac{\pi m(i_T-1)}{N}}{\sin \frac{i_T \pi m}{N}} \end{aligned} \quad (5.58)$$

For $m = 0$ (see equation (5.51)) we have to set:

$$F_1(0, n) = G_1(0, n) \quad \text{for } 0 \leq n \leq N - 1$$

$$F_2(0, n) = G_2(0, n) \quad \text{for } 0 \leq n \leq N - 1 \quad (5.59)$$

To restore the image, we use $F_1(m, n)$ and $F_2(m, n)$ as the real and the imaginary parts of the Fourier transform of the undegraded image and take the inverse Fourier transform. As we are trying to recover a real image, we expect that this inverse transform will yield a zero imaginary part, while the real part will be the restored image. It turns out that, if we simply take the inverse DFT of $F_1(m, n) + jF_2(m, n)$, both real and imaginary parts consist of irrelevant huge positive and negative numbers. The real part of the result is shown in 5.5d. Note the strong saturation of the vertical lines. They are due to the presence of infinitely large positive and negative values that are truncated to the extreme allowable grey values.

This result is totally wrong because in equations (5.58) we divide by 0 for several values of m .

Indeed, the denominator $\sin \frac{i_T \pi m}{N}$ becomes 0 every time $\frac{i_T \pi m}{N}$ is a multiple of π :

$$\frac{i_T \pi m}{N} = k\pi \Rightarrow m = \frac{kN}{i_T} \quad \text{where } k = 1, 2, \dots \quad (5.60)$$

Our image is 128×128 , ie $N = 128$, and $i_T = 10$. Therefore, we divide by 0 when $m = 12.8, 25.6, 38.4$, etc. As m takes only integer values, the denominator becomes very small for $m = 13, 26, 38$, etc. It is actually exactly 0 only for $m = 64$. Let us omit this value for m , ie let us use:

$$F_1(64, n) = G_1(64, n) \quad \text{for } 0 \leq n \leq 127$$

$$F_2(64, n) = G_2(64, n) \quad \text{for } 0 \leq n \leq 127 \quad (5.61)$$

The rest of the values of $F_1(m, n)$ and $F_2(m, n)$ are as defined by equations (5.58). If we take the inverse Fourier transform now, we obtain as real part the image in figure 5.5e, with the imaginary part being very nearly 0. The most striking characteristic of this image is the presence of some vertical stripes.

Example 5.14

Restore the image of figure 5.5a using inverse filtering and setting the frequency response function of the degradation process equal to 1 after its first 0.

When we select certain frequencies to handle them in a different way from the way we handle other frequencies, we must be careful to treat in the same way positive and negative frequencies, so that the restored image we get at the end is real. From the way we have defined the DFT we are using here (see equation (5.42), on page 409), it is not obvious which frequencies correspond to the negative frequencies, ie which values of m in (5.58) are paired frequencies.

According to example 5.13, the blurring is along the horizontal direction only. So, our problem is really only 1D: we may restore the image line by line, as if each image line were a separate signal that needed restoration. We shall treat formulae (5.58) as if they were 1D, ie consider n fixed, identifying only the image line we are restoring. Let us remind ourselves what the DFT of a 1D signal $f(k)$ looks like:

$$\hat{F}(m) \equiv \frac{1}{N} \sum_{k=0}^{N-1} f(k) e^{-j \frac{2\pi m}{N} k} \quad (5.62)$$

Since m takes values from 0 to 127, we appreciate that the negative frequencies must be frequencies mirrored from the 127 end of the range. Let us manipulate (5.62) to identify the exact frequency correspondence:

$$\begin{aligned} \hat{F}(m) &= \frac{1}{N} \sum_{k=0}^{N-1} f(k) e^{-j \frac{2\pi(m-N+N)}{N} k} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} f(k) e^{-j \frac{2\pi(m-N)}{N} k} \underbrace{e^{-j \frac{2\pi N}{N} k}}_{=1} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} f(k) e^{j \frac{2\pi(N-m)}{N} k} = \hat{F}^*(N-m) \end{aligned} \quad (5.63)$$

It is obvious from this that if we wish to obtain a real image, whatever we do to frequencies lower than m_0 , we must also do it to the frequencies higher than $N - m_0$. In our example, the first zero of the frequency response function is for $k = 1$, ie for $m = \frac{N}{t_T} = 12.8$ (see (5.60)). We use formulae (5.58), therefore, only for $0 \leq m \leq 12$ and $116 \leq m \leq 127$, and for $0 \leq n \leq 127$. Otherwise we use:

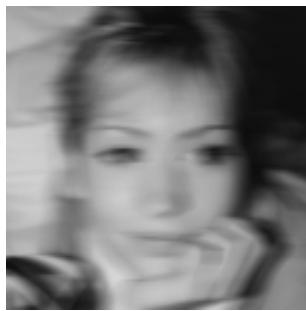
$$\left. \begin{array}{l} F_1(m, n) = G_1(m, n) \\ F_2(m, n) = G_2(m, n) \end{array} \right\} \quad \begin{array}{ll} \text{for} & 13 \leq m \leq 115 \\ & 0 \leq n \leq 127 \end{array} \quad (5.64)$$

If we now take the inverse Fourier transform of $F_1(m, n) + jF_2(m, n)$, we obtain the image shown in figure 5.5f (the imaginary part is again virtually 0). This image looks better than the previous, but more blurred, with the vertical lines (the horizontal

interfering frequency) still there, but less prominent. The blurring is understandable: we have effectively done nothing to improve the frequencies above $m = 12$, so the high frequencies of the image, responsible for any sharp edges, remain degraded.



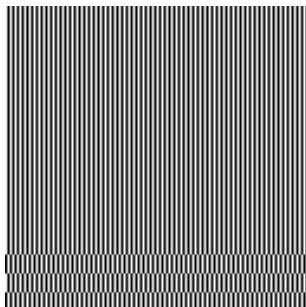
(a) Original image



(b) Realistic blurring
 $MSE = 893$



(c) Blurring with cylindri-
cal boundary condition
 $MSE = 1260$



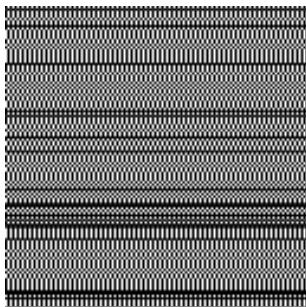
(d) Real part of inverse
filtering of (b)
 $MSE = 19962$



(e) Inverse filtering of (b)
omitting division by 0
 $MSE = 7892$



(f) Inverse filtering of (b)
omitting division with
terms beyond the first 0
 $MSE = 2325$



(g) Real part of inverse
filtering of (c)
 $MSE = 20533$



(h) Inverse filtering of (c)
omitting division by 0
 $MSE = 61$



(i) Inverse filtering of (c)
omitting division with
terms beyond the first 0
 $MSE = 194$

Figure 5.5: Restoring “Dionisia” with inverse filtering.

Example 5.15

Explain the presence of the vertical stripes in restored images 5.5e and 5.5f.

We observe that we have almost 13 vertical stripes in an image of width 128, ie they repeat every 10 pixels. They are due to the boundary effect: the Fourier transform assumes that the image is repeated ad infinitum in all directions. So it assumes that the pixels on the left of the blurred image carry the true values of the pixels on the right of the image. In reality, of course, this is not the case, as the blurred pixels on the left carry the true values of some points further left that do not appear in the image. For example, figure 5.6 shows the results of restoring the same original image when blurred with $i_T = 5, 6, 8$. One can clearly count the interfering stripes being $128/5 \simeq 26$, $128/6 = 24$ and $128/8 = 16$ in these images, respectively. To demonstrate further that this explanation is correct, we blurred the original image with $i_T = 10$, assuming cylindrical boundary conditions, ie assuming that the image is repeated on the left. The result is the blurred image of figure 5.5c. The results of restoring this image by the three versions of inverse filtering are shown in the bottom row of figure 5.5. The vertical lines have disappeared entirely and we have a remarkably good restoration in 5.5h, obtained by simply omitting the frequency for which the frequency response function is exactly 0. The only noise present in this image is quantisation noise: the restored values are not necessarily integer and they are mapped to the nearest integer. Unfortunately, in real situations, the blurring is going to be like that of figure 5.5b and the restoration results are expected to be more like those in figures 5.5e and 5.5f, rather than those in 5.5h and 5.5i.



(a) $i_T = 5$, $MSE = 11339$ (b) $i_T = 6$, $MSE = 6793$ (c) $i_T = 8$, $MSE = 2203$

Figure 5.6: Restored versions of Dionisia blurred with different number i_T of columns recorded on top of each other. The interfering horizontal frequency depends on i_T . The restorations are by simply omitting division by 0.

Note that the quality of the restorations in figure 5.6 does not follow what we would instinctively expect: we would expect the restoration of the image blurred with $i_T = 8$ to be worse than that of the restoration of the image blurred with $i_T = 5$. And yet,

the opposite is true. This is because 8 is an exact divisor of 128, so we have several divisions by 0 and we omit all of them. On the contrary, when $i_T = 5$, we have not a single frequency for which the denominator in (5.58) becomes exactly 0, and so we do not omit any frequency. However, we have many frequencies, where the denominator in (5.58) becomes near 0. Those frequencies are amplified unnecessarily and unrealistically, and cause the problem.

Example 5.16

Quantify the quality of the restoration you achieved in examples 5.13, 5.14 and 5.15 by computing the mean square error (*MSE*) of each restoration. Comment on the suitability of this measure for image quality assessment.

The mean square error is the sum of the square differences between the corresponding pixels of the original (undistorted image) and the restored one, divided by the total number of pixels in the image. The values of *MSE* are given in the captions of figures 5.6 and 5.5. *MSE* is not suitable for image quality assessment in practical applications, because it requires the availability of the perfect image, used as reference. It is only suitable for evaluating and comparing different algorithms using simulating situations. One, however, has to be careful what one measures. The restored images contain errors and interfering frequencies. These result in out of range grey values. So, the restored image matrix we obtain contains real values in a range broader than [0, 255]. To visualise this matrix as an image, we have to decide on whether we truncate the values outside the range to the extreme 0 and 255 values, or we map the full range to the [0, 255] range. The results shown in figures 5.6 and 5.5 were produced by clipping the out of range values. That is why the stripes due to the interfering frequencies have high contrast. These extreme valued pixels contribute significantly to the *MSE* we compute. Figure 5.7 shows the histograms of the original image 5.5a and the two restored versions of it shown in 5.5e and 5.5f. We can see the dominant peaks at grey values 0 and 255, which contribute to the extreme values of *MSE* for these images. Figure 5.8 shows the results we would have obtained if the whole range of obtained values had been mapped to the range [0, 255]. The value of *MSE* now is lower, but the images are not necessarily better.

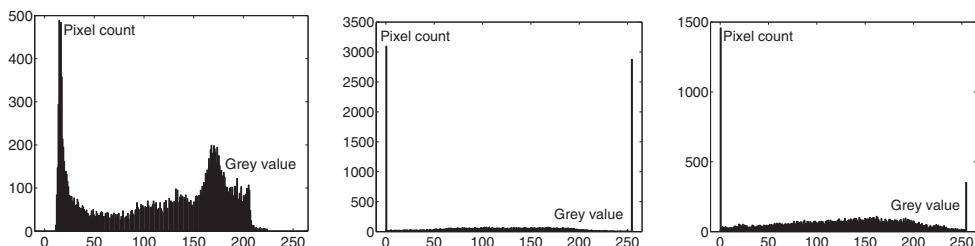


Figure 5.7: Histograms of the original image 5.5a (left), and the restored images 5.5e (middle) and 5.5f (right).

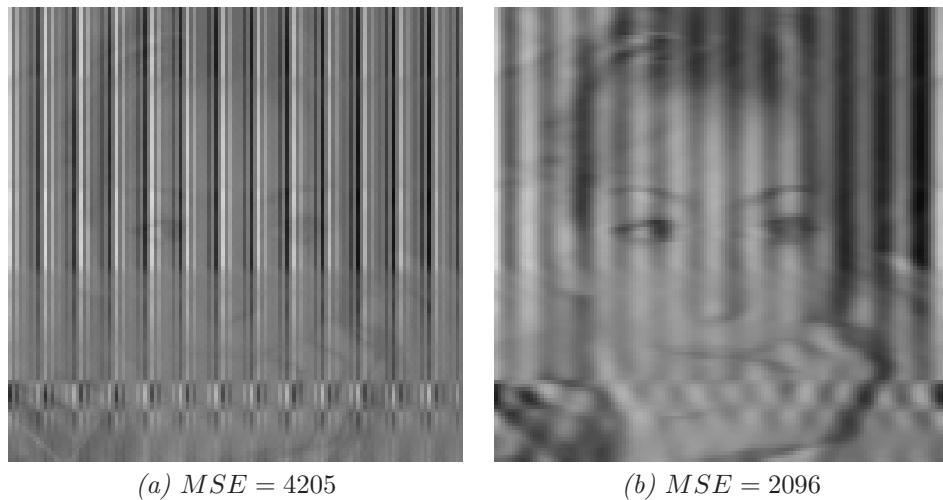


Figure 5.8: Images obtained by mapping the full range of restored values to $[0, 255]$. They should be compared with 5.5e and 5.5f, respectively, as those were produced by clipping the out of range values. We note that scaling linearly the full range of values tends to produce images of low contrast.

Can we define a filter that will automatically take into consideration the noise in the blurred image?

Yes. One such filter is the **Wiener filter**, which treats the image restoration problem as an estimation problem and solves it in the least square error sense. This will be discussed in the next section.

Example 5.17

Restore the blurred and noisy images shown in figure 5.9. They were produced by adding white Gaussian noise with standard deviation 10 or 20 to the blurred images 5.5b and 5.5c.

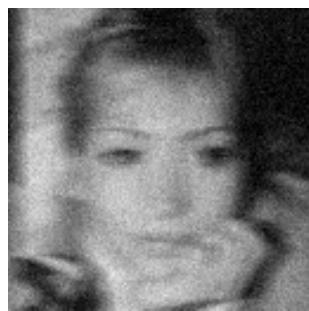
The results shown in figures 5.9d–5.9f are really very bad: High frequencies dominated by noise are amplified by the filter to the extent that they dominate the restored image. When the filter is truncated beyond its first 0, the results, shown in figures 5.9g–5.9i are reasonable.



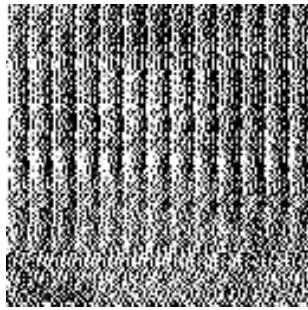
(a) Realistic blurring with added Gaussian noise with $\sigma = 10$.
 $MSE = 994$



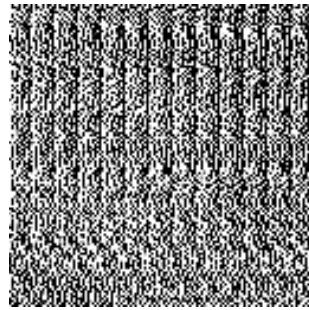
(b) Realistic blurring with added Gaussian noise with $\sigma = 20$.
 $MSE = 1277$



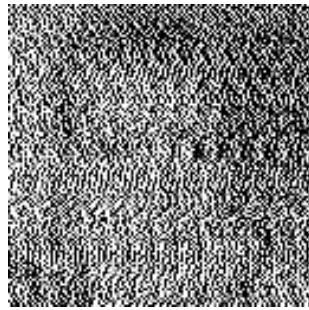
(c) Blurring (cylindrical boundary condition) plus Gaussian noise ($\sigma = 10$)
 $MSE = 1364$



(d) Inverse filtering of (a), omitting division by 0
 $MSE = 15711$



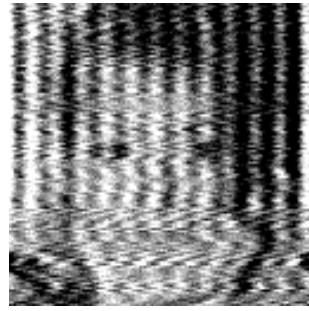
(e) Inverse filtering of (b), omitting division by 0
 $MSE = 18010$



(f) Inverse filtering of (c), omitting division by 0
 $MSE = 14673$



(g) Inverse filtering of (a), but omitting division with terms beyond the first 0
 $MSE = 2827$



(h) Inverse filtering of (b), but omitting division with terms beyond the first 0
 $MSE = 3861$



(i) Inverse filtering of (c), but omitting division with terms beyond the first 0
 $MSE = 698$

Figure 5.9: Restoring the image of noisy Dionisia, with inverse filtering. All MSE values have been computed in relation to 5.5a.

5.2 Homogeneous linear image restoration: Wiener filtering

How can we express the problem of image restoration as a least square error estimation problem?

If $\hat{f}(\mathbf{r})$ is an estimate of the original undegraded image $f(\mathbf{r})$, we wish to calculate $\hat{f}(\mathbf{r})$ so that the norm of the residual image $f(\mathbf{r}) - \hat{f}(\mathbf{r})$ is minimal over all possible versions of image $f(\mathbf{r})$. This is equivalent to saying that we wish to identify $\hat{f}(\mathbf{r})$ which minimises:

$$e^2 \equiv E \left\{ [f(\mathbf{r}) - \hat{f}(\mathbf{r})]^2 \right\} \quad (5.65)$$

Can we find a linear least squares error solution to the problem of image restoration?

Yes, by imposing the constraint that the solution $\hat{f}(\mathbf{r})$ is a linear function of the degraded image $g(\mathbf{r})$. This constraint is valid if the process of image degradation is assumed to be linear, ie modelled by an equation like (5.1), on page 396. Clearly, if this assumption is wrong, the solution found this way will not give the absolute minimum of e^2 but it will make e^2 minimum *within the limitations of the constraints imposed*.

The solution of a linear problem is also linear, so we may express $\hat{f}(\mathbf{r})$ as a linear function of the grey levels of the degraded image, ie

$$\hat{f}(\mathbf{r}) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r}, \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \quad (5.66)$$

where $m(\mathbf{r}, \mathbf{r}')$ is the function we want to determine and which gives the weight by which the grey level value of the degraded image g at position \mathbf{r}' affects the value of the estimated image \hat{f} at position \mathbf{r} . If the degradation process is further modelled as a homogeneous process (ie modelled by an equation like (5.2)), then the solution may also be obtained by a homogeneous process and the weighting function $m(\mathbf{r}, \mathbf{r}')$ will depend only on the difference of \mathbf{r} and \mathbf{r}' as opposed to depending on them separately. In that case (5.66) may be written as:

$$\hat{f}(\mathbf{r}) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \quad (5.67)$$

This equation means that we wish to identify a filter $m(\mathbf{r})$ with which to convolve the degraded image $g(\mathbf{r}')$ in order to obtain an estimate $\hat{f}(\mathbf{r})$ of the undegraded image $f(\mathbf{r})$. In order to avoid the failures of inverse filtering, we have to account for the noise, and so instead of equation (5.66), we should model the degradation process as

$$g(\mathbf{r}) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\mathbf{r} - \mathbf{r}') f(\mathbf{r}') d\mathbf{r}' + \nu(\mathbf{r}) \quad (5.68)$$

where $g(\mathbf{r})$, $f(\mathbf{r})$ and $\nu(\mathbf{r})$ are considered to be random fields, with $\nu(\mathbf{r})$ being the noise field.

What is the linear least mean square error solution of the image restoration problem?

If $\hat{M}(u, v)$ is the Fourier transform of filter $m(\mathbf{r})$, it can be shown (see Box 5.3, on page 428) that the linear solution of equation (5.65) can be obtained if

$$\hat{M}(u, v) = \frac{\hat{H}^*(u, v)}{|\hat{H}(u, v)|^2 + \frac{S_{\nu\nu}(u, v)}{S_{ff}(u, v)}} \quad (5.69)$$

where $\hat{H}(u, v)$ is the Fourier transform of the point spread function of the degradation process, $\hat{H}^*(u, v)$ its complex conjugate, $S_{\nu\nu}(u, v)$ is the spectral density of the noise field and $S_{gg}(u, v)$ is the spectral density of the undegraded image. $\hat{M}(u, v)$ is known as the **Wiener filter** for image restoration.

Box 5.1. The least squares error solution

We shall show that if $m(\mathbf{r} - \mathbf{r}')$ satisfies

$$E \left\{ \left[f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right] g(\mathbf{s}) \right\} = 0 \quad (5.70)$$

then it minimises the error defined by equation (5.65).

Intuitively, we can see that this is true, because equation (5.70) says that the error of the estimation (expressed by the quantity inside the square bracket) is orthogonal to the data. This is what least squares error estimation does. (Remember how when we fit a least squares error line to some points, we minimise the sum of the distances of these points from the line.) Next, we shall prove this mathematically.

If we substitute equation (5.67) into equation (5.65) we obtain:

$$e^2 = E \left\{ \left[f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right]^2 \right\} \quad (5.71)$$

Consider now another function $m'(\mathbf{r})$ which does not satisfy (5.70). We shall show that $m'(\mathbf{r})$ when used for the restoration of the image, will produce an estimate $\hat{f}'(\mathbf{r})$ with error e'^2 , greater than the error of the estimate obtained by $m(\mathbf{r})$ which does satisfy (5.70). Error e'^2 will be:

$$e'^2 \equiv E \left\{ \left[f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m'(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right]^2 \right\} \quad (5.72)$$

Inside the integrand in (5.72) we add to and subtract from $m(\mathbf{r} - \mathbf{r}')$ function $m'(\mathbf{r} - \mathbf{r}')$.

We split the integral into two parts and then expand the square:

$$\begin{aligned}
e'^2 &= E \left\{ \left[f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} [m'(\mathbf{r} - \mathbf{r}') + m(\mathbf{r} - \mathbf{r}') - m(\mathbf{r} - \mathbf{r}')] g(\mathbf{r}') d\mathbf{r}' \right]^2 \right\} \\
&= E \left\{ \left[\left(f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right) \right. \right. \\
&\quad \left. \left. + \left(\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} [m(\mathbf{r} - \mathbf{r}') - m'(\mathbf{r} - \mathbf{r}')] g(\mathbf{r}') d\mathbf{r}' \right) \right]^2 \right\} \\
&= E \underbrace{\left\{ \left(f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right)^2 \right\}}_{e^2} \\
&\quad + E \underbrace{\left\{ \left(\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} [m(\mathbf{r} - \mathbf{r}') - m'(\mathbf{r} - \mathbf{r}')] g(\mathbf{r}') d\mathbf{r}' \right)^2 \right\}}_{\text{a non-negative number}} \\
&\quad + 2E \left\{ \left(f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right) \right. \\
&\quad \left. \underbrace{\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} [m(\mathbf{r} - \mathbf{r}') - m'(\mathbf{r} - \mathbf{r}')] g(\mathbf{r}') d\mathbf{r}' \right\} \\
&\quad \text{rename } \mathbf{r}' \rightarrow \mathbf{s} \tag{5.73}
\end{aligned}$$

The expectation value of the first term is e^2 and clearly the expectation value of the second term is a non-negative number. In the last term, in the second factor, change the dummy variable of integration from \mathbf{r}' to \mathbf{s} . The factor with the expectation operator in the last term on the right-hand side of (5.73) may then be written as:

$$E \left\{ \left(f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right) \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} [m(\mathbf{r} - \mathbf{s}) - m'(\mathbf{r} - \mathbf{s})] g(\mathbf{s}) d\mathbf{s} \right\} \tag{5.74}$$

The first factor in the above expression does not depend on \mathbf{s} and thus it can be put inside the double integral sign:

$$E \left\{ \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \left(f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right) [m(\mathbf{r} - \mathbf{s}) - m'(\mathbf{r} - \mathbf{s})] g(\mathbf{s}) d\mathbf{s} \right\} \tag{5.75}$$

The difference $[m(\mathbf{r} - \mathbf{s}) - m'(\mathbf{r} - \mathbf{s})]$ is not a random field but the difference of two specific functions. If we exchange the order of integrating and taking the expectation value, the expectation is not going to affect this factor, so this term will become:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} E \left\{ \left[f(\mathbf{r}) - \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') g(\mathbf{r}') d\mathbf{r}' \right] g(\mathbf{s}) \right\} [m(\mathbf{r} - \mathbf{s}) - m'(\mathbf{r} - \mathbf{s})] d\mathbf{s} \tag{5.76}$$

However, the expectation value in the above term is 0 according to (5.70), so from (5.73) we get:

$$e'^2 = e^2 + \text{a non-negative term} \quad (5.77)$$

We conclude that the error of the restoration created with the $m'(\mathbf{r})$ function is greater than or equal to the error of the restoration created with $m(\mathbf{r})$. So, $m(\mathbf{r})$, that satisfies equation (5.70), minimises the error defined by equation (5.65).

Example B5.18

If $\hat{F}(u, v)$, $\hat{H}(u, v)$ and $\hat{G}(u, v)$ are the Fourier transforms of real functions $f(\mathbf{r})$, $h(\mathbf{r})$ and $g(\mathbf{r})$ respectively, and

$$g(\mathbf{r}) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\mathbf{t} - \mathbf{r}) f(\mathbf{t}) d\mathbf{t} \quad (5.78)$$

show that

$$\hat{G}(u, v) = \hat{H}^*(u, v) \hat{F}(u, v) \quad (5.79)$$

where $\hat{H}^*(u, v)$ is the complex conjugate of $\hat{H}(u, v)$.

Assume that $\mathbf{r} = (x, y)$ and $\mathbf{t} = (\tilde{x}, \tilde{y})$. The Fourier transforms of the three functions are:

$$\hat{G}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(x, y) e^{-j(ux+vy)} dx dy \quad (5.80)$$

$$\hat{F}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) e^{-j(ux+vy)} dx dy \quad (5.81)$$

$$\hat{H}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x, y) e^{-j(ux+vy)} dx dy \quad (5.82)$$

The complex conjugate of $\hat{H}(u, v)$ is

$$\hat{H}^*(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x, y) e^{j(ux+vy)} dx dy \quad (5.83)$$

since $h(x, y)$ is real.

Let us substitute $g(x, y)$ from (5.78) into the right-hand side of (5.80):

$$\hat{G}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\tilde{x} - x, \tilde{y} - y) f(\tilde{x}, \tilde{y}) d\tilde{x} d\tilde{y} e^{-j(ux+vy)} dx dy \quad (5.84)$$

We define new variables of integration $s_1 \equiv \tilde{x} - x$ and $s_2 \equiv \tilde{y} - y$ to replace integration over x and y . Since $dx = -ds_1$ and $dy = -ds_2$, $dxdy = ds_1ds_2$. Also, as the limits of both s_1 and s_2 are from $+\infty$ to $-\infty$, we can change their order without worrying about a change of sign:

$$\hat{G}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(s_1, s_2) f(\tilde{x}, \tilde{y}) d\tilde{x} d\tilde{y} e^{-j(u(\tilde{x}-s_1)+v(\tilde{y}-s_2))} ds_1 ds_2 \quad (5.85)$$

The two double integrals are separable:

$$\hat{G}(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(s_1, s_2) e^{j(us_1+vs_2)} ds_1 ds_2 \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(\tilde{x}, \tilde{y}) e^{-j(u\tilde{x}+v\tilde{y})} d\tilde{x} d\tilde{y} \quad (5.86)$$

On the right-hand side of this equation we recognise the product of $\hat{F}(u, v)$ and $\hat{H}^*(u, v)$ from equations (5.81) and (5.83), respectively. Therefore, equation (5.79) has been proven.

Example B5.19

If \hat{R}_{ff} and \hat{R}_{gf} are the Fourier transforms of the autocorrelation function of field $f(\mathbf{r})$, and the cross-correlation function between fields $g(\mathbf{r})$ and $f(\mathbf{r})$, respectively, related by equation (5.68), show that

$$\hat{R}_{gf}(u, v) = \hat{H}^*(u, v) \hat{R}_{ff}(u, v) \quad (5.87)$$

where $\hat{H}^*(u, v)$ is the complex conjugate of the Fourier transform of function $h(\mathbf{r})$ and (u, v) are the frequencies along axes x and y , with $\mathbf{r} = (x, y)$. Assume that $f(\mathbf{r})$ and $\nu(\mathbf{r})$ are uncorrelated and the noise is zero-mean.

To create the cross-correlation between random fields $g(\mathbf{r})$ and $f(\mathbf{r})$, we multiply both sides of equation (5.68), on page 419, with $f(\mathbf{r} + \mathbf{s})$ and take the expectation value:

$$\underbrace{E\{g(\mathbf{r})f(\mathbf{r} + \mathbf{s})\}}_{R_{gf}(\mathbf{s})} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\mathbf{r} - \mathbf{r}') \underbrace{E\{f(\mathbf{r}')f(\mathbf{r} + \mathbf{s})\}}_{R_{ff}(\mathbf{r} + \mathbf{s} - \mathbf{r}')} d\mathbf{r}' + E\{f(\mathbf{r} + \mathbf{s})\nu(\mathbf{r})\} \quad (5.88)$$

The last term in the above equation is 0, because, due to the uncorrelatedness of $f(\mathbf{r})$ and $\nu(\mathbf{r})$, it may be written as $E\{f(\mathbf{r} + \mathbf{s})\}E\{\nu(\mathbf{r})\}$ and $E\{\nu(\mathbf{r})\} = 0$. Therefore:

$$R_{gf}(\mathbf{s}) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\mathbf{r} - \mathbf{r}') R_{ff}(\mathbf{r} - \mathbf{r}' + \mathbf{s}) d\mathbf{r}' \quad (5.89)$$

Let us define a new vector of integration $\mathbf{r} - \mathbf{r}' + \mathbf{s} \equiv \tilde{\mathbf{s}}$. Because each vector represents two independent variables of integration, $d\mathbf{r}' = d\tilde{\mathbf{s}}$, ie no sign change, and also the

change in the limits of integration will not introduce any change of sign, (5.89) may be written as:

$$R_{gf}(\mathbf{s}) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\tilde{\mathbf{s}} - \mathbf{s}) R_{ff}(\tilde{\mathbf{s}}) d\tilde{\mathbf{s}} \quad (5.90)$$

According to (5.79), this equation may be written as

$$\hat{R}_{gf}(u, v) = \hat{H}^*(u, v) \hat{R}_{ff}(u, v) \quad (5.91)$$

in terms of Fourier transforms, where (u, v) are the frequencies along the two axes.

Example B5.20

If $\hat{R}_{gg}(u, v)$, $\hat{R}_{fg}(u, v)$ and $\hat{R}_{\nu g}(u, v)$ are the Fourier transforms of the autocorrelation function of the homogeneous random field $g(x, y)$, and the cross-correlation functions between the homogeneous random fields $f(x, y)$ and $g(x, y)$, and $\nu(x, y)$ and $g(x, y)$, respectively, $\hat{H}(u, v)$ is the Fourier transform of $h(x, y)$, and

$$g(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) f(\tilde{x}, \tilde{y}) d\tilde{x} d\tilde{y} + \nu(x, y) \quad (5.92)$$

show that

$$\hat{R}_{gg}(u, v) = \hat{H}^*(u, v) \hat{R}_{fg}(u, v) + \hat{R}_{\nu g}(u, v) \quad (5.93)$$

If we multiply both sides of equation (5.92) with $g(x + s_1, y + s_2)$ and take the ensemble average over all versions of random field $g(x, y)$, we obtain:

$$\begin{aligned} E\{g(x, y)g(x + s_1, y + s_2)\} &= \\ E\left\{g(x + s_1, y + s_2) \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) f(\tilde{x}, \tilde{y}) d\tilde{x} d\tilde{y}\right\} & \\ + E\{g(x + s_1, y + s_2)\nu(x, y)\} & \end{aligned} \quad (5.94)$$

Since $g(x, y)$ is a homogeneous random field, we recognise on the left-hand side the autocorrelation function of g , $R_{gg}(s_1, s_2)$, with shifting arguments s_1 and s_2 . The noise random field $\nu(x, y)$ is also homogeneous, so the last term on the right-hand side is the cross-correlation $R_{\nu g}(s_1, s_2)$ between random fields g and ν . Further, $g(x + s_1, y + s_2)$ does not depend on the variables of integration \tilde{x} and \tilde{y} , so it may go inside the integral in the first term of the right-hand side:

$$\begin{aligned} R_{gg}(s_1, s_2) &= \\ E\left\{\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) f(\tilde{x}, \tilde{y}) g(x + s_1, y + s_2) d\tilde{x} d\tilde{y}\right\} + R_{\nu g}(s_1, s_2) & \end{aligned} \quad (5.95)$$

Taking the expectation value and integrating are two linear operations that may be interchanged. The expectation operator operates only on random fields f and g , while it leaves unaffected function h . Therefore, we may write:

$$R_{gg}(s_1, s_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) E\{f(\tilde{x}, \tilde{y})g(x + s_1, y + s_2)\} d\tilde{x}d\tilde{y} + R_{\nu g}(s_1, s_2) \quad (5.96)$$

We recognise inside the integral the cross-correlation R_{fg} between fields f and g , calculated for shifting values $x + s_1 - \tilde{x}$ and $y + s_2 - \tilde{y}$:

$$R_{gg}(s_1, s_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) R_{fg}(x - \tilde{x} + s_1, y - \tilde{y} + s_2) d\tilde{x}d\tilde{y} + R_{\nu g}(s_1, s_2) \quad (5.97)$$

We may define new variables of integration: $x - \tilde{x} \equiv \alpha$, $y - \tilde{y} \equiv \beta$. Then $d\tilde{x}d\tilde{y} = d\alpha d\beta$, and the change of sign of the two sets of limits of integration cancel each other out:

$$R_{gg}(s_1, s_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\alpha, \beta) R_{fg}(\alpha + s_1, \beta + s_2) d\alpha d\beta + R_{\nu g}(s_1, s_2) \quad (5.98)$$

We may change variables of integration again, to $w \equiv \alpha + s_1$, $z \equiv \beta + s_2$. Then $\alpha = w - s_1$, $\beta = z - s_2$, $d\alpha d\beta = dw dz$ and the limits of integration are not affected:

$$R_{gg}(s_1, s_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(w - s_1, z - s_2) R_{fg}(w, z) dw dz + R_{\nu g}(s_1, s_2) \quad (5.99)$$

If we take the Fourier transform of both sides of this expression, and make use of (5.79), we may write:

$$\hat{R}_{gg}(u, v) = \hat{H}^*(u, v) \hat{R}_{fg}(u, v) + \hat{R}_{\nu g}(u, v) \quad (5.100)$$

Example B5.21

If $\hat{R}_{ff}(u, v)$ and $\hat{R}_{fg}(u, v)$ are the Fourier transforms of the autocorrelation function of the homogeneous random field $f(x, y)$, and the cross-correlation function between the homogeneous random fields $f(x, y)$ and $g(x, y)$, respectively, $\hat{H}(u, v)$ is the Fourier transform of $h(x, y)$, and

$$g(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) f(\tilde{x}, \tilde{y}) d\tilde{x}d\tilde{y} + \nu(x, y) \quad (5.101)$$

where $\nu(x, y)$ is a zero-mean homogeneous random field uncorrelated with $f(x, y)$, show that:

$$\hat{R}_{fg}(u, v) = \hat{H}(u, v)\hat{R}_{ff}(u, v) \quad (5.102)$$

We multiply both sides of (5.101) with $f(x - s_1, y - s_2)$ and take the expectation value. The reason we multiply with $f(x - s_1, y - s_2)$ and not with $f(x + s_1, y + s_2)$ is in order to be consistent with example 5.19. In that example, we formed the shifting arguments of R_{gf} by subtracting the arguments of g from the arguments of f . Following the same convention here will result in positive arguments for R_{fg} .

With this proviso, on the left-hand side of the resultant equation then we shall have the cross-correlation between random fields $f(x, y)$ and $g(x, y)$, $R_{fg}(s_1, s_2)$. On the right-hand side we exchange the order of expectation and integration and observe that the expectation operator is applied only to the random components:

$$\begin{aligned} R_{fg}(s_1, s_2) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) E\{f(\tilde{x}, \tilde{y})f(x - s_1, y - s_2)\} d\tilde{x}d\tilde{y} \\ &\quad + E\{f(x - s_1, y - s_2)\nu(x, y)\} \end{aligned} \quad (5.103)$$

Random fields f and ν are uncorrelated and ν has 0 mean. Then:

$$E\{f(x - s_1, y - s_2)\nu(x, y)\} = E\{f(x - s_1, y - s_2)\} E\{\nu(x, y)\} = 0 \quad (5.104)$$

Inside the integral on the right-hand side of (5.103) we recognise the autocorrelation function of random field f , computed for shifting argument $(\tilde{x} - x + s_1, \tilde{y} - y + s_2)$. The reason we subtract the arguments of $f(x - s_1, y - s_2)$ from the arguments of $f(\tilde{x}, \tilde{y})$, and not the other way round, is because on the left-hand side we subtracted the arguments of the “new” function from the arguments of the existing one (ie the arguments of $f(x - s_1, y - s_2)$ from the arguments of $g(x, y)$) to form R_{fg} , and we adopt the same convention here. So, we have:

$$R_{fg}(s_1, s_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) R_{ff}(\tilde{x} - x + s_1, \tilde{y} - y + s_2) d\tilde{x}d\tilde{y} \quad (5.105)$$

We define new variables of integration $\alpha \equiv x - \tilde{x}$, $\beta \equiv y - \tilde{y}$:

$$R_{fg}(s_1, s_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\alpha, \beta) R_{ff}(s_1 - \alpha, s_2 - \beta) d\alpha d\beta \quad (5.106)$$

The above equation is a straightforward convolution, and in terms of Fourier transforms it is written as (5.102).

Example B5.22

If $\hat{R}_{\nu\nu}(u, v)$ and $\hat{R}_{\nu g}(u, v)$ are the Fourier transforms of the autocorrelation function of the homogeneous random field $\nu(x, y)$, and the cross-correlation function between the homogeneous random fields $\nu(x, y)$ and $g(x, y)$, and

$$g(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) f(\tilde{x}, \tilde{y}) d\tilde{x} d\tilde{y} + \nu(x, y) \quad (5.107)$$

where $h(x, y)$ is some real function, and $f(x, y)$ is a homogeneous random field uncorrelated with $\nu(x, y)$, which has zero mean, show that:

$$\hat{R}_{\nu g}(u, v) = \hat{R}_{\nu\nu}(u, v) \quad (5.108)$$

We multiply both sides of equation (5.107) with $\nu(x - s_1, y - s_2)$ and take the expectation value:

$$R_{\nu g}(s_1, s_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(x - \tilde{x}, y - \tilde{y}) E \{ f(\tilde{x}, \tilde{y}) \nu(x - s_1, y - s_2) \} d\tilde{x} d\tilde{y} + R_{\nu\nu}(s_1, s_2) \quad (5.109)$$

The integral term vanishes because the two fields are uncorrelated and at least one of them has zero mean. Then, taking the Fourier transform of both sides yields (5.108).

Box 5.2. From the Fourier transform of the correlation functions of images to their spectral densities

The autocorrelation and the cross-correlation functions that were computed in examples 5.19–5.22 were computed in the ensemble sense. If we invoke the ergodicity assumption, we may say that these correlations are the same as the spatial correlations of the corresponding images (treated as random fields). Then, we may apply the Wiener-Khinchine theorem (see Box 4.5, on page 325) to identify the Fourier transforms of the correlation functions with the corresponding power spectral densities of the images. Thus, equations (5.87), (5.93), (5.102) and (5.108) may be replaced with

$$S_{gf}(u, v) = \hat{H}^*(u, v) S_{ff}(u, v) \quad (5.110)$$

$$S_{gg}(u, v) = \hat{H}^*(u, v) S_{fg}(u, v) + S_{\nu g}(u, v) \quad (5.111)$$

$$S_{fg}(u, v) = \hat{H}(u, v) S_{ff}(u, v) \quad (5.112)$$

$$S_{\nu g}(u, v) = S_{\nu\nu}(u, v) \quad (5.113)$$

where $S_{fg}(u, v)$, $S_{gf}(u, v)$ and $S_{\nu g}(u, v)$ are the cross-spectral densities between the real and the observed image, the observed and the real image and the noise field and the observed image, respectively, with the convention that the arguments of the first field are subtracted from the second to yield the shifting variable. $S_{gg}(u, v)$, $S_{ff}(u, v)$ and $S_{\nu\nu}(u, v)$ are the power spectral densities of the observed image, the unknown image and the noise field, respectively.

In general, however, the fields are not ergodic. Then, it is *postulated* that the Fourier transforms of the auto- and cross-correlation functions are the spectral and cross-spectral densities respectively, of the corresponding random fields.

The above equations are used in the development of the Wiener filter, and so the ergodicity assumption is tacitly made in this derivation.

Box 5.3. Derivation of the Wiener filter

In order to identify filter $m(\mathbf{r} - \mathbf{r}')$ in equation (5.67), we must make some extra assumption: the noise and the true image are uncorrelated and at least one of the two has zero mean. This assumption is a plausible one: we expect the process that gives rise to the image to be entirely different from the process that gives rise to the noise. Further, if the noise has a biasing, ie it does not have zero mean, we can always identify and subtract this biasing to make it have zero mean.

Since $f(\mathbf{r})$ and $\nu(\mathbf{r})$ are uncorrelated and since $E\{\nu(\mathbf{r})\} = 0$, we may write:

$$E\{f(\mathbf{r})\nu(\mathbf{r})\} = E\{f(\mathbf{r})\}E\{\nu(\mathbf{r})\} = 0 \quad (5.114)$$

In Box 5.1, on page 420, we saw that filter $m(\mathbf{r})$, that minimises (5.65), has to satisfy equation (5.70). This equation may be written as

$$E\{f(\mathbf{r})g(\mathbf{s})\} - E\left\{\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}')g(\mathbf{r}')g(\mathbf{s})d\mathbf{r}'\right\} = 0 \quad (5.115)$$

where $g(\mathbf{s})$ has gone inside the integral because it does not depend on \mathbf{r}' . The expectation operator applied to the second term operates really only on the random fields $g(\mathbf{r}')$ and $g(\mathbf{s})$. Therefore, we may write:

$$\underbrace{E\{f(\mathbf{r})g(\mathbf{s})\}}_{R_{gf}(\mathbf{r}, \mathbf{s})} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') \underbrace{E\{g(\mathbf{r}')g(\mathbf{s})\}}_{R_{gg}(\mathbf{r}', \mathbf{s})} d\mathbf{r}' \quad (5.116)$$

In this expression we recognise the definitions of the autocorrelation and cross-correlation functions of the random fields, so we may write:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') R_{gg}(\mathbf{r}', \mathbf{s}) d\mathbf{r}' = R_{gf}(\mathbf{r}, \mathbf{s}) \quad (5.117)$$

We have seen that for homogeneous random fields, the correlation function can be written as a function of the difference of its two arguments (example 3.24, page 196). So:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\mathbf{r} - \mathbf{r}') R_{gg}(\mathbf{r}' - \mathbf{s}) d\mathbf{r}' = R_{gf}(\mathbf{r} - \mathbf{s}) \quad (5.118)$$

We introduce some new variables: $\mathbf{r}' - \mathbf{s} \equiv \mathbf{t}$ and $\mathbf{r} - \mathbf{s} \equiv \boldsymbol{\tau}$. Therefore, $d\mathbf{r}' = d\mathbf{t}$ and $\mathbf{r} - \mathbf{r}' = \boldsymbol{\tau} - \mathbf{t}$. Then:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} m(\boldsymbol{\tau} - \mathbf{t}) R_{gg}(\mathbf{t}) d\mathbf{t} = R_{gf}(\boldsymbol{\tau}) \quad (5.119)$$

This is a convolution between the autocorrelation function of the degraded image and the sought filter. According to the convolution theorem, the effect is equivalent to the multiplication of the Fourier transforms of the two functions:

$$\hat{M}(u, v) S_{gg}(u, v) = S_{gf}(u, v) \quad (5.120)$$

Here S_{gg} and S_{fg} are the spectral density of the degraded image and the cross-spectral density of the undegraded and degraded images, respectively, ie the Fourier transforms of the autocorrelation function of g and cross-correlation of f and g functions, respectively (see Box 5.2, on page 427). Therefore:

$$\hat{M}(u, v) = \frac{S_{gf}(u, v)}{S_{gg}(u, v)} \quad (5.121)$$

The Fourier transform of the optimal restoration filter, which minimises the mean square error between the real image and the reconstructed one, is equal to the ratio of the cross-spectral density of the degraded image and the true image, over the spectral density of the degraded image.

If we substitute from equations (5.112) and (5.113) into equation (5.111), we obtain:

$$S_{gg}(u, v) = S_{ff}(u, v) |\hat{H}(u, v)|^2 + S_{\nu\nu}(u, v) \quad (5.122)$$

If we substitute then equations (5.110) and (5.122) into (5.121), we obtain

$$\hat{M}(u, v) = \frac{\hat{H}^*(u, v) S_{ff}(u, v)}{S_{ff}(u, v) |\hat{H}(u, v)|^2 + S_{\nu\nu}(u, v)} \quad (5.123)$$

or:

$$\hat{M}(u, v) = \frac{\hat{H}^*(u, v)}{|\hat{H}(u, v)|^2 + \frac{S_{\nu\nu}(u, v)}{S_{ff}(u, v)}} \quad (5.124)$$

This equation gives the Fourier transform of the **Wiener filter** for image restoration.

What is the relationship between Wiener filtering and inverse filtering?

If we multiply the numerator and denominator of (5.124) with $\hat{H}(u, v)$, we obtain:

$$\hat{M}(u, v) = \frac{1}{\hat{H}(u, v)} \times \frac{|\hat{H}(u, v)|^2}{|\hat{H}(u, v)|^2 + \frac{S_{\nu\nu}(u, v)}{S_{ff}(u, v)}} \quad (5.125)$$

In the absence of noise, $S_{\nu\nu}(u, v) = 0$ and the Wiener filter given by equation (5.125) becomes the inverse frequency response function filter of equation (5.38), on page 407. So, the linear least square error approach simply determines a correction factor with which the inverse frequency response function of the degradation process has to be multiplied before it is used as a filter, so that the effect of noise is taken care of.

How can we determine the spectral density of the noise field?

We usually make the assumption that the noise is white, ie that

$$S_{\nu\nu}(u, v) = \text{constant} = S_{\nu\nu}(0, 0) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} R_{\nu\nu}(x, y) dx dy \quad (5.126)$$

Since the noise is assumed to be ergodic, we can obtain $R_{\nu\nu}(x, y)$ from a single pure noise image, the recorded image $g(x, y)$, when the imaged surface is uniformly coloured of some intermediate brightness level. Ideally, we should record the noise field for $f(x, y) = 0$. However, as no negative values are recorded by the sensor, the distribution of the noise field in this case will be distorted. For example, if the noise were zero-mean Gaussian, it would not appear as such because all negative values most likely would have been mapped to the 0 value. If, however, $f(x, y) = 120$, for every (x, y) , say, we can easily remove the mean grey value from all recorded values and proceed to work out the statistics of the noise field.

How can we possibly use Wiener filtering, if we know nothing about the statistical properties of the unknown image?

If we do not know anything about the statistical properties of the image we want to restore, ie we do not know $S_{ff}(u, v)$, we may replace the term $\frac{S_{\nu\nu}(u, v)}{S_{ff}(u, v)}$ in equation (5.125) with a constant Γ and experiment with various values of Γ .

This is clearly rather an oversimplification, as the ratio $\frac{S_{\nu\nu}(u, v)}{S_{ff}(u, v)}$ is a function of (u, v) and not a constant. So, we may try to estimate both the spectrum of the noise and the spectrum of the undegraded image from the spectrum of the degraded image. Let us assume for simplicity that all functions that appear in filter (5.69), on page 420, are functions of $\sqrt{u^2 + v^2} \equiv \omega$. Let us also say that the first zero of $\hat{H}(u, v)\hat{H}(u, v)$ happens for $\omega_0 \equiv \sqrt{u_0^2 + v_0^2}$. Then there will be a strip of frequencies around frequency ω_0 , inside which $\hat{H}(u, v)$ stops being reliable and the noise effects become serious. Let us also consider two frequencies, ω_1 and ω_2 , such that for frequencies $\omega < \omega_1$, $\hat{H}(u, v)$ behaves well and we may use inverse filtering, while for frequencies $\omega > \omega_2$, the power spectrum we observe is totally dominated by noise. This assumption is valid, as long as the noise is assumed white, and so it has a constant power spectrum, while the unknown image is assumed to have a spectrum that decays fast for high frequencies. We may then consider the power spectrum of the observed image beyond

frequency ω_2 and use it to estimate the power spectrum of the noise, by taking, for example, its average over all frequencies beyond ω_2 . Further, we may make the assumption that the power spectrum of the unknown image decays exponentially beyond frequency ω_1 . We may apply then (5.122) at frequency ω_1 to work out the model parameters for S_{ff} :

$$S_{gg}(\omega_1) = S_{ff}(\omega_1)|\hat{H}(\omega_1)|^2 + S_{\nu\nu} \quad (5.127)$$

Note that $S_{\nu\nu}$ now is assumed to be a known constant. $S_{gg}(\omega_1)$ may be estimated from the observed degraded image, and $\hat{H}(\omega_1)$ is also assumed known. Then:

$$S_{ff}(\omega_1) = \frac{S_{gg}(\omega_1) - S_{\nu\nu}}{|\hat{H}(\omega_1)|^2} \quad (5.128)$$

Assuming an exponential decay for $S_{ff}(\omega)$ when $\omega > \omega_1$, we may write

$$S_{ff}(\omega) = S_{ff}(\omega_1)e^{-\alpha(\omega-\omega_1)} \quad (5.129)$$

where α is some positive constant. We may then define a filter as follows:

$$\hat{M}(\omega) = \begin{cases} \frac{1}{\hat{H}(\omega)} & \text{if } \omega < \omega_1 \\ \frac{\hat{H}^*(\omega)}{|\hat{H}(\omega)|^2 + S(\omega)} & \text{if } \omega \geq \omega_1 \end{cases} \quad (5.130)$$

Here:

$$\begin{aligned} S(\omega) &\equiv \frac{S_{\nu\nu}}{S_{ff}(\omega_1)e^{-\alpha(\omega-\omega_1)}} - \frac{S_{\nu\nu}}{S_{ff}(\omega_1)} \\ &= \frac{S_{\nu\nu}}{S_{ff}(\omega_1)} \left[e^{\alpha(\omega-\omega_1)} - 1 \right] \end{aligned} \quad (5.131)$$

Note that when $\omega = \omega_1$, the Wiener branch of this filter coincides with the inverse filter. For $\omega \gg \omega_1$ the -1 in the denominator of the second branch of the filter is negligible in comparison with the exponential term, and so the filter behaves like the Wiener filter. Parameter α should be selected so that $S_{ff}(\omega_1)e^{-\alpha(\omega_2-\omega_1)} \ll S_{\nu\nu}$.

Figure 5.10 shows schematically how this filter is defined.

How do we apply Wiener filtering in practice?

In summary, Wiener filtering may be implemented as follows.

Step 0: Somehow work out the Fourier transform of the point spread function of the degradation process, $\hat{H}(u, v)$.

Step 1: Take the Fourier transform of the observed degraded image, $\hat{G}(u, v)$.

Step 2: Select a value for constant Γ and multiply $\hat{G}(u, v)$ point by point with

$$\hat{M}(u, v) = \frac{\hat{H}^*(u, v)}{|\hat{H}(u, v)|^2 + \Gamma} \quad (5.132)$$

Step 3: Take the inverse Fourier transform to recover the restored image.

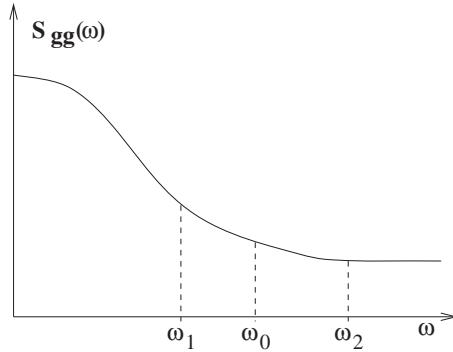


Figure 5.10: $S_{gg}(\omega)$ is the power spectrum of the observed image. Let us say that the frequency response function with which we model the degradation process has its first 0 at frequency ω_0 . We observe that for frequencies beyond ω_2 the spectrum flattens out, and, therefore, under the assumption of white noise, we may infer that at those frequencies the spectrum is dominated by the noise. We may thus compute the constant power spectrum of the noise, $S_{\nu\nu}$, by averaging the observed values of $S_{gg}(\omega)$ beyond frequency ω_2 . Let us accept that for frequencies smaller than ω_1 , inverse filtering may be used. We must, however, use Wiener filtering for frequencies around ω_0 .

If we wish to use the filter given by (5.131), we must use the following algorithm.

Step 0: Somehow work out the Fourier transform of the point spread function of the degradation process, $\hat{H}(u, v)$.

Step 1: Identify the frequencies u_0 and v_0 which correspond to the first zeros of $\hat{H}(u, v)$.

Step 2: Take the Fourier transform of the observed degraded image, $\hat{G}(u, v)$.

Step 3: Take the Fourier spectrum $S_{gg}(u, v)$ of the degraded image.

Step 4: Identify some frequencies $u_2 > u_0$ and $v_2 > v_0$, beyond which the spectrum is flat. Average the values of the spectrum for those frequencies to obtain $S_{\nu\nu}$.

Step 5: Identify some frequencies $u_1 < u_0$ and $v_1 < v_0$, compute $S_{gg}(u_1, v_1)$ and set:

$$S_{ff}(u_1, v_1) = \frac{S_{gg}(u_1, v_1) - S_{\nu\nu}}{|\hat{H}(u_1, v_1)|^2} \quad (5.133)$$

Step 6: Select a value for α so that

$$S_{ff}(u_1, v_1)e^{-\alpha(\sqrt{u_2^2+v_2^2}-\sqrt{u_1^2+v_1^2})} \simeq 0.1S_{\nu\nu} \quad (5.134)$$

Step 7: Multiply $\hat{G}(u, v)$ point by point with

$$\hat{M}(u, v) = \begin{cases} \frac{1}{\hat{H}(u, v)} & \text{if } u < u_1 \text{ and } v < v_1 \\ \frac{\hat{H}^*(u, v)}{|\hat{H}(u, v)|^2 + \frac{S_{\nu\nu}}{S_{ff}(u_1, v_1)} \left[e^{\alpha(\sqrt{u^2+v^2}-\sqrt{u_1^2+v_1^2})} - 1 \right]} & \text{if } u \geq u_1 \text{ or } v \geq v_1 \end{cases} \quad (5.135)$$

Step 8: Take the inverse Fourier transform to recover the restored image.

Example 5.23

Restore the blurred images of figures 5.5a, on page 414, 5.9a and 5.9b, on page 418, by using Wiener filtering.

From equation (5.50), on page 410, we have:

$$|\hat{H}(m, n)|^2 = \frac{1}{i_T^2 \sin^2 \frac{\pi m}{N}} \sin^2 \frac{i_T \pi m}{N} \quad (5.136)$$

We shall use the Wiener filter as given by equation (5.132), with the ratio of the spectral densities in the denominator replaced by a constant Γ :

$$\hat{M}(m, n) = \frac{\frac{1}{i_T} \frac{\sin(\frac{\pi m}{N} i_T)}{\sin \frac{\pi m}{N}} e^{j \frac{\pi m(i_T - 1)}{N}}}{\frac{1}{i_T^2} \frac{\sin^2 \frac{\pi m}{N} i_T}{\sin^2 \frac{\pi m}{N}} + \Gamma} \quad (5.137)$$

Or:

$$\hat{M}(m, n) = \frac{i_T \sin \frac{\pi m}{N} \sin \left(\frac{\pi m}{N} i_T \right)}{\sin^2 \left(\frac{\pi m}{N} i_T \right) + \Gamma i_T^2 \sin^2 \frac{\pi m}{N}} e^{j \frac{\pi m(i_T - 1)}{N}} \quad (5.138)$$

We must be careful for the case $m = 0$, when we have:

$$\hat{M}(0, n) = \frac{1}{1 + \Gamma} \quad \text{for } 0 \leq n \leq N - 1 \quad (5.139)$$

If we multiply with this function the Fourier transform of the blurred image, as defined by equation (5.54), on page 411, we obtain:

$$\hat{F}(m, n) = \frac{i_T \sin \frac{\pi m}{N} \sin \frac{i_T \pi m}{N} \sqrt{G_1^2(m, n) + G_2^2(m, n)}}{\sin^2 \frac{i_T \pi m}{N} + \Gamma i_T^2 \sin^2 \frac{\pi m}{N}} e^{j \left(\phi(m, n) + \frac{(i_T - 1)\pi m}{N} \right)} \quad (5.140)$$

For the case $m = 0$, we have:

$$\hat{F}(0, n) = \frac{\sqrt{G_1^2(0, n) + G_2^2(0, n)}}{1 + \Gamma} e^{j \phi(0, n)} \quad \text{for } 0 \leq n \leq N - 1 \quad (5.141)$$

The real and the imaginary parts of $\hat{F}(m, n)$ are given by:

$$\begin{aligned}
F_1(m, n) &= \frac{i_T \sin \frac{\pi m}{N} \sin \frac{i_T \pi m}{N} \sqrt{G_1^2(m, n) + G_2^2(m, n)}}{\sin^2 \frac{i_T \pi m}{N} + \Gamma i_T^2 \sin^2 \frac{\pi m}{N}} \cos \left(\phi(m, n) + \frac{(i_T - 1)\pi m}{N} \right) \\
F_2(m, n) &= \frac{i_T \sin \frac{\pi m}{N} \sin \frac{i_T \pi m}{N} \sqrt{G_1^2(m, n) + G_2^2(m, n)}}{\sin^2 \frac{i_T \pi m}{N} + \Gamma i_T^2 \sin^2 \frac{\pi m}{N}} \sin \left(\phi(m, n) + \frac{(i_T - 1)\pi m}{N} \right)
\end{aligned} \tag{5.142}$$

If we use formulae $\cos(a + b) = \cos a \cos b - \sin a \sin b$ and $\sin(a + b) = \sin a \cos b + \cos a \sin b$ and substitute $\cos \phi(m, n)$ and $\sin \phi(m, n)$ from equations (5.55), we obtain:

$$\begin{aligned}
F_1(m, n) &= \frac{i_T \sin \frac{\pi m}{N} \sin \frac{i_T \pi m}{N}}{\sin^2 \frac{i_T \pi m}{N} + \Gamma i_T^2 \sin^2 \frac{\pi m}{N}} \left[G_1(m, n) \cos \frac{(i_T - 1)\pi m}{N} - G_2(m, n) \sin \frac{(i_T - 1)\pi m}{N} \right] \\
F_2(m, n) &= \frac{i_T \sin \frac{\pi m}{N} \sin \frac{i_T \pi m}{N}}{\sin^2 \frac{i_T \pi m}{N} + \Gamma i_T^2 \sin^2 \frac{\pi m}{N}} \left[G_1(m, n) \sin \frac{(i_T - 1)\pi m}{N} + G_2(m, n) \cos \frac{(i_T - 1)\pi m}{N} \right]
\end{aligned} \tag{5.143}$$

For $m = 0$ we must remember to use:

$$\begin{aligned}
F_1(0, n) &= \frac{G_1(0, n)}{1 + \Gamma} \quad \text{for } 0 \leq n \leq N - 1 \\
F_2(0, n) &= \frac{G_2(0, n)}{1 + \Gamma} \quad \text{for } 0 \leq n \leq N - 1
\end{aligned} \tag{5.144}$$

If we take the inverse Fourier transform, using functions $F_1(m, n)$ and $F_2(m, n)$ as the real and the imaginary part of the transform, respectively, we obtain the restored image shown in figure 5.11a. This image should be compared with images 5.5e and 5.5f, on page 414, which were obtained by inverse filtering.

The restoration of the noisy images of figures 5.9a and 5.9b by Wiener filtering is shown in figures 5.11b and 5.11c. These images should be compared with figures 5.9g and 5.9h, respectively. In all cases, Wiener filtering produces superior results. We note, that if we use too small a value of Γ , the effect of the correction term in the denominator of the filter is insignificant. If we use too high Γ , the restored image is very smoothed. For the case with no noise, we obtained acceptable results for Γ in the range from about 0.01 to 0.03. For $\sigma = 10$, we obtained acceptable results for Γ from about 0.025 to 0.05, while for $\sigma = 20$, the best results were obtained for Γ from about 0.03 to 0.06.

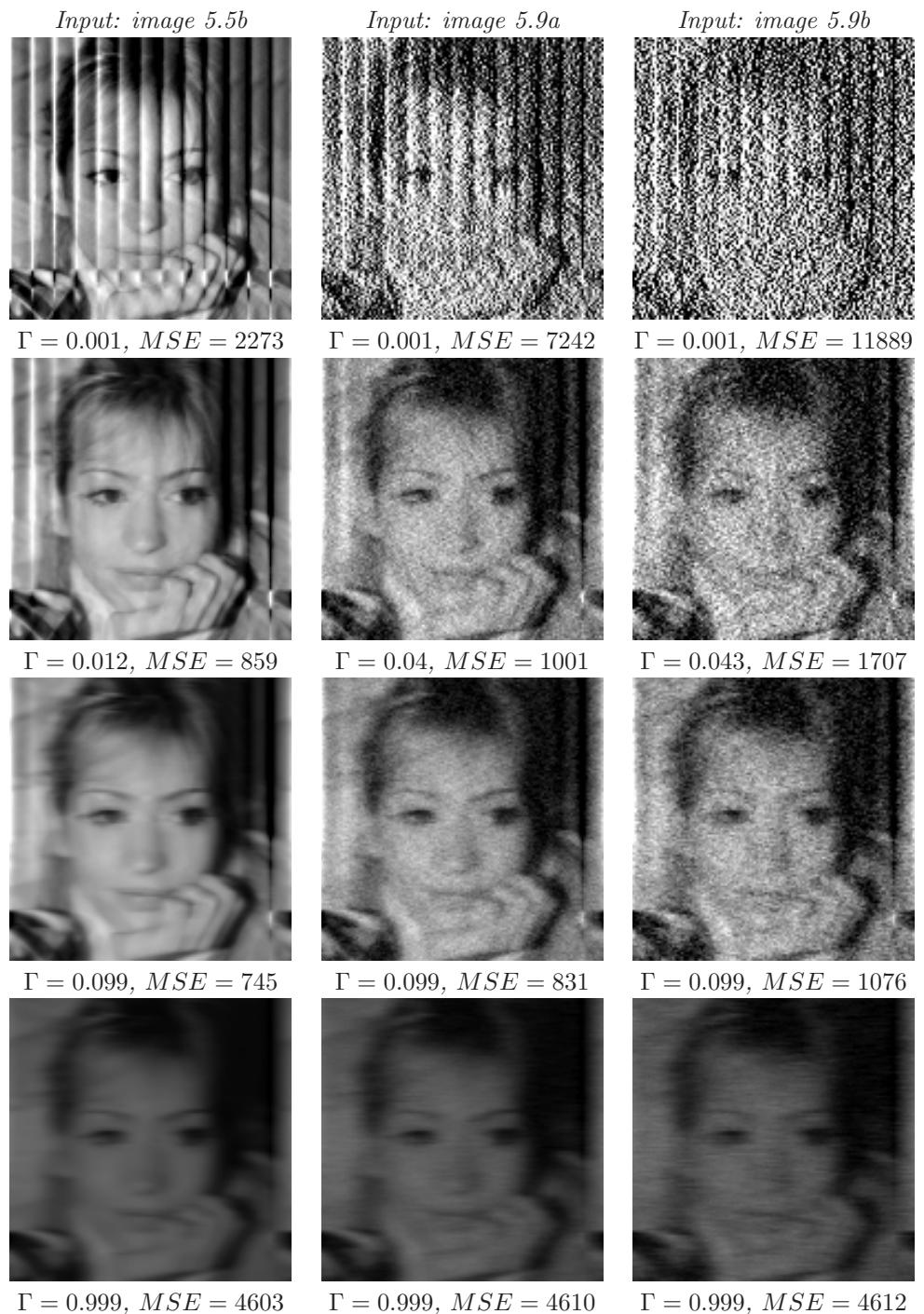


Figure 5.11: Dionisia restored with Wiener filtering.

5.3 Homogeneous linear image restoration: Constrained matrix inversion

If the degradation process is assumed linear, why don't we solve a system of linear equations to reverse its effect instead of invoking the convolution theorem?

Indeed, the system of linear equations we must invert is given in matrix form by equation (1.38), on page 19, $\mathbf{g} = H\mathbf{f}$. However, \mathbf{g} is expected to be noisy, so we shall rewrite this equation including an explicit noise term:

$$\mathbf{g} = H\mathbf{f} + \boldsymbol{\nu} \quad (5.145)$$

Here $\boldsymbol{\nu}$ is the noise field written in vector form.

Since we assumed that we have some knowledge about the point spread function of the degradation process, matrix H is assumed to be known. Then

$$\mathbf{f} = H^{-1}\mathbf{g} - H^{-1}\boldsymbol{\nu} \quad (5.146)$$

where H is an $N^2 \times N^2$ matrix, and \mathbf{f} , \mathbf{g} and $\boldsymbol{\nu}$ are $N^2 \times 1$ vectors, for an $N \times N$ image.

Equation (5.146) seems pretty straightforward, why bother with any other approach?

There are two major problems with equation (5.146).

- 1) It is extremely sensitive to noise. It has been shown that one needs impossibly low levels of noise for the method to work.
- 2) The solution of equation (5.146) requires the inversion of an $N^2 \times N^2$ square matrix, with N typically being 500, which is a formidable task even for modern computers.

Example 5.24

Demonstrate the sensitivity to noise of the inverse matrix restoration.

Let us consider the signal given by:

$$f(x) = 25 \sin \frac{2\pi x}{30} \quad \text{for } x = 0, 1, \dots, 29 \quad (5.147)$$

Assume that this signal is blurred by a function that averages every three samples after multiplying them with some weights. We can express this by saying that the discrete signal is multiplied with matrix

$$H = \begin{pmatrix} 0.4 & 0.3 & 0 & 0 & \dots & 0 & 0 & 0 & 0.3 \\ 0.3 & 0.4 & 0.3 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 0.4 & 0.3 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0.4 & 0.3 \\ 0.3 & 0 & 0 & 0 & \dots & 0 & 0 & 0.3 & 0.4 \end{pmatrix} \quad (5.148)$$

to produce a blurred signal $g(x)$. In a digital system, the elements of $g(x)$ are rounded to the nearest integer. To recover the original signal we multiply the blurred signal $g(x)$ with the inverse of matrix H . The original and the restored signals are shown in figure 5.12.

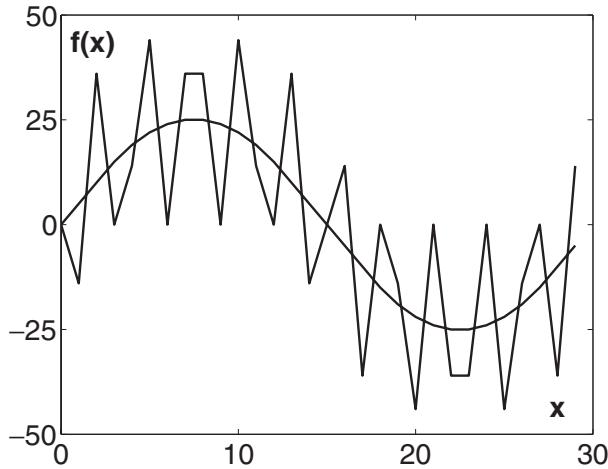


Figure 5.12: An original signal (smooth line) and its restored version by direct matrix inversion, using the exact inverse of the matrix that caused the distortion in the first place. The noise in the signal was only due to rounding errors.

Is there any way by which matrix H can be inverted?

Yes, for the case of homogeneous linear degradation, matrix H can easily be inverted because it is a **block circulant matrix**.

When is a matrix block circulant?

A matrix H is **block circulant** if it has the following structure:

$$H = \begin{pmatrix} H_0 & H_{M-1} & H_{M-2} & \dots & H_1 \\ H_1 & H_0 & H_{M-1} & \dots & H_2 \\ H_2 & H_1 & H_0 & \dots & H_3 \\ \vdots & \vdots & \vdots & & \vdots \\ H_{M-1} & H_{M-2} & H_{M-3} & & H_0 \end{pmatrix} \quad (5.149)$$

Here H_0, H_1, \dots, H_{M-1} are partitions of matrix H , and they are themselves **circulant matrices**.

When is a matrix circulant?

A matrix D is **circulant** if it has the following structure:

$$D = \begin{pmatrix} d(0) & d(M-1) & d(M-2) & \dots & d(1) \\ d(1) & d(0) & d(M-1) & \dots & d(2) \\ d(2) & d(1) & d(0) & \dots & d(3) \\ \vdots & \vdots & \vdots & & \vdots \\ d(M-1) & d(M-2) & d(M-3) & \dots & d(0) \end{pmatrix} \quad (5.150)$$

In such a matrix, each column can be obtained from the previous one by shifting all elements one place down and putting the last element at the top.

Why can block circulant matrices be inverted easily?

Circulant and block circulant matrices can easily be inverted because we can find easily their eigenvalues and eigenvectors.

Which are the eigenvalues and eigenvectors of a circulant matrix?

We define the set of scalars

$$\begin{aligned} \lambda(k) &\equiv d(0) + d(M-1) \exp\left[\frac{2\pi j}{M}k\right] + d(M-2) \exp\left[\frac{2\pi j}{M}2k\right] \\ &\quad + \dots + d(1) \exp\left[\frac{2\pi j}{M}(M-1)k\right] \end{aligned} \quad (5.151)$$

and the set of vectors

$$\mathbf{w}(k) \equiv \frac{1}{\sqrt{M}} \begin{pmatrix} 1 \\ \exp\left[\frac{2\pi j}{M}k\right] \\ \exp\left[\frac{2\pi j}{M}2k\right] \\ \vdots \\ \exp\left[\frac{2\pi j}{M}(M-1)k\right] \end{pmatrix} \quad (5.152)$$

where k takes up values $k = 0, 1, 2, \dots, M-1$. It can be shown then by direct substitution that

$$D\mathbf{w}(k) = \lambda(k)\mathbf{w}(k) \quad (5.153)$$

i.e. $\lambda(k)$ are the eigenvalues of matrix D (defined by equation (5.150)) and $\mathbf{w}(k)$ are its corresponding eigenvectors.

How does the knowledge of the eigenvalues and the eigenvectors of a matrix help in inverting the matrix?

If we form matrix W which has the eigenvectors of matrix D as its columns, we know that we can write

$$D = W\Lambda W^{-1} \quad (5.154)$$

where W^{-1} has elements (see example 5.25)

$$W^{-1}(k, i) = \frac{1}{\sqrt{M}} \exp \left[-j \frac{2\pi}{M} ki \right] \quad (5.155)$$

and Λ is a diagonal matrix with the eigenvalues along its diagonal. Then, the inversion of matrix D is trivial:

$$D^{-1} = (W\Lambda W^{-1})^{-1} = (W^{-1})^{-1} \Lambda^{-1} W^{-1} = W\Lambda^{-1} W^{-1} \quad (5.156)$$

Example 5.25

Consider matrix W the columns of which $w(0), w(1), \dots, w(M-1)$ are given by equation (5.152). Show that matrix Z with elements

$$Z(k, i) = \frac{1}{\sqrt{M}} \exp \left(-\frac{2\pi j}{M} ki \right) \quad (5.157)$$

is the inverse of matrix W .

We have:

$$W = \frac{1}{\sqrt{M}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & e^{\frac{2\pi j}{M}} & e^{\frac{2\pi j}{M}2} & \dots & e^{\frac{2\pi j}{M}(M-1)} \\ 1 & e^{\frac{2\pi j}{M}2} & e^{\frac{2\pi j}{M}4} & \dots & e^{\frac{2\pi j}{M}2(M-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{\frac{2\pi j}{M}(M-1)} & e^{\frac{2\pi j}{M}2(M-1)} & \dots & e^{\frac{2\pi j}{M}(M-1)^2} \end{pmatrix} \quad (5.158)$$

$$Z = \frac{1}{\sqrt{M}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & e^{-\frac{2\pi j}{M}} & e^{-\frac{2\pi j}{M}2} & \dots & e^{-\frac{2\pi j}{M}(M-1)} \\ 1 & e^{-\frac{2\pi j}{M}2} & e^{-\frac{2\pi j}{M}4} & \dots & e^{-\frac{2\pi j}{M}2(M-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{-\frac{2\pi j}{M}(M-1)} & e^{-\frac{2\pi j}{M}2(M-1)} & \dots & e^{-\frac{2\pi j}{M}(M-1)^2} \end{pmatrix} \quad (5.159)$$

$$ZW = \frac{1}{M} \begin{pmatrix} M & \sum_{k=0}^{M-1} e^{\frac{2\pi j}{M} k} & \dots & \sum_{k=0}^{M-1} e^{\frac{2\pi j}{M} (M-1)k} \\ \sum_{k=0}^{M-1} e^{-\frac{2\pi j}{M} k} & M & \dots & \sum_{k=0}^{M-1} e^{-\frac{2\pi j}{M} 2(M-2)k} \\ \vdots & \vdots & & \vdots \\ \sum_{k=0}^{M-1} e^{-\frac{2\pi j}{M} (M-1)k} & \sum_{k=0}^{M-1} e^{-\frac{2\pi j}{M} 2(M-2)k} & \dots & M \end{pmatrix} \quad (5.160)$$

All the off-diagonal elements of this matrix are of the form: $\sum_{k=0}^{M-1} e^{\frac{2\pi j t}{M} k}$ where t is some positive or negative integer. We may then apply (2.167), on page 95, with $m \equiv k$ and $S \equiv M$ to show that all the off-diagonal elements are 0 and thus recognise that ZW is the identity matrix, ie that $Z = W^{-1}$.

Example 5.26

For $M = 3$ show that $\lambda(k)$ defined by equation (5.151) and $w(k)$ defined by (5.152) are the eigenvalues and eigenvectors, respectively, of matrix (5.150), for $k = 0, 1, 2$.

Let us redefine matrix D for $M = 3$ as:

$$D = \begin{pmatrix} d_0 & d_2 & d_1 \\ d_1 & d_0 & d_2 \\ d_2 & d_1 & d_0 \end{pmatrix} \quad (5.161)$$

We also have:

$$w(k) = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ e^{\frac{2\pi j}{3} k} \\ e^{\frac{2\pi j}{3} 2k} \end{pmatrix} \quad \text{for } k = 0, 1, 2 \quad (5.162)$$

$$\lambda(k) = d_0 + d_2 e^{\frac{2\pi j}{3} k} + d_1 e^{\frac{2\pi j}{3} 2k} \quad \text{for } k = 0, 1, 2 \quad (5.163)$$

We must show that:

$$Dw(k) = \lambda(k)w(k) \quad (5.164)$$

We compute first the left-hand side of this expression:

$$Dw(k) = \begin{pmatrix} d_0 & d_2 & d_1 \\ d_1 & d_0 & d_2 \\ d_2 & d_1 & d_0 \end{pmatrix} \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ e^{\frac{2\pi j}{3} k} \\ e^{\frac{2\pi j}{3} 2k} \end{pmatrix} = \frac{1}{\sqrt{3}} \begin{pmatrix} d_0 + d_2 e^{\frac{2\pi j}{3} k} + d_1 e^{\frac{2\pi j}{3} 2k} \\ d_1 + d_0 e^{\frac{2\pi j}{3} k} + d_2 e^{\frac{2\pi j}{3} 2k} \\ d_2 + d_1 e^{\frac{2\pi j}{3} k} + d_0 e^{\frac{2\pi j}{3} 2k} \end{pmatrix} \quad (5.165)$$

We also compute the right-hand side of (5.164):

$$\lambda(k)\mathbf{w}(k) = \frac{1}{\sqrt{3}} \begin{pmatrix} d_0 + d_2 e^{\frac{2\pi j}{3}k} + d_1 e^{\frac{2\pi j}{3}2k} \\ d_0 e^{\frac{2\pi j}{3}k} + d_2 e^{\frac{2\pi j}{3}2k} + d_1 e^{\frac{2\pi j}{3}3k} \\ d_0 e^{\frac{2\pi j}{3}2k} + d_2 e^{\frac{2\pi j}{3}3k} + d_1 e^{\frac{2\pi j}{3}4k} \end{pmatrix} \quad (5.166)$$

If we compare the elements of the matrices on the right-hand sides of expressions (5.165) and (5.166) one by one, and take into consideration the fact that

$$e^{2\pi jk} = 1 \quad \text{for any integer } k \quad (5.167)$$

and

$$e^{\frac{2\pi j}{3}4k} = e^{\frac{2\pi j}{3}3k} e^{\frac{2\pi j}{3}k} = e^{2\pi jk} e^{\frac{2\pi j}{3}k} = e^{\frac{2\pi j}{3}k}, \quad (5.168)$$

we see that equation (5.164) is correct.

Example 5.27

Find the inverse of the following matrix:

$$\begin{pmatrix} -1 & 0 & 2 & 3 \\ 3 & -1 & 0 & 2 \\ 2 & 3 & -1 & 0 \\ 0 & 2 & 3 & -1 \end{pmatrix} \quad (5.169)$$

This is a circulant matrix and according to the notation of equation (5.150), we have $M = 4$, $d(0) = -1$, $d(1) = 3$, $d(2) = 2$, $d(3) = 0$. Then, applying formulae (5.151) and (5.152), we obtain:

$$\lambda(0) = -1 + 2 + 3 = 4 \Rightarrow \lambda(0)^{-1} = \frac{1}{4} \quad (5.170)$$

$$\lambda(1) = -1 + 2e^{\frac{2\pi j}{4}2} + 3e^{\frac{2\pi j}{4}3} = -1 - 2 - 3j = -3 - 3j \Rightarrow \lambda^{-1}(1) = \frac{-3+3j}{18} = \frac{-1+j}{6}$$

$$\lambda(2) = -1 + 2e^{\frac{2\pi j}{4}4} + 3e^{\frac{2\pi j}{4}6} = -1 + 2 - 3 = -2 \Rightarrow \lambda^{-1}(2) = -\frac{1}{2}$$

$$\lambda(3) = -1 + 2e^{\frac{2\pi j}{4}6} + 3e^{\frac{2\pi j}{4}9} = -1 - 2 + 3j = -3 + 3j \Rightarrow \lambda^{-1}(3) = \frac{-3-3j}{18} = \frac{-1-j}{6}$$

$$\begin{aligned}
 \mathbf{w}^T(0) &= \frac{1}{2} (1 \quad 1 \quad 1 \quad 1) \\
 \mathbf{w}^T(1) &= \frac{1}{2} (1 \quad e^{\frac{2\pi j}{4}} \quad e^{\frac{2\pi j}{4}2} \quad e^{\frac{2\pi j}{4}3}) = \frac{1}{2} (1 \quad j \quad -1 \quad -j) \\
 \mathbf{w}^T(2) &= \frac{1}{2} (1 \quad e^{\frac{2\pi j}{4}2} \quad e^{\frac{2\pi j}{4}4} \quad e^{\frac{2\pi j}{4}6}) = \frac{1}{2} (1 \quad -1 \quad 1 \quad -1) \\
 \mathbf{w}^T(3) &= \frac{1}{2} (1 \quad e^{\frac{2\pi j}{4}3} \quad e^{\frac{2\pi j}{4}6} \quad e^{\frac{2\pi j}{4}9}) = \frac{1}{2} (1 \quad -j \quad -1 \quad j)
 \end{aligned} \tag{5.171}$$

We use these vectors to construct matrices W and W^{-1} and then apply formula (5.156):

$$\begin{aligned}
 D^{-1} &= \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{pmatrix} \begin{pmatrix} \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{-1+j}{6} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{-1-j}{6} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \\
 &= \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{pmatrix} \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{-1+j}{6} & \frac{1+j}{6} & \frac{1-j}{6} & \frac{-1-j}{6} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{-1-j}{6} & \frac{1-j}{6} & \frac{1+j}{6} & \frac{-1+j}{6} \end{pmatrix} \\
 &= \frac{1}{4} \begin{pmatrix} -\frac{7}{12} & \frac{13}{12} & \frac{1}{12} & \frac{5}{12} \\ \frac{5}{12} & -\frac{7}{12} & \frac{13}{12} & \frac{1}{12} \\ \frac{1}{12} & \frac{5}{12} & -\frac{7}{12} & \frac{13}{12} \\ \frac{13}{12} & \frac{1}{12} & \frac{5}{12} & -\frac{7}{12} \end{pmatrix} = \frac{1}{48} \begin{pmatrix} -7 & 13 & 1 & 5 \\ 5 & -7 & 13 & 1 \\ 1 & 5 & -7 & 13 \\ 13 & 1 & 5 & -7 \end{pmatrix}
 \end{aligned} \tag{5.172}$$

Example B5.28

The elements of a matrix W_N are given by

$$W_N(k, n) = \frac{1}{\sqrt{N}} \exp\left(\frac{2\pi j}{N} kn\right) \tag{5.173}$$

where k and n take values $0, 1, 2, \dots, N-1$. The elements of the inverse matrix W_N^{-1} (see example 5.25) are given by:

$$W_N^{-1}(k, n) = \frac{1}{\sqrt{N}} \exp\left(-\frac{2\pi j}{N} kn\right) \tag{5.174}$$

We define matrix W as the Kronecker product of W_N with itself. Show that the inverse of matrix W is formed by the Kronecker product of matrix W_N^{-1} with itself.

Let us consider an element $W(m, l)$ of matrix W . We write integers m and l in terms of their quotients and remainders when divided with N :

$$\begin{aligned} m &\equiv m_1 N + m_2 \\ l &\equiv l_1 N + l_2 \end{aligned} \quad (5.175)$$

Since W is $W_N \otimes W_N$ we have:

$$W(m, l) = \frac{1}{N} e^{\frac{2\pi i}{N} m_1 l_1} e^{\frac{2\pi i}{N} m_2 l_2} \quad (5.176)$$

Indices (m_1, l_1) identify the partition of matrix W to which element $W(m, l)$ belongs. Indices m_2 and l_2 vary inside each partition, taking all their possible values (see figure 5.13).

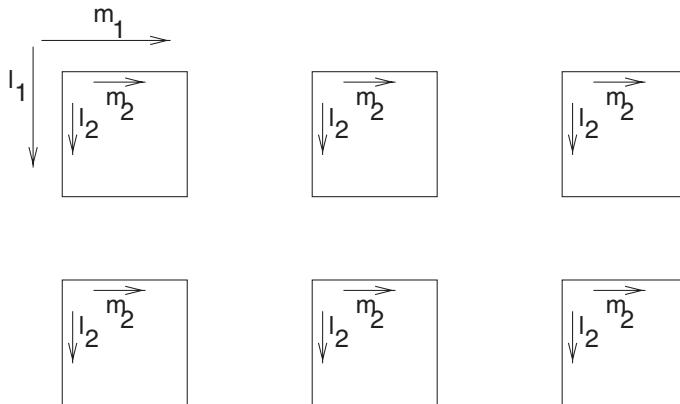


Figure 5.13: There are $N \times N$ partitions, enumerated by indices (m_1, l_1) , and inside each partition there are $N \times N$ elements, enumerated by indices (m_2, l_2) .

In a similar way, we can write an element of matrix $Z \equiv W_N^{-1} \otimes W_N^{-1}$, by writing $t \equiv t_1 N + t_2$ and $n \equiv n_1 N + n_2$:

$$Z(t, n) = \frac{1}{N} e^{-\frac{2\pi i}{N} t_1 n_1} e^{-\frac{2\pi i}{N} t_2 n_2} \quad (5.177)$$

An element of the product matrix $A \equiv WZ$ is given by:

$$\begin{aligned}
A(k, n) &= \sum_{t=0}^{N^2-1} W(k, t) Z(t, n) \\
&= \frac{1}{N^2} \sum_{t=0}^{N^2-1} e^{\frac{2\pi j}{N} k_1 t_1} e^{\frac{2\pi j}{N} k_2 t_2} e^{-\frac{2\pi j}{N} t_1 n_1} e^{-\frac{2\pi j}{N} t_2 n_2} \\
&= \frac{1}{N^2} \sum_{t=0}^{N^2-1} e^{\frac{2\pi j}{N} (k_1 - n_1) t_1} e^{\frac{2\pi j}{N} (k_2 - n_2) t_2}
\end{aligned} \tag{5.178}$$

If we write again $t \equiv t_1 N + t_2$, we can break the sum over t into two sums, one over t_1 and one over t_2 :

$$A(k, n) = \frac{1}{N^2} \sum_{t_1=0}^{N-1} \left\{ e^{\frac{2\pi j}{N} (k_1 - n_1) t_1} \sum_{t_2=0}^{N-1} e^{\frac{2\pi j}{N} (k_2 - n_2) t_2} \right\} \tag{5.179}$$

We apply formula (2.164), on page 95, for the inner sum first, with $S \equiv N$, $m \equiv t_2$ and $t \equiv k_2 - n_2$, and the outer sum afterwards, with $S \equiv N$, $m \equiv t_1$ and $t \equiv k_1 - n_1$:

$$\begin{aligned}
A(k, n) &= \frac{1}{N^2} \sum_{t_1=0}^{N-1} e^{\frac{2\pi j}{N} (k_1 - n_1) t_1} \delta(k_2 - n_2) N \\
&= \frac{1}{N} \delta(k_2 - n_2) N \delta(k_1 - n_1) \\
&= \delta(k_2 - n_2) \delta(k_1 - n_1) \\
&= \delta(k - n)
\end{aligned} \tag{5.180}$$

Therefore, matrix A has all its elements 0, except the diagonal ones (obtained for $k = n$), which are equal to 1. So, A is the unit matrix and this proves that matrix Z , with elements given by (5.177), is the inverse of matrix W , with elements given by (5.176).

How do we know that matrix H that expresses the linear degradation process is block circulant?

We saw in Chapter 1, that matrix H , which corresponds to a shift invariant linear operator expressed by equation (1.17), on page 13, may be partitioned into submatrices as expressed by equation (1.39), on page 19.

Let us consider one of the partitions of the partitioned matrix H (see equation (5.149), on page 437). Inside every partition, the values of l and j remain constant; ie $j - l$ is constant inside each partition. The value of $i - k$ along each line runs from i to $i - N + 1$, taking all

integer values in between. When i is incremented by 1 in the next row, all the values of $i - k$ are shifted by one position to the right (see equations (1.39) and (5.149)). So, each partition submatrix of H is characterised by the value of $j - l \equiv u$ and has a circulant form:

$$H_u \equiv \begin{pmatrix} h(0, u) & h(N-1, u) & h(N-2, u) & \dots & h(1, u) \\ h(1, u) & h(0, u) & h(N-1, u) & \dots & h(2, u) \\ h(2, u) & h(1, u) & h(0, u) & \dots & h(3, u) \\ \vdots & \vdots & \vdots & & \vdots \\ h(N-1, u) & h(N-2, u) & h(N-3, u) & \dots & h(0, u) \end{pmatrix} \quad (5.181)$$

Notice that here we assume that $h(v, u)$ is periodic with period N in each of its arguments, and so $h(1-N, u) = h((1-N)+N, u) = h(1, u)$ etc.

The full matrix H may be written in the form

$$H = \begin{pmatrix} H_0 & H_{-1} & H_{-2} & \dots & H_{-M+1} \\ H_1 & H_0 & H_{-1} & \dots & H_{-M+2} \\ H_2 & H_1 & H_0 & \dots & H_{-M+3} \\ \vdots & \vdots & \vdots & & \vdots \\ H_{M-1} & H_{M-2} & H_{M-3} & \dots & H_0 \end{pmatrix} \quad (5.182)$$

where again owing to the periodicity of $h(v, u)$, $H_{-1} = H_{M-1}$, $H_{-M+1} = H_1$ etc.

How can we diagonalise a block circulant matrix?

Define a matrix with elements

$$W_N(k, n) \equiv \frac{1}{\sqrt{N}} \exp \left[\frac{2\pi j}{N} kn \right] \quad (5.183)$$

and matrix

$$W \equiv W_N \otimes W_N \quad (5.184)$$

where \otimes is the Kronecker product of the two matrices (see example 1.26, on page 38). The inverse of $W_N(k, n)$ is a matrix with elements:

$$W_N^{-1}(k, n) = \frac{1}{\sqrt{N}} \exp \left[-\frac{2\pi j}{N} kn \right] \quad (5.185)$$

The inverse of W is given by (see example 5.28):

$$W_N^{-1}(k, n) = W_N^{-1} \otimes W_N^{-1} \quad (5.186)$$

We also define a diagonal matrix Λ as

$$\Lambda(k, i) = \begin{cases} N^2 \hat{H} \left(\text{mod}_N(k), \lfloor \frac{k}{N} \rfloor \right) & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases} \quad (5.187)$$

where \hat{H} is the discrete Fourier transform of the point spread function h :

$$\hat{H}(u, v) = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} h(x, y) e^{-2\pi j \left(\frac{ux}{N} + \frac{vy}{N} \right)} \quad (5.188)$$

It can be shown then, by direct matrix multiplication, that:

$$H = W \Lambda W^{-1} \Rightarrow H^{-1} = W \Lambda^{-1} W^{-1} \quad (5.189)$$

Thus, H can be inverted easily since it has been written as the product of matrices the inversion of which is trivial.

Box 5.4. Proof of equation (5.189)

First we have to find how an element $H(f, g)$ of matrix H is related to the point spread function $h(x, y)$. Let us write indices f and g as multiples of the dimension N of one of the partitions, plus a remainder:

$$\begin{aligned} f &\equiv f_1 N + f_2 \\ g &\equiv g_1 N + g_2 \end{aligned} \quad (5.190)$$

As f and g scan all possible values from 0 to $N^2 - 1$ each, we can visualise the $N \times N$ partitions of matrix H , indexed by subscript $u \equiv f_1 - g_1$, as follows:

$f_1 = 0$	$f_1 = 0$	$f_1 = 0$	\dots
$g_1 = 0$	$g_1 = 1$	$g_1 = 2$	
$u = 0$	$u = -1$	$u = -2$	
$f_1 = 1$	$f_1 = 1$	$f_1 = 1$	\dots
$g_1 = 0$	$g_1 = 1$	$g_1 = 2$	
$u = 1$	$u = 0$	$u = -1$	
\dots	\dots	\dots	\dots

We observe that each partition is characterised by index $u = f_1 - g_1$ and inside each partition the elements computed from $h(x, y)$ are computed for various values of $f_2 - g_2$. We conclude that:

$$H(f, g) = h(f_2 - g_2, f_1 - g_1) \quad (5.191)$$

Let us consider next an element of matrix $W\Lambda W^{-1}$:

$$A(m, n) \equiv \sum_{l=0}^{N^2-1} \sum_{t=0}^{N^2-1} W_{ml} \Lambda_{lt} W_{tn}^{-1} \quad (5.192)$$

Since matrix Λ is diagonal, the sum over t collapses to values of $t = l$ only. Then:

$$A(m, n) = \sum_{l=0}^{N^2-1} W_{ml} \Lambda_{ll} W_{ln}^{-1} \quad (5.193)$$

Λ_{ll} is a scalar and therefore it may change position inside the summand:

$$A(m, n) = \sum_{l=0}^{N^2-1} W_{ml} W_{ln}^{-1} \Lambda_{ll} \quad (5.194)$$

In example 5.28 we saw how the elements of matrices W_{ml} and W_{ln}^{-1} can be written if we write their indices in terms of their quotients and remainders when divided by N :

$$\begin{aligned} m &\equiv Nm_1 + m_2 \\ l &\equiv Nl_1 + l_2 \\ n &\equiv Nn_1 + n_2 \end{aligned} \quad (5.195)$$

Using these expressions and the definition of Λ_{ll} as $N^2 \hat{H}(l_2, l_1)$ from equation (5.187), we obtain:

$$A(m, n) = \sum_{l=0}^{N^2-1} e^{\frac{2\pi i}{N} m_1 l_1} e^{\frac{2\pi i}{N} m_2 l_2} \frac{1}{N^2} e^{-\frac{2\pi i}{N} l_1 n_1} e^{-\frac{2\pi i}{N} l_2 n_2} N^2 \hat{H}(l_2, l_1) \quad (5.196)$$

On rearranging, we have:

$$A(m, n) = \sum_{l_1=0}^{N-1} \sum_{l_2=0}^{N-1} \hat{H}(l_2, l_1) e^{\frac{2\pi i}{N} (m_1 - n_1) l_1} e^{\frac{2\pi i}{N} (m_2 - n_2) l_2} \quad (5.197)$$

We recognise this expression as the inverse Fourier transform of $h(m_2 - n_2, m_1 - n_1)$. Therefore:

$$A(m, n) = h(m_2 - n_2, m_1 - n_1) \quad (5.198)$$

By comparing equations (5.191) and (5.198) we can see that the elements of matrices H and $W\Lambda W^{-1}$ have been shown to be identical, and so equation (5.189) has been proven.

Box 5.5. What is the transpose of matrix H ?

We shall show that $H^T = W\Lambda^*W^{-1}$, where Λ^* is the complex conjugate of matrix Λ . According to equation (5.191) of Box 5.4, an element of the transpose of matrix H will be given by:

$$H^T(f, g) = h(g_2 - f_2, g_1 - f_1) \quad (5.199)$$

(The roles of f and g are exchanged in the formula.) An element $A(m, n)$ of matrix $W\Lambda^*W^{-1}$ will be given by an equation similar to (5.197), but instead of having factor $\hat{H}(l_2, l_1)$, it will have factor $\hat{H}(-l_2, -l_1)$, coming from the element of Λ_{ll}^* being defined in terms of the complex conjugate of the Fourier transform $\hat{H}(u, v)$ given by equation (5.188):

$$A(m, n) = \sum_{l_1=0}^{N-1} \sum_{l_2=0}^{N-1} \hat{H}(-l_2, -l_1) e^{\frac{2\pi j}{N}(m_1-n_1)l_1} e^{\frac{2\pi j}{N}(m_2-n_2)l_2} \quad (5.200)$$

We change the dummy variables of summation to:

$$\tilde{l}_1 \equiv -l_1 \text{ and } \tilde{l}_2 \equiv -l_2 \quad (5.201)$$

Then:

$$A(m, n) = \sum_{\tilde{l}_1=0}^{-N+1} \sum_{\tilde{l}_2=0}^{-N+1} \hat{H}(\tilde{l}_2, \tilde{l}_1) e^{\frac{2\pi j}{N}(-m_1+n_1)\tilde{l}_1} e^{\frac{2\pi j}{N}(-m_2+n_2)\tilde{l}_2} \quad (5.202)$$

Since we are dealing with periodic functions summed over a period, the range over which we sum does not really matter, as long as N consecutive values are considered. Then we can write:

$$A(m, n) = \sum_{\tilde{l}_1=0}^{N-1} \sum_{\tilde{l}_2=0}^{N-1} \hat{H}(\tilde{l}_2, \tilde{l}_1) e^{\frac{2\pi j}{N}(-m_1+n_1)\tilde{l}_1} e^{\frac{2\pi j}{N}(-m_2+n_2)\tilde{l}_2} \quad (5.203)$$

We recognise on the right-hand side of the above expression the inverse Fourier transform of $\hat{H}(\tilde{l}_2, \tilde{l}_1)$, computed at $(n_2 - m_2, n_1 - m_1)$:

$$A(m, n) = h(n_2 - m_2, n_1 - m_1) \quad (5.204)$$

By direct comparison with equation (5.199), we prove that matrices H^T and $W\Lambda^*W^{-1}$ are equal, element by element.

Example 5.29

Show that the Laplacian, ie the sum of the second derivatives, of a discrete image at a pixel position (i, j) may be estimated by:

$$\Delta^2 f(i, j) = f(i - 1, j) + f(i, j - 1) + f(i + 1, j) + f(i, j + 1) - 4f(i, j) \quad (5.205)$$

At inter-pixel position $(i + 0.5, j)$, the first derivative of the image function along the i axis is approximated by the first difference:

$$\Delta_i f(i + 0.5, j) = f(i + 1, j) - f(i, j) \quad (5.206)$$

Similarly, the first difference at $(i - 0.5, j)$ along the i axis is:

$$\Delta_i f(i - 0.5, j) = f(i, j) - f(i - 1, j) \quad (5.207)$$

The second derivative at (i, j) along the i axis may be approximated by the first difference of the first differences, computed at positions $(i + 0.5, j)$ and $(i - 0.5, j)$, that is:

$$\begin{aligned} \Delta_i^2 f(i, j) &= \Delta_i f(i + 0.5, j) - \Delta_i f(i - 0.5, j) \\ &= f(i + 1, j) - 2f(i, j) + f(i - 1, j) \end{aligned} \quad (5.208)$$

Similarly, the second derivative at (i, j) along the j axis may be approximated by:

$$\begin{aligned} \Delta_j^2 f(i, j) &= \Delta_j f(i, j + 0.5) - \Delta_j f(i, j - 0.5) \\ &= f(i, j + 1) - 2f(i, j) + f(i, j - 1) \end{aligned} \quad (5.209)$$

Adding equations (5.208) and (5.209) by parts we obtain the result.

Example 5.30

Consider a 3×3 image represented by a column vector \mathbf{f} . Identify a 9×9 matrix L such that if we multiply vector \mathbf{f} by it, the output will be a vector with the estimate of the value of the Laplacian at each position. Assume that image f is periodic in each direction with period 3. What type of matrix is L ?

From example 5.29 we know that the point spread function of the operator that returns the estimate of the Laplacian at each position is:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix} \quad (5.210)$$

To avoid boundary effects, we first extend the image in all directions periodically:

$$\begin{matrix} f_{13} & \begin{pmatrix} f_{31} & f_{32} & f_{33} \\ f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \\ f_{11} & f_{12} & f_{13} \end{pmatrix} & f_{11} \\ f_{21} & & f_{21} \\ f_{31} & & f_{31} \end{matrix} \quad (5.211)$$

By observing which values will contribute to the value of the Laplacian at a pixel position, and with what weight, we construct the 9×9 matrix with which we must multiply the column vector \mathbf{f} to obtain its Laplacian:

$$\begin{pmatrix} -4 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & -4 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & -4 \end{pmatrix} \begin{pmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{pmatrix} \quad (5.212)$$

This matrix is a block circulant matrix with easily identifiable partitions of size 3×3 .

Example B5.31

Using the matrix defined in example 5.30, estimate the Laplacian of the following image:

$$\begin{pmatrix} 3 & 2 & 1 \\ 2 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.213)$$

Then re-estimate the Laplacian of the above image using the formula of example 5.29.

$$\begin{pmatrix} -4 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & -4 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & -4 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 0 \\ 2 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -7 \\ -4 \\ 6 \\ -4 \\ 5 \\ 3 \\ 3 \\ 0 \\ -2 \end{pmatrix} \quad (5.214)$$

If we use the formula, we need to augment first the image by writing explicitly the boundary pixels:

$$\begin{matrix} 0 & 0 & 1 \\ 1 & \begin{pmatrix} 3 & 2 & 1 \\ 2 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} & 3 \\ 1 & & 2 \\ 1 & & 0 \\ 3 & 2 & 1 \end{matrix} \quad (5.215)$$

The Laplacian is:

$$\begin{pmatrix} 1+2+2-4\times 3 & 3+1-4\times 2 & 1+2+1+3-4\times 1 \\ 3+1-4\times 2 & 2+2+1 & 1+1+2-4\times 1 \\ 2+1+3 & 2+1 & 1+1-4\times 1 \end{pmatrix} = \begin{pmatrix} -7 & -4 & 3 \\ -4 & 5 & 0 \\ 6 & 3 & -2 \end{pmatrix} \quad (5.216)$$

Note that we obtain the same answer, whether we use the local formula or matrix multiplication.

Example B5.32

Find the eigenvalues and eigenvectors of the matrix worked out in example 5.30.

Matrix L worked out in example 5.30 is:

$$L = \begin{pmatrix} -4 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & -4 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & -4 \end{pmatrix} \quad (5.217)$$

This matrix is a block circulant matrix with easily identifiable 3×3 partitions. To find its eigenvectors, we first use equation (5.152), on page 438, for $M = 3$ to define vectors \mathbf{w} :

$$\mathbf{w}(0) = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad \mathbf{w}(1) = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ e^{\frac{2\pi j}{3}} \\ e^{\frac{4\pi j}{3}} \end{pmatrix} \quad \mathbf{w}(2) = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ e^{\frac{4\pi j}{3}} \\ e^{\frac{8\pi j}{3}} \end{pmatrix} \quad (5.218)$$

These vectors are used as columns to construct the matrix defined by equation (5.183):

$$W_3 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & e^{\frac{2\pi j}{3}} & e^{\frac{4\pi j}{3}} \\ 1 & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} \end{pmatrix} \quad (5.219)$$

We take the Kronecker product of this matrix with itself to create matrix W as defined by equation (5.184), on page 445:

$$W = \frac{1}{3} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{\frac{2\pi j}{3}} & e^{\frac{4\pi j}{3}} & 1 & e^{\frac{2\pi j}{3}} & e^{\frac{4\pi j}{3}} & 1 & e^{\frac{2\pi j}{3}} & e^{\frac{4\pi j}{3}} \\ 1 & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} & 1 & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} & 1 & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} \\ 1 & 1 & 1 & e^{\frac{2\pi j}{3}} & e^{\frac{2\pi j}{3}} & e^{\frac{2\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{4\pi j}{3}} \\ 1 & e^{\frac{2\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{2\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{6\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{6\pi j}{3}} & e^{\frac{8\pi j}{3}} \\ 1 & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{2\pi j}{3}} & e^{\frac{6\pi j}{3}} & e^{\frac{10\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{12\pi j}{3}} \\ 1 & e^{\frac{8\pi j}{3}} & e^{\frac{2\pi j}{3}} & e^{\frac{6\pi j}{3}} & e^{\frac{10\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{2\pi j}{3}} & e^{\frac{8\pi j}{3}} \\ 1 & 1 & 1 & e^{\frac{4\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{8\pi j}{3}} \\ 1 & e^{\frac{2\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{6\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{10\pi j}{3}} & e^{\frac{12\pi j}{3}} \\ 1 & e^{\frac{4\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{4\pi j}{3}} & e^{\frac{12\pi j}{3}} & e^{\frac{8\pi j}{3}} & e^{\frac{12\pi j}{3}} & e^{\frac{12\pi j}{3}} & e^{\frac{16\pi j}{3}} \end{pmatrix} \quad (5.220)$$

The columns of this matrix are the eigenvectors of matrix L . These eigenvectors are the same for all block circulant matrices with the same structure, independent of what the exact values of the elements are. The inverse of matrix W can be constructed using equation (5.185), on page 445, ie by taking the complex conjugate of matrix W . (Note that for a general unitary matrix we must take the complex conjugate of its transpose

in order to construct its inverse. This is not necessary here as W is a symmetric matrix and therefore it is equal to its transpose.)

$$W^{-1} = \frac{1}{3} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{4\pi j}{3}} & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{4\pi j}{3}} & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{4\pi j}{3}} \\ 1 & e^{-\frac{4\pi j}{3}} & e^{-\frac{8\pi j}{3}} & 1 & e^{-\frac{4\pi j}{3}} & e^{-\frac{8\pi j}{3}} & 1 & e^{-\frac{4\pi j}{3}} & e^{-\frac{8\pi j}{3}} \\ 1 & 1 & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{4\pi j}{3}} \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{6\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{6\pi j}{3}} & e^{-\frac{8\pi j}{3}} \\ 1 & e^{-\frac{4\pi j}{3}} & e^{-\frac{8\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{6\pi j}{3}} & e^{-\frac{10\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{8\pi j}{3}} & e^{-\frac{12\pi j}{3}} \\ 1 & 1 & 1 & e^{-\frac{4\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{8\pi j}{3}} & e^{-\frac{8\pi j}{3}} & e^{-\frac{8\pi j}{3}} \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{4\pi j}{3}} & e^{-\frac{6\pi j}{3}} & e^{-\frac{8\pi j}{3}} & e^{-\frac{10\pi j}{3}} & e^{-\frac{12\pi j}{3}} & e^{-\frac{16\pi j}{3}} \end{pmatrix} \quad (5.221)$$

The eigenvalues of matrix L may be computed from its Fourier transform, using equation (5.187), on page 446. First, however, we need to identify the kernel $l(x, y)$ of the operator represented by matrix L and take its Fourier transform $\hat{L}(u, v)$ using equation (5.188), on page 446. From example 5.30 we know that the kernel function is:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix} \quad (5.222)$$

We can identify then the following values for the discrete function $l(x, y)$:

$$\begin{aligned} l(0, 0) &= -4, \quad l(-1, -1) = 0, \quad l(-1, 0) = 1, \quad l(-1, 1) = 0 \\ l(0, -1) &= 1, \quad l(0, 1) = 1, \quad l(1, -1) = 0, \quad l(1, 0) = 1, \quad l(1, 1) = 0 \end{aligned} \quad (5.223)$$

However, these values cannot be directly used in equation (5.188), which assumes a function $h(x, y)$ defined with positive values of its arguments only. We therefore need a shifted version of our kernel, one that puts the value -4 at the top left corner of the matrix representation of the kernel. We can obtain such a version by reading the first column of matrix L and wrapping it around to form a 3×3 matrix:

$$\begin{matrix} -4 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{matrix} \quad (5.224)$$

Then we have:

$$\begin{aligned} l(0, 0) &= -4, \quad l(0, 1) = 1, \quad l(0, 2) = 1, \quad l(1, 0) = 1 \\ l(2, 0) &= 1, \quad l(1, 1) = 0, \quad l(1, 2) = 0, \quad l(2, 1) = 0, \quad l(2, 2) = 0 \end{aligned} \quad (5.225)$$

We can use these values in equation (5.188) to derive:

$$\hat{L}(u, v) = \frac{1}{9} \left[-4 + e^{-\frac{2\pi j}{3}v} + e^{-\frac{2\pi j}{3}2v} + e^{-\frac{2\pi j}{3}u} + e^{-\frac{2\pi j}{3}2u} \right] \quad (5.226)$$

Formula (5.187) says that the eigenvalues of matrix L , which appear along the diagonal of matrix $\Lambda(k, i)$, are the values of the Fourier transform $\hat{L}(u, v)$, computed for $u = \text{mod}_3(k)$ and $v = [\frac{k}{3}]$, where $k = 0, 1, \dots, 8$. These values may be computed using formula (5.226):

$$\begin{aligned}
\hat{L}(0,0) &= 0 \\
\hat{L}(0,1) &= \frac{1}{9} \left[-4 + e^{-\frac{2\pi j}{3}} + e^{-\frac{4\pi j}{3}} + 1 + 1 \right] = \frac{1}{9}[-2 - 2 \cos 60^\circ] = -\frac{1}{3} \\
\hat{L}(0,2) &= \frac{1}{9} \left[-4 + e^{-\frac{4\pi j}{3}} + e^{-\frac{8\pi j}{3}} + 2 \right] = \frac{1}{9}[-2 + e^{-\frac{4\pi j}{3}} + e^{-\frac{2\pi j}{3}}] = -\frac{1}{3} \\
\hat{L}(1,0) &= \hat{L}(0,1) = -1 \\
\hat{L}(1,1) &= \frac{1}{9} \left[-4 + 2e^{-\frac{2\pi j}{3}} + 2e^{-\frac{4\pi j}{3}} \right] = \frac{1}{9}[-4 - 4 \cos 60^\circ] = -\frac{2}{3} \\
\hat{L}(1,2) &= \frac{1}{9} \left[-4 + e^{-\frac{4\pi j}{3}} + e^{-\frac{8\pi j}{3}} + e^{-\frac{2\pi j}{3}} + e^{-\frac{4\pi j}{3}} \right] = -\frac{2}{3} \\
\hat{L}(2,0) &= \hat{L}(0,2) = -\frac{1}{3} \\
\hat{L}(2,1) &= \hat{L}(1,2) = -\frac{2}{3} \\
\hat{L}(2,2) &= \frac{1}{9} \left[-4 + 2e^{-\frac{4\pi j}{3}} + 2e^{-\frac{8\pi j}{3}} \right] = -\frac{2}{3} \tag{5.227}
\end{aligned}$$

Here we made use of the following:

$$\begin{aligned}
 e^{-\frac{2\pi j}{3}} &= -\cos 60^\circ - j \sin 60^\circ = -\frac{1}{2} - j \frac{\sqrt{3}}{2} \\
 e^{-\frac{4\pi j}{3}} &= -\cos 60^\circ + j \sin 60^\circ = -\frac{1}{2} + j \frac{\sqrt{3}}{2} \\
 e^{-\frac{6\pi j}{3}} &= 1 \\
 e^{-\frac{8\pi j}{3}} &= e^{-\frac{2\pi j}{3}} = -\cos 60^\circ - j \sin 60^\circ = -\frac{1}{2} - j \frac{\sqrt{3}}{2}
 \end{aligned} \tag{5.228}$$

Note that the first eigenvalue of matrix L is 0. This means that matrix L is singular, and even though we can diagonalise it using equation (5.189), we cannot invert it by taking the inverse of this equation. This should not be surprising as matrix L expresses the Laplacian operator on an image, and we know that from the knowledge of the Laplacian alone we can never recover the original image.

Applying equation (5.187) we define matrix Λ_L for L to be:

Having defined matrices W , W^{-1} and Λ we can then write:

$$L = W\Lambda_L W^{-1} \quad (5.230)$$

This equation may be confirmed by direct substitution. First we compute matrix $\Lambda_L W^{-1}$:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -e^{-\frac{2\pi j}{3}} & -e^{-\frac{4\pi j}{3}} & -1 & -e^{-\frac{2\pi j}{3}} & -e^{-\frac{4\pi j}{3}} & -1 & -e^{-\frac{2\pi j}{3}} \\ -1 & -e^{-\frac{4\pi j}{3}} & -e^{-\frac{8\pi j}{3}} & -1 & -e^{-\frac{4\pi j}{3}} & -e^{-\frac{8\pi j}{3}} & -1 & -e^{-\frac{4\pi j}{3}} \\ -1 & -1 & -1 & -e^{-\frac{2\pi j}{3}} & -e^{-\frac{2\pi j}{3}} & -e^{-\frac{2\pi j}{3}} & -e^{-\frac{4\pi j}{3}} & -e^{-\frac{4\pi j}{3}} \\ -2 & -2e^{-\frac{2\pi j}{3}} & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{2\pi j}{3}} & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{6\pi j}{3}} & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{6\pi j}{3}} \\ -2 & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{8\pi j}{3}} & -2e^{-\frac{2\pi j}{3}} & -2e^{-\frac{6\pi j}{3}} & -2e^{-\frac{10\pi j}{3}} & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{8\pi j}{3}} \\ -1 & -1 & -1 & -e^{-\frac{4\pi j}{3}} & -e^{-\frac{4\pi j}{3}} & -e^{-\frac{4\pi j}{3}} & -e^{-\frac{8\pi j}{3}} & -e^{-\frac{8\pi j}{3}} \\ -2 & -2e^{-\frac{2\pi j}{3}} & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{6\pi j}{3}} & -2e^{-\frac{8\pi j}{3}} & -2e^{-\frac{8\pi j}{3}} & -2e^{-\frac{12\pi j}{3}} \\ -2 & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{8\pi j}{3}} & -2e^{-\frac{4\pi j}{3}} & -2e^{-\frac{8\pi j}{3}} & -2e^{-\frac{12\pi j}{3}} & -2e^{-\frac{8\pi j}{3}} & -2e^{-\frac{16\pi j}{3}} \end{bmatrix}$$

If we take into consideration that

$$\begin{aligned} e^{-\frac{10\pi j}{3}} &= e^{-\frac{4\pi j}{3}} = -\cos 60^\circ + j \sin 60^\circ = -\frac{1}{2} + j \frac{\sqrt{3}}{2} \\ e^{-\frac{12\pi j}{3}} &= 1 \\ e^{-\frac{16\pi j}{3}} &= e^{-\frac{4\pi j}{3}} = -\cos 60^\circ + j \sin 60^\circ = -\frac{1}{2} + j \frac{\sqrt{3}}{2} \end{aligned} \quad (5.231)$$

and multiply the above matrix with W from the left, we recover matrix L .

How can we overcome the extreme sensitivity of matrix inversion to noise?

We can do it by imposing a smoothness constraint to the solution, so that it does not fluctuate too much. Let us say that we would like the second derivative of the reconstructed image to be small overall. At each pixel, the sum of the second derivatives of the image along each axis, known as the **Laplacian**, may be approximated by $\Delta^2 f(i, k)$ given by equation (5.205) derived in example 5.29. The constraint we choose to impose then is for the sum of the squares of the Laplacian values at each pixel position to be minimal:

$$\sum_{k=0}^{N-1} \sum_{i=0}^{N-1} [\Delta^2 f(i, k)]^2 = \text{minimal} \quad (5.232)$$

The value of the Laplacian at each pixel position may be computed by using the Laplacian operator which has the form of an $N^2 \times N^2$ matrix acting on column vector \mathbf{f} (of size $N^2 \times 1$),

$L\mathbf{f}$. $L\mathbf{f}$ is a vector. The sum of the squares of its elements are given by $(L\mathbf{f})^T L\mathbf{f}$. The constraint then is:

$$(L\mathbf{f})^T L\mathbf{f} = \text{minimal} \quad (5.233)$$

How can we incorporate the constraint in the inversion of the matrix?

Let us write again in matrix form the equation we want to solve for \mathbf{f} :

$$\mathbf{g} = H\mathbf{f} + \boldsymbol{\nu} \quad (5.234)$$

We assume that the noise vector $\boldsymbol{\nu}$ is not known but some of its statistical properties are known; say we know that:

$$\boldsymbol{\nu}^T \boldsymbol{\nu} = \varepsilon \quad (5.235)$$

This quantity ε is related to the variance of the noise and it could be estimated from the image itself using areas of uniform brightness only. If we substitute $\boldsymbol{\nu}$ from (5.234) into (5.235), we have:

$$(\mathbf{g} - H\mathbf{f})^T (\mathbf{g} - H\mathbf{f}) = \varepsilon \quad (5.236)$$

The problem then is to minimise (5.233) under the constraint (5.236). The solution of this problem is a filter with Fourier transform (see Box 5.6, on page 459, and example 5.36):

$$\hat{M}(u, v) = \frac{\hat{H}^*(u, v)}{|\hat{H}(u, v)|^2 + \gamma |\hat{L}(u, v)|^2} \quad (5.237)$$

By multiplying numerator and denominator with $\hat{H}(u, v)$, we can bring this filter into a form directly comparable with the inverse and the Wiener filters:

$$\hat{M}(u, v) = \frac{1}{\hat{H}(u, v)} \times \frac{|\hat{H}(u, v)|^2}{|\hat{H}(u, v)|^2 + \gamma |\hat{L}(u, v)|^2} \quad (5.238)$$

Here γ is a constant and $\hat{L}(u, v)$ is the Fourier transform of an $N \times N$ matrix L , with the following property: if we use it to multiply the image (written as a vector) from the left, the output will be an array, the same size as the image, with an estimate of the value of the Laplacian at each pixel position. The role of parameter γ is to strike the balance between smoothing the output and paying attention to the data.

Example B5.33

If \mathbf{f} is an $N \times 1$ real vector and A is an $N \times N$ matrix, show that

$$\frac{\partial \mathbf{f}^T A \mathbf{f}}{\partial \mathbf{f}} = (A + A^T) \mathbf{f} \quad (5.239)$$

Using the results of example 3.65, on page 269, we can easily see that:

$$\begin{aligned}
 \frac{\partial \mathbf{f}^T A \mathbf{f}}{\partial \mathbf{f}} &= \frac{\partial \mathbf{f}^T (A\mathbf{f})}{\partial \mathbf{f}} + \frac{\partial (\mathbf{f}^T A)\mathbf{f}}{\partial \mathbf{f}} \\
 &= A\mathbf{f} + \frac{\partial (A^T \mathbf{f})^T \mathbf{f}}{\partial \mathbf{f}} \\
 &= A\mathbf{f} + A^T \mathbf{f} = (A + A^T)\mathbf{f}
 \end{aligned} \tag{5.240}$$

Here we made use of the fact that $A\mathbf{f}$ and $A^T \mathbf{f}$ are vectors.

Example B5.34

If \mathbf{g} is the column vector that corresponds to a 3×3 image G and matrix W^{-1} is defined as in example 5.28 for $N = 3$, show that vector $W^{-1}\mathbf{g}$ is proportional to the discrete Fourier transform \hat{G} of G .

Assume that:

$$G = \begin{pmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{pmatrix} \quad \text{and} \quad W_3^{-1} = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} \\ 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} \end{pmatrix} \tag{5.241}$$

Then:

$$\begin{aligned}
 W^{-1} &= W_3^{-1} \otimes W_3^{-1} = \\
 \frac{1}{3} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} \\ 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} \\ 1 & 1 & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}3} & e^{-\frac{2\pi j}{3}3} & e^{-\frac{2\pi j}{3}3} \\ 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}3} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}4} & e^{-\frac{2\pi j}{3}3} \\ 1 & 1 & 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}3} & e^{-\frac{2\pi j}{3}4} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}3} \\ 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}4} & e^{-\frac{2\pi j}{3}3} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}3} & e^{-\frac{2\pi j}{3}2} \end{pmatrix} \tag{5.242}
 \end{aligned}$$

If we use $e^{-\frac{2\pi j}{3}3} = e^{-2\pi j} = 1$ and $e^{-\frac{2\pi j}{3}4} = e^{-\frac{2\pi j}{3}3}e^{-\frac{2\pi j}{3}} = e^{-\frac{2\pi j}{3}}$, this matrix simplifies to:

fies somehow. So we get:

$$\begin{aligned}
 & W^{-1}g = \\
 & \frac{1}{3} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} \\ 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} \\ 1 & 1 & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & 1 & e^{-\frac{2\pi j}{3}2} & 1 & e^{-\frac{2\pi j}{3}} \\ 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & 1 & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & 1 \\ 1 & 1 & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}} \\ 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & 1 & e^{-\frac{2\pi j}{3}} & e^{-\frac{2\pi j}{3}2} & 1 \end{pmatrix} \begin{pmatrix} g_{11} \\ g_{21} \\ g_{31} \\ g_{12} \\ g_{22} \\ g_{32} \\ g_{13} \\ g_{23} \\ g_{33} \end{pmatrix} = \\
 & \frac{1}{3} \begin{pmatrix} g_{11} + g_{21} + g_{31} + g_{12} + g_{22} + g_{32} + g_{13} + g_{23} + g_{33} \\ g_{11} + g_{21}e^{-\frac{2\pi j}{3}} + g_{31}e^{-\frac{2\pi j}{3}2} + g_{12} + g_{22}e^{-\frac{2\pi j}{3}} + g_{32}e^{-\frac{2\pi j}{3}2} + g_{13} + g_{23}e^{-\frac{2\pi j}{3}} + g_{33}e^{-\frac{2\pi j}{3}2} \\ \vdots \\ g_{11} + g_{21} + g_{31} + g_{12}e^{-\frac{2\pi j}{3}2} + g_{22}e^{-\frac{2\pi j}{3}} + g_{32}e^{-\frac{2\pi j}{3}2} + g_{13}e^{-\frac{2\pi j}{3}} + g_{23}e^{-\frac{2\pi j}{3}} + g_{33}e^{-\frac{2\pi j}{3}} \end{pmatrix} \quad (5.243)
 \end{aligned}$$

Careful examination of the elements of this vector shows that they are the Fourier components of G , multiplied with 3, computed at various combinations of frequencies (u, v) , for $u = 0, 1, 2$ and $v = 0, 1, 2$, and arranged as follows:

$$3 \times \begin{pmatrix} \hat{G}(0, 0) \\ \hat{G}(1, 0) \\ \hat{G}(2, 0) \\ \hat{G}(0, 1) \\ \hat{G}(1, 1) \\ \hat{G}(2, 1) \\ \hat{G}(0, 2) \\ \hat{G}(1, 2) \\ \hat{G}(2, 2) \end{pmatrix} \quad (5.244)$$

This shows that $W^{-1}\mathbf{g}$ yields N times the Fourier transform of G , as a column vector.

Example B5.35

Show that, if matrix Λ is defined by equation (5.187), then $\Lambda^*\Lambda$ is a diagonal matrix with its k^{th} element along the diagonal being $N^4|\hat{H}(k_2, k_1)|^2$, where $k_2 \equiv \text{mod}_N(k)$ and $k_1 \equiv \lfloor \frac{k}{N} \rfloor$.

From the definition of Λ , equation (5.187), we can write:

$$\Lambda = \begin{pmatrix} N^2\hat{H}(0,0) & 0 & 0 & \dots & 0 \\ 0 & N^2\hat{H}(1,0) & 0 & \dots & 0 \\ 0 & 0 & N^2\hat{H}(2,0) & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & N^2\hat{H}(N-1,N-1) \end{pmatrix} \quad (5.245)$$

Then:

$$\Lambda^* = \begin{pmatrix} N^2\hat{H}^*(0,0) & 0 & 0 & \dots & 0 \\ 0 & N^2\hat{H}^*(1,0) & 0 & \dots & 0 \\ 0 & 0 & N^2\hat{H}^*(2,0) & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & N^2\hat{H}^*(N-1,N-1) \end{pmatrix} \quad (5.246)$$

Obviously:

$$\Lambda^*\Lambda = \begin{pmatrix} N^4|\hat{H}(0,0)|^2 & 0 & 0 & \dots & 0 \\ 0 & N^4|\hat{H}(1,0)|^2 & 0 & \dots & 0 \\ 0 & 0 & N^4|\hat{H}(2,0)|^2 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & N^4|\hat{H}(N-1,N-1)|^2 \end{pmatrix} \quad (5.247)$$

Box 5.6. Derivation of the constrained matrix inversion filter

We must find the solution of the problem: minimise $(Lf)^T Lf$ with the constraint:

$$[\mathbf{g} - H\mathbf{f}]^T [\mathbf{g} - H\mathbf{f}] = \varepsilon \quad (5.248)$$

According to the method of **Lagrange multipliers** (see Box 3.8, on page 268), the solution must satisfy

$$\frac{\partial}{\partial \mathbf{f}} \left[\mathbf{f}^T L^T L \mathbf{f} + \lambda (\mathbf{g} - H \mathbf{f})^T (\mathbf{g} - H \mathbf{f}) \right] = 0 \quad (5.249)$$

where λ is a constant. This differentiation is with respect to a vector and it will yield a system of N^2 equations (one for each component of vector \mathbf{f}) which, with equation (5.248), form a system of N^2+1 equations, for the N^2+1 unknowns: the N^2 components of \mathbf{f} plus λ .

If \mathbf{a} is a vector and \mathbf{b} another one, then it can be shown (example 3.65, page 267) that:

$$\frac{\partial \mathbf{f}^T \mathbf{a}}{\partial \mathbf{f}} = \mathbf{a} \quad (5.250)$$

$$\frac{\partial \mathbf{b}^T \mathbf{f}}{\partial \mathbf{f}} = \mathbf{b} \quad (5.251)$$

Also, if A is an $N^2 \times N^2$ square matrix, then (see example 5.33, on page 456):

$$\frac{\partial \mathbf{f}^T A \mathbf{f}}{\partial \mathbf{f}} = (A + A^T) \mathbf{f} \quad (5.252)$$

We apply equations (5.250), (5.251) and (5.252) to (5.249) to perform the differentiation:

$$\frac{\partial}{\partial \mathbf{f}} \left[\underbrace{\mathbf{f}^T (L^T L) \mathbf{f}}_{\substack{\text{eqn(5.252)} \\ \text{with } A \equiv L^T L}} + \lambda (\mathbf{g}^T \mathbf{g} - \underbrace{\mathbf{g}^T H \mathbf{f}}_{\substack{\text{eqn(5.251)} \\ \text{with } \mathbf{b} \equiv H^T \mathbf{g}}}) - \underbrace{\mathbf{f}^T H^T \mathbf{g}}_{\substack{\text{eqn(5.250)} \\ \text{with } \mathbf{a} \equiv H^T \mathbf{g}}} + \underbrace{\mathbf{f}^T H^T H \mathbf{f}}_{\substack{\text{eqn(5.252) with} \\ A \equiv H^T H}} \right] = 0$$

$$\Rightarrow (2L^T L) \mathbf{f} + \lambda (-H^T \mathbf{g} - H^T \mathbf{g} + 2H^T H \mathbf{f}) = 0$$

$$\Rightarrow (H^T H + \gamma L^T L) \mathbf{f} = H^T \mathbf{g} \quad (5.253)$$

Here $\gamma \equiv \frac{1}{\lambda}$. Equation (5.253) can easily be solved in terms of block circulant matrices. Then:

$$\mathbf{f} = [H^T H + \gamma L^T L]^{-1} H^T \mathbf{g} \quad (5.254)$$

Parameter γ may be specified by substitution in equation (5.248).

Example B5.36

Solve equation (5.253).

Since H and L are block circulant matrices (see examples 5.30 and 5.32, and Box 5.5, on page 448), they may be written as:

$$\begin{aligned} H &= W\Lambda_H W^{-1} & H^T &= W\Lambda_H^* W^{-1} \\ L &= W\Lambda_L W^{-1} & L^T &= W\Lambda_L^* W^{-1} \end{aligned} \quad (5.255)$$

Then:

$$\begin{aligned} H^T H + \gamma L^T L &= W\Lambda_H^* W^{-1} W\Lambda_H W^{-1} + \gamma W\Lambda_L^* W^{-1} W\Lambda_L W^{-1} \\ &= W\Lambda_H^* \Lambda_H W^{-1} + \gamma W\Lambda_L^* \Lambda_L W^{-1} \\ &= W(\Lambda_H^* \Lambda_H + \gamma \Lambda_L^* \Lambda_L) W^{-1} \end{aligned} \quad (5.256)$$

We substitute from (5.255) and (5.256) into (5.253) to obtain:

$$W(\Lambda_H^* \Lambda_H + \gamma \Lambda_L^* \Lambda_L) W^{-1} \mathbf{f} = W\Lambda_H^* W^{-1} \mathbf{g} \quad (5.257)$$

First we multiply both sides of the equation from the left with W^{-1} , to get:

$$(\Lambda_H^* \Lambda_H + \gamma \Lambda_L^* \Lambda_L) W^{-1} \mathbf{f} = \Lambda_H^* W^{-1} \mathbf{g} \quad (5.258)$$

Notice that as Λ_H^* , $\Lambda_H^* \Lambda_H$ and $\Lambda_L^* \Lambda_L$ are diagonal matrices, this equation expresses a relationship between the corresponding elements of vectors $W^{-1} \mathbf{f}$ and $W^{-1} \mathbf{g}$ one by one.

Applying the result of example 5.35, we may write

$$\Lambda_H^* \Lambda_H = N^4 |\hat{H}(u, v)|^2 \quad \text{and} \quad \Lambda_L^* \Lambda_L = N^4 |\hat{L}(u, v)|^2 \quad (5.259)$$

where $\hat{L}(u, v)$ is the Fourier transform of matrix L . Also, by applying the results of example 5.34, we may write:

$$W^{-1} \mathbf{f} = N \hat{F}(u, v) \quad \text{and} \quad W^{-1} \mathbf{g} = N \hat{G}(u, v) \quad (5.260)$$

Finally, we replace Λ_H^* by its definition, equation (5.187), so that (5.258) becomes:

$$\begin{aligned} N^4 \left[|\hat{H}(u, v)|^2 + \gamma |\hat{L}(u, v)|^2 \right] N \hat{F}(u, v) &= N^2 \hat{H}^*(u, v) N \hat{G}(u, v) \Rightarrow \\ N^2 \frac{|\hat{H}(u, v)|^2 + \gamma |\hat{L}(u, v)|^2}{\hat{H}^*(u, v)} \hat{F}(u, v) &= \hat{G}(u, v) \end{aligned} \quad (5.261)$$

Note that when we work fully in the discrete domain, we have to use the form of the convolution theorem that applies to DFTs (see equation (2.208), on page 108). Then the correct form of equation (5.3), on page 396, is $\hat{G}(u, v) = N^2 \hat{H}(u, v) \hat{G}(u, v)$. This means that the filter with which we have to multiply the DFT of the degraded image in order to obtain the DFT of the original image is given by equation (5.237).

What is the relationship between the Wiener filter and the constrained matrix inversion filter?

Both filters look similar (see equations (5.125) and (5.238)), but they differ in many ways.

1. The Wiener filter is designed to optimise the restoration in an average statistical sense over a large ensemble of similar images. The constrained matrix inversion deals with one image only and imposes constraints on the solution sought.
2. The Wiener filter is based on the assumption that the random fields involved are homogeneous with known spectral densities. In the constrained matrix inversion it is assumed that we know only some statistical property of the noise.

In the constrained matrix restoration approach, various filters may be constructed using the same formulation, by simply changing the smoothing criterion. For example, one may try to minimise the sum of the squares of the first derivatives at all positions as opposed to the second derivatives. The only difference from formula (5.237) will be in matrix L .

Example B5.37

Calculate the DFT of the $N \times N$ matrix L' defined as:

$$L' \equiv \begin{pmatrix} -4 & 1 & 1 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix} \quad (5.262)$$

By applying formula (5.188) for $L'(x, y)$, we obtain:

$$\begin{aligned} \hat{L}'(u, v) &= \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} L'(x, y) e^{-2\pi j \left(\frac{ux}{N} + \frac{vy}{N}\right)} \\ &= \frac{1}{N^2} \left[-4 + \sum_{x=1}^{N-1} e^{-2\pi j \frac{ux}{N}} + e^{-2\pi j \frac{v}{N}} + e^{-2\pi j \frac{(N-1)v}{N}} \right] \\ &= \frac{1}{N^2} \left[-4 + \sum_{x=0}^{N-1} e^{-2\pi j \frac{ux}{N}} - 1 + e^{-2\pi j \frac{v}{N}} + e^{-2\pi j \frac{(N-1)v}{N}} \right] \\ &= \frac{1}{N^2} \left[-5 + N\delta(u) + e^{-2\pi j \frac{v}{N}} + e^{-2\pi j \frac{(N-1)v}{N}} \right] \end{aligned} \quad (5.263)$$

Here we made use of the geometric progression formula (2.165), on page 95.

Example B5.38

Calculate the magnitude of the DFT of matrix L' defined by equation (5.262).

The real and the imaginary parts of the DFT computed in example 5.37 are:

$$\begin{aligned} L_1(m, n) &\equiv \frac{1}{N^2} \left[-5 + N\delta(m) + \cos \frac{2\pi n}{N} + \cos \frac{2\pi(N-1)n}{N} \right] \\ L_2(m, n) &\equiv \frac{1}{N^2} \left[-\sin \frac{2\pi n}{N} - \sin \frac{2\pi(N-1)n}{N} \right] \end{aligned} \quad (5.264)$$

Then:

$$L_1(m, n) = \begin{cases} \frac{1}{N^2} \left[N - 5 + \cos \frac{2\pi n}{N} + \cos \frac{2\pi(N-1)n}{N} \right] & m = 0, n = 0, 1, \dots, N-1 \\ \frac{1}{N^2} \left[-5 + \cos \frac{2\pi n}{N} + \cos \frac{2\pi(N-1)n}{N} \right] & m \neq 0, n = 0, 1, \dots, N-1 \end{cases} \quad (5.265)$$

Then:

$$\begin{aligned} L_1^2(0, n) + L_2^2(0, n) &= \frac{1}{N^4} \left[(N-5)^2 + 2 + 2(N-5) \cos \frac{2\pi n}{N} \right. \\ &\quad \left. + 2(N-5) \cos \frac{2\pi(N-1)n}{N} + 2 \cos \frac{2\pi n}{N} \cos \frac{2\pi(N-1)n}{N} \right. \\ &\quad \left. + 2 \sin \frac{2\pi n}{N} \sin \frac{2\pi(N-1)n}{N} \right] \\ &= \frac{1}{N^4} \left[(N-5)^2 + 2(N-5) \cos \frac{2\pi n}{N} + 2(N-5) \cos \frac{2\pi(N-1)n}{N} \right. \\ &\quad \left. + 2 \cos \frac{2\pi(N-2)n}{N} \right] \end{aligned} \quad (5.266)$$

And:

$$L_1^2(m, n) + L_2^2(m, n) = \frac{1}{N^4} \left[25 + 2 - 10 \cos \frac{2\pi n}{N} - 10 \cos \frac{2\pi(N-1)n}{N} + 2 \cos \frac{2\pi(N-2)n}{N} \right] \quad (5.267)$$

How do we apply constrained matrix inversion in practice?

Apply the following algorithm.

Step 0: Select a smoothing operator and compute $|\hat{L}(u, v)|^2$. If you select to use the Laplacian, use formulae (5.266) and (5.267) to compute $|\hat{L}(u, v)|^2$.

Step 1: Select a value for parameter γ . It has to be higher for higher levels of noise in the image. The rule of thumb is that γ should be selected such that the two terms in the denominator of (5.237) are roughly of the same order of magnitude.

Step 2: Compute the mean grey value of the degraded image.

Step 3: Compute the DFT of the degraded image.

Step 4: Multiply the DFT of the degraded image with function $\hat{M}(u, v)$ of equation (5.237), point by point.

Step 5: Take the inverse DFT of the result.

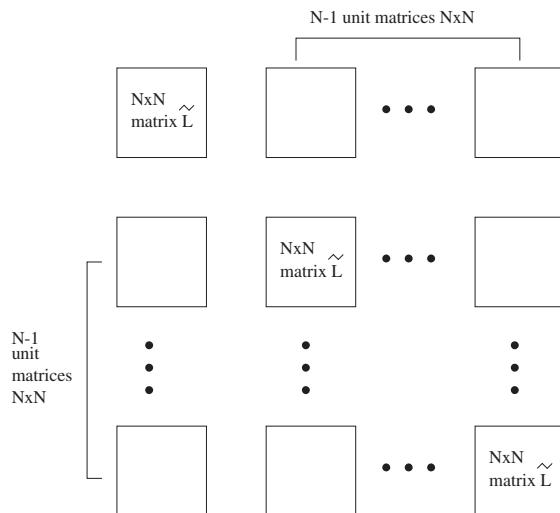
Step 6: Add the mean grey value of the degraded image to all elements of the result, to obtain the restored image.

Example 5.39

Restore the images of figures 5.5a, on page 414, and 5.9a and 5.9b, on page 418, using constrained matrix inversion.

We must first define matrix $\hat{L}(u, v)$ which expresses the constraint.

Following the steps of example 5.29, on page 449, we can see that matrix $L(i, j)$, with which we have to multiply an $N \times N$ image in order to obtain the value of the Laplacian at each position, is given by an $N^2 \times N^2$ matrix of the following structure:



Matrix $\tilde{\hat{L}}$ has the following form:

$$\tilde{L} = \begin{pmatrix} -4 & 1 & \overbrace{0 & 0 & \dots & 0}^{N-3 \text{ zeros}} & 1 \\ 1 & -4 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -4 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & -4 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & -4 & 1 \\ 1 & 0 & 0 & 0 & \dots & 1 & -4 \end{pmatrix} \quad (5.268)$$

To form the kernel we require, we must take the first column of matrix L and wrap it to form an $N \times N$ matrix. The first column of matrix L consists of the first column of matrix \tilde{L} (N elements) plus the first columns of $N-1$ unit matrices of size $N \times N$. These N^2 elements have to be written as N columns of size N next to each other, to form an $N \times N$ matrix L' , say:

$$L' = \begin{pmatrix} -4 & 1 & 1 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix} \quad (5.269)$$

It is the Fourier transform of this matrix that appears in the constrained matrix inversion filter. This Fourier transform may be computed analytically easily (see examples 5.37 and 5.38). Note that

$$|\hat{L}(m, n)|^2 = L_1^2(m, n) + L_2^2(m, n) \quad (5.270)$$

where \hat{L} is the Fourier transform of L' . This quantity has a factor $1/128^4$. Let us omit it, as it may be easily incorporated in the constant γ that multiplies it. For simplicity, let us call this modified function $A(m, n)$. From example 5.38, $A(m, n)$ is given by:

$$A(m, n) \equiv \begin{cases} 15131 + 246 \cos \frac{2\pi n}{128} + 246 \cos \frac{2\pi 127n}{128} + 2 \cos \frac{2\pi 126n}{128} & m = 0 \\ & n = 0, 1, \dots, 127 \\ 27 - 10 \cos \frac{2\pi n}{128} - 10 \cos \frac{2\pi 127n}{128} + 2 \cos \frac{2\pi 126n}{128} & m = 1, 2, \dots, 127 \\ & n = 0, 1, \dots, 127 \end{cases} \quad (5.271)$$

The frequency response function of the filter we must use is then given by substituting the frequency response function (5.50), on page 410, into equation (5.237):

$$\hat{M}(m, n) = \frac{\frac{1}{i_T \sin \frac{\pi m}{N}} \sin \frac{i_T \pi m}{N}}{\frac{\sin^2 \frac{i_T \pi m}{N}}{i_T^2 \sin^2 \frac{\pi m}{N}} + \gamma A(m, n)} e^{j \frac{\pi m}{N} (i_T - 1)} \quad (5.272)$$

For $m = 0$ we must use:

$$\hat{M}(0, n) = \frac{1}{1 + \gamma A(0, n)} \quad \text{for } 0 \leq n \leq N - 1 \quad (5.273)$$

Note from equation (5.271) that $A(0, 0)$ is much larger than 1, making the dc component of filter \hat{M} virtually 0. So, when we multiply the DFT of the degraded image with \hat{M} , we kill its dc component. This is because the constraint we have imposed did not have a dc component. To restore, therefore, the dc component of the image, after filtering, we have to compute the dc component of the input image and add it to the result, before we visualise it as an image.

Working as for the case of the Wiener filtering, we can work out that the real and imaginary parts of the Fourier transform of the original image are given by:

$$F_1(m, n) = \frac{i_T \sin \frac{\pi m}{N} \sin \frac{i_T \pi m}{N} \left[G_1(m, n) \cos \frac{(i_T - 1)\pi m}{N} - G_2(m, n) \sin \frac{(i_T - 1)\pi m}{N} \right]}{\sin^2 \frac{i_T \pi m}{N} + \gamma A(m, n) i_T^2 \sin^2 \frac{\pi m}{N}}$$

$$F_2(m, n) = \frac{i_T \sin \frac{\pi m}{N} \sin \frac{i_T \pi m}{N} \left[G_1(m, n) \sin \frac{(i_T - 1)\pi m}{N} + G_2(m, n) \cos \frac{(i_T - 1)\pi m}{N} \right]}{\sin^2 \frac{i_T \pi m}{N} + \gamma A(m, n) i_T^2 \sin^2 \frac{\pi m}{N}} \quad (5.274)$$

These formulae are valid for $0 < m \leq N - 1$ and $0 \leq n \leq N - 1$. For $m = 0$ we must use formulae:

$$F_1(0, n) = \frac{G_1(0, n)}{1 + \gamma A(0, n)}$$

$$F_2(0, n) = \frac{G_2(0, n)}{1 + \gamma A(0, n)} \quad (5.275)$$

If we take the inverse Fourier transform using functions $F_1(m, n)$ and $F_2(m, n)$ as the real and the imaginary parts, and add the dc component, we obtain the restored image. The results of restoring images 5.5b, 5.9a and 5.9b are shown in figure 5.14. Note that different values of γ , ie different levels of smoothing, have to be used for different levels of noise in the image.



5.4 Inhomogeneous linear image restoration: the whirl transform

How do we model the degradation of an image if it is linear but inhomogeneous?

In the general case, equation (1.15), on page 13, applies:

$$g(i, j) = \sum_{k=1}^N \sum_{l=1}^N f(k, l)h(k, l, i, j) \quad (5.276)$$

We have shown in Chapter 1 that this equation can be written in matrix form (see equation (1.38), on page 19):

$$\mathbf{g} = H\mathbf{f} \quad (5.277)$$

For inhomogeneous linear distortions, matrix H is not circulant or block circulant. In order to solve system (5.277) we can no longer use filtering. Instead, we must solve it by directly inverting matrix H . However, this will lead to a noisy solution, so some regularisation process must be included.

Example 5.40

In a notorious case in 2007, a criminal was putting on the Internet images of himself while committing crimes, with his face scrambled in a whirl pattern. Work out the distortion he might have been applying to the subimage of his face.

First we have to create a whirl scanning pattern. This may be given by coordinates (x, y) defined as

$$\begin{aligned} x(t) &= x_0 + \alpha t \cos(\beta t) \\ y(t) &= y_0 + \alpha t \sin(\beta t) \end{aligned} \quad (5.278)$$

where (x_0, y_0) is the “eye” of the whirl, ie its starting point, t is a parameter incremented along the scanning path, and α and β are parameters that define the exact shape of the whirl. For example, for a tight whirl pattern, α must be small. The integer coordinates (i, j) of the image that will make up the scanning path will be given by

$$\begin{aligned} i &= \lfloor i_0 + \alpha t \cos(\beta t) + 0.5 \rfloor \\ j &= \lfloor j_0 + \alpha t \sin(\beta t) + 0.5 \rfloor \end{aligned} \quad (5.279)$$

where (i_0, j_0) are the coordinates of the starting pixel, and α and β are chosen to be much smaller than 1. Parameter t is allowed to take positive integer values starting from 0. Once we have the sequence of pixels that make up the scanning pattern, we may smear their values by, for example, averaging the previous K values and assigning the result to the current pixel of the scanning sequence. For example, if the values of three successive pixels are averaged and assigned to the most recent pixel in the sequence, ($K = 3$), the values of the scrambled image \tilde{g} will be computed according to:

$$\begin{aligned} \tilde{g}([i_0 + \alpha t \cos(\beta t) + 0.5], [j_0 + \alpha t \sin(\beta t) + 0.5]) = \\ \frac{1}{3} \{ g([i_0 + \alpha(t-2) \cos[\beta(t-2)] + 0.5], [j_0 + \alpha(t-2) \sin[\beta(t-2)] + 0.5]) + \\ g([i_0 + \alpha(t-1) \cos[\beta(t-1)] + 0.5], [j_0 + \alpha(t-1) \sin[\beta(t-1)] + 0.5]) + \\ g([i_0 + \alpha t \cos[\beta t] + 0.5], [j_0 + \alpha t \sin[\beta t] + 0.5]) \} \quad (5.280) \end{aligned}$$

Example 5.41

Use the scrambling pattern of example 5.40 to work out the elements of matrix H with which one should operate on an $M \times N$ image in order to scramble it in a whirl-like way. Assume that in the scrambling pattern the values of $K + 1$ successive pixels are averaged.

Remember that the H mapping should be of size $MN \times MN$, because it will operate on the image written as a column vector with its columns written one under the other. To compute the elements of this matrix we apply the following algorithm.

Step 1: Create an array H of size $MN \times MN$ with all its elements 0.

Step 2: Choose (i_0, j_0) to be the coordinates of a pixel near the centre of the image.

Step 3: Select values for α and β , say $\alpha = 0.1$ and $\beta = \frac{2\pi}{360}5$. Select the maximum value of t you will use, say $t_{max} = 10,000$.

Step 4: Create a 1D array S of MN samples, all with flag 0. This array will be used to keep track which rows of matrix H have all their elements 0.

Step 5: Starting with $t = 0$ and carrying on with $t = 1, 2, \dots, t_{max}$, or until all elements of array S have their flag down, perform the following computations.

Compute the indices of the 2D image pixels that will have to be mixed to yield the value of output pixel (i_c, j_c) :

$$\begin{aligned} i_c &= [i_0 + \alpha t \cos(\beta t) + 0.5] & j_c &= [j_0 + \alpha t \sin(\beta t) + 0.5] \\ i_1 &= [i_0 + \alpha(t-1) \cos[\beta(t-1)] + 0.5] & j_1 &= [j_0 + \alpha(t-1) \sin[\beta(t-1)] + 0.5] \end{aligned}$$

$$\begin{aligned}
i_2 &= \lfloor i_0 + \alpha(t-2) \cos[\beta(t-2)] + 0.5 \rfloor & j_2 &= \lfloor j_0 + \alpha(t-2) \sin[\beta(t-2)] + 0.5 \rfloor \\
&\dots \\
i_K &= \lfloor i_0 + \alpha(t-K) \cos[\beta(t-K)] + 0.5 \rfloor & j_K &= \lfloor j_0 + \alpha(t-K) \sin[\beta(t-K)] + 0.5 \rfloor
\end{aligned} \tag{5.281}$$

In the above we must make sure that the values of the coordinates do not go out of range, ie i_x should take values between 1 and M and j_x should take values between 1 and N . To ensure that, we use

$$\begin{aligned}
i_k &= \min\{i_k, M\} \\
i_k &= \max\{i_k, 1\} \\
j_k &= \min\{j_k, N\} \\
j_k &= \max\{j_k, 1\}
\end{aligned}$$

for every $k = 1, 2, \dots, K$.

Step 6: Convert the coordinates computed in Step 5 into the indices of the column vector we have created from the input image by writing its columns one under the other. Given that a column has M elements indexed by i , with first value 1, the pixels identified in (5.281) will have the following indices in the column image:

$$\begin{aligned}
I_c &= (i_c - 1)M + j_c \\
I_1 &= (i_1 - 1)M + j_1 \\
I_2 &= (i_2 - 1)M + j_2 \\
&\dots \\
I_K &= (i_K - 1)M + j_K
\end{aligned} \tag{5.282}$$

Step 7: If $S(I_c) = 0$, we proceed to apply (5.284).

If $S(I_c) \neq 0$, the elements of the I_c row of matrix H have already been computed. We wish to retain, however, the most recent scrambling values, so we set them all again to 0:

$$H(I_c, J) = 0 \quad \text{for all } J = 1, 2, \dots, MN \tag{5.283}$$

Then we proceed to apply (5.284):

$$\begin{aligned}
S(I_c) &= 1 \\
H(I_c, I_c) &= H(I_c, I_1) = H(I_c, I_2) = H(I_c, I_3) = \dots = H(I_c, I_K) = 1
\end{aligned} \tag{5.284}$$

There will be some rows of H that have all their elements 0. This means that the output pixel, that corresponds to such a row, will have value 0. We may decide to allow this, in which case the scrambling we perform will not be easily invertible, as matrix H will be singular. Alternatively, we may use the following fix.

Step 8: Check all rows of matrix H , and in a row that contains only 0s, set the

diagonal element equal to 1. For example, if the 5th row contains only 0s, set the 5th element of this row to 1. This means that the output pixel, that corresponds to this row, will have the same value as the input pixel and matrix H will not be singular.

Step 9: Normalise each row of matrix H so that its elements sum up to 1.

After you have computed matrix H , you may produce the scrambled image $\tilde{\mathbf{g}}$ in column form, from the input image \mathbf{g} , also in column form, by using:

$$\tilde{\mathbf{g}} = H\mathbf{g} \quad (5.285)$$

Example 5.42

Figure 5.15a shows the image of a criminal that wishes to hide his face. Use a window of 70×70 around his face to scramble it using the algorithm of example 5.41.

In this case, $M = N = 70$. We select $t_{max} = 50,000$, $\alpha = 0.001$ and $\beta = (2\pi/360) \times 2$. The value of α was chosen small so that the spiral is tight and, therefore, more likely to pass through most, if not all, pixels. The value of β was selected to mean that each time parameter t was incremented by 1, the spiral was rotated by 2° . The value of t was selected high enough so that the spiral covers the whole square we wish to scramble. After matrix H has been created and equation (5.285) applied to the 70×70 patch written as a 4900 vector, the result is wrapped again to form a 70×70 patch which is embedded in the original image. The result is shown in figure 5.15b.

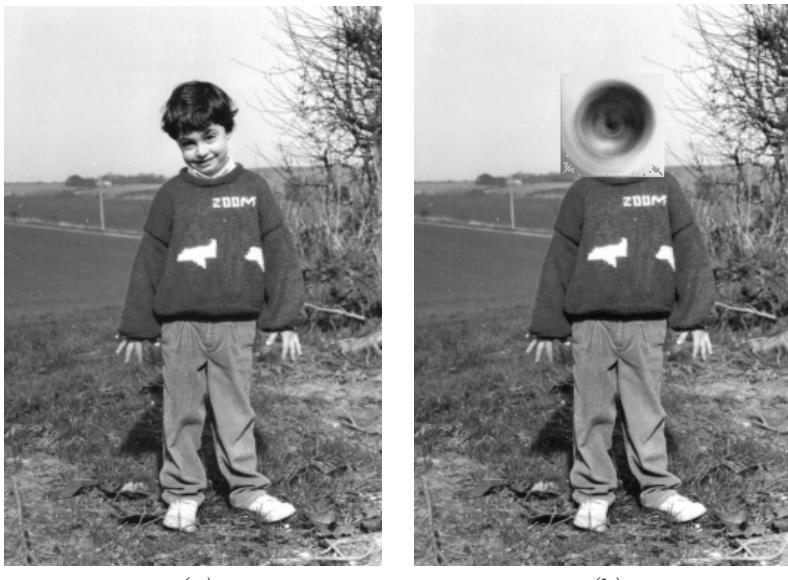


Figure 5.15: (a) “Zoom” (size 360×256). (b) After a patch of size 70×70 around the face region is scrambled.

Example B5.43

Instead of using the spiral of example 5.41 to scramble a subimage, use concentric circles to scan a square subimage of size $M \times M$.

Step 1: Create an array H of size $M^2 \times M^2$ with all its elements 0.

Step 2: Choose (i_0, j_0) to be the coordinates of a pixel near or at the centre of the image.

Step 3: Create a 1D array S of M^2 samples, all with flag 0. This array will be used to keep track which rows of matrix H have all their elements 0.

Step 4: Set $\beta = (2\pi/360)x$ where x is a small number like 1 or 2. Select a value of K , say $K = 10$.

Step 5: For α taking values from 1 to $\lfloor M/2 \rfloor$ in steps of 1, do the following.

Step 6: Starting with $t = 0$ and carrying on with $t = 1, 2, \dots, 359$, perform the following computations.

Compute the indices of the 2D image pixels that will have to be mixed to yield the value of output pixel (i_c, j_c) :

$$\begin{aligned} i_c &= \lfloor i_0 + \alpha \cos(\beta t) + 0.5 \rfloor & j_c &= \lfloor j_0 + \alpha \sin(\beta t) + 0.5 \rfloor \\ i_1 &= \lfloor i_0 + \alpha \cos[\beta(t - 1)] + 0.5 \rfloor & j_1 &= \lfloor j_0 + \alpha \sin[\beta(t - 1)] + 0.5 \rfloor \\ i_2 &= \lfloor i_0 + \alpha \cos[\beta(t - 2)] + 0.5 \rfloor & j_2 &= \lfloor j_0 + \alpha \sin[\beta(t - 2)] + 0.5 \rfloor \\ &\vdots \\ i_K &= \lfloor i_0 + \alpha \cos[\beta(t - K)] + 0.5 \rfloor & j_K &= \lfloor j_0 + \alpha \sin[\beta(t - K)] + 0.5 \rfloor \end{aligned} \tag{5.286}$$

In the above, we must make sure that the values of the coordinates do not go out of range, ie i_x and j_x should take values between 1 and M . To ensure that, we use

$$\begin{aligned} i_k &= \min\{i_k, M\} \\ i_k &= \max\{i_k, 1\} \\ j_k &= \min\{j_k, M\} \\ j_k &= \max\{j_k, 1\} \end{aligned}$$

for every $k = 1, 2, \dots, K$.

Step 7: Convert the coordinates computed in Step 5 into the indices of the column vector we have created from the input image by writing its columns one under the other. Given that a column has M elements indexed by i , with first value 1, the pixels identified in (5.286) will have the following indices in the column image:

$$\begin{aligned}
 I_c &= (i_c - 1)M + j_c \\
 I_1 &= (i_1 - 1)M + j_1 \\
 I_2 &= (i_2 - 1)M + j_2 \\
 &\dots \\
 I_K &= (i_K - 1)M + j_K
 \end{aligned} \tag{5.287}$$

Step 8: If $S(I_c) = 0$, proceed to apply (5.289).

If $S(I_c) \neq 0$, the elements of the I_c row of matrix H have already been computed. We wish to retain, however, the most recent scrambling values, so we set them all again to 0:

$$H(I_c, J) = 0 \quad \text{for all } J = 1, 2, \dots, M^2 \tag{5.288}$$

We then set:

$$\begin{aligned}
 S(I_c) &= 1 \\
 H(I_c, I_c) &= H(I_c, I_1) = H(I_c, I_2) = H(I_c, I_3) = \dots = H(I_c, I_K) = 1
 \end{aligned} \tag{5.289}$$

Step 9: Check all rows of matrix H , and in a row that contains only 0s, set the diagonal element equal to 1.

Step 10: Normalise each row of matrix H so that its elements sum up to 1.

Example B5.44

Construct a scrambling matrix that averages the values of $2K + 1$ pixels on the circle of an arc centred at each pixel of an $M \times M$ patch of an image and leaves no pixel unchanged.

Let us say that we wish to scramble a patch centred at pixel (i_0, j_0) . Consider a pixel (i, j) of the patch. The polar coordinates of this pixel with respect to the patch centre are

$$r = \sqrt{(i - i_0)^2 + (j - j_0)^2} \tag{5.290}$$

and θ , such that:

$$i = r \cos \theta \quad \text{and} \quad j = r \sin \theta \tag{5.291}$$

Then points on the arc of the same circle centred at this pixel and symmetrically placed on either side of it, have coordinates

$$\begin{aligned}
 i_k &= i_0 + r \cos(\theta + k\phi) \\
 j_k &= j_0 + r \sin(\theta + k\phi)
 \end{aligned} \tag{5.292}$$

where k takes values $-K, -K + 1, \dots, 0, \dots, K - 1, K$ and ϕ is the angle subtended by a single pixel on the circle of radius r , with its vertex at the centre of the circle, measured in rads: $\phi = 1/r$.

Then the algorithm of creating matrix H is as follows.

Step 1: Create an array H of size $M^2 \times M^2$ with all its elements 0.

Step 2: Choose (i_0, j_0) to be the coordinates of a pixel near or at the centre of the image.

Step 3: Scan every pixel (i, j) inside the subimage you wish to scramble, and compute for it its polar coordinates using equations (5.290) and (5.291), and set $\phi = 1/r$.

Step 4: Compute the indices of the 2D image pixels that will have to be mixed to yield the value of output pixel (i, j)

$$i_k = \lfloor i_0 + r \cos(\theta + k\phi) + 0.5 \rfloor \quad j_k = \lfloor j_0 + r \sin(\theta + k\phi) + 0.5 \rfloor \quad (5.293)$$

for $k = -K, -K + 1, \dots, 0, \dots, K - 1, K$.

In the above we must make sure that the values of the coordinates do not go out of range, ie i_k and j_k should take values between 1 and M . To ensure that, we use

$$\begin{aligned} i_k &= \min\{i_k, M\} \\ i_k &= \max\{i_k, 1\} \\ j_k &= \min\{j_k, M\} \\ j_k &= \max\{j_k, 1\} \end{aligned} \quad (5.294)$$

for every k .

Step 5: Convert the coordinates computed in Step 4 into the indices of the column vector we have created from the input image by writing its columns one under the other. Given that a column has M elements indexed by i , with first value 1, the pixels identified in (5.293) will have the following indices in the column image:

$$\begin{aligned} I &= (i - 1)M + j \\ I_k &= (i_k - 1)M + j_k \end{aligned} \quad (5.295)$$

Step 6: Set:

$$H(I, I_c) = H(I, I_{-K}) = H(I, I_{-K+1}) = \dots = H(I, I_0) = \dots = H(I, I_K) = 1 \quad (5.296)$$

Step 7: Check all rows of matrix H , and in a row that contains only 0s, set the diagonal element equal to 1.

Step 8: Normalise each row of matrix H so that its elements sum up to 1.

Example B5.45

Show how a thick whirl-like scrambling pattern might be created.

To keep it simple, when we compute a scrambled value for pixel (i_c, j_c) we assign it to all pixels around it inside a window of size $(2L + 1) \times (2L + 1)$. At first sight this may appear to create image patches with the same value, but due to the continuous and slow rotation of the spiral pattern, large parts of each square patch are continually over-written and the effect disappears.

Example B5.46

Use the algorithms you developed in examples 5.43 and 5.45 to scramble the face of figure 5.15a. Show the scrambled patterns and compare them with the one produced in example 5.42.

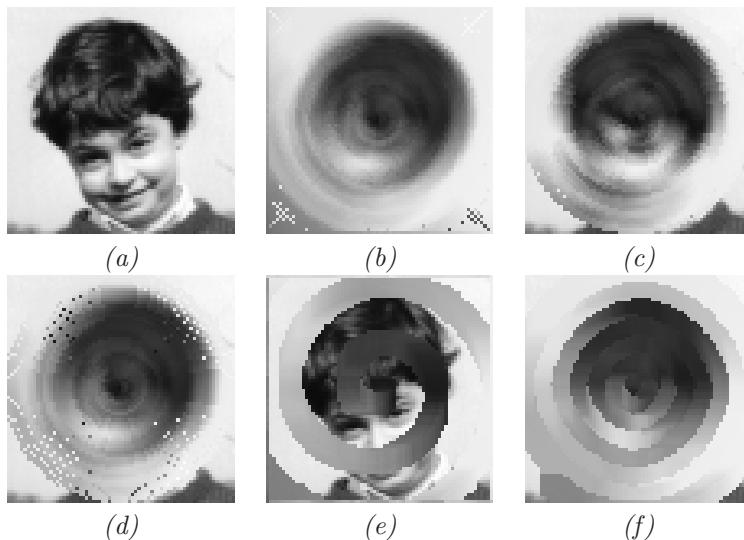


Figure 5.16: (a) The original image to be scrambled (size 70×70). (b) The scrambling obtained in example 5.42. (c) The scrambling obtained with the algorithm of example 5.43, with $x = 1$. (d) The scrambling obtained with the algorithm of example 5.43, with $x = 2$. (e) The scrambling obtained with the algorithm of example 5.45, with $\alpha = 0.1$, $x = 2$, $K = 50$, $L = 3$ and $t_{max} = 50,000$. (f) The scrambling obtained with the algorithm of example 5.45, with $\alpha = 0.03$, $x = 2$, $K = 50$, $L = 3$ and $t_{max} = 50,000$.

Example 5.47

Apply the algorithm of example 5.45 to construct matrix H with which an 8×8 image may be scrambled. Consider as the eye of the whirl pixel $(4, 4)$. Use this matrix then to scramble the flower image. Take the inverse of matrix H and apply it to the scrambled image to reconstruct the flower.

As the image we have to scramble is small, only the inner part of the whirl will be used. A large part of the whirl remains close to the central pixel, so although the image is small, we have to use a large value of K . This is because K really represents the steps along the whirl we use for averaging, not necessarily the number of distinct pixels, as many of these steps are mapped to the same pixel. After trial and error, the following parameters gave good results: $\alpha = 0.01$, $x = 1$, $K = 10$, $L = 1$ and $t_{max} = 5,000$. Matrix H , of size 64×64 , is shown in figure 5.17. Every black cell in this matrix represents a nonzero value. The values along each row of the matrix are all equal and sum up to 1. Every white cell represents a 0. We can see that there is no particular structure in this matrix.

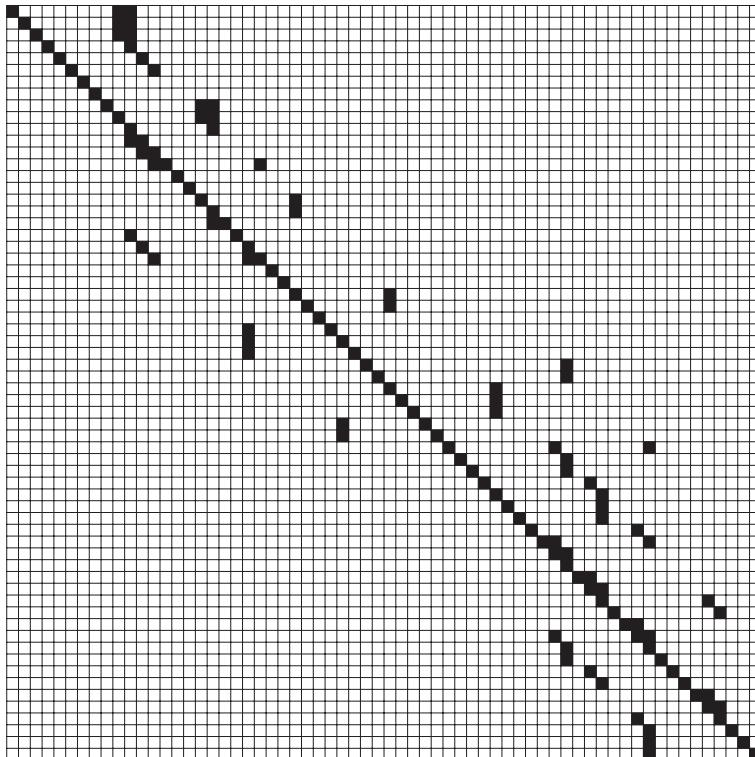


Figure 5.17: Matrix H with which an 8×8 image may be scrambled. All black cells represent nonzero positive numbers that along each row are equal and sum up to 1. White cells represent value 0.

Figure 5.18 shows the original image, the scrambled one and the unscrambled obtained by operating on the scrambled image with matrix H^{-1} . The sum of the squares of the errors of the reconstructed image is 219. This error is computed from the raw output values, where negative and higher than 255 grey values are allowed. This error is only due to the quantisation errors introduced by representing the scrambled image with integers, as matrix H could be inverted exactly. We observe that although we knew exactly what the scrambling matrix was and we applied its inverse exactly on the scrambled image, we did not get back the flower image exactly. This is because in equation (5.285), on page 471, the $\tilde{\mathbf{g}}$ is given to us as an image with all its elements rounded to the nearest integer, while the result of $H\mathbf{g}$ is actually a vector with non-integer elements. So, the application of H^{-1} to $\tilde{\mathbf{g}}$ is not the inverse of equation (5.285).

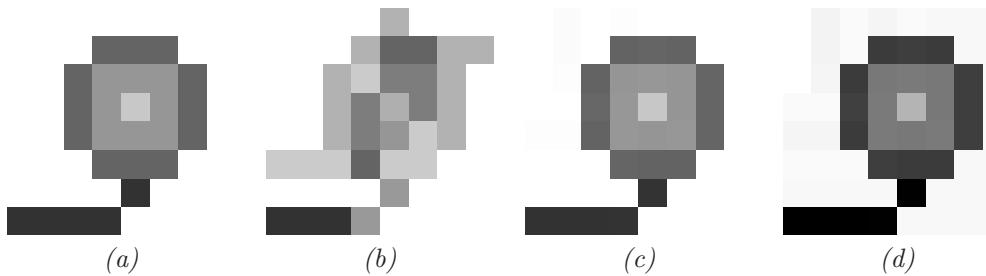


Figure 5.18: (a) Original image of size 8×8 . (b) Image scrambled with matrix H shown in figure 5.17. (c) Image recovered by operating on the scrambled image with the inverse of matrix H . The output has all its values beyond 255 and below 0 truncated to 255 and 0, respectively. The square error is 79. (d) Image recovered by operating on the scrambled image with the inverse of matrix H . The output values are mapped to the range 0 to 255, without any truncation. The square error is 39957.

How may we use constrained matrix inversion when the distortion matrix is not circulant?

Here the regularisation term has to be applied to the whole matrix, and not in the form of a filter. To do that, we have to solve the following problem: if $\tilde{\mathbf{g}}$ is the observed distorted image, \mathbf{g} is the undistorted image we are seeking and H is the distortion matrix that was used to produce $\tilde{\mathbf{g}}$, minimise $L\mathbf{g}$ and $|H^{-1}\tilde{\mathbf{g}} - \mathbf{g}|$ simultaneously. Here L is a matrix similar to the one we worked out in example 5.30, on page 449. If it operates on a column image, it yields the Laplacian at each position of the image. However, other regularisation constraints may also be used (see examples 5.48 and 5.49). We formulate the problem as one in which the norm of vectors $L\mathbf{g}$ and $H^{-1}\tilde{\mathbf{g}} - \mathbf{g}$ have to be minimised simultaneously, and express that by a so called **cost function** that has to be minimised:

$$U(\mathbf{g}) \equiv \|H^{-1}\tilde{\mathbf{g}} - \mathbf{g}\|^2 + \lambda \|L\mathbf{g}\|^2 \quad (5.297)$$

Here parameter λ allows us to control how much smoothing we want to impose to the data. Our problem now is to specify values for \mathbf{g} so that $U(\mathbf{g})$ is minimum. Note that $U(\mathbf{g})$ is quadratic with respect to all the unknowns (the elements of vector \mathbf{g}), so this is a classical minimisation problem: we must find the first derivatives of $U(\mathbf{g})$ with respect to the unknowns, set them to 0 and solve the linear system of equations that will result from that. So, our first task is to write $U(\mathbf{g})$ explicitly in terms of the components of vector \mathbf{g} and then work out its derivatives with respect to these components.

For simplicity, let us call $H^{-1}\tilde{\mathbf{g}} \equiv \mathbf{d}$. Vector \mathbf{d} is the noisy estimate we have for the elements of the original image. Now we want to select values g_i which will be close to d_i and at the same time they will change smoothly from one pixel position to the next. Assuming that the image is $M \times N$, $U(\mathbf{g})$ may be written as:

$$U(\mathbf{g}) \equiv \sum_{i=1}^{MN} (d_i - g_i)^2 + \lambda \sum_{i=1}^{MN} \left[\sum_{j=1}^{MN} L_{ij} g_j \right]^2 \quad (5.298)$$

We have to set the first derivative of $U(\mathbf{g})$ with respect to one of the components of \mathbf{g} , say g_k , equal to 0:

$$\begin{aligned} \frac{\partial U}{\partial g_k} &= 0 \\ \Rightarrow 2(d_k - g_k)(-1) + \lambda 2 \sum_{i=1}^{MN} \left[\sum_{j=1}^{MN} L_{ij} g_j \right] L_{ik} &= 0 \\ \Rightarrow g_k + \lambda \sum_{i=1}^{MN} \left[\sum_{j=1}^{MN} L_{ij} \right] L_{ik} g_j &= d_k \end{aligned} \quad (5.299)$$

Example 5.48

A column image consisting of 3 elements has been scrambled by a 3×3 matrix, the inverse of which is matrix A . We wish to restore the image, using as a regularisation constraint the requirement that every pixel has a value as similar as possible with the value of the next pixel. Work out the system of equations you will have to solve in this case.

Let us call the scrambled image we have $\tilde{\mathbf{g}}$. Operating on it with matrix A will yield a noisy estimate of the original image. Let us call it \mathbf{d} :

$$\begin{aligned} d_1 &= a_{11}\tilde{g}_1 + a_{12}\tilde{g}_2 + a_{13}\tilde{g}_3 \\ d_2 &= a_{21}\tilde{g}_1 + a_{22}\tilde{g}_2 + a_{23}\tilde{g}_3 \\ d_3 &= a_{31}\tilde{g}_1 + a_{32}\tilde{g}_2 + a_{33}\tilde{g}_3 \end{aligned} \quad (5.300)$$

We wish to estimate better values for the original image \mathbf{g} which will be such that the squares of first differences $g_1 - g_2$, $g_2 - g_3$ and $g_3 - g_1$ are as small as possible, and at

the same time g_i will be as near as possible to d_i . Note that we assumed here periodic boundary conditions, ie that the image is repeated ad infinitum, so that the last pixel g_3 has as its next neighbour the first pixel g_1 . Then the cost function we have to minimise is:

$$U = (d_1 - g_1)^2 + (d_2 - g_2)^2 + (d_3 - g_3)^2 + \lambda(g_1 - g_2)^2 + \lambda(g_2 - g_3)^2 + \lambda(g_3 - g_1)^2 \quad (5.301)$$

Parameter λ is used to control the balance between remaining faithful to the original estimate and imposing the smoothness constraint we selected to use. The derivatives of this function with respect to the three unknowns are:

$$\begin{aligned} \frac{\partial U}{\partial g_1} &= 2(d_1 - g_1)(-1) + 2\lambda(g_1 - g_2) - 2\lambda(g_3 - g_1) \\ \frac{\partial U}{\partial g_2} &= 2(d_2 - g_2)(-1) - 2\lambda(g_1 - g_2) + 2\lambda(g_2 - g_3) \\ \frac{\partial U}{\partial g_3} &= 2(d_3 - g_3)(-1) - 2\lambda(g_2 - g_3) + 2\lambda(g_3 - g_1) \end{aligned} \quad (5.302)$$

The right-hand sides of these equations should be set to 0. Then, after some manipulation, the system of equations we have to solve becomes:

$$\begin{aligned} g_1(1 + 2\lambda) - \lambda g_2 - \lambda g_3 &= d_1 \\ -\lambda g_1 + g_2(1 + 2\lambda) - \lambda g_3 &= d_2 \\ -\lambda g_1 - \lambda g_2 + (1 + 2\lambda)g_3 &= d_3 \end{aligned} \quad (5.303)$$

Example 5.49

Verify that the result of example 5.48 could have been obtained by applying formula (5.299).

We first work out matrix L , which, when it operates on the image, yields at each position the first difference with the next pixel. It is easy to see that:

$$\begin{pmatrix} g_1 - g_2 \\ g_2 - g_3 \\ g_3 - g_1 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} g_1 \\ g_2 \\ g_3 \end{pmatrix} \quad (5.304)$$

So, matrix L is:

$$\begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{pmatrix} \quad (5.305)$$

For $MN = 3$ and for $k = 1, 2, 3$, formula (5.299) has the form:

$$\begin{aligned} g_1 + \lambda \sum_{i=1}^3 \sum_{j=1}^3 L_{ij} L_{i1} g_j &= d_1 \\ g_2 + \lambda \sum_{i=1}^3 \sum_{j=1}^3 L_{ij} L_{i2} g_j &= d_2 \\ g_3 + \lambda \sum_{i=1}^3 \sum_{j=1}^3 L_{ij} L_{i3} g_j &= d_3 \end{aligned} \quad (5.306)$$

Or:

$$\begin{aligned} g_1 + \lambda \sum_{i=1}^3 (L_{i1} L_{i1} g_1 + L_{i2} L_{i1} g_2 + L_{i3} L_{i1} g_3) &= d_1 \\ g_2 + \lambda \sum_{i=1}^3 (L_{i1} L_{i2} g_1 + L_{i2} L_{i2} g_2 + L_{i3} L_{i2} g_3) &= d_2 \\ g_3 + \lambda \sum_{i=1}^3 (L_{i1} L_{i3} g_1 + L_{i2} L_{i3} g_2 + L_{i3} L_{i3} g_3) &= d_3 \end{aligned} \quad (5.307)$$

Or:

$$\begin{aligned} g_1 + \lambda(L_{11} L_{11} g_1 + L_{12} L_{11} g_2 + L_{13} L_{11} g_3 \\ + L_{21} L_{21} g_1 + L_{22} L_{21} g_2 + L_{23} L_{21} g_3 \\ + L_{31} L_{31} g_1 + L_{32} L_{31} g_2 + L_{33} L_{31} g_3) &= d_1 \\ g_2 + \lambda(L_{11} L_{12} g_1 + L_{12} L_{12} g_2 + L_{13} L_{12} g_3 \\ + L_{21} L_{22} g_1 + L_{22} L_{22} g_2 + L_{23} L_{22} g_3 \\ + L_{31} L_{32} g_1 + L_{32} L_{32} g_2 + L_{33} L_{32} g_3) &= d_2 \\ g_3 + \lambda(L_{11} L_{13} g_1 + L_{12} L_{13} g_2 + L_{13} L_{13} g_3 \\ + L_{21} L_{23} g_1 + L_{22} L_{23} g_2 + L_{23} L_{23} g_3 \\ + L_{31} L_{33} g_1 + L_{32} L_{33} g_2 + L_{33} L_{33} g_3) &= d_3 \end{aligned} \quad (5.308)$$

Substituting the values of L_{ij} , we finally get:

$$\begin{aligned} g_1 + \lambda(g_1 - g_2 + g_1 - g_3) &= d_1 \\ g_2 + \lambda(-g_1 + g_2 + g_2 - g_3) &= d_2 \\ g_3 + \lambda(-g_2 + g_3 - g_1 + g_3) &= d_3 \end{aligned} \quad (5.309)$$

This set of equations is the same as (5.303).

What happens if matrix H is really very big and we cannot take its inverse?

In such cases, instead of estimating $\mathbf{d} \equiv H^{-1}\tilde{\mathbf{g}}$ and trying to estimate \mathbf{g} so that it is as close as possible to \mathbf{d} , we try to estimate \mathbf{g} so that $H\mathbf{g}$ is as close as possible to $\tilde{\mathbf{g}}$. The function we have to minimise now has the form:

$$U(\mathbf{g}) \equiv \|\tilde{\mathbf{g}} - H\mathbf{g}\|^2 + \lambda \|L\mathbf{g}\|^2 \quad (5.310)$$

In terms of components, $U(\mathbf{g})$ may be written as:

$$U(\mathbf{g}) \equiv \sum_{i=1}^{MN} \left(\tilde{g}_i - \sum_{j=1}^{MN} H_{ij}g_j \right)^2 + \lambda \sum_{i=1}^{MN} \left[\sum_{j=1}^{MN} L_{ij}g_j \right]^2 \quad (5.311)$$

We have to work out the first derivative of $U(\mathbf{g})$ with respect to a component of \mathbf{g} , say g_k , and set it equal to 0:

$$\begin{aligned} \frac{\partial U}{\partial g_k} &= 0 \\ \Rightarrow 2 \sum_{i=1}^{MN} \left[\tilde{g}_i - \sum_{j=1}^{MN} H_{ij}g_j \right] (-H_{ik}) + 2\lambda \sum_{i=1}^{MN} \sum_{j=1}^{MN} L_{ij}g_j L_{ik} &= 0 \\ \Rightarrow -\sum_{i=1}^{MN} \tilde{g}_i H_{ik} + \sum_{i=1}^{MN} \sum_{j=1}^{MN} H_{ij}H_{ik}g_j + \lambda \sum_{i=1}^{MN} \sum_{j=1}^{MN} L_{ij}L_{ik}g_j &= 0 \\ \Rightarrow \sum_{j=1}^{MN} g_j \left[\sum_{i=1}^{MN} H_{ij}H_{ik} \right] + \sum_{j=1}^{MN} g_j \left[\lambda \sum_{i=1}^{MN} L_{ij}L_{ik} \right] &= \sum_{i=1}^{MN} H_{ik}\tilde{g}_i \\ \Rightarrow \sum_{j=1}^{MN} g_j \underbrace{\sum_{i=1}^{MN} (H_{ij}H_{ik} + \lambda L_{ij}L_{ik})}_{\equiv A_{kj}} &= \underbrace{\sum_{i=1}^{MN} H_{ik}\tilde{g}_i}_{\equiv b_k} \\ \Rightarrow \sum_{j=1}^{MN} A_{kj}g_j &= b_k \\ \Rightarrow Ag &= \mathbf{b} \end{aligned} \quad (5.312)$$

Matrix $A \equiv H^T H + \lambda L^T L$ with elements

$$A_{kj} \equiv \sum_{i=1}^{MN} (H_{ik}H_{ij} + \lambda L_{ik}L_{ij}) \quad (5.313)$$

and vector $\mathbf{b} \equiv H^T \tilde{\mathbf{g}}$ with elements

$$b_k \equiv \sum_{i=1}^{MN} H_{ik}\tilde{g}_i \quad (5.314)$$

may easily be computed from the distorted image $\tilde{\mathbf{g}}$, the distortion matrix H and the regularisation matrix L .

However, system (5.312) is still very difficult to solve, because matrix A cannot be inverted easily. It is neither circulant nor block circulant and it is of size $MN \times MN$, where MN typically could be of the order of 250,000. Approximate iterative methods may be used in such a case, like **Jacobi's method** and its more elaborate version, the **Gauss-Seidel method**.

Box 5.7. Jacobi's method for inverting large systems of linear equations

Consider the system of N linear equations with N unknowns

$$A\mathbf{x} = \mathbf{b} \quad (5.315)$$

where A is an $N \times N$ known matrix and \mathbf{b} is an $N \times 1$ known vector. If matrix A has its largest values along its diagonal, then system (5.315) may be solved iteratively, as follows.

Step 1: Write matrix A as the sum of three matrices, one containing its elements along its diagonal, one containing its elements in the upper triangle of it, and the other containing its elements in the lower triangle of it:

$$D \equiv \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & a_{NN} \end{pmatrix} \quad U \equiv \begin{pmatrix} 0 & a_{12} & \dots & a_{1N} \\ 0 & 0 & \dots & a_{2N} \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \quad L \equiv \begin{pmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ a_{N1} & a_{N2} & \dots & 0 \end{pmatrix} \quad (5.316)$$

Step 2: Make an initial guess for the values of the elements of vector \mathbf{x} . Say, set them all to 0.

Step 3: Update the values of \mathbf{x} until convergence, using the iterative formula

$$D\mathbf{x}^{(k+1)} = -(L + U)\mathbf{x}^{(k)} + \mathbf{b} \quad (5.317)$$

where superscript (k) means the value at iteration k .

Note that in practice we do not use matrices, as the computer may not be able to handle such large matrices: we instead consider each equation in turn, as follows.

Step 2 (realistic): Initialise all elements of \mathbf{x} :

$$x_i^{(0)} = \frac{b_i}{D_{ii}} \quad \text{for } i = 1, 2, \dots, N \quad (5.318)$$

Step 3 (realistic): Update the values of x_i , iteratively, until convergence:

$$x_i^{(k+1)} = \frac{1}{D_{ii}} \left(b_i - \sum_{j=1, j \neq i}^N A_{ij} x_j^{(k)} \right) \quad \text{for } i = 1, 2, \dots, N \quad (5.319)$$

The essence of this algorithm is to consider only the dominant term in each equation, assuming that this is the first term for the first equation, the second term for the second equation, the third term for the third equation, and so on, and express the unknown of the dominant term in terms of the other unknowns. Then compute an improved value of the dominant unknown in each equation from the estimated values of the other unknowns (see example 5.50). Convergence is guaranteed when the absolute value of the diagonal element in each row of matrix A is larger than the sum of the absolute values of the other elements.

Example B5.50

Consider the system of equations $Ax = b$, or explicitly

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \quad (5.320)$$

where $|a_{11}| > |a_{12}| + |a_{13}|$, $|a_{22}| > |a_{21}| + |a_{23}|$ and $|a_{33}| > |a_{31}| + |a_{32}|$. Solve it for x_1 , x_2 and x_3 , iteratively.

First we keep the dominant term in each equation, and express it in terms of the other terms:

$$\begin{aligned} a_{11}x_1 &= b_1 - a_{12}x_2 - a_{13}x_3 = b_1 - 0x_1 - a_{12}x_2 - a_{13}x_3 \\ a_{22}x_2 &= b_2 - a_{21}x_1 - a_{23}x_3 = b_2 - a_{21}x_1 - 0x_2 - a_{23}x_3 \\ a_{33}x_3 &= b_3 - a_{31}x_1 - a_{32}x_2 = b_3 - a_{31}x_1 - a_{32}x_2 - 0x_3 \end{aligned} \quad (5.321)$$

In terms of matrices, this system of equations may be written as:

$$\begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} - \begin{pmatrix} 0 & a_{12} & a_{13} \\ a_{21} & 0 & a_{23} \\ a_{31} & a_{32} & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad (5.322)$$

Note that the matrix that appears on the left-hand side is the matrix of the diagonal elements of matrix A , while the matrix that appears on the right-hand side is the sum of the upper and the lower triangular matrices of matrix A . The values of the unknowns on the right-hand side are to be the values we have estimated so far, while the values we shall compute this way will be improved estimates. This way of solving the problem is formally given in Box 5.7 as **Jacobi's method**.

Example B5.51

In example 5.50 observe that when you solve system (5.321), at each iteration you find first an improved value for x_1 , then for x_2 , and finally for x_3 . Use the improved value of x_1 when you work out the new estimate for x_2 , and the improved values of x_1 and x_2 when you work out the new estimate for x_3 .

Let us call the improved values we work out at an iteration $(\xi_1, \xi_2, \xi_3)^T$. The system (5.321) may be written as:

$$\begin{aligned} a_{11}\xi_1 &= b_1 - 0x_1 - a_{12}x_2 - a_{13}x_3 \\ a_{22}\xi_2 &= b_2 - a_{21}x_1 - 0x_2 - a_{23}x_3 \\ a_{33}\xi_3 &= b_3 - a_{31}x_1 - a_{32}x_2 - 0x_3 \end{aligned} \quad (5.323)$$

Now consider that when you compute ξ_2 , instead of using on the right-hand side the old value of x_1 , you use the newly estimated value ξ_1 , and when you work out ξ_3 , you make use of the values ξ_1 and ξ_2 , instead of x_1 and x_2 . Then this system becomes:

$$\begin{aligned} a_{11}\xi_1 &= b_1 - 0x_1 - a_{12}x_2 - a_{13}x_3 \\ a_{22}\xi_2 &= b_2 - a_{21}\xi_1 - 0x_2 - a_{23}x_3 \\ a_{33}\xi_3 &= b_3 - a_{31}\xi_1 - a_{32}\xi_2 - 0x_3 \end{aligned} \quad (5.324)$$

In terms of matrices, this may be written as:

$$\begin{aligned} \begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix} &= \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix} \\ &\quad - \begin{pmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \end{aligned} \quad (5.325)$$

Collecting all the new estimates on the left-hand side, we eventually get:

$$\left\{ \begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{pmatrix} \right\} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} - \begin{pmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad (5.326)$$

This formulation of the problem is known as the **Gauss-Seidel method** (see Box 5.8).

Box 5.8. Gauss-Seidel method for inverting large systems of linear equations

Note that in Jacobi's method (Box 5.7) we may solve the system sequentially, using for some of the unknowns the improved values we have already estimated in the current iteration, instead of waiting to update all values using those of the previous iteration (see example 5.51). In that case, the algorithm is like that in Box 5.7, but with step 3 replaced with:

Step 3 (Gauss-Seidel): Update the values of \mathbf{x} until convergence, using the iterative formula

$$(D + L)\mathbf{x}^{(k+1)} = -U\mathbf{x}^{(k)} + \mathbf{b} \quad (5.327)$$

where superscript (k) means the value at iteration k .

Again, this step in practice is performed by handling each equation separately:

Step 3 (Gauss-Seidel, realistic): Update the values of x_i until convergence, using:

$$x_i^{(k+1)} = \frac{1}{D_{ii}} \left(b_i - \sum_{j=1, j > i}^N A_{ij} x_j^{(k)} - \sum_{j=1, j < i}^N A_{ij} x_j^{(k+1)} \right) \quad \text{for } i = 1, 2, \dots, N \quad (5.328)$$

Does matrix H as constructed in examples 5.41, 5.43, 5.44 and 5.45 fulfil the conditions for using the Gauss-Seidel or the Jacobi method?

Only marginally. The diagonal elements of matrix H are not larger than the non-diagonal elements. We may, however, construct scrambling matrices H which fulfil this constraint, if we add the following step at the end of each algorithm:

Step easy: Count the nonzero elements of each row of matrix H . If it is only 1, do nothing. If there are $N > 1$ nonzero elements, set the value of the diagonal element to 0.55. Set the values of the other nonzero elements to $0.45/(N - 1)$.

Example 5.52

Degraded the face of image 5.15a with the algorithm of example 5.45 with the extra step that makes the scrambling matrix dominated by its diagonal values. Then unscramble it using the Jacobi and the Gauss-Seidel methods.

Figure 5.19a shows the original image 5.16a. Its degraded version, shown in 5.19b, was obtained by running the thick whirl algorithm with parameters $\alpha = 0.01$, $x = 2$, $K = 25$, $L = 1$ and using 5000 steps, and subsequently modifying the scrambling

matrix H so that the dominant element of each row is the diagonal element, with a value larger than the sum of the values of the remaining elements. This is necessary for the restoration to be possible with the Jacobi or the Gauss-Seidel method. To achieve this, we set all diagonal elements of the matrix to 0.55 (unless the diagonal element was the only nonzero element in its row, in which case it was left to be 1), and distributed the remaining 0.45 (necessary to make all elements of a row to sum up to 1) equally between the remaining nonzero elements of the same row. The H matrix created this way is not really effective in scrambling, as the face can be easily recognised. Nevertheless, the face is degraded, and our task is to recover its undegraded version.

Figures 5.19c and 5.19d show the recovered images by using the Jacobi and the Gauss-Seidel algorithm, respectively. The algorithms were run with the following stopping criterion: “stop either when the number of iterations reaches 35, or when the sum of the squares of the differences in the values of the pixels between successive iterations is less than 0.1”. Note that the values of the pixels were in the range [0, 1] and not [0, 255]. The Jacobi algorithm stopped after 29 iterations with a sum square error 0.09, while the Gauss-Seidel algorithm stopped after 11 iterations with sum square error 0.011. Although one cannot see any difference in the quality of the results, the Gauss-Seidel algorithm converged faster, to a more accurate solution.

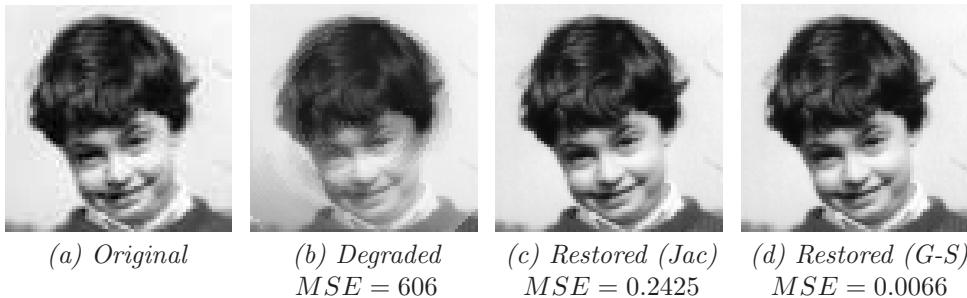


Figure 5.19: An original image, its degraded version and the recovered images by the Jacobi and the Gauss-Seidel algorithms. The mean square errors were computed using pixel values in the range [0, 255].

What happens if matrix H does not satisfy the conditions for the Gauss-Seidel method?

In that case we can minimise the cost function (5.297) by using gradient descent. Since the function is quadratic in all the unknowns, it has a well defined minimum that can be reached by simply starting from an initial guess of the solution and then updating the values of the unknowns, one at a time, so that we always move towards the minimum. This is shown schematically in figure 5.20.

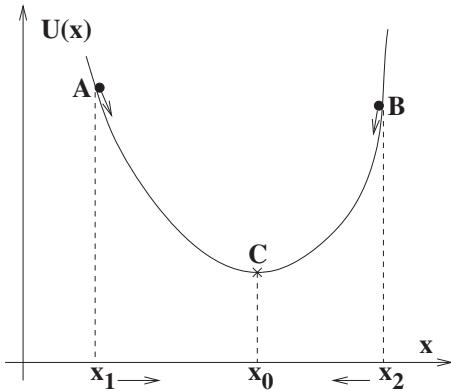


Figure 5.20: A function $U(x)$, which is quadratic in its argument, has one well defined minimum at point C , for $x = x_0$. If we guess an initial value for x_0 as $x = x_1$, we are at point A , where, for increasing value of x , the function decreases, ie $dU/dx < 0$. Then if we increase the value of the original guess, we shall move it towards x_0 . If we guess an initial value for x_0 as $x = x_2$, we are at point B , where, for increasing value of x , the function also increases, ie $dU/dx > 0$. Then if we decrease the value of the original guess, we shall move it towards x_0 . In either case, we change the value of the original guess by $-dU/dx$.

How do we apply the gradient descent algorithm in practice?

Step 1: Guess an initial restored image. You may set it to be the same as the distorted image, or equal to a flat image where all pixels have value 128.

Step 2: Consider in turn, one pixel at a time. Compute the value of the first derivative of function (5.310) with respect to the pixel value under consideration

$$\frac{\partial U}{g_k} = \sum_{j=1}^{MN} g_j A_{kj} - b_k \quad (5.329)$$

where parameters A_{kj} and b_k are given by (5.313) and (5.314), respectively.

Step 3: If $\frac{\partial U}{g_k} > 0$, decrease the value of pixel k by 1.

If $\frac{\partial U}{g_k} < 0$, increase the value of pixel k by 1.

If $\frac{\partial U}{g_k} = 0$, do nothing.

You may set as a termination criterion a fixed number of updates. For an image with MN pixels, and since we change the value of each pixel by 1 at each update, and if we start with all pixels having value 128, we may estimate that we shall need roughly $128MN$ updates, ie 128 iterations.

Example 5.53

The image shown in figure 5.21a was produced with the thick whirl algorithm with parameter setting $\alpha = 0.01$, $x = 2$, $K = 25$, $L = 1$ and by using 5000 steps. Unscramble it with the gradient descent algorithm.

To unscramble this image we assume we know matrix H that produced it. We then have to work out matrix L , used in the regularisation term of the cost function that we have to minimise. To do that, we have to generalise for an $N \times N$ image the L matrix we worked out in example 5.30, on page 449, for a 3×3 image. It turns out that the $N^2 \times N^2$ matrix L we need consists of N^2 partitions of size $N \times N$ each, with the following arrangement:

$$L = \begin{pmatrix} A & B & C & C & \dots & C & C & C & B \\ B & A & B & C & \dots & C & C & C & C \\ C & B & A & B & \dots & C & C & C & C \\ C & C & B & A & \dots & C & C & C & C \\ \vdots & \vdots \\ C & C & C & C & \dots & B & A & B & C \\ C & C & C & C & \dots & C & B & A & B \\ B & C & C & C & \dots & C & C & B & A \end{pmatrix} \quad (5.330)$$

Partition type B is the unit matrix of size $N \times N$. Partition type C has all its elements 0. Partition type A is:

$$A = \begin{pmatrix} -4 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 1 \\ 1 & -4 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 1 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -4 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & -4 & 1 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & -4 \end{pmatrix} \quad (5.331)$$

To compute matrix L in practice, we create first an $N^2 \times N^2$ matrix with all its elements 0. Then we visit each element with indices (i, j) and compute:

$$\begin{aligned} i_1 &\equiv \left\lfloor \frac{i-1}{N} \right\rfloor + 1 & i_2 &\equiv i - (i_1 - 1)N \\ j_1 &\equiv \left\lfloor \frac{j-1}{N} \right\rfloor + 1 & j_2 &\equiv j - (j_1 - 1)N \end{aligned} \quad (5.332)$$

This way, we have analysed indices i and j into their quotient and remainder with respect to N , so that indices (i_1, j_1) tell us to which partition element (i, j) belongs and indices (i_2, j_2) tell us which element of partition (i_1, j_1) element (i, j) is. Note

that, indices i and j are assumed to take values from 1 to N . Then, we use the following rules to change or not the value of element L_{ij} .

Rule 1: If $i = j$, set $L_{ij} = -4$

Rule 2: If $i_1 = j_1$ and $|i_2 - j_2| = 1$, set $L_{ij} = 1$

Rule 3: If $i_1 = j_1$ and $[(i_2 = 1 \text{ and } j_2 = N) \text{ or } (i_2 = N \text{ and } j_2 = 1)]$, set $L_{ij} = 1$

Rule 4: If $|i - j| = N$, set $L_{ij} = 1$

Rule 5: If $j_1 = N$ and $i_1 = 1$ and $i_2 = j_2$, set $L_{ij} = 1$

Rule 6: If $j_1 = 1$ and $i_1 = N$ and $i_2 = j_2$, set $L_{ij} = 1$

Assuming that we know or can guess scrambling matrix H , we then use equation (5.313) to compute matrix A for a selected value of λ , and equation (5.314) to compute vector \mathbf{b} from the input image $\tilde{\mathbf{g}}$. These computations are slow, but they have to be done only once, before the iterations of the gradient descent algorithm start.

Figures 5.21b, 5.21c and 5.21d show the results obtained after 250 iterations for $\lambda = 0.01$, 0.001 and 0.0001, respectively. The mean square error for each image and the final value of the cost function in each case are also given. In all cases the final value was reached with much fewer iterations than 250 (at around 150 iterations). Letting the program run for more iterations than necessary simply made the solution oscillate about the minimum. The starting solution in all cases was the input image, and the initial value of the cost function was of the order of 10 – 100.

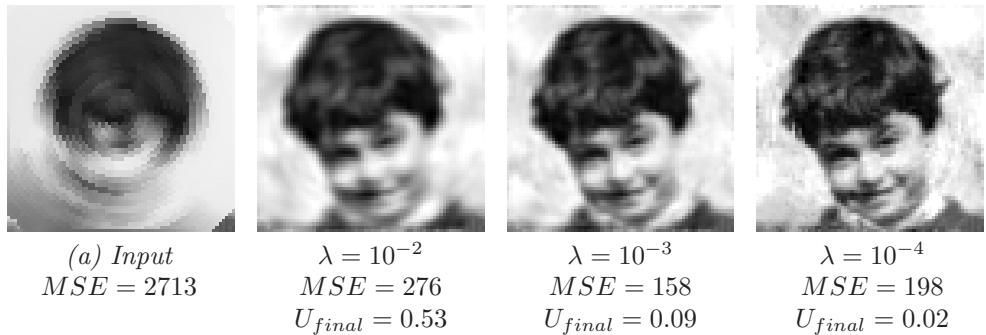


Figure 5.21: Results of the gradient descent algorithm. The scrambling matrix is assumed known. The value of λ controls the level of smoothing imposed on the solution.

What happens if we do not know matrix H ?

In that case we have also to guess the values of H . Any prior knowledge or educated guess concerning its structure might be used to guide the process so it is not totally random. The reconstruction will have to proceed in alternating steps:

Step 1: Guess a matrix H

Step 2: Restore the image using the guessed H . According to the result you get, update your guess concerning matrix H and go to Step 1.

5.5 Nonlinear image restoration: MAP estimation

What does MAP estimation mean?

MAP stands for **maximum a posteriori probability** estimation. In general, it tries to answer the question: “what is the most probable solution, given the data and the model we use?”

How do we formulate the problem of image restoration as a MAP estimation?

Any combination of grey values assigned to the image pixels is a potential solution to the restoration problem. We wish to identify the particular configuration that is most probable, given the data and the degradation model. This is a global optimisation problem, as it seeks the joint configuration of pixel values that is best, rather than trying to recover the image by performing filtering that relies on local operations.

Thus, the MAP solution to the image restoration problem is

$$f = \arg \max_x \{p(x|g, \text{model})\} \quad (5.333)$$

where f is the restored image, x is a combination of pixel values, g is the damaged image, and *model* encompasses all knowledge we have on the degradation process, including the point spread function of any blurring operator involved, any nonlinear degradation, the statistical properties and the nature of noise, etc. Here $p(x|g, \text{model})$ is the a posteriori probability density function of image x to be the image that has given rise to the observed image g , given the degradation model.

How do we select the most probable configuration of restored pixel values, given the degradation model and the degraded image?

To do that, we need to write down an expression for the a posteriori probability density function $p(x|g, \text{model})$ that appears in (5.333) (see example 5.55).

We usually consider $p(x|g, \text{model})$ to have the following form

$$p(x|g, \text{model}) = \frac{1}{Z'} e^{-H(x; g)} \quad (5.334)$$

where $H(x; g)$ is a non-negative function of configuration x , often referred to as the **cost function**, or the **energy function**, or the **Hamiltonian** of the configuration, and Z' is a normalising constant, called the **partition function**.

It is clear from (5.334) that in order to chose x so that $p(x|g, \text{model})$ is maximised, it is enough to choose x so that $H(x; g)$ is minimised.

Box 5.9. Probabilities: prior, a priori, posterior, a posteriori, conditional

Prior or a priori probability is the probability of an event to happen when we have no information concerning any special circumstances.

Posterior or a posteriori probability is the probability of an event to happen when we have some information concerning the circumstances.

Conditional probability is the probability of an event to happen, subject to the condition that another event happens first.

Example B5.54

Estimate the probability that the first person somebody meets when she walks out in the street is Chinese. Also, estimate the probability that the first person somebody meets when she walks out in the street is Chinese, if this person lives in Girona in Spain. Finally, estimate the probability that the person you will see when you open your front door is Chinese, given that the person you saw from your entrance video ringing your bell was Chinese.

In absence of any information, the prior probability that the first person somebody saw when they walked out in the street would be Chinese is 0.2. This is because there are roughly 1.2 billion Chinese in the world and the world has roughly 6 billion people. If one is in Girona, however, the posterior probability for the first person she sees when she walks out in the street being Chinese is rather slim, tending to 0. The conditional probability that the person at the doorstep is Chinese, given that the person who rung the bell was Chinese, is near 1.

Is the minimum of the cost function unique?

In general, not, unless the cost function is quadratic in the unknowns, like function (5.297), on page 477. Notice that the cost function takes a single value for a combination of values of thousands of unknowns. (For a typical 500×500 image, $H(x; g)$ depends on 250,000 unknowns.) If the function is not quadratic, it is very likely that different combinations of values of the unknowns may lead to the same value of the cost function. In some cases (see example 5.56), the cost function may take a unique, well defined minimum, which, however, corresponds to the trivial solution.

Example 5.55

Assuming that an image has been degraded by iid (independent identically distributed) additive Gaussian noise with zero mean and variance σ^2 , work out an expression for $p(x|g, \text{model})$ in (5.333).

Since the noise is additive, the true value x_{ij} of pixel (i, j) and its observed value g_{ij} are related by

$$x_{ij} = g_{ij} + n_{ij} \quad (5.335)$$

where n_{ij} is the noise value at this pixel. We may write:

$$n_{ij} = x_{ij} - g_{ij} \quad (5.336)$$

Since the noise is Gaussian, noise value n_{ij} for pixel (i, j) was drawn with probability:

$$p(n_{ij}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{n_{ij}^2}{2\sigma^2}} \quad (5.337)$$

Then the joint probability density function of the combination of noise values in all pixels is simply the product of all such probabilities, since the noise values are independent:

$$\begin{aligned} p(n) &\equiv p(n_{11}, n_{12}, n_{13}, \dots, n_{NM}) \\ &= \prod_{(i,j)} p(n_{ij}) \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^{NM} \prod_{(i,j)} e^{-\frac{n_{ij}^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^{NM} e^{-\sum_{(i,j)} \frac{n_{ij}^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^{NM} e^{-\frac{1}{2\sigma^2} \sum_{(i,j)} n_{ij}^2} \end{aligned} \quad (5.338)$$

Here n is the noise field and the image is assumed to be $N \times M$ in size. If x is the undegraded image and g is the observed image, obviously $n = x - g$. So, $p(n)$ is actually $p(x - g)$ and it is the joint probability of pixel values x to have given rise to the observed values g . In other words, $p(x - g)$ is nothing else than the a posteriori probability of x given g and the model (since we used the noise model to work out that $n = x - g$ and the form of $p(n)$). We can then write:

$$p(x|g, \text{model}) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^{NM} e^{-\frac{1}{2\sigma^2} \sum_{(i,j)} (x_{ij} - g_{ij})^2} \quad (5.339)$$

How can we select then one solution from all possible solutions that minimise the cost function?

We select the most probable solution, according to the general knowledge we have for the world. For example, we know that the world is, in general, smooth. This means that we may work out a probability $p(x)$ for any particular configuration x to arise in the first place (a priori probability), depending on its smoothness. We can then select the configuration that maximises this probability. We usually adopt for this probability the form

$$p(x) = \frac{1}{Z''} e^{-\tilde{H}(x)} \quad (5.340)$$

where Z'' is a normalising constant and $\tilde{H}(x)$ is some non-negative function that measures the roughness of configuration x .

Can we combine the posterior and the prior probabilities for a configuration x ?

Yes. We may simply multiply them, as the processes that gave rise to the prior probability $p(x)$, concerning configurations, are independent from the degradation processes that gave rise to the a posteriori probability $p(x|g, \text{model})$. So, we may write

$$p(x|g, \text{model})p(x) = \frac{1}{Z} e^{-H(x;g) - \tilde{H}(x)} \quad (5.341)$$

where Z is a normalising constant. This combination is slightly naive, as it does not consider that the two functions that appear in the exponent may take values of totally different scale. For example, if $H(x;g)$ takes values in the range $[0, 10]$ while $\tilde{H}(x)$ takes values in the range $[0.01, 0.1]$, simply adding them implies that any processing we do will be dominated by $H(x;g)$, which will drive $p(x|g, \text{model})p(x)$ down due to its high value, no matter what value $\tilde{H}(x)$ takes. So, it is better if we introduce a positive scaling constant in the exponent of one of the probabilities when we multiply them:

$$p(x|g, \text{model})p(x) = \frac{1}{Z} e^{-H(x;g) - \lambda \tilde{H}(x)} \quad (5.342)$$

Note that the insertion of a scaling factor in the exponent of (5.340) makes no difference to the relative ranking of the various configurations in terms of how probable they are. For example, the configuration that maximises $p(x)$ remains the same. What changes is the sharpness of one of the probability density functions, in relation to the other. This way we avoid large patches of the configuration space over which the value of one of the probability density functions dominates over the other. This is shown schematically in figure 5.22.

In general, we always use a scaling parameter in the exponent of functions $p(x|g, \text{model})$, $p(x)$ or $p(x|g, \text{model})p(x)$. This scaling constant is known as **temperature parameter**. So, in general, we write

$$p(x|g, \text{model})p(x) = \frac{1}{Z} e^{-\frac{1}{T} U(x;g)} \quad (5.343)$$

where $T > 0$ is the temperature parameter and $U(x;g)$ is the over all cost function.

The cost function then, that we have to minimise in order to identify the most probable configuration, is:

$$U(x;g) \equiv H(x;g) + \lambda \tilde{H}(x) \quad \lambda > 0 \quad (5.344)$$

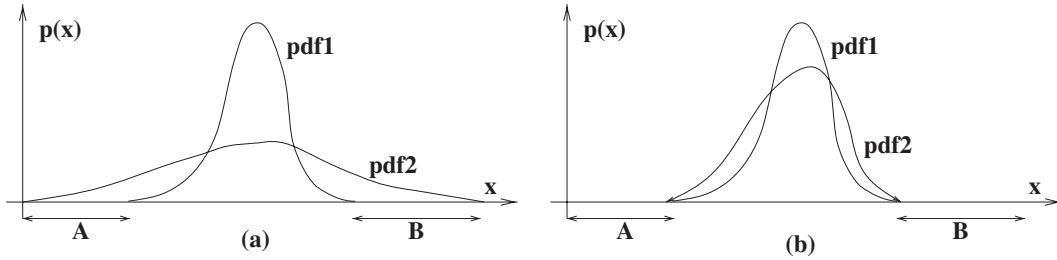


Figure 5.22: Two probability density functions (pdfs). The area under the curve of each of them is 1. (a) One of them is sharper than the other. This means that for a large range of values of variable x , pdf1 has a “veto” since when we multiply the two pdfs, pdf1 is virtually 0 in ranges A and B, forcing the product to be 0 too, irrespective of what pdf2 says. (b) We may multiply the exponent of pdf2 with a constant λ , so that we make it nonzero for roughly the same range of values of x as pdf1. The area under the curve of pdf2 will remain 1, but because its range is now restricted, it is forced to become more peaky. When we multiply these two pdfs now, none of them has the veto over the other.

This function consists of two terms: $H(x; g)$ is the term that expresses how faithful to the data the solution is. The second term expresses the prejudice we have for the world. That is, the expectation we have that the solution has to have certain characteristics in order to be acceptable. Another way of putting it, is to say that the second term incorporates any prior knowledge we have concerning the solution. Often, this term in the cost function is referred to as a **regularisation** term, because it tends to be chosen so that it makes the solution smoother. Parameter λ allows us to control the relative importance we give to prior model $\tilde{H}(x)$ in relation to the faithfulness to the data term.

Example 5.56

Work out the cost function for the problem of example 5.55 and identify the configuration that minimises it. Comment on your answer.

The cost function is the function that appears in the exponential of the probability density function defined by equation (5.339), without the minus sign. It is, therefore:

$$H(x; g) = \sum_{(i,j)} (x_{ij} - g_{ij})^2 \quad (5.345)$$

This is a non-negative function. Its minimum value is 0 and it is obtained when $x_{ij} = g_{ij}$ for all pixels (i, j) .

This is a trivial solution, that does not help us restore the image.

Example 5.57

Write down a joint probability density function that favours configurations in which a pixel has similar value to its horizontal neighbours.

Let us consider a pixel x_{ij} . We would like this value to be chosen from a probability density function that favours values similar to the values of the neighbouring pixels (ie pixels $x_{i-1,j}$ and $x_{i+1,j}$). Let us consider

$$p(x_{ij}) = \frac{1}{Z'} e^{-(x_{ij} - x_{i-1,j})^2 - (x_{ij} - x_{i+1,j})^2} \quad (5.346)$$

where Z' is a normalising constant. According to this model, the value of pixel $x_{i-1,j}$ will be chosen according to

$$p(x_{i-1,j}) = \frac{1}{Z''} e^{-(x_{i-1,j} - x_{i-2,j})^2 - (x_{i-1,j} - x_{ij})^2} \quad (5.347)$$

where Z'' is another normalising constant. Note that $p(x_{ij})$ is not independent from $p(x_{i-1,j})$, as factor $e^{-(x_{i-1,j} - x_{ij})^2}$ appears in both expressions. So, in order to find the joint probability density function $p(x_{ij}, x_{i-1,j})$, we cannot multiply the two probabilities. If, however, we consider only one of the two pairs a pixel (i,j) forms with its neighbours, say only the pair with its right neighbour, then the two probabilities will be independent and they can be multiplied to form the joint probability. So, instead of using (5.346) and (5.347), we use:

$$p(x_{ij}) = \frac{1}{Z'} e^{-(x_{ij} - x_{i+1,j})^2} \quad \text{and} \quad p(x_{i-1,j}) = \frac{1}{Z''} e^{-(x_{i-1,j} - x_{ij})^2} \quad (5.348)$$

Then the joint probability density function for the whole configuration x is

$$p(x) = \frac{1}{Z} e^{-\sum_{(i,j)} (x_{ij} - x_{i+1,j})^2} \quad (5.349)$$

where Z is a normalising constant.

Example 5.58

Work out the cost function for the problem of example 5.57 and identify the configuration that minimises it. Comment on your answer.

The cost function is:

$$\tilde{H}(x) = \sum_{(i,j)} (x_{ij} - x_{i+1,j})^2 \quad (5.350)$$

This is a non-negative function. Its minimum value is 0 and it is obtained when $x_{ij} = x_{i+1,j}$ for all pixels (i,j) . This is a trivial solution that yields a flat image: all pixels have to have the same value.

Example 5.59

Write down a cost function which, when minimised, will restore an image that has suffered from iid additive Gaussian noise and at the same time favours solutions that make a pixel similar to its horizontal neighbours.

This should be the weighted sum of the results of examples 5.56 and 5.58:

$$U(x; g) = H(x; g) + \lambda \tilde{H}(x) = \sum_{(i,j)} [(x_{ij} - g_{ij})^2 + \lambda(x_{ij} - x_{i+1,j})^2] \quad (5.351)$$

Example 5.60

Write down a cost function which, when minimised, will restore an image that has suffered from iid additive Gaussian noise and at the same time favours solutions that make a pixel similar to its vertical and horizontal neighbours.

$$U(x; g) = \sum_{(i,j)} [(x_{ij} - g_{ij})^2 + a(x_{ij} - x_{i+1,j})^2 + b(x_{ij} - x_{i,j+1})^2] \quad (5.352)$$

Here a and b are some scaling constants, which allow us to control the relative importance we give to each term. For example, if $a > b > 0$, or if $a > 0$ and $b < 0$, a pixel will tend to be more similar to its horizontal neighbours than to its vertical neighbours. The image we shall construct then will tend to have horizontal stripes.

Box 5.10. Parseval's theorem

If x_n is a real N -sample long signal and \hat{x}_k is its Fourier transform, we have:

$$\sum_{k=0}^{N-1} \hat{x}_k \hat{x}_k^* = \frac{1}{N} \sum_{n=0}^{N-1} x_n^2 \quad (5.353)$$

For a 2D $N \times M$ image g_{nm} , we have:

$$\sum_{k=0}^{N-1} \sum_{l=0}^{M-1} \hat{g}_{kl} \hat{g}_{kl}^* = \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g_{nm}^2 \quad (5.354)$$

Example B5.61

Prove Parseval's theorem for a real N -sample long signal x_n .

The DFT of a digital signal x_n is given by:

$$\hat{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn} \quad (5.355)$$

For a real signal, the complex conjugate of \hat{x}_k is:

$$\hat{x}_k^* = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{j\frac{2\pi}{N}kn} \quad (5.356)$$

Then:

$$\begin{aligned} \sum_{k=0}^{N-1} \hat{x}_k \hat{x}_k^* &= \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x_n x_m e^{-j\frac{2\pi}{N}k(n-m)} \\ &= \frac{1}{N^2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x_n x_m \sum_{k=0}^{N-1} e^{-j\frac{2\pi}{N}k(n-m)} \\ &= \frac{1}{N^2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x_n x_m N \delta(m-n) \\ &= \frac{1}{N} \sum_{n=0}^{N-1} x_n^2 \end{aligned} \quad (5.357)$$

Here, we made use of (2.164), on page 95, with $S \equiv N$, $m \equiv k$ and $t \equiv m - n$.

Example B5.62

Using Parseval's theorem, work out how the cost function defined in (5.352) may be defined in the frequency domain.

Let us consider the energy function $U(x; g)$ as defined in (5.352) in the frequency domain. First of all, the sum of the squares of function $x_{ij} - g_{ij}$ according to Parseval's theorem is equal to the sum of the squares of the Fourier coefficients of the same function, ie

$$\sum_{(i,j)} (x_{ij} - g_{ij})^2 = \frac{1}{NM} \sum_{(k,l)} (\hat{x}_{kl} - \hat{g}_{kl})^2 \quad (5.358)$$

where (k, l) are frequency indices and the hats identify the DFTs of the functions. Next, we observe that $x_{i+1,j}$ is function x_{ij} shifted by a single position along the i axis. This means that the DFT of $x_{i+1,j}$ is the DFT of x_{ij} multiplied with $e^{-j\frac{2\pi k}{N}}$. So, the DFT of function $F_{ij} \equiv x_{ij} - x_{i+1,j}$ is $\hat{x}_{kl} \left(1 - e^{-j\frac{2\pi k}{N}}\right)$. Let us consider that we can write the DFT of x_{ij} in terms of amplitude and phase as $A(k, l)e^{-j\Phi(k, l)}$. Then the DFT of function F_{ij} is:

$$\begin{aligned} \hat{F}_{kl} &= A(k, l)e^{-j\Phi(k, l)} \left(1 - e^{-j\frac{2\pi k}{N}}\right) \\ &= A(k, l)e^{-j\Phi(k, l)} - A(k, l)e^{-j(\Phi(k, l) + \frac{2\pi k}{N})} \end{aligned} \quad (5.359)$$

Then obviously,

$$\hat{F}_{kl}^* = A(k, l)e^{j\Phi(k, l)} - A(k, l)e^{j(\Phi(k, l) + \frac{2\pi k}{N})} \quad (5.360)$$

According to Parseval's theorem, sum $\sum_{(i,j)} (x_{ij} - x_{i+1,j})^2$ is equal to $(1/(NM)) \sum_{(k,l)} \hat{F}_{kl} \hat{F}_{kl}^*$:

$$\begin{aligned} \hat{F}_{kl} \hat{F}_{kl}^* &= A(k, l)^2 - A(k, l)^2 e^{j\frac{2\pi k}{N}} - A(k, l)^2 e^{-j\frac{2\pi k}{N}} + A(k, l)^2 \\ &= 2A(k, l)^2 - A(k, l)^2 \left(e^{j\frac{2\pi k}{N}} + e^{-j\frac{2\pi k}{N}}\right) \\ &= 2A(k, l)^2 - 2A(k, l)^2 \cos \frac{2\pi k}{N} \\ &= 2A(k, l)^2 \left(1 - \cos \frac{2\pi k}{N}\right) \\ &= 2\hat{x}_{kl}^2 \left(1 - \cos \frac{2\pi k}{N}\right) \end{aligned} \quad (5.361)$$

So, we may write:

$$\sum_{(i,j)} (x_{ij} - x_{i,j+1})^2 = \frac{1}{NM} 2 \sum_{(k,l)} \hat{x}_{kl}^2 \left(1 - \cos \frac{2\pi k}{M}\right) \quad (5.362)$$

Then the cost function that we have to minimise to restore the image may be written in the frequency domain as

$$U(x; g) = \sum_{(k,l)} \left[(\hat{x}_{kl} - \hat{g}_{kl})^2 + 2a\hat{x}_{kl}^2 \left(1 - \cos \frac{2\pi k}{N}\right) + 2b\hat{x}_{kl}^2 \left(1 - \cos \frac{2\pi l}{M}\right) \right] \quad (5.363)$$

where we omitted factor $1/(NM)$ as not relevant to the minimisation problem.

Example B5.63

By working in the frequency domain, work out the solution that minimises energy function (5.352).

Cost function (5.352) expressed in the frequency domain is given by (5.363). This expression involves only quantities that refer to the same site, so we may differentiate it with respect to the unknowns \hat{x}_{kl} and set the first derivative to 0 in order to find the values that minimise it:

$$\frac{\partial U(x; g)}{\partial \hat{x}_{nm}} = 2(\hat{x}_{nm} - \hat{g}_{nm}) + 4a\hat{x}_{nm} \left(1 - \cos \frac{2\pi n}{N}\right) + 4b\hat{x}_{nm} \left(1 - \cos \frac{2\pi m}{M}\right) \quad (5.364)$$

Set this derivative to 0 and solve for \hat{x}_{nm} :

$$\begin{aligned} \hat{x}_{nm} \left[1 + 2a \left(1 - \cos \frac{2\pi n}{N}\right) + 2b \left(1 - \cos \frac{2\pi m}{M}\right) \right] &= \hat{g}_{nm} \\ \Rightarrow \hat{x}_{nm} &= \frac{1}{1 + 2a \left(1 - \cos \frac{2\pi n}{N}\right) + 2b \left(1 - \cos \frac{2\pi m}{M}\right)} \times \hat{g}_{nm} \end{aligned} \quad (5.365)$$

The operation, therefore, of minimising cost function (5.352) is equivalent to using a filter in the frequency domain defined as:

$$M(k, l) \equiv \frac{1}{1 + 2a \left(1 - \cos \frac{2\pi k}{N}\right) + 2b \left(1 - \cos \frac{2\pi l}{M}\right)} \quad (5.366)$$

This filter is very similar to the filter we worked out in the constrained matrix inversion section (see equation (5.272), on page 466). As we do not assume any image blurring here, we only have in the denominator the term that is used for regularisation.

How do we model in general the cost function we have to minimise in order to restore an image?

Assume that the original value of each pixel has been distorted by:

- (i) a known linear process h ;
- (ii) a known nonlinear process ϕ ;
- (iii) additive or multiplicative white Gaussian noise with known mean μ and standard deviation σ .

Then it has been shown that the cost function that we have to minimise in order to restore the image is given by

$$U(x; g) = \tilde{H}(x) + \frac{1}{2\sigma^2} \sum_{(i,j)} [\mu - \Phi^{-1}(g_{ij}, \phi(h(x)))]^2 \quad (5.367)$$

where Φ^{-1} implies division of its arguments if the noise is multiplicative and subtraction of its arguments if the noise is additive, and $\tilde{H}(x)$ is the regularisation term.

Example 5.64

An image was blurred by a linear process with a point spread function given by:

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix} \quad (5.368)$$

The image was also subject to a nonlinear degradation process where instead of recording the value of the pixel, the square root of the value was recorded. Finally, the image was damaged by additive white Gaussian noise with mean μ and standard deviation σ . Use formula (5.367) to work out the cost function for restoring the image.

The degradation by the linear process means that if the true value at pixel (i, j) was x_{ij} , this value was replaced by the average of the 3×3 neighbourhood around position (i, j) . So, $h(x)$ in (5.367) means:

$$h(x) \rightarrow \frac{1}{9}(x_{i-1,j-1} + x_{i-1,j} + x_{i-1,j+1} + x_{i,j-1} + x_{i,j} + x_{i,j+1} + x_{i+1,j-1} + x_{i+1,j} + x_{i+1,j+1}) \quad (5.369)$$

This value was corrupted by a nonlinear process $\phi(\cdot) = \sqrt{\cdot}$. So, $\phi(h(x))$ in (5.367) means:

$$\phi(h(x)) \rightarrow \sqrt{\frac{x_{i-1,j-1} + x_{i-1,j} + x_{i-1,j+1} + x_{i,j-1} + x_{i,j} + x_{i,j+1} + x_{i+1,j-1} + x_{i+1,j} + x_{i+1,j+1}}{9}} \quad (5.370)$$

However, this was not the value that was recorded, but rather the sum of this value with a random noise component:

$$g_{ij} = n_{ij} + \frac{1}{3}\sqrt{x_{i-1,j-1} + x_{i-1,j} + x_{i-1,j+1} + x_{i,j-1} + x_{i,j} + x_{i,j+1} + x_{i+1,j-1} + x_{i+1,j} + x_{i+1,j+1}} \quad (5.371)$$

As the noise was additive, its inverse (denoted by Φ^{-1} in (5.367)) means taking the difference between the recorded value g_{ij} and the $\phi(h(x))$ value, given by (5.370). Then, the cost function of the restoration problem is:

$$U(x; g) = \tilde{H}(x) + \frac{1}{2\sigma^2} \sum_{(i,j)} \left[\mu - \left(g_{ij} - \frac{1}{3}\sqrt{x_{i-1,j-1} + x_{i-1,j} + x_{i-1,j+1} + x_{i,j-1} + x_{i,j} + x_{i,j+1} + x_{i+1,j-1} + x_{i+1,j} + x_{i+1,j+1}} \right) \right] \quad (5.372)$$

What is the reason we use a temperature parameter when we model the joint probability density function, since its does not change the configuration for which the probability takes its maximum?

The temperature parameter does not affect the solution of the problem as it is obvious from (5.343), on page 493. That is why it is omitted from the expression of the cost function we minimise in order to identify the most probable configuration (function (5.344)). However, the temperature parameter allows us to control how different the probabilities that correspond to two different configurations are. If we can control that, we can make the differentiation between configurations more or less sharp, at will. For example, we may start by allowing all configurations to be more or less equally probable, as if our “vision” were blurred at the beginning. Then, as we explore the solution space and we gradually get a better understanding of our problem, we may allow the various configurations to be more and more distinct, as if our “vision” sharpens and we can focus better to the most probable solution we seek.

How does the temperature parameter allow us to focus or defocus in the solution space?

Consider two configurations:

$$\begin{aligned} x &\equiv (x_{11}, x_{12}, \dots, x_{NN}) \\ \tilde{x} &\equiv (\tilde{x}_{11}, \tilde{x}_{12}, \dots, \tilde{x}_{NN}) \end{aligned} \quad (5.373)$$

Their corresponding probabilities of existence are:

$$p(x) = \frac{1}{Z} e^{-\frac{U(x)}{T}} \quad (5.374)$$

$$p(\tilde{x}) = \frac{1}{Z} e^{-\frac{U(\tilde{x})}{T}} \quad (5.375)$$

Let us assume that $U(x) > U(\tilde{x})$. Then the *relative* probability q of the two configurations is:

$$q \equiv \frac{p(x)}{p(\tilde{x})} = e^{-\frac{U(x)-U(\tilde{x})}{T}} \quad (5.376)$$

If $T \rightarrow +\infty \Rightarrow e^{-\frac{U(x)-U(\tilde{x})}{T}} \rightarrow 1 \Rightarrow \frac{p(x)}{p(\tilde{x})} \rightarrow 1$: both configurations are equally likely.

If $T \rightarrow 0 \Rightarrow e^{-\frac{U(x)-U(\tilde{x})}{T}} \rightarrow 0 \Rightarrow \frac{p(x)}{p(\tilde{x})} \rightarrow 0$: the configuration with the highest energy ($U(x)$) is much less probable than the other.

So, the temperature parameter allows us to control the differentiation between the different states of the system.

How do we model the prior probabilities of configurations?

This question is equivalent to asking how we define function $\tilde{H}(x)$ in (5.344) and in (5.367), ie how we define the regularisation term in the cost function. This is largely arbitrary. We may decide that we would like to favour configurations in which the first differences between neighbouring pixels are minimised. Such a prior model is known as the **membrane** model:

$$\tilde{H}(x) = \sum_{(i,j)} [(x_{ij} - x_{i+1,j})^2 + (x_{ij} - x_{i,j+1})^2] \quad (5.377)$$

Alternatively, we may decide that we would like to favour configurations in which the second differences between neighbouring pixels are minimised. Such a prior model is known as the **thin plate** model:

$$\tilde{H}(x) = \sum_{(i,j)} [(x_{i-1,j} - 2x_{ij} + x_{i+1,j})^2 + (x_{i,j-1} - 2x_{ij} + x_{i,j+1})^2] \quad (5.378)$$

These two regularisation models are schematically shown in figure 5.23.

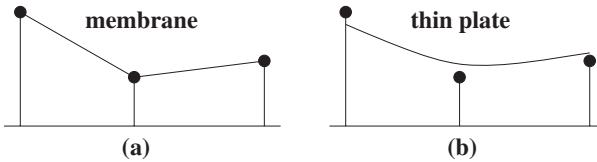


Figure 5.23: (a) For any two fixed points, the membrane model acts as if an elastic band is stretched between two pegs. (b) For any three fixed points, the thin plate model acts as if a stiff metal rod is bent to pass through them as close as possible. In 2D, the elastic band corresponds to a membrane and the stiff metal rod to a stiff metal plate.

What happens if the image has genuine discontinuities?

The genuine discontinuities of the image will be smoothed out by the membrane and the thin plate models, and the restored image will look blurred. This is because these models impose the “I know the world is by and large smooth” prejudice, everywhere in the image. This is a gross oversimplification. The following two more sophisticated models have been used to deal with this problem.

1) The imposition of a so called **line process** that is defined in between pixels and takes binary values. Its effect is to switch off the smoothness constraint when large discontinuities in the pixel values are observed. In this case, (5.377) takes the form

$$H(x) = \sum_{(i,j)} [l_{ij;i+1,j}(x_{ij} - x_{i+1,j})^2 + l_{ij;i,j+1}(x_{ij} - x_{i,j+1})^2] \quad (5.379)$$

where

$$l_{ij;mn} = \begin{cases} 1 & \text{if } |x_{ij} - x_{mn}| \leq t \\ 0 & \text{otherwise} \end{cases} \quad (5.380)$$

with t being a threshold. Note that when the difference in value between two successive pixels exceeds a threshold, the l factor sets to 0 the corresponding term that tries to drive the solution towards those two pixels having similar values.

2) The use of an edge preserving adaptive regularisation term. The line process defined above is discrete, not taking into consideration the strength of the discontinuity between two neighbouring pixels, other than thresholding it. We may define instead a continuous function that adapts according to the observed difference in value. In this case, (5.377) takes the form

$$H(x) = \sum_{(i,j)} [f(x_{ij}, x_{i+1,j})(x_{ij} - x_{i+1,j})^2 + f(x_{ij}, x_{i,j+1})(x_{ij} - x_{i,j+1})^2] \quad (5.381)$$

where

$$f(x_{ij}, x_{mn}) = \frac{2}{1 + e^{\alpha|x_{ij} - x_{mn}|}} \quad (5.382)$$

with $\alpha > 0$ acting as a “soft” threshold. Note that if $\alpha|x_{ij} - x_{mn}| \rightarrow +\infty$, $e^{\alpha|x_{ij} - x_{mn}|} \rightarrow +\infty$ and $f(x_{ij}, x_{mn}) \rightarrow 0$, so the smoothing is switched off. If $\alpha|x_{ij} - x_{mn}| \rightarrow 0$, $e^{\alpha|x_{ij} - x_{mn}|} \rightarrow 1$ and $f(x_{ij}, x_{mn}) \rightarrow 1$ and the smoothing term of these two pixels is allowed to play a role in the minimisation of the cost function.

How do we minimise the cost function?

The cost function typically depends on hundreds of thousands of variables, all the x_{ij} , and in general is not quadratic. As a result, it has lots of local minima. If we were to use any conventional optimisation technique, like gradient descent for example, what we would achieve would be to identify the nearest to the starting point local minimum, which could be well away from the sought global minimum. To avoid that, we have to use stochastic methods which allow the solution to escape from such minima. Most of the methods people use come under the general term **Monte Carlo Markov Chain (MCMC)** methods. This is because they create a chain of successive configurations in the solution space, each one differing from the previous usually in the value of a single pixel (see figure 5.24). The way this chain of successive possible solutions is constructed differentiates one method from the other. The term “Monte Carlo” is used to indicate the stochastic nature of the method. The term “Markov” is used to indicate that each new solution is created from the previous one by an explicit process, while its dependence to all other previously considered solutions is only implicit. When the temperature parameter in the probability density function is used to sharpen up the solution space gradually, as our chain of configurations grows, the method is called **simulated annealing**. This is because the gradual reduction in the value of T in (5.343) imitates the way physicists grow, for example, crystals or agglomerates of ferromagnetic materials, by gradually lowering the temperature of the physical system, in order to allow them to reach their minimum energy states.

How do we create a possible new solution from the previous one?

Method 1: Select a pixel. Select a possible new value for it uniformly distributed over all its possible values. If the new value reduces the cost function, accept the change. If the value *increases* the cost function, accept the change with probability q . This is called **Metropolis sampler**. It tends to be very slow, but it can escape from local minima (see figure 5.25).

Method 2: From the regularisation term you have adopted, work out the local conditional probability density function for a pixel to take a value s given the values of its neighbours: $p(x_{ij} = s | \text{values of neighbours})$. Select a pixel. Select the most probable of values s for this pixel and assign it to the pixel. This is called **iterative conditional modes** method. It tends to get stuck to local minima.

Method 3: From the regularisation term you have adopted, work out the local conditional probability density function for a pixel to take a value s given the values of its neighbours: $p(x_{ij} = s | \text{values of neighbours})$. Use a temperature parameter T in the exponent of this function, so that you can control the sharpness of your configuration space. Select a new value for pixel (i, j) according to this probability. This is called **Gibbs sampler**.

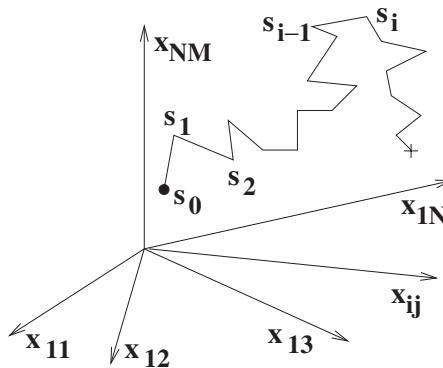


Figure 5.24: The configuration (solution) space for the restoration of an $N \times M$ image. In the configuration space we have as many axes as pixels in the image. Along each axis we measure the value of one pixel. Every point in this space represents an image. The black dot represents the starting configuration, possibly the degraded image. The cross represents the solution we are trying to find, ie the configuration which minimises the cost function globally. Stochastic optimisation methods create a chain of possible solutions, aiming to end up at the global minimum of the cost function. As each new possible solution s_i is created from the previous one s_{i-1} , this chain is a Markov chain. Some methods keep the value of the cost function for each configuration in the chain, and when they stop, they select the solution with the minimum value of the cost function. Some methods are cleverer than this and try to direct the construction of the chain so that the final configuration is the one that makes the cost function globally minimal.

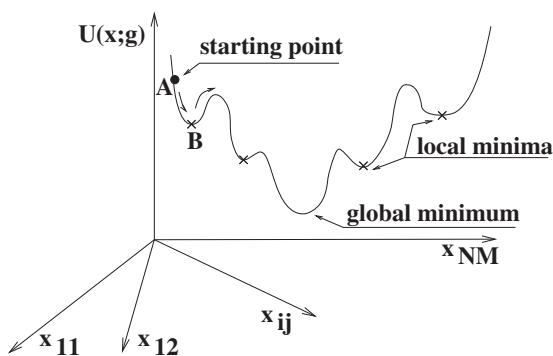


Figure 5.25: The cost function is a function of thousands of variables, as many as the pixels in the image. Here we plot the value of the cost function in terms of the values of the pixels of an $N \times M$ image. If, when we minimise the cost function, we allow only new configurations that reduce the cost function, starting from point A we shall end up to point B, which is the nearest local minimum of function $U(x; g)$. By allowing configurations that increase the value of the cost function, the Metropolis algorithm can escape from local minima.

The pixel we select to process may be chosen at random. However, a systematic approach, according to which we visit each pixel of the image in turn, is preferable. When we have visited all pixels of the image once, we say we have one iteration of the algorithm. In all methods, it is necessary to scan the image in two passes when we use the membrane model. In each pass we should update pixels that do not belong to the neighbourhood of each other. Figure 5.26a shows how the image should be split when we use the membrane model. The division of the image in two such disjoint sets is known as **coding** or **colouring** of the image. If we update the values of neighbouring pixels in the same pass, we tend to move in circles. Consider the case where we have obtained an improved value of pixel *A* in figure 5.26b on the basis of the values of its neighbours, and then we consider pixel *B*. If we update the value of *B* using the old value of pixel *A*, we may end up losing any gain we had achieved by changing the value of *A* in the output image.

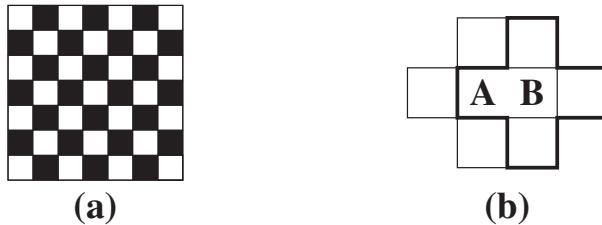


Figure 5.26: (a) The coding or colouring scheme that should be used for the restoration of an image using the membrane model. The white pixels are neighbours of the black pixels. For all sampling methods, we have to update the values of the white pixels first, taking into consideration the values of their neighbours. This constitutes one *pass* of the image. Then, with the white pixels having new values, we update the values of the black pixels taking into consideration the values of their neighbours. This is a second *pass* of the image. The two passes together constitute an *iteration*. (b) Pixels *A* and *B* should not have their values updated in the same pass. Each pixel and its neighbourhood is delineated with a line of constant thickness.

How do we know when to stop the iterations?

If T in Method 3 is fixed, we have a so called **fixed temperature annealing** (actually a contradiction in terms, but that is what it is called!). The method consists of a fixed number of passes (until one runs out of patience and time!). The value of the cost function for each constructed configuration is used to keep track of the configuration with the minimum cost value that was encountered. When the process ends, the configuration that produced the minimum cost is identified as the desired solution.

If the probability q with which we accept a configuration that *increases* the cost function in Method 1 reduces from one iteration to the next, or if T in Method 3 reduces from one iteration to the next, we have the method of **simulated annealing**. In the first case, we have simulated annealing with the Metropolis sampler, in the second, simulated annealing with the Gibbs sampler. The iterations stop when the algorithm converges, ie when the improvement in the cost function from one iteration to the next is below a threshold.

How do we reduce the temperature in simulated annealing?

The formula we adopt for reducing the temperature is known as **cooling schedule**. It has been shown that if we use

$$T(k) = \frac{C}{\ln(1+k)} \quad \text{for } k = 1, 2, \dots \quad (5.383)$$

where k is the iteration number and C is some positive constant, the simulated annealing algorithm converges to the global minimum of the cost function. This cooling schedule, however, is extremely slow. That is why, often, alternative suboptimal cooling schedules are adopted:

$$T(k) = aT(k-1) \quad \text{for } k = 1, 2, \dots \quad (5.384)$$

Here a is chosen to be a number near 1, but just below it. Typical values are 0.99 or 0.999. The starting value $T(0)$ is selected to be high enough to make all configurations more or less equally probable. One has to have a look at the typical values the cost function takes in order to select an appropriate value of $T(0)$. Typical values are of the order of 10.

How do we perform simulated annealing with the Metropolis sampler in practice?

You are advised to scale the image values from 0 to 1 when using this algorithm. The algorithm below is for the membrane model.

Step 0: Create an array the same size as the original image and set its values equal to the original image. Select a cooling schedule, a value for the cooling parameter a and a starting value for the temperature parameter. Set the current configuration x_c to be the same as the degraded image g . Compute the cost function $U(x_c = g) \equiv U_{old}$. Set $threshold = 0.001$ (or any other value of your choice).

Step 1: Consider the two subsets of pixels of the image, as identified in figure 5.26a, one after the other.

Step 2: Visit every pixel of the colouring you consider of the current configuration x_c in turn (say the white pixels in figure 5.26a). For the pixel you are currently visiting, select a new value uniformly distributed in the range of acceptable values, $[0, 1]$.

Step 2.1: Using the new value for this pixel, compute the value of the cost function. Call it U_{new} .

Step 2.2: If $U_{new} \leq U_{old}$, accept the new value for this pixel and use it to replace the value in the output array. Set $U_{old} = U_{new}$. Go to Step 2.6.

Step 2.3: If $U_{new} > U_{old}$, compute

$$q \equiv e^{-\frac{1}{T}(U_{new} - U_{old})} \quad (5.385)$$

Step 2.4: Draw a random number ξ uniformly distributed in the range $[0, 1]$.

Step 2.5: If $\xi \leq q$, accept the new value for this pixel and use it to replace the value in the output array. Set $U_{old} = U_{new}$.

Step 2.6: If you have not finished visiting all pixels of the image of the current colouring, go to the next pixel (Step 2). If you have finished with all pixels of this colouring, set the current configuration x_c equal to the output configuration and go to Step 1 in order to carry on with the pixels of the other colouring.

Step 3: When all pixels of the image have been visited, compute the global cost function of

current configuration x_c , U_{new} . If $|U_{old} - U_{new}| < threshold \times U_{old}$, exit. If not, go to Step 4.

Step 4: Set current array x_c equal to the output array. Reduce the temperature according to the cooling schedule. Set $U_{old} = U_{new}$. Go to Step 1.

Note that at the beginning of the iterations, and as T is large, the ratio q of the probability of the configuration with the new pixel value over the configuration with the old pixel value is just below 1. It is highly likely then for the number ξ we draw uniformly distributed in the range $[0, 1]$ (Step 2.4) to be smaller than q . This means that the new value of the pixel is accepted with high probability, even though it makes the configuration worse (increases the cost function). As the iterations progress and T becomes smaller, when the new configuration is less probable than the old configuration, q becomes smaller than 1 but not near 0. It rather tends to be near 0 because the lower temperature allows us to differentiate more sharply between the different configurations. Thus, when we draw ξ uniformly distributed in the range $[0, 1]$, we have much lower probability to draw a number smaller than q than we had at the beginning of the algorithm. This means that as the iterations advance, we accept configurations that worsen the cost function with smaller and smaller probability. This way, at the beginning of the algorithm, the system is “hot”, ie anything goes! We are prepared to accept bad configurations much more easily than later on. The solution is allowed to jump around in the configuration space rather easily. This allows the solution to avoid being trapped in local minima of the cost function. As the system cools down, and the values of the pixels begin to gel, we do not allow the solution to jump to other parts of the configuration space so much, as we consider that we are approaching the global minimum and we do not want to miss it by moving away from it.

Some tips concerning the implementation of this algorithm follow.

- When we compute the value of the cost function for the proposed new value of a pixel, we only compute the terms of the cost function that are affected by the value of the pixel under consideration and replace their old contribution to the cost function by the new one. This saves considerable computation time.
- We may keep a track of how many times in succession we pass through steps 2.4 and 2.5. If we keep going through that loop and not getting a new pixel value that genuinely reduces the cost function, we may decide to exit the algorithm and accept as solution the most recent configuration we had for which the cost function genuinely reduced. You may, therefore, use as an extra exiting criterion, the following: “if for X successive pixels you did not manage to draw a new value that improved the cost function, go back to the configuration that you had before the latest X pixels were visited and output it as the solution”. X here could be 50, or 100, or a similar number.
- Typically, hundreds if not thousands of iterations are required.

How do we perform simulated annealing with the Gibbs sampler in practice?

You are advised to scale the image values from 0 to 1 when using this algorithm. The algorithm below is for the membrane model.

Step 0: Create an array the same size as the original image and set its values equal to those of the original image. Select a cooling schedule, a value for the cooling parameter and a starting value for the temperature parameter. Set the current configuration x_c to be the same as the degraded image g . Compute the cost function $U(x_c = g) \equiv U_{old}$. Set $threshold = 0.001$, or another similar value of your choice.

Step 1: Visit every odd pixel of the current configuration x_c in turn (the white pixels in figure 5.26a). For every pixel, given the values of its neighbours, draw a new value with probability:

$$p(x_{ij} = x_{new} | x_{i-1,j;c}, x_{i,j-1;c}, x_{i+1,j;c}, x_{i,j+1;c}) = \frac{1}{Z} e^{-\frac{1}{T}[(x_{new} - x_{i-1,j;c})^2 + (x_{new} - x_{i,j-1;c})^2 + (x_{new} - x_{i+1,j;c})^2 + (x_{new} - x_{i,j+1;c})^2]} \quad (5.386)$$

Use this value to replace the value of this pixel in the output array. When you have finished visiting all white pixels, set the current array x_c equal to the output array and go to Step 2.

Step 2: Visit every even pixel of the current configuration x_c in turn (the black pixels in figure 5.26a). For every pixel, given the values of its neighbours, draw a new value as in Step 1. Use this value to replace the value of this pixel in the output array.

Step 3: Compute the global cost function of current configuration x_c , U_{new} . If $|U_{old} - U_{new}| < threshold \times U_{old}$, exit. If not, go to Step 4.

Step 4: Set current array x_c equal to the output array. Reduce the temperature according to the cooling schedule. Set $U_{old} = U_{new}$. Go to Step 1.

Typically, hundreds or even thousands of iterations are required.

Box 5.11. How can we draw random numbers according to a given probability density function?

Most computers have programs that can produce uniformly distributed random numbers. Let us say that we wish to draw random numbers x according to a given probability density function $p_x(x)$. Let us also say that we know how to draw random numbers y with a uniform probability density function defined in the range $[A, B]$. We may formulate the problem as follows.

Define a transformation $y = g(x)$ which is one-to-one and which is such that if y is drawn from a uniform probability density function in the range $[A, B]$, samples x are distributed according to the given probability density function $p_x(x)$.

Since we assume that relationship $y = g(x)$ is one-to-one, we may schematically depict it as shown in figure 5.27.

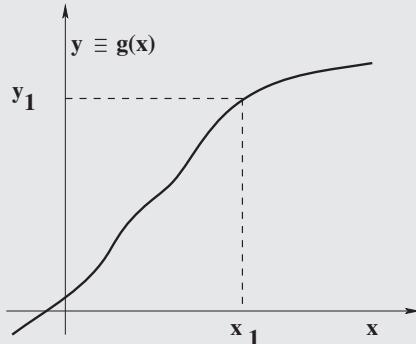


Figure 5.27: A one-to-one relationship between x and y .

It is obvious from figure 5.27 that distributions $P_y(y_1)$ and $P_x(x_1)$ of the two variables are identical, since whenever y is less than $y_1 \equiv g(x_1)$, x is less than x_1 :

$$P_y(y_1) \equiv \mathcal{P}(y \leq y_1) = \mathcal{P}(x \leq x_1) \equiv P_x(x_1) \quad (5.387)$$

The distribution of x is known, since the probability density function of x is known:

$$P_x(x_1) \equiv \int_{-\infty}^{x_1} p_x(x) dx \quad (5.388)$$

The probability density function of y is given by:

$$p_y(y) = \begin{cases} \frac{1}{B-A} & \text{for } A \leq y \leq B \\ 0 & \text{otherwise} \end{cases} \quad (5.389)$$

The distribution of y is then easily obtained:

$$P_y(y_1) \equiv \int_{-\infty}^{y_1} p_y(y) dy = \begin{cases} 0 & \text{for } y_1 \leq A \\ \frac{y_1 - A}{B - A} & \text{for } A \leq y_1 \leq B \\ 1 & \text{for } B \leq y_1 \end{cases} \quad (5.390)$$

Upon substitution in (5.387), we obtain

$$\frac{y_1 - A}{B - A} = P_x(x_1) \quad (5.391)$$

which leads to:

$$y_1 = (B - A)P_x(x_1) + A \quad (5.392)$$

Random number generators usually produce uniformly distributed numbers in the range $[0, 1]$. For $A = 0$ and $B = 1$ we have:

$$y_1 = P_x(x_1) \quad (5.393)$$

So, to produce random numbers x , distributed according to a given probability density function $p_x(x)$, we follow these steps: we compute the distribution of x , $P_x(x_1)$ using equation (5.388); we tabulate pairs of numbers $(x_1, P_x(x_1))$; we draw uniformly distributed numbers y_1 in the range $[0, 1]$; we use our tabulated numbers as a look-up table where for each $y_1 = P_x(x_1)$ we look up the corresponding x_1 . These x_1 numbers are our random samples distributed according to the way we wanted.

Example B5.65

Explain how you are going to draw random numbers according to the probability density function (5.386).

We start by rewriting the probability density function. For simplicity we call $p(x_{ij} = x_{new}|x_{i-1,j;c}, x_{i,j-1;c}, x_{i+1,j;c}, x_{i,j+1;c}) \equiv p_n$, $x_{new} \equiv x_n$ and we drop the explicit dependence on the values of the neighbours in the current configuration, ie we drop ;c from (5.386). We also expand the squares in the exponent and collect all similar terms together. Then:

$$p_n = \frac{1}{Z} e^{-\frac{1}{T} [4x_n^2 + x_{i-1,j}^2 + x_{i,j-1}^2 + x_{i+1,j}^2 + x_{i,j+1}^2 - 2x_n(x_{i-1,j} + x_{i,j-1} + x_{i+1,j} + x_{i,j+1})]} \quad (5.394)$$

Call:

$$\begin{aligned} \frac{4}{T} &\equiv A \\ \frac{1}{T}(x_{i-1,j} + x_{i,j-1} + x_{i+1,j} + x_{i,j+1}) &\equiv B \\ \frac{1}{T}(x_{i-1,j}^2 + x_{i,j-1}^2 + x_{i+1,j}^2 + x_{i,j+1}^2) &\equiv \Gamma \end{aligned} \quad (5.395)$$

Then we have:

$$p_n = \frac{1}{Z} e^{-(Ax_n^2 - 2Bx_n + \Gamma)} \quad (5.396)$$

To draw random numbers according to this probability density function, we have to compute the cumulative distribution $P_n(z)$ of p_n :

$$\begin{aligned} P_n(z) &= \int_0^z p_n dx_n \\ &= \frac{1}{Z} \int_0^z e^{-(Ax_n^2 - 2Bx_n + \Gamma)} dx_n \end{aligned} \quad (5.397)$$

We define a new variable of integration $y \equiv \sqrt{A}x_n \Rightarrow dx_n = dy/\sqrt{A}$. Then

$$P_n(z) = \frac{1}{Z\sqrt{A}} \int_0^{\sqrt{A}z} e^{-(y^2 - 2\Delta y + \Gamma)} dy \quad (5.398)$$

where we set $\Delta \equiv B/\sqrt{A}$. Then we complete the square in the exponent of the integrand

$$\begin{aligned} P_n(z) &= \frac{1}{Z\sqrt{A}} \int_0^{\sqrt{A}z} e^{-(y^2 - 2\Delta y + \Gamma + \Delta^2 - \Delta^2)} dy \\ &= \frac{1}{Z\sqrt{A}} e^{-(\Gamma - \Delta^2)} \int_0^{\sqrt{A}z} e^{-(y - \Delta)^2} dy \\ &= \frac{1}{Z\sqrt{A}} e^{-(\Gamma - \Delta^2)} \int_0^{\sqrt{A}z - \Delta} e^{-t^2} dt \end{aligned} \quad (5.399)$$

where we set $t \equiv y - \Delta$. We remember that the error function is defined as:

$$\text{erf}(z) \equiv \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \quad (5.400)$$

Then:

$$\begin{aligned} P_n(z) &= \underbrace{\frac{1}{Z\sqrt{A}} e^{-(\Gamma-\Delta^2)} \frac{\sqrt{\pi}}{2} \text{erf}(\sqrt{A}z - \Delta)}_{\Theta} \\ &= \Theta \text{erf}(\sqrt{A}z - \Delta) \end{aligned} \quad (5.401)$$

If we want to draw random numbers according to p_n , we must draw random numbers uniformly distributed between 0 and 1, treat them as values of $P_n(z)$, and from them work out the values of z . The values of z we shall work out this way will be distributed according to p_n .

To deal with the unknown scaling constant Θ , that multiplies the error function in (5.401), we sample the z variable using, say 100 points, equally spaced between 0 and 1. For each value of z , we compute $\text{erf}(\sqrt{A}z - \Delta)$. We scale the values of $\text{erf}(\sqrt{A}z - \Delta)$ to vary between 0 and 1. This way, we have a table of corresponding values of z and $P_n(z)$. Then, when we need to draw a number according to p_n , we draw a number from a uniform distribution between 0 and 1, and use it as the value of $P_n(z)$, to look up the corresponding value of z . This is the number we use in the Gibbs sampler as a new value for the pixel under consideration.

Why is simulated annealing slow?

Simulated annealing is slow because it allows the long range interactions in the image to evolve through a large number of *local* interactions. This is counter-intuitive, as in every day life, we usually try to get the rough picture right first, ie the gross long range and low frequency characteristics, and then proceed to the details.

How can we accelerate simulated annealing?

We may speed up simulated annealing by using a multiresolution approach. The philosophy of multiresolution approaches is as follows. We impose some “graininess” to the solution space. Each “grain” represents many configurations of fine resolution. Each “grain” is represented by a single configuration of coarse resolution. We then scan the configuration space by jumping from grain to grain! That is, we find a solution by using only the coarse representative configurations. Then we search for the final solution of fine resolution only among the configurations of the final grain, ie only among the configurations that are compatible with the coarse solution we found. This is schematically shown in figure 5.28.

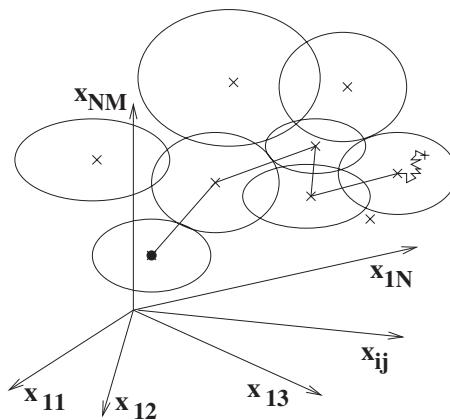


Figure 5.28: The configuration space with a coarse graininess imposed to it. Each blob represents thousands of possible solutions and is represented by a single coarse solution, indicated by an \times here. The gross resolution allows us to jump from blob to blob, thus taking large strides in the configuration space. If we have done the coarsening correctly, the solution we seek, marked here by a cross, is within the last grain, ie it is compatible with the coarse solution we find. Starting from an upsampled version then of that coarse solution, we may proceed with careful small steps towards the final solution, examining only configurations that are compatible with the coarse solution.

How can we coarsen the configuration space?

Very often people coarsen the image by blurring and subsampling it. Then they find a solution for the downsampled image by using *the same* model they had adopted for the original image, and upsample it (often by pixel replication) and use it as a starting point of finding the solution for the next higher resolution. This approach, however, is rather simplistic: the configuration space depends on the data *and* the model. It is not correct to coarsen the data only. For a start, coarsening the data only does not guarantee that the global minimum of the cost function we are seeking will be inside the final “grain” to which we stop in the configuration space (see figure 5.28). Furthermore, when we adopt a model for the image, we explicitly model the direct interaction between neighbouring pixels, but at the same time, we model *implicitly* interactions at all scales, since a pixel is influenced by its immediate neighbours, but those are influenced by their own neighbours, and so on. So, when we coarsen the configuration space, the model implied for the coarsened space is already defined and we have no freedom to arbitrarily redefine it. The correct way to proceed is to coarsen the data and the model together. This method is known as **renormalisation group transform**. The use of the renormalisation group transform guarantees that the global solution sought is among the solutions represented by the gross solution we derive using the coarse data and model representations. It also allows one to define the model for the coarse data that is compatible with the model we originally adopted for the full image. However, the practical implementation of the renormalisation group transform is quite difficult, and some shortcuts have been proposed, like the **super-coupling transform**. These methods are quite involved and beyond the scope of this book.

5.6 Geometric image restoration

How may geometric distortion arise?

Geometric distortion may arise because of the imperfections of the lens or because of the irregular movement of the sensor during image capture. In the former case, the distortion looks regular like those shown in figure 5.29. The latter case arises, for example, when an aeroplane photographs the surface of the Earth with a line scan camera: as the aeroplane wobbles, the captured image may be *inhomogeneously* distorted, with pixels displaced by as much as 4–5 interpixel distances away from their true positions and in random directions.

Why do lenses cause distortions?

Lenses cause distortions because they are not ideal. They are usually very complicated devices designed to minimise other lens undesirable effects. This is sometimes done at the expense of geometric fidelity. For example, a telephoto lens may cause the **pin-cushion distortion**, while a wide-angle lens may cause the **barrel distortion** (see figure 5.29).

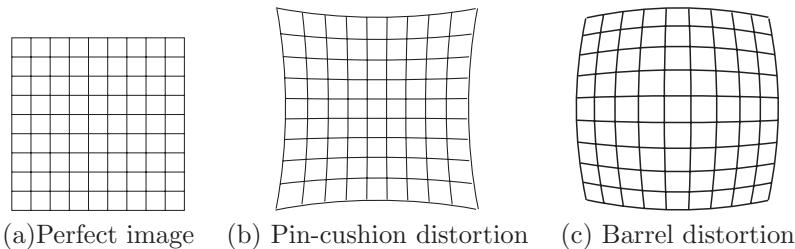


Figure 5.29: Examples of geometric distortions caused by the lens.

How can a geometrically distorted image be restored?

We start by creating an empty array of numbers the same size as the distorted image. This array will become the corrected image. Our purpose is to assign grey values to the elements of this array. This can be achieved by performing a two-stage operation: spatial transformation followed by grey level interpolation (see figure 5.30).

How do we perform the spatial transformation?

Assume that the correct position of a pixel is (x_c, y_c) and the distorted position is (x_d, y_d) (see figure 5.30). In general, there will be a transformation which leads from one set of coordinates to the other, say:

$$x_d = \mathcal{O}_x(x_c, y_c) \quad y_d = \mathcal{O}_y(x_c, y_c) \quad (5.402)$$

The type of transformation we choose depends on the type of distortion. For lens distortion we use a global transformation. For inhomogeneous distortion we use a mosaic of local transformations.

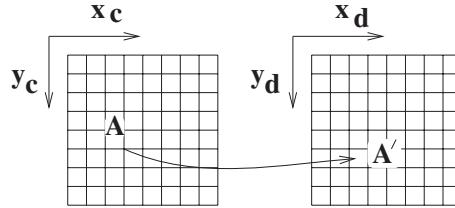


Figure 5.30: In this figure the pixels correspond to the nodes of the grids. Pixel A of the corrected grid corresponds to inter-pixel position A' of the original image. The spatial transformation is needed to identify the location of point A' and the interpolation is needed to assign it a grey value using the grey values of the surrounding pixels, as, in general, A' will not coincide with a pixel position.

How may we model the lens distortions?

The distortions caused by the lens tend to be radially symmetric. So, the transformation we have to apply in order to correct for that should also be radially symmetric. If we define $r \equiv \sqrt{x^2 + y^2}$ measured from the image centre, transformation (5.402) takes the form

$$r_d = \tilde{f}(r_c) \quad (5.403)$$

with the obvious interpretation for r_d and r_c . Function $\tilde{f}(r_c)$ has to have certain properties:

- it has to yield 0 distortion at the image centre, with the level of distortion increasing as we move towards the image periphery;
- it has to be invertible, so that one can easily move between the distorted and the undistorted coordinates;
- it has to be antisymmetric in x and y , so that negative coordinate positions move away from the centre by the same amount as the corresponding positive coordinate positions.

For all the above reasons, the model adopted usually is:

$$r_d = r_c f(r_c) \Leftrightarrow \begin{cases} x_d = x_c f(r_c) \\ y_d = y_c f(r_c) \end{cases} \quad (5.404)$$

Typical functions $f(r_c)$ used are:

$$f(r_c) = 1 + k_1 r_c \quad (5.405)$$

$$f(r_c) = 1 + k_1 r_c^2 \quad (5.406)$$

$$f(r_c) = 1 + k_1 r_c + k_2 r_c^2 \quad (5.407)$$

$$f(r_c) = 1 + k_1 r_c^2 + k_2 r_c^4 \quad (5.408)$$

$$f(r_c) = \frac{1}{1 + k_1 r_c} \quad (5.409)$$

$$f(r_c) = \frac{1}{1 + k_1 r_c^2} \quad (5.410)$$

$$f(r_c) = \frac{1 + k_1 r_c}{1 + k_2 r_c^2} \quad (5.411)$$

$$f(r_c) = \frac{1}{1 + k_1 r_c + k_2 r_c^2} \quad (5.412)$$

In these formulae k_1 and k_2 are model parameters that have to be specified. Often, when considering the individual components of such a transformation, as in (5.404), the parameters have different values for the x coordinate and the y coordinate, as the pixels are not always perfectly square.

Example B5.66

It has been noticed that the lens distortion models tend to change the scale of the image, either making it larger (in the case of the barrel distortion), or smaller (in the case of the pin-cushion distortion). In order to avoid this problem, the following spatial transformation model has been proposed:

$$\begin{aligned} x_c &= x_d(1 + ky_d^2) \\ y_c &= y_d(1 + kx_d^2) \end{aligned} \quad (5.413)$$

Explain why this model may not be acceptable.

This model is not analytically invertible to yield the distorted coordinates as functions of the correct coordinates. If we use the second of equations (5.413) to solve for y_d and substitute in the first, we obtain:

$$\begin{aligned} x_c &= x_d \left(1 + k \left(\frac{y_c}{1 + kx_d^2} \right)^2 \right) \\ \Rightarrow x_c(1 + kx_d^2)^2 &= x_d(1 + kx_d^2)^2 + kx_dy_c^2 \\ \Rightarrow x_c + x_ck^2x_d^4 + 2x_ckx_d^2 &= x_d + 2kx_d^3 + k^2x_d^5 + kx_dy_c^2 \\ \Rightarrow -k^2x_d^5 + x_ck^2x_d^4 - 2kx_d^3 + 2x_ckx_d^2 - (1 + ky_c^2)x_d &= 0 \end{aligned} \quad (5.414)$$

This is a fifth order equation in terms of x_d : not easily solvable.

In addition, this model is not radially symmetric, although lens distortion tends to be radially symmetric.

How can we model the inhomogeneous distortion?

In the general case we also consider a parametric spatial transformation model, only now we apply it locally. A commonly used transformation is

$$\begin{aligned} x_d &= c_1x_c + c_2y_c + c_3x_cy_c + c_4 \\ y_d &= c_5x_c + c_6y_c + c_7x_cy_c + c_8 \end{aligned} \quad (5.415)$$

where c_1, c_2, \dots, c_8 are some parameters. Alternatively, we may assume a nonlinear transformation, where squares of the coordinates x_c and y_c appear on the right-hand sides of the above equations.

How can we specify the parameters of the spatial transformation model?

In all cases, the values of the parameters of the spatial transformation model have to be determined from the known positions of pixels in the distorted and the undistorted grid. Such points are called **tie points**. For example, in aerial photographs of the surface of the Earth, the values of parameters c_1, \dots, c_8 in (5.415) can be determined from the known positions of specific land mark points. There are several such points scattered all over the surface of the Earth. We may use, for example, four such points to find the values of the above eight parameters and assume that these transformation equations, with the derived parameter values, hold inside the whole quadrilateral region defined by these four tie points. Then, we apply the transformation to find the position A' of point A of the corrected image in the distorted image.

For the case of transformation (5.404), a calibration pattern is necessary (see example 5.70).

Why is grey level interpolation needed?

It is likely that point A' will not have integer coordinates even though the coordinates of point A in the (x_c, y_c) space are integer. This means that we do not actually know the grey value at position A' . That is when the grey level interpolation process comes into play. The grey value at position A' can be estimated from the values at its four nearest neighbouring pixels in the (x_d, y_d) space by some method, for example by **bilinear interpolation**. We assume that inside each little square the grey value is a simple function of the positional coordinates

$$g(x_d, y_d) = \alpha x_d + \beta y_d + \gamma x_d y_d + \delta \quad (5.416)$$

where α, \dots, δ are some parameters. We apply this formula to the four corner pixels to derive values for α, β, γ and δ and then use these values to calculate $g(x_d, y_d)$ at the position of A' point.

Figure 5.31 shows in magnification the neighbourhood of point A' in the distorted image with the four nearest pixels at the neighbouring positions with integer coordinates.

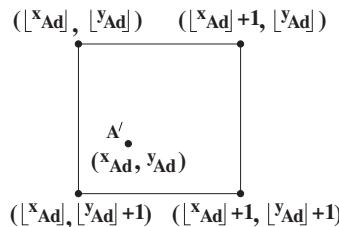


Figure 5.31: The non-integer position of point A' is surrounded by four pixels at integer positions, with known grey values ($\lfloor x \rfloor$ means integer part of x).

Simpler, as well as more sophisticated, methods of interpolation may be employed. For example, the simplest method that can be used is the **nearest neighbour interpolation** method, where A' gets the grey value of the pixel which is nearest to it. A more sophisticated method is to fit a higher order surface through a larger patch of pixels around A' and find the value at A' from the equation of that surface.

Example 5.67

In the figure below, the grid on the right is a geometrically distorted image and has to be registered with the reference image on the left, using points A, B, C and D as tie points. The entries in the image on the left indicate coordinate positions. Assuming that the distortion within the rectangle $ABCD$ can be modelled by bilinear interpolation and the grey value at an interpixel position can be modelled by bilinear interpolation too, find the grey value at pixel position $(2, 2)$ in the reference image.

A	B				A					
$\boxed{(0, 0)}$	(1, 0)	(2, 0)	$\boxed{(3, 0)}$	(4, 0)		3	2	5	1	
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)	3	3	4	\boxed{B}	4	(5.417)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	0	5	5	3	3	
$C \boxed{(0, 3)}$	(1, 3)	(2, 3)	$\boxed{(3, 3)}$	D	(4, 3)	0	\boxed{C}	2	1	2
(0, 4)	(1, 4)	(2, 4)	(3, 4)	(4, 4)	2	1	0	3	\boxed{D}	

Assume that position (x_d, y_d) of a pixel in the distorted image is given in terms of its position (x_c, y_c) in the reference image by:

$$\begin{aligned} x_d &= c_1 x_c + c_2 y_c + c_3 x_c y_c + c_4 \\ y_d &= c_5 x_c + c_6 y_c + c_7 x_c y_c + c_8 \end{aligned} \quad (5.418)$$

We have the following set of the corresponding coordinates between the two grids, using the four tie points:

	Distorted (x_d, y_d) coords.	Reference (x_c, y_c) coords.
Pixel A	(0, 0)	(0, 0)
Pixel B	(3, 1)	(3, 0)
Pixel C	(1, 3)	(0, 3)
Pixel D	(4, 4)	(3, 3)

We can use these to calculate the values of parameters c_1, \dots, c_8 :

$$\begin{aligned} \text{Pixel A:} \quad & c_4 = 0 \\ & c_8 = 0 \\ \text{Pixel B:} \quad & \left. \begin{array}{l} 3 = 3c_1 \\ 1 = 3c_5 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} c_1 = 1 \\ c_5 = \frac{1}{3} \end{array} \right. \\ \text{Pixel C:} \quad & \left. \begin{array}{l} 1 = 3c_2 \\ 3 = 3c_6 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} c_2 = \frac{1}{3} \\ c_6 = 1 \end{array} \right. \\ \text{Pixel D:} \quad & \left. \begin{array}{l} 4 = 3 + 3 \times \frac{1}{3} + 9c_3 \\ 4 = 3 \times \frac{1}{3} + 3 + 9c_7 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} c_3 = 0 \\ c_7 = 0 \end{array} \right. \end{aligned} \quad (5.420)$$

The distorted coordinates, therefore, of any pixel within rectangle $ABDC$, are given by:

$$x_d = x_c + \frac{y_c}{3}, \quad y_d = \frac{x_c}{3} + y_c \quad (5.421)$$

For $x_c = y_c = 2$ we have $x_d = 2 + \frac{2}{3}$, $y_d = \frac{2}{3} + 2$. So, the coordinates of pixel $(2, 2)$ in the distorted image are $(2\frac{2}{3}, 2\frac{2}{3})$. This position is located between pixels in the distorted image and actually between pixels with grey values as shown in figure 5.32.

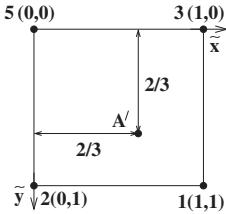


Figure 5.32: A local coordinate system (\tilde{x}, \tilde{y}) is defined to facilitate the calculation of the parameters of the bilinear interpolation between pixels at locations $(2, 2)$, $(3, 2)$, $(2, 3)$ and $(3, 3)$ of the original distorted grid, with corresponding grey values 5, 3, 2 and 1.

We define a local coordinate system (\tilde{x}, \tilde{y}) , so that the pixel at the top left corner has coordinate position $(0, 0)$, the pixel at the top right corner has coordinates $(1, 0)$, the one at the bottom left $(0, 1)$ and the one at the bottom right $(1, 1)$.

Assuming that the grey value between four pixels can be computed from the grey values in the four corner pixels, with bilinear interpolation, we have:

$$g(\tilde{x}, \tilde{y}) = \alpha\tilde{x} + \beta\tilde{y} + \gamma\tilde{x}\tilde{y} + \delta \quad (5.422)$$

Applying this for the four neighbouring pixels, we have:

$$\begin{aligned} 5 &= \alpha \times 0 + \beta \times 0 + \gamma \times 0 + \delta \Rightarrow \delta = 5 \\ 3 &= \alpha \times 1 + \beta \times 0 + \gamma \times 0 + 5 \Rightarrow \alpha = -2 \\ 2 &= (-2) \times 0 + \beta \times 1 + \gamma \times 0 + 5 \Rightarrow \beta = -3 \\ 1 &= (-2) \times 1 + (-3) \times 1 + \gamma \times 1 \times 1 + 5 \Rightarrow \gamma = 1 \end{aligned} \quad (5.423)$$

Therefore:

$$g(\tilde{x}, \tilde{y}) = -2\tilde{x} - 3\tilde{y} + \tilde{x}\tilde{y} + 5 \quad (5.424)$$

We apply this for $\tilde{x} = \frac{2}{3}$ and $\tilde{y} = \frac{2}{3}$ to obtain:

$$g\left(\frac{2}{3}, \frac{2}{3}\right) = -2 \times \frac{2}{3} - 3 \times \frac{2}{3} + \frac{2}{3} \times \frac{2}{3} + 5 = -\frac{4}{3} - \frac{6}{3} + \frac{4}{9} + 5 = 2\frac{1}{9} \quad (5.425)$$

So, the grey value at position (2, 2) should be $2\frac{1}{9}$. Rounding it to the nearest integer we have 2. If the nearest neighbour interpolation were used, we would have assigned to this position grey value 1, as the nearest neighbour is the pixel at the bottom right corner.

Example 5.68

It was established that the image captured was rotated in relation to the x axis of the image by an angle ϕ in the counter-clockwise direction. Correct the image so that it has 0 rotation with respect to the image axes.

Clearly, the correct coordinates of a pixel (x_c, y_c) will have to be rotated by angle ϕ in order to coincide with the distorted coordinates (x_d, y_d) (see figure 5.33):

$$\begin{aligned} x_d &= x_c \cos \phi + y_c \sin \phi \\ y_d &= -x_c \sin \phi + y_c \cos \phi \end{aligned} \quad (5.426)$$

We create first an empty grid, the same size as the original image. For every pixel (x_c, y_c) of this grid, we apply transformation (5.426) to identify the corresponding position (x_d, y_d) in the captured image. We then interpolate the values of pixels $(\lfloor x_d \rfloor, \lfloor y_d \rfloor)$, $(\lfloor x_d \rfloor, \lfloor y_d \rfloor + 1)$, $(\lfloor x_d \rfloor + 1, \lfloor y_d \rfloor)$ and $(\lfloor x_d \rfloor + 1, \lfloor y_d \rfloor + 1)$ using bilinear interpolation, to work out a grey value for position (x_d, y_d) , which we assign to pixel (x_c, y_c) of the corrected image.

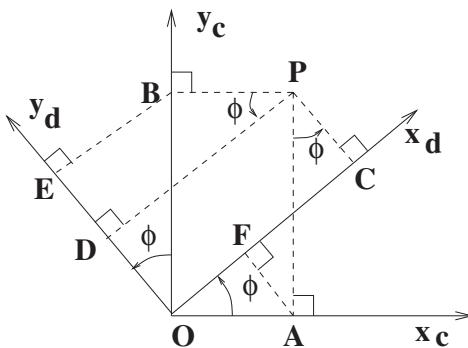


Figure 5.33: The coordinates of a point P in the correct coordinate system are (OA, OB) . In the rotated coordinate system, the coordinates of the point are (OC, OD) . From simple geometry: $OC = OA \cos \phi + AP \sin \phi$ and $OD = OB \cos \phi - BP \sin \phi$. Then equations (5.426) follow.

Box 5.12. The Hough transform for line detection

The equation of a straight line may be written as

$$p = x \cos \phi + y \sin \phi \quad (5.427)$$

where (ϕ, p) are some parameters that fully define the line, defined as shown in figure 5.34a.

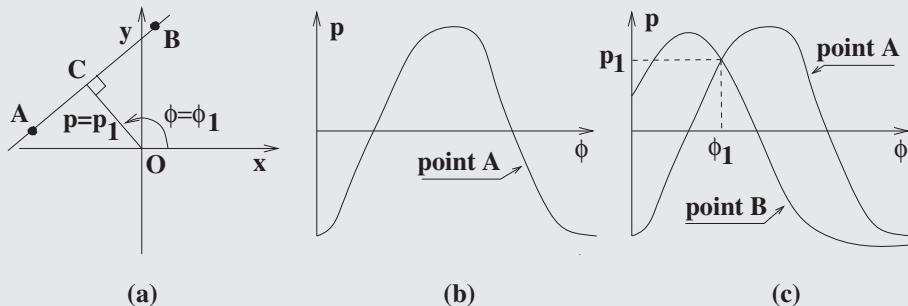


Figure 5.34: (a) A line may be fully defined if we know length $p = OC$ of the normal from the image centre O to the line, and the angle ϕ it forms with the reference direction Ox . (b) A point A in Oxy space corresponds to a curve in the (ϕ, p) space. (c) Two points A and B fully define a line and their corresponding curves in the (ϕ, p) space intersect exactly at the point that defines the parameters of the line.

Let us consider a pixel (x_i, y_i) which belongs to this line:

$$p = x_i \cos \phi + y_i \sin \phi \quad (5.428)$$

This equation represents a trigonometric curve when plotted in the (ϕ, p) space, as shown in figure 5.34b. If we consider another point (x_j, y_j) , which belongs to the same line, we shall have another trigonometric curve in space (ϕ, p) , crossing the first one. The point where the two curves cross specifies the parameters of the line defined by the two points, as shown in 5.34c. Using this principle, we may identify alignments of pixels in a binary image by using the following algorithm. This algorithm is for identifying alignments of black pixels in a binary image.

Step 0: Decide the number of bins you will use to quantise parameters (ϕ, p) . Let us say that the input image is $M \times N$ and that $m = \min\{M, N\}$. Let us say that we shall allow p to take values up to $p_{max} = \lfloor m/2 \rfloor$, in steps of 1, so that $K \equiv p_{max}$. Let us also say that we shall allow ϕ to take values from 0 to 359° in steps of 1° , so that $L \equiv 360$.

Step 1: Create an empty accumulator array A , of size $K \times L$, as shown in figure 5.35.

Step 2: Consider every black pixel in the image, with coordinates (x_i, y_i) and for every value of ϕ we allow, compute the corresponding value of p , using (5.428).

Step 3: For every value of p_l you compute, corresponding to angle $\phi = l$, identify the bin to which it belongs in the accumulator array A , as $k = \lfloor p_l \rfloor$.

Step 4: Increase the cell of the accumulator array by 1: $A(l+1, k+1) = A(l+1, k+1)+1$. Here we assumed that the elements of the accumulator array are identified by indices l and k that take values $1, 2, \dots, L$ and $1, 2, \dots, K$, respectively.

Step 5: Identify the peaks in the accumulator array. The coordinates of these peaks yield the parameters of the straight lines in your image. If you expected to find I lines, consider the I strongest peaks.

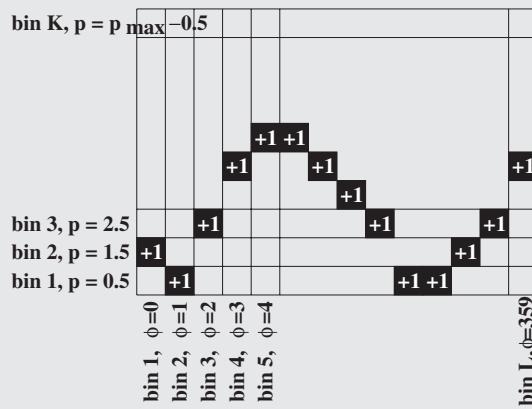


Figure 5.35: An accumulators array is a 2D histogram where we count how many curves enter each bin. Each bin has width 1 and the values given correspond to the centre of each bin. A bin with coordinates (l, k) has as central values $\phi = l - 1$ and $p = k - 0.5$ and contains all values in the range $[l - 0.5, l + 0.5)$ and $[k - 1, k)$. A single point in the binary image creates a curve that passes through many bins (the black bins shown). Every time a curve passes through a bin, the value of the bin is increased by 1. The more curves pass through a bin, the more points “vote” for the corresponding values of ϕ and p . All those points form a straight line in the image defined by the parameters of the centre of this bin.

Example B5.69

A line may be defined in terms of parameters (ϕ, p) , as shown in figure 5.34. Work out the coordinates of the intersection of two lines.

Let us assume that the equations of the two lines are:

$$\begin{aligned} p_1 &= x \cos \phi_1 + y \sin \phi_1 \\ p_2 &= x \cos \phi_2 + y \sin \phi_2 \end{aligned} \quad (5.429)$$

To solve this system of equations for x , we multiply the first one with $\sin \phi_2$ and the second with $\sin \phi_1$ and subtract them by parts:

$$\begin{aligned} p_1 \sin \phi_2 - p_2 \sin \phi_1 &= x(\cos \phi_1 \sin \phi_2 - \cos \phi_2 \sin \phi_1) \\ &= x \sin(\phi_2 - \phi_1) \Rightarrow \\ x &= \frac{p_1 \sin \phi_2 - p_2 \sin \phi_1}{\sin(\phi_2 - \phi_1)} \end{aligned} \quad (5.430)$$

To solve the system for y , we multiply the first equation with $\cos \phi_2$ and the second with $\cos \phi_1$ and subtract them by parts:

$$\begin{aligned} p_1 \cos \phi_2 - p_2 \cos \phi_1 &= y(\sin \phi_1 \cos \phi_2 - \sin \phi_2 \cos \phi_1) \\ &= y \sin(\phi_1 - \phi_2) \Rightarrow \\ y &= \frac{p_1 \cos \phi_2 - p_2 \cos \phi_1}{\sin(\phi_1 - \phi_2)} \end{aligned} \quad (5.431)$$

Note that if the two lines are parallel, $\phi_1 = \phi_2$ and formulae (5.430) and (5.431) will contain divisions by 0, which will yield infinite values, indicating that the two lines meet at infinity, which would be correct.

Example B5.70

Explain how you can work out the distortion parameters for the lens of the camera of your mobile, assuming the spatial transformation model given by (5.405), on page 514.

First I will construct a drawing consisting of L equidistant vertical black lines crossed by L equidistant horizontal lines, as shown in figure 5.29a. I will then photograph this drawing from a distance, say 5cm, with the centre of the drawing being roughly at the centre of the image. Let us say that this way I will create an image of size $M \times N$ pixels. This will be a colour image. I will convert it into a binary image by setting all pixels with colour values R , G and B to 255 if $R + G + B > 3 \times 127 = 381$, and all pixels with $R + G + B \leq 381$ to 0.

I will then apply the **Hough transform** (see Box 5.12) to identify all straight lines in the image. I expect that I shall have several lines with angle ϕ around 0° and 180° (corresponding to the vertical lines of the grid), and several lines with angle ϕ around 90° and 270° (corresponding to the horizontal lines of the grid). If I had managed to photograph the grid exactly aligned with the axes of the image plane and if there were no discretisation problems, obviously, the lines would have been observed with exactly these values.

To identify any rotation of the grid in relation to the image axes, I will subtract 180°

from all the angles that are near 180° and average the resultant values with the values that are near 0° . Let us say that this way I will find an angle ϕ_V . I will also subtract 180° from all the values that are around 270° and average the resultant angles with all those that are about 90° . Let us say that this way I will find an angle ϕ_H . Then I will average ϕ_V and $\phi_H - 90^\circ$ in order to identify the angle by which the grid I photographed is misaligned with the image axes. Let us say that I identified an angle ϕ_R . I must first assess whether this angle is worth worrying about or not. To do that, I will multiply $\tan \phi_R$ with $\min\{M/2, N/2\}$ to see how much shift this creates near the periphery of the image. If this shift is more than 0.5 (ie more than half a pixel), then I must correct for rotation.

How we correct for rotation is shown in example 5.68. I will have to apply the rotation to the original colour image I captured, then binarise the rotation-corrected image and perform again the Hough transform. This time the detected lines must have angles ϕ very close to 0° , 180° , 90° and 270° .

Let us consider the four lines identified by the Hough transform algorithm with the minimum values of parameter p , ie the four lines nearest to the image centre. Let us say that the image centre has coordinates (i_0, j_0) . I would like to identify the corners of the smallest rectangle that surrounds the centre. I can do that by combining the four lines two by two and computing their intersection point using formulae (5.430) and (5.431) of example 5.69. The coordinates of the intersection points are rounded to the nearest integer. Let us say that in this way we identify the four corners of the smallest grid cell around the centre of the image, to be: (i_{TL}, j_{TL}) , (i_{BL}, j_{BL}) , (i_{TR}, j_{TR}) and (i_{BR}, j_{BR}) . We know that the lens distortion is negligible near the image centre, so we may infer that the positions of these four points are the correct positions. Then I may find the length of the sides of the rectangle in terms of pixels, by using:

$$\begin{aligned} d_{LV} &= \sqrt{(i_{TL} - i_{BL})^2 + (j_{TL} - j_{BL})^2} \\ d_{RV} &= \sqrt{(i_{TR} - i_{BR})^2 + (j_{TR} - j_{BR})^2} \\ d_{TH} &= \sqrt{(i_{TL} - i_{TR})^2 + (j_{TL} - j_{TR})^2} \\ d_{BH} &= \sqrt{(i_{BL} - i_{BR})^2 + (j_{BL} - j_{BR})^2} \end{aligned} \quad (5.432)$$

We average the two vertical and the two horizontal lengths to have a better estimate:

$$d_H = \frac{d_{TH} + d_{BH}}{2} \quad d_V = \frac{d_{LV} + d_{RV}}{2} \quad (5.433)$$

Next, let us identify the corner of this rectangle that is closest to the image centre (i_0, j_0) , and let us shift that corner to coincide with the image centre. Let us call this new image centre $(\tilde{i}_0, \tilde{j}_0)$. These steps are shown schematically in figure 5.36.

Now we know that one of the corners of the grid I drew and photographed coincides with the image centre. We also know that the outer corners of the grid should be at

coordinates which are known integer multiples of d_H and d_V . These coordinates are expressed in terms of pixels and they are measured from the image centre. Let us call these locations of the four outer corners of the photographed grid their correct locations (see figure 5.37):

$$\begin{aligned}x_{TLc} &= \tilde{i}_0 - d_H L_1 \\y_{TLc} &= \tilde{j}_0 - d_V L_3 \\x_{BLc} &= \tilde{i}_0 - d_H L_1 \\y_{BLc} &= \tilde{j}_0 + d_V L_4 \\x_{TRc} &= \tilde{i}_0 + d_H L_2 \\y_{TRc} &= \tilde{j}_0 - d_V L_3 \\x_{BRc} &= \tilde{i}_0 + d_H L_2 \\y_{BRc} &= \tilde{j}_0 + d_V L_4\end{aligned}\tag{5.434}$$

By considering now the four lines with the maximum value of p identified by the Hough transform, we can work out the observed coordinates of the four outer corners of the grid. Let us call them:

$$(x_{TLD}, y_{TLD}) \quad (x_{BLD}, y_{BLD}) \quad (x_{TRD}, y_{TRD}) \quad (x_{BRD}, y_{BRD})\tag{5.435}$$

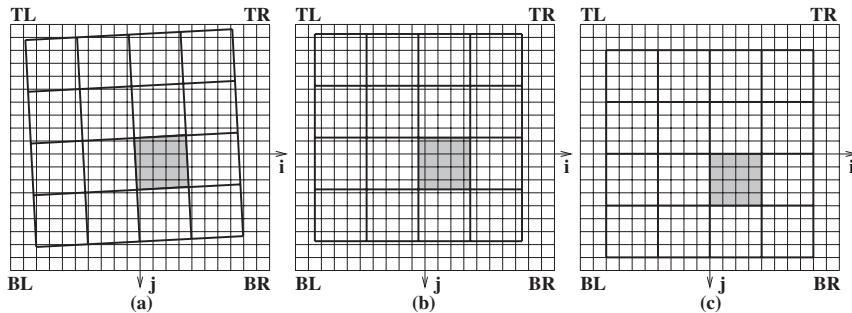


Figure 5.36: The thick lines represent the imaged grid. The intersections of the thin lines represent the pixel centres. The four corners of any rectangle are identified as top left (TL), top right (TR), bottom left (BL) and bottom right (BR). (a) The imaged grid may not be perfectly aligned with the image axes. (b) After we correct the image by rotating it about the image centre. (c) After we shift the corner of the inner-most rectangle (highlighted with grey) that is nearest to the image centre to coincide with the image centre. In this case it is the TL corner of the rectangle.

The assumed transformation model is given by (5.404) and (5.405) as

$$x_d = x_c(1 + k_1 r_c)$$

$$y_d = y_c(1 + k_2 r_c) \quad (5.436)$$

where I decided to use different parameter value for the vertical and horizontal distortion. I can solve these equations for k_1 and k_2 :

$$\begin{aligned} k_1 &= \frac{x_d - x_c}{x_c \sqrt{x_c^2 + y_c^2}} \\ k_2 &= \frac{y_d - y_c}{y_c \sqrt{x_c^2 + y_c^2}} \end{aligned} \quad (5.437)$$

For each one of these equations I shall have four pairs of correct and observed (distorted) values given by (5.434) and (5.435), respectively. If I substitute them in (5.437) I shall obtain four values for k_1 and four values for k_2 . I will average the four values to work out a single value for k_1 and a single value for k_2 .

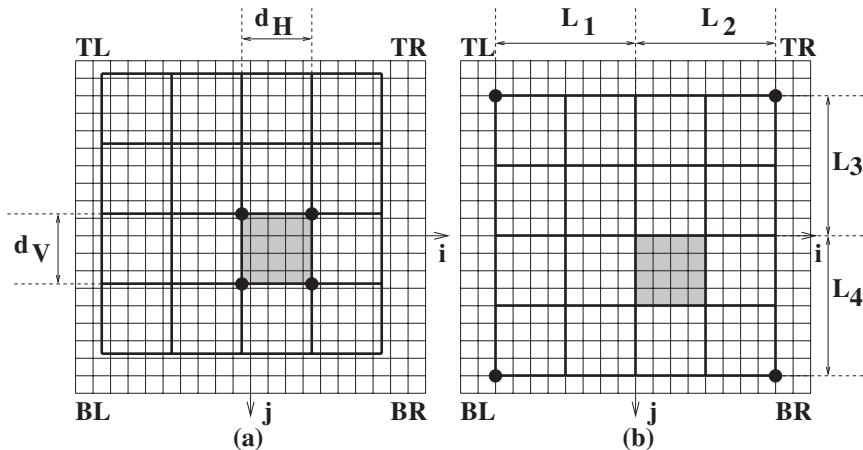


Figure 5.37: (a) The four black dots represent the inner-most cell of the grid, around the image centre, that has suffered the least lens distortion. From its sides we can compute the true size of each cell in terms of pixels (the values of d_H and d_V). (b) After we have shifted the nearest corner of the inner-most cell to coincide with the image centre, we can compute the true locations of the four outer corners of the grid (marked by black dots), as integer multiples of d_H and d_V . The number of integer multiples we need in each case is specified by parameters L_1 , L_2 , L_3 and L_4 .

What is the “take home” message of this chapter?

This chapter explored some techniques used to correct (ie restore) the damaged values of an image. The problem of restoration requires some prior knowledge concerning the original uncorrupted signal or the imaged scene, and in that way differs from the image enhancement problem. Geometric restoration of an image requires knowledge of the correct location of some reference points.

Grey level restoration of an image requires knowledge of some statistical properties of the corrupting noise and the perfect image itself, as well as a model for the blurring process. Often, we bypass the requirement for knowing the statistical properties of the original image by imposing some spatial smoothness constraints on the solution, based on the heuristic that “the world is largely smooth”. Having chosen the correct model for the degradation process and the uncorrupted image, we have then to solve the problem of recovering the original image values.

The full problem of image restoration is a very difficult one as it is nonlinear. It can be solved with the help of global optimisation approaches. However, simpler solutions can be found, in the form of convolution filters, if we make the assumption that the degradation process is shift invariant.

Chapter 6

Image Segmentation and Edge Detection

What is this chapter about?

This chapter is about the image processing techniques that are used to prepare an image as an input to an automatic vision system. These techniques perform **image segmentation** and **edge detection**, and their purpose is to extract information from an image in such a way that the output image contains much less information than the original one, but the little information it contains is much more relevant to the other modules of an automatic vision system than the discarded information.

What exactly is the purpose of image segmentation and edge detection?

The purpose of image segmentation and edge detection is to extract the outlines of different regions in the image, that is to divide the image into regions which are made up of pixels which have something in common. For example, they may have similar brightness, or colour, which may indicate that they belong to the same object or facet of an object. An example is shown in figure 6.1.

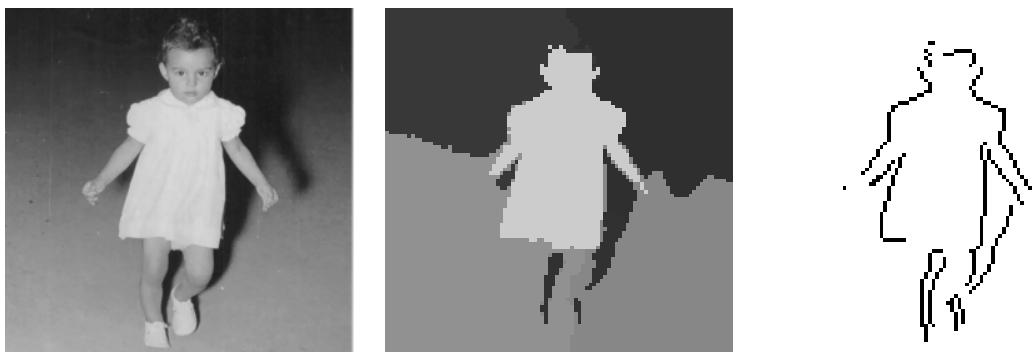


Figure 6.1: An original image, its segmentation and its edge map.

6.1 Image segmentation

How can we divide an image into uniform regions?

One of the simplest methods is that of histogramming and **thresholding**. If we plot the number of pixels which have a specific grey value versus that value, we create the histogram of the image. Properly normalised, the histogram is essentially the probability density function of the grey values of the image (see page 236).

Assume that we have an image consisting of a bright object on a dark background and assume that we want to extract the object. For such an image, the histogram will have two peaks and a valley between them. We can choose as the threshold then the grey value which corresponds to the valley of the histogram, indicated by t_0 in figure 6.2a, and **label** all pixels with grey values greater than t_0 as object pixels and all pixels with grey values smaller than t_0 as background pixels.

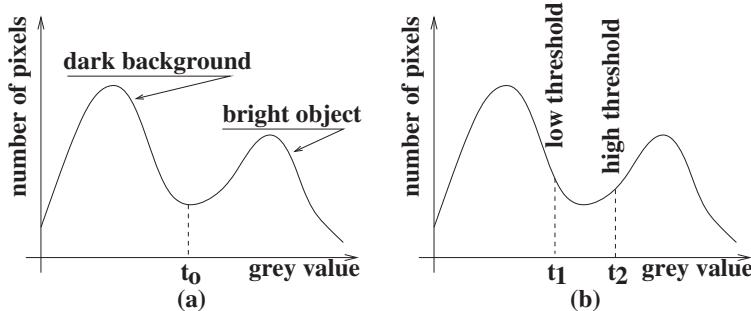


Figure 6.2: The histogram of an image with a bright object on a dark background.

What do we mean by “labelling” an image?

When we say we “extract” an object in an image, we mean that we identify the pixels that make the object up. To express this information, we create an array of the same size as the original image and we give to each pixel a **label**. All pixels that make up the object are given the same label and all pixels that make up the background are given a different label. The label is usually a number, but it could be anything: a letter or a colour. It is essentially a name and it has symbolic meaning only. Labels, therefore, cannot be treated as numbers. Label images cannot be processed in the same way as grey level images. Often label images are also referred to as **classified images**, as they indicate the **class** to which each pixel belongs.

What can we do if the valley in the histogram is not very sharply defined?

If there is no clear valley in the histogram of an image, it means that there are several pixels in the background which have the same grey value as pixels in the object and vice versa.

Such pixels are particularly encountered near the boundaries of the objects which may be fuzzy and not sharply defined. One may use then what is called **hysteresis thresholding**: instead of one, two threshold values are chosen on either side of the valley (see figure 6.2b).

The highest of the two thresholds is used to define the “hard core” of the object. The lowest is used in conjunction with spatial proximity of the pixels: a pixel with intensity value greater than the smaller threshold but less than the larger threshold is labelled as object pixel only if it is adjacent to a pixel which is a core object pixel. Sometimes we use different rules: we may label such a pixel as an object pixel only if the majority of its neighbouring pixels have already been labelled as object pixels.

Figure 6.3 shows an image depicting a dark object on a bright background and its histogram. In 6.3c the image is segmented with a single threshold, marked with a t in the histogram, while in 6.3d it has been segmented using two thresholds marked t_1 and t_2 in the histogram.

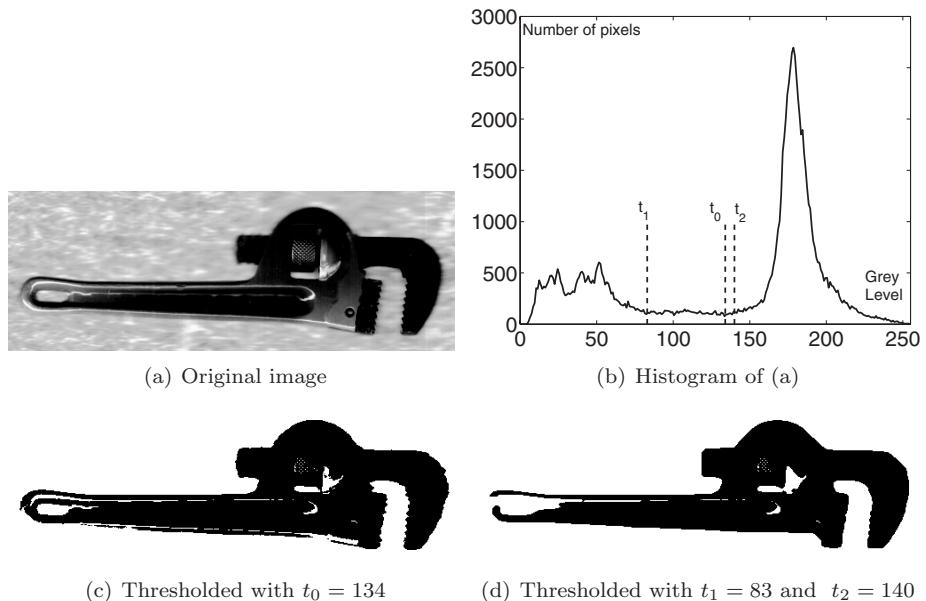


Figure 6.3: Simple thresholding versus hysteresis thresholding.

Alternatively, we may try to choose the global threshold value in an optimal way, ie by trying to minimise the number of misclassified pixels.

How can we minimise the number of misclassified pixels?

We can minimise the number of misclassified pixels if we have some prior knowledge about the distributions of the grey values that make up the object and the background. For example, if we know that the object occupies a certain fraction θ of the area of the picture, then this θ is the prior probability for a pixel to be an object pixel. Clearly, the background pixels will occupy $1 - \theta$ of the area and a pixel will have $1 - \theta$ prior probability to be a background pixel. We may choose the threshold then, so that, the pixels we classify as object pixels are

a fraction θ of the total number of pixels. This method is called **p-tile** method. Further, if we also happen to know the probability density functions of the grey values of the object pixels and the background pixels, then we may choose the threshold that exactly minimises the error.

How can we choose the minimum error threshold?

Let us assume that the pixels which make up the object are distributed according to the probability density function $p_o(x)$ and the pixels which make up the background are distributed according to function $p_b(x)$.

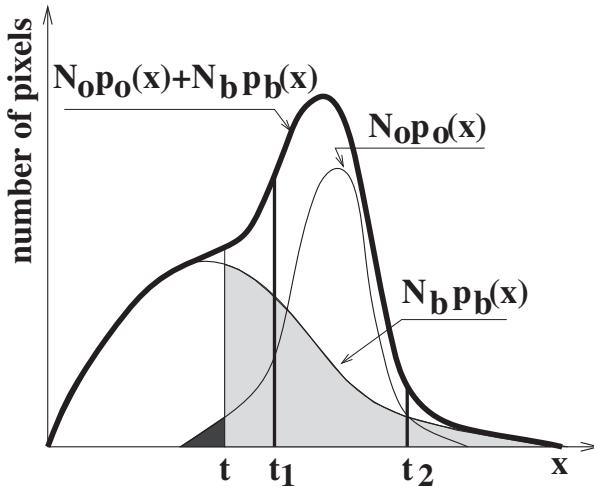


Figure 6.4: The probability density functions of the grey values of the pixels that make up the object ($p_o(x)$) and the background ($p_b(x)$). Their weighted sum, ie $p_o(x)$ and $p_b(x)$ multiplied with the total number of pixels that make up the object, N_o , and the background, N_b , respectively, and added, is the histogram of the image (depicted here by the thick black line). If we use threshold t , the pixels that make up the cross-over tails of the two probability density functions (shaded grey in this figure) will be misclassified. The minimum error threshold method tries to minimise the total number of pixels misclassified and it may yield two thresholds: one on either side of the narrower probability density function, implying that the pixels that are on the far right have more chance to come from the long tail of the “fat” probability density function, than from the short tail of the narrow probability density function.

Assume that we choose a threshold value t (see figure 6.4). Then the error committed by misclassifying object pixels as background pixels will be given by

$$\int_{-\infty}^t p_o(x) dx \quad (6.1)$$

and the error committed by misclassifying background pixels as object pixels is:

$$\int_t^{+\infty} p_b(x) dx \quad (6.2)$$

In other words, the error that we commit arises from misclassifying the two tails of the two probability density functions on either side of threshold t . Let us also assume that the fraction of the pixels that make up the object is θ , and, by inference, the fraction of the pixels that make up the background is $1 - \theta$. Then, the total error is:

$$E(t) = \theta \int_{-\infty}^t p_o(x)dx + (1 - \theta) \int_t^{+\infty} p_b(x)dx \quad (6.3)$$

We would like to choose t so that $E(t)$ is minimum. We take the first derivative of $E(t)$ with respect to t (see Box 4.9, on page 348) and set it to zero:

$$\begin{aligned} \frac{\partial E}{\partial t} &= \theta p_o(t) - (1 - \theta)p_b(t) = 0 \Rightarrow \\ \theta p_o(t) &= (1 - \theta)p_b(t) \end{aligned} \quad (6.4)$$

The solution of this equation gives the minimum error threshold, for any type of probability density functions that are used to model the two pixel populations.

Example B6.1

Derive equation (6.4) from (6.3).

We apply Leibniz rule given by equation (4.114), on page 348, to perform the differentiation of $E(t)$ given by equation (6.3). We have the following correspondences:

Parameter λ corresponds to t .

For the first integral:

$$\begin{aligned} a(\lambda) &\rightarrow -\infty && (a \text{ constant, with zero derivative}) \\ b(\lambda) &\rightarrow t \\ f(x; \lambda) &\rightarrow p_o(x) && (\text{independent from the parameter with respect} \\ &&& \text{to which we differentiate}) \end{aligned}$$

For the second integral:

$$\begin{aligned} a(\lambda) &\rightarrow t \\ b(\lambda) &\rightarrow +\infty && (a \text{ constant, with zero derivative}) \\ f(x; \lambda) &\rightarrow p_b(x) && (\text{independent from } t) \end{aligned}$$

Equation (6.4) then follows.

Example 6.2

The grey values of the object and the background pixels are distributed according to probability density function

$$p(x) = \begin{cases} \frac{3}{4a^3} [a^2 - (x-b)^2] & \text{for } b-a \leq x \leq b+a \\ 0 & \text{otherwise} \end{cases} \quad (6.5)$$

with $a = 1$ and $b = 5$ for the background, and $a = 2$ and $b = 7$ for the object. Sketch the two probability density functions and determine the range of possible thresholds.

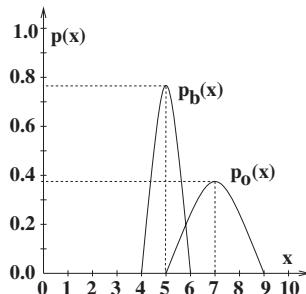


Figure 6.5: The range of possible thresholds is from 5 to 6.

Example 6.3

If the object pixels are eight-ninths ($8/9$) of the total number of pixels, determine the threshold that minimises the fraction of misclassified pixels for the problem of example 6.2.

We substitute into equation (6.4) the following:

$$\begin{aligned} \theta &= \frac{8}{9} \Rightarrow 1 - \theta = \frac{1}{9} \\ p_b(t) &= \frac{3}{4}(-t^2 - 24 + 10t) \quad p_o(t) = \frac{3}{32}(-t^2 - 45 + 14t) \end{aligned} \quad (6.6)$$

Then:

$$\begin{aligned} \frac{1}{9} \times \frac{3}{4}(-t^2 - 24 + 10t) &= \frac{8}{9} \times \frac{3}{32}(-t^2 - 45 + 14t) \\ \Rightarrow -24 + 10t &= -45 + 14t \Rightarrow 4t = 21 \Rightarrow t = \frac{21}{4} = 5.25 \end{aligned} \quad (6.7)$$

Example 6.4

The grey values of the object and the background pixels are distributed according to the probability density function

$$p(x) = \begin{cases} \frac{\pi}{4a} \cos \frac{(x-x_0)\pi}{2a} & \text{for } x_0 - a \leq x \leq x_0 + a \\ 0 & \text{otherwise} \end{cases} \quad (6.8)$$

with $x_0 = 1$ and $a = 1$ for the objects, and $x_0 = 3$ and $a = 2$ for the background. Sketch the two probability density functions. If one-third of the total number of pixels are object pixels, determine the fraction of misclassified object pixels by optimal thresholding.

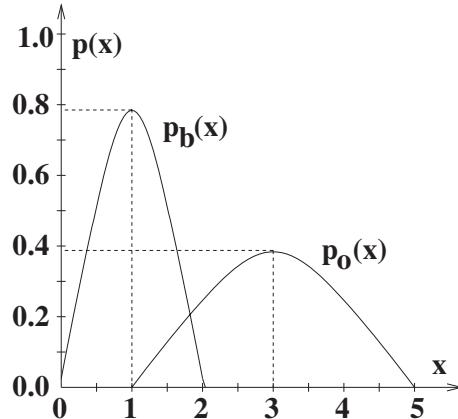


Figure 6.6: The range of possible thresholds is from 1 to 2.

Apply formula (6.4) with:

$$\theta = \frac{1}{3} \Rightarrow 1 - \theta = \frac{2}{3}$$

$$p_0(x) = \frac{\pi}{4} \cos \frac{(x-1)\pi}{2} \quad p_b(x) = \frac{\pi}{8} \cos \frac{(x-3)\pi}{4} \quad (6.9)$$

Equation (6.4) becomes:

$$\frac{1}{3} \times \frac{\pi}{4} \cos \frac{(t-1)\pi}{2} = \frac{2}{3} \times \frac{\pi}{8} \cos \frac{(t-3)\pi}{4}$$

$$\Rightarrow \cos \frac{(t-1)\pi}{2} = \cos \frac{(t-3)\pi}{4}$$

$$\Rightarrow \frac{(t-1)\pi}{2} = \pm \frac{(t-3)\pi}{4} \quad (6.10)$$

Consider first $\frac{t-1}{2}\pi = \frac{t-3}{4}\pi \Rightarrow 2t - 2 = t - 3 \Rightarrow t = -1$.

This value is outside the acceptable range, so it is a meaningless solution.

Then:

$$\frac{(t-1)\pi}{2} = -\frac{(t-3)\pi}{4} \Rightarrow 2t - 2 = -t + 3 \Rightarrow 3t = 5 \Rightarrow t = \frac{5}{3} \quad (6.11)$$

This is the threshold for minimum error. The fraction of misclassified object pixels will be given by all those object pixels that have grey value greater than $\frac{5}{3}$. We define a new variable of integration $y \equiv x - 1$, to obtain:

$$\begin{aligned} \int_{\frac{5}{3}}^2 \frac{\pi}{4} \cos \frac{(x-1)\pi}{2} dx &= \frac{\pi}{4} \int_{\frac{2}{3}}^1 \cos \frac{y\pi}{2} dy \\ &= \frac{\pi}{4} \left[\frac{\sin \frac{y\pi}{2}}{\frac{\pi}{2}} \right]_{\frac{2}{3}}^1 \\ &= \frac{1}{2} \left(\sin \frac{\pi}{2} - \sin \frac{\pi}{3} \right) \\ &= \frac{1}{2}(1 - \sin 60^\circ) \\ &= \frac{1}{2} \left(1 - \frac{\sqrt{3}}{2} \right) \\ &= \frac{2 - 1.7}{4} \\ &= \frac{0.3}{4} \\ &= 0.075 = 7.5\% \end{aligned} \quad (6.12)$$

What is the minimum error threshold when object and background pixels are normally distributed?

Let us assume that the pixels that make up the object are normally distributed with mean μ_o and standard deviation σ_o and the pixels that make up the background are normally distributed with mean μ_b and standard deviation σ_b :

$$\begin{aligned} p_o(x) &= \frac{1}{\sqrt{2\pi}\sigma_o} \exp \left[-\frac{(x - \mu_o)^2}{2\sigma_o^2} \right] \\ p_b(x) &= \frac{1}{\sqrt{2\pi}\sigma_b} \exp \left[-\frac{(x - \mu_b)^2}{2\sigma_b^2} \right] \end{aligned} \quad (6.13)$$

Upon substitution into equation (6.4), we obtain:

$$\begin{aligned}
\theta \frac{1}{\sqrt{2\pi}\sigma_o} \exp \left[-\frac{(t - \mu_o)^2}{2\sigma_o^2} \right] &= (1 - \theta) \frac{1}{\sqrt{2\pi}\sigma_b} \exp \left[-\frac{(t - \mu_b)^2}{2\sigma_b^2} \right] \Rightarrow \\
\exp \left[-\frac{(t - \mu_o)^2}{2\sigma_o^2} + \frac{(t - \mu_b)^2}{2\sigma_b^2} \right] &= \frac{1 - \theta}{\theta} \frac{\sigma_o}{\sigma_b} \Rightarrow \\
-\frac{(t - \mu_o)^2}{2\sigma_o^2} + \frac{(t - \mu_b)^2}{2\sigma_b^2} &= \ln \left[\frac{\sigma_o}{\sigma_b} \frac{1 - \theta}{\theta} \right] \Rightarrow \\
(t^2 + \mu_b^2 - 2t\mu_b)\sigma_o^2 - (t^2 + \mu_o^2 - 2t\mu_o)\sigma_b^2 &= 2\sigma_o^2\sigma_b^2 \ln \left[\frac{\sigma_o}{\sigma_b} \frac{1 - \theta}{\theta} \right] \Rightarrow \\
(\sigma_o^2 - \sigma_b^2)t^2 + 2(-\mu_b\sigma_o^2 + \mu_o\sigma_b^2)t + \mu_b^2\sigma_o^2 - \mu_o^2\sigma_b^2 - 2\sigma_o^2\sigma_b^2 \ln \left[\frac{\sigma_o}{\sigma_b} \frac{1 - \theta}{\theta} \right] &= 0 \quad (6.14)
\end{aligned}$$

This is a quadratic equation in t . It has two solutions in general, except when the two populations have the same standard deviation. If $\sigma_o = \sigma_b$, the above expression takes the form:

$$\begin{aligned}
2(\mu_o - \mu_b)\sigma_o^2 t + (\mu_b^2 - \mu_o^2)\sigma_o^2 - 2\sigma_o^4 \ln \left(\frac{1 - \theta}{\theta} \right) &= 0 \Rightarrow \\
t = \frac{\sigma_o^2}{\mu_o - \mu_b} \ln \left(\frac{1 - \theta}{\theta} \right) - \frac{\mu_o + \mu_b}{2} &\quad (6.15)
\end{aligned}$$

This is the minimum error threshold.

What is the meaning of the two solutions of the minimum error threshold equation?

When $\sigma_o \neq \sigma_b$, the quadratic term in (6.14) does not vanish and we have two thresholds, t_1 and t_2 . These turn out to be one on either side of the sharpest probability density function. Let us assume that the sharpest probability density function is that of the object pixels (see figure 6.4). Then the correct thresholding will be to label as object pixels only those pixels with grey value x such that $t_1 < x < t_2$.

The meaning of the second threshold is that the flatter probability density function has such a long tail, that the pixels with grey values $x \geq t_2$ are more likely to belong to the long tail of the flat probability density function, than to the sharper probability density function.

Example 6.5

The grey values of the object pixels are distributed according to probability density function

$$p_o = \frac{1}{2\sigma_o} \exp \left(-\frac{|x - \mu_o|}{\sigma_o} \right) \quad (6.16)$$

while the grey values of the background pixels are distributed according to probability density function:

$$p_b = \frac{1}{2\sigma_b} \exp\left(-\frac{|x - \mu_b|}{\sigma_b}\right) \quad (6.17)$$

If $\mu_o = 60$, $\mu_b = 40$, $\sigma_o = 10$ and $\sigma_b = 5$, find the thresholds that minimise the fraction of misclassified pixels, when we know that the object occupies two-thirds of the area of the image.

We substitute in equation (6.4):

$$\begin{aligned} \frac{\theta}{2\sigma_o} e^{-\frac{|t-\mu_o|}{\sigma_o}} &= \frac{1-\theta}{2\sigma_b} e^{-\frac{|t-\mu_b|}{\sigma_b}} \\ \Rightarrow \exp\left(-\frac{|t-\mu_o|}{\sigma_o} + \frac{|t-\mu_b|}{\sigma_b}\right) &= \frac{\sigma_o(1-\theta)}{\sigma_b\theta} \end{aligned} \quad (6.18)$$

We have $\theta = \frac{2}{3}$ (therefore, $1-\theta = \frac{1}{3}$) and $\sigma_o = 10$, $\sigma_b = 5$. Then:

$$-\frac{|t-\mu_o|}{\sigma_o} + \frac{|t-\mu_b|}{\sigma_b} = \ln \frac{10 \times \frac{1}{3}}{5 \times \frac{2}{3}} = \ln 1 = 0 \quad (6.19)$$

We have the following cases:

$$\begin{aligned} t < \mu_b < \mu_o &\Rightarrow |t - \mu_o| = -t + \mu_o \quad \text{and} \quad |t - \mu_b| = -t + \mu_b \\ &\Rightarrow \frac{t - \mu_o}{\sigma_o} + \frac{-t + \mu_b}{\sigma_b} = 0 \Rightarrow (\sigma_b - \sigma_o)t = \mu_o\sigma_b - \sigma_o\mu_b \\ &\Rightarrow t = \frac{\mu_o\sigma_b - \sigma_o\mu_b}{\sigma_b - \sigma_o} = \frac{60 \times 5 - 10 \times 40}{-5} \Rightarrow t_1 = 20 \\ \mu_b < t < \mu_o &\Rightarrow |t - \mu_o| = -t + \mu_o \quad \text{and} \quad |t - \mu_b| = t - \mu_b \\ &\Rightarrow \frac{t - \mu_o}{\sigma_o} + \frac{t - \mu_b}{\sigma_b} = 0 \Rightarrow (\sigma_o + \sigma_b)t = \mu_o\sigma_b + \sigma_o\mu_b \\ &\Rightarrow t = \frac{\mu_o\sigma_b + \sigma_o\mu_b}{\sigma_b + \sigma_o} = \frac{60 \times 5 + 10 \times 40}{15} \Rightarrow t_2 = 47 \\ \mu_b < \mu_o &< t \Rightarrow |t - \mu_o| = t - \mu_o \quad \text{and} \quad |t - \mu_b| = t - \mu_b \\ &\Rightarrow -\frac{t - \mu_o}{\sigma_o} + \frac{t - \mu_b}{\sigma_b} = 0 \Rightarrow (\sigma_o - \sigma_b)t = \sigma_o\mu_b - \mu_o\sigma_b \\ &\Rightarrow t = \frac{-\mu_o\sigma_b + \sigma_o\mu_b}{\sigma_o - \sigma_b} = \frac{-60 \times 5 + 10 \times 40}{5} = 20 < \mu_o \end{aligned} \quad (6.20)$$

The last solution is rejected because t was assumed greater than μ_o . So, there are two thresholds, $t_1 = 20$ and $t_2 = 47$.

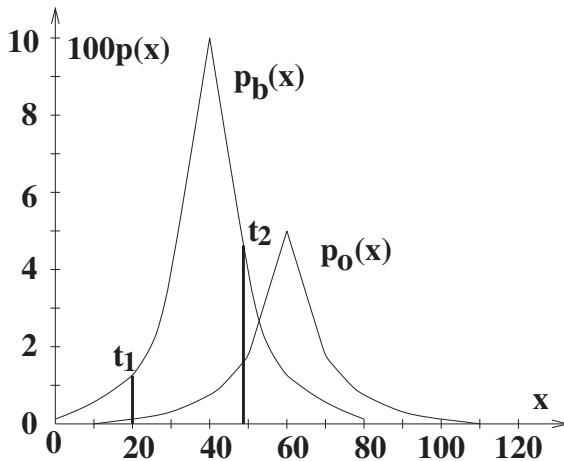


Figure 6.7: Only pixels with grey values between t_1 and t_2 should be classified as background pixels in order to minimise the error. Notice how these two thresholds do not appear very intuitive. This is because the object pixels are twice as many as the background pixels, and this shifts the thresholds away from the intuitive positions, in order to minimise the total number of misclassified pixels.

How can we estimate the parameters of the Gaussian probability density functions that represent the object and the background?

In general, a multimodal histogram may be modelled by a **Gaussian mixture model (GMM)**. The parameters of such a model may be estimated by using the **expectation maximisation (EM) algorithm**. In the case of a single object in a uniform background, the mixture model consists of two Gaussians. The following EM algorithm is for two classes.

Step 0: Decide the number of Gaussians you want to fit to your data. Say $K = 2$. Guess initial values for their parameters, ie μ_1 , μ_2 , σ_1 and σ_2 . Select a value for the threshold of convergence X , say $X = 0.1$.

Step 1: For every pixel, (i, j) , with grey value g_{ij} , compute the probability with which it belongs to the two classes:

$$\tilde{p}_{ij1} \equiv \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(g_{ij}-\mu_1)^2}{2\sigma_1^2}} \quad \tilde{p}_{ij2} \equiv \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{(g_{ij}-\mu_2)^2}{2\sigma_2^2}} \quad (6.21)$$

Step 2: Divide \tilde{p}_{ij1} and \tilde{p}_{ij2} with $\tilde{p}_{ij1} + \tilde{p}_{ij2}$, to produce the normalised probabilities p_{ij1} and p_{ij2} , respectively.

Step 3: For every class, compute updated values of its parameters, taking into consideration the probability with which each pixel belongs to that class:

$$\begin{aligned}\mu'_1 &= \frac{\sum_{(i,j)} p_{ij1} g_{ij}}{\sum_{(i,j)} p_{ij1}} & \sigma'_1 &= \sqrt{\frac{\sum_{(i,j)} p_{ij1} (g_{ij} - \mu'_1)^2}{\sum_{(i,j)} p_{ij1}}} \\ \mu'_2 &= \frac{\sum_{(i,j)} p_{ij2} g_{ij}}{\sum_{(i,j)} p_{ij2}} & \sigma'_2 &= \sqrt{\frac{\sum_{(i,j)} p_{ij2} (g_{ij} - \mu'_2)^2}{\sum_{(i,j)} p_{ij2}}}\end{aligned}\quad (6.22)$$

Step 4: Check whether the new parameter values are very different from the previous values:
If $|\mu_1 - \mu'_1| < X\mu_1$ and $|\mu_2 - \mu'_2| < X\mu_2$ and $|\sigma_1 - \sigma'_1| < X\sigma_1$ and $|\sigma_2 - \sigma'_2| < X\sigma_2$, exit
Else, set $\mu_1 = \mu'_1$, $\mu_2 = \mu'_2$, $\sigma_1 = \sigma'_1$ and $\sigma_2 = \sigma'_2$ and go to Step 1.

Step 5: Compute the fraction of pixels that belong to the first Gaussian:

$$\theta = \frac{1}{N} \sum_{(i,j)} p_{ij1} \quad (6.23)$$

where N is the total number of pixels in the image.

Example 6.6

Assuming Gaussian probability density functions for the object and the background in image 6.3a, estimate their parameters with the EM algorithm, derive the optimal threshold and use it to threshold the image.

The EM algorithm was initialised with $\mu_o = 64$, $\mu_b = 192$ and $\sigma_o = \sigma_b = 3$. It took two iterations to converge with $X = 0.1$ to: $\mu_o = 53.3$, $\mu_b = 182.0$, $\sigma_o = 31.7$, $\sigma_b = 16.5$, $A_o = 588$, $A_b = 2207$ and $\theta = 0.332$. The optimal threshold was estimated by solving:

$$731.88t^2 - 336800t + 3.1783 \times 10^7 = 0 \quad (6.24)$$

This equation had two roots: $t_1 = 132.54$ and $t_2 = 327.65$. Using t_1 as the optimal threshold, we obtained the result shown in figure 6.8.

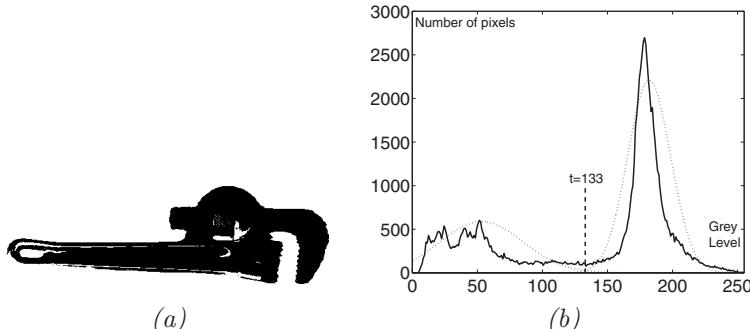


Figure 6.8: (a) The image thresholded with threshold 133. (b) The histogram of the image with the two Gaussians, estimated by the EM algorithm, overlaid and the chosen threshold marked.

Example 6.7

Use simulated annealing to fit the two peaks of the histogram of figure 6.3a with a mixture of two Gaussians.

Let us assume that the pairs of values of the histogram are (x_i, y_i) where x_i is the value of the centre of bin i and y_i is the bin value. It is assumed that $\sum_{i=1}^N y_i \Delta y = 1$, that is it is assumed that the histogram has been normalised by dividing all its entries with the total number of pixels and the bin width Δy (see page 236). Here N is the number of bins of the histogram. Since the histogram has two main peaks, we shall try to fit it with the mixture of two Gaussians. We wish to select the parameters of the two Gaussians and their mixing proportions so that the square of the difference between the value obtained by the mixture of the Gaussians and the true histogram value is minimised. That is, we define the cost function of the problem as follows:

$$E \equiv \sum_{i=1}^N \left[y_i - \frac{\frac{\theta}{\sqrt{2\pi}\sigma_1} e^{-\frac{(x_i-\mu_1)^2}{2\sigma_1^2}} + \frac{1-\theta}{\sqrt{2\pi}\sigma_2} e^{-\frac{(x_i-\mu_2)^2}{2\sigma_2^2}}}{\sum_{m=1}^N \left[\frac{\theta}{\sqrt{2\pi}\sigma_1} e^{-\frac{(x_m-\mu_1)^2}{2\sigma_1^2}} + \frac{1-\theta}{\sqrt{2\pi}\sigma_2} e^{-\frac{(x_m-\mu_2)^2}{2\sigma_2^2}} \right]} \right]^2 \quad (6.25)$$

Here θ is the mixing proportion with which the first Gaussian contributes to the mixture and $1-\theta$ is the mixing proportion with which the second Gaussian contributes to the mixture. Obviously, $0 < \theta < 1$. If we assume that the histogram has been constructed with one bin per grey value, $\Delta y = 1$ and $N = 256$. Also, parameters μ_1 , μ_2 , σ_1 and σ_2 must take values in the range $[0, 255]$. The simulated annealing algorithm then, which we may use to identify the values of all five parameters of the problem, is as follows.

Step 1: Select plausible values for parameters μ_1 , μ_2 , σ_1 , σ_2 and θ , by inspecting the histogram you wish to fit. For the particular problem, we selected: $\mu_1 = 239$, $\mu_2 = 22$, $\sigma_1 = 5$, $\sigma_2 = 5$ and $\theta = 0.5$.

Step 2: Compute E from (6.25), and call it E^{old} . Set $E^{min} = E^{old}$. Set μ_1^{min} , μ_2^{min} , σ_1^{min} , σ_2^{min} and θ^{min} equal to the current values of these parameters.

Step 3: Select a starting temperature $T_0 = 10$. Set $k = 0$. Set $\alpha = 0.99$.

Step 4: Increase k by 1 and set $T_k = \alpha T_{k-1}$.

Step 5: Consider one of the parameters. Select randomly a new value for it, uniformly distributed in the range of its plausible values.

Step 6: For the new set of parameter values, compute E from (6.25), and call it E^{new} .

Step 7: If $E^{new} \leq E^{old}$, accept the new parameter value. Set $E^{old} = E^{new}$. If $E^{new} < E^{min}$, set $E^{min} = E^{new}$ and μ_1^{min} , μ_2^{min} , σ_1^{min} , σ_2^{min} and θ^{min} equal to the parameter values with which E^{new} was computed.

If all parameters have been considered in turn, go to Step 4. If not all parameters have been considered in this iteration step, go to Step 5.

Step 8: If $E^{new} > E^{old}$, compute $q \equiv e^{-\frac{E^{new}-E^{old}}{T_k}}$. Then draw a random number ζ , uniformly distributed between 0 and 1. If $\zeta \leq q$, accept the new parameter value. Set $E^{old} = E^{new}$.

If all parameters have been considered in turn, go to Step 4. If not all parameters have been considered in this iteration step, go to Step 5.

Step 9: If $\zeta > q$, retain the old parameter value. If all parameters have been considered in turn, go to Step 4. If not all parameters have been considered in this iteration step, go to Step 5.

Exit the algorithm when a certain number of iterations has been performed, or when $(|E^{old} - E^{new}|)/E^{old} < 0.01$. In this example, we used as termination criterion the fixed number of iterations, set to 1000. Figure 6.9a shows how the value of the cost function E changes as a function of the iterations. The output values of the algorithm are μ_1^{min} , μ_2^{min} , σ_1^{min} , σ_2^{min} and θ^{min} , ie the values of the parameters for which E obtained its minimum value during the whole process. For this example, these values were: $\mu_1^{min} = 179.0$, $\mu_2^{min} = 8.1$, $\sigma_1^{min} = 8.2$, $\sigma_2^{min} = 120.1$ and $\theta^{min} = 0.35$. Figure 6.9b shows how the mixture of the two Gaussians fits the histogram of the image.

Once we have fitted the histogram with the two Gaussians, we can apply formula (6.14) to derive the minimum error threshold. For this example, two valid roots were obtained: $t_1 = 159.5$ and $t_2 = 200.1$. This is because, the Gaussian, identified by the algorithm for modelling the part of the histogram that corresponds to the background grey values, is much narrower than the Gaussian that models the object. So, we have to use both these thresholds to identify the object pixels.

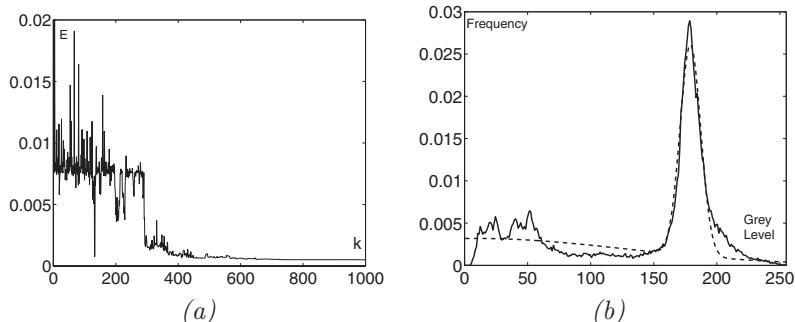


Figure 6.9: (a) The value of the cost function versus the number of iterations used. (b) The two Gaussians that are identified to model the object and the background.

The result is shown in figure 6.10. This result is not satisfactory. The use of the second threshold to restrict the background pixels in the range [160, 200] clearly created false object regions in the background. This is not because the significance of the second threshold is wrongly understood, but because the models do not fit very well the two

humps of the histogram. So, although the theory is correct, the practice is wrong because we model with Gaussians two probability density functions that are not Gaussians.



Figure 6.10: In white, the pixels with grey values in the range [160, 200].

What are the drawbacks of the minimum error threshold method?

The method has various drawbacks. For a start, we must know the prior probabilities for the pixels to belong to the object or the background, ie we must know θ . Next, we must know the distributions of the two populations. Often, it is possible to approximate these probability density functions with normal probability density functions, but even in that case one would have to estimate the parameters σ and μ of each function.

Is there any method that does not depend on the availability of models for the distributions of the object and the background pixels?

A method which does not depend on modelling the probability density functions is the **Otsu method**. Unlike the previous analysis, this method has been developed directly in the discrete domain. The method identifies the threshold that maximises the distinctiveness of the two populations to which it divides the image. This distinctiveness is expressed by the interclass variance (see Box 6.1) that can be shown to be

$$\sigma_B^2(t) = \frac{[\mu(t) - \mu\theta(t)]^2}{\theta(t)[1 - \theta(t)]} \quad (6.26)$$

where $p(x)$ are the values of the image histogram, μ is the mean grey value of the image, t is a hypothesised threshold and:

$$\mu(t) \equiv \sum_{x=1}^t xp_x \quad \text{and} \quad \theta(t) \equiv \sum_{x=1}^t p_x \quad (6.27)$$

The idea is then to start from the beginning of the histogram and test each grey value t for the possibility of being the threshold that maximises $\sigma_B^2(t)$, by calculating the values of $\mu(t)$ and $\theta(t)$ and substituting them into equation (6.26). We stop testing once the value of σ_B^2 starts decreasing. This way we identify t for which $\sigma_B^2(t)$ becomes maximal. This method tacitly assumes that function $\sigma_B^2(t)$ is well-behaved, ie that it has only one maximum.

Box 6.1. Derivation of Otsu's threshold

Consider that we have an image with L grey levels in total and its normalised histogram, so that for each grey value x , p_x represents the frequency with which the particular value arises. Then assume that we set the threshold to t . Let us assume that we are dealing with the case of a bright object on a dark background. The fraction of pixels that will be classified as background ones will be:

$$\theta(t) = \sum_{x=1}^t p_x \quad (6.28)$$

The fraction of pixels that will be classified as object pixels will be:

$$1 - \theta(t) = \sum_{x=t+1}^L p_x \quad (6.29)$$

The mean grey value of the background pixels and the object pixels, respectively, will be:

$$\begin{aligned} \mu_b &= \frac{\sum_{x=1}^t x p_x}{\sum_{x=1}^t p_x} \equiv \frac{\mu(t)}{\theta(t)} \\ \mu_o &= \frac{\sum_{x=t+1}^L x p_x}{\sum_{x=t+1}^L p_x} = \frac{\sum_{x=1}^L x p_x - \sum_{x=1}^t x p_x}{1 - \theta(t)} = \frac{\mu - \mu(t)}{1 - \theta(t)} \end{aligned} \quad (6.30)$$

Here we defined $\mu(t) \equiv \sum_{x=1}^t x p_x$, and μ is the mean grey value over the whole image, defined as:

$$\mu \equiv \frac{\sum_{x=1}^L x p_x}{\sum_{x=1}^L p_x} \quad (6.31)$$

Similarly, we may define the variance of each of the two populations, created by the choice of a threshold t , as:

$$\begin{aligned} \sigma_b^2 &\equiv \frac{\sum_{x=1}^t (x - \mu_b)^2 p_x}{\sum_{x=1}^t p_x} = \frac{1}{\theta(t)} \sum_{x=1}^t (x - \mu_b)^2 p_x \\ \sigma_o^2 &\equiv \frac{\sum_{x=t+1}^L (x - \mu_o)^2 p_x}{\sum_{x=t+1}^L p_x} = \frac{1}{1 - \theta(t)} \sum_{x=t+1}^L (x - \mu_o)^2 p_x \end{aligned} \quad (6.32)$$

Let us consider next the total variance of the distribution of the pixels in the image:

$$\sigma_T^2 = \sum_{x=1}^L (x - \mu)^2 p_x \quad (6.33)$$

We may split this sum into two:

$$\sigma_T^2 = \sum_{x=1}^t (x - \mu)^2 p_x + \sum_{x=t+1}^L (x - \mu)^2 p_x \quad (6.34)$$

As we would like eventually to involve the statistics defined for the two populations, we add and subtract inside each sum the corresponding mean:

$$\begin{aligned} \sigma_T^2 &= \sum_{x=1}^t (x - \mu_b + \mu_b - \mu)^2 p_x + \sum_{x=t+1}^L (x - \mu_o + \mu_o - \mu)^2 p_x \\ &= \sum_{x=1}^t (x - \mu_b)^2 p_x + \sum_{x=1}^t (\mu_b - \mu)^2 p_x + 2 \sum_{x=1}^t (x - \mu_b)(\mu_b - \mu) p_x + \\ &\quad \sum_{x=t+1}^L (x - \mu_o)^2 p_x + \sum_{x=t+1}^L (\mu_o - \mu)^2 p_x + 2 \sum_{x=t+1}^L (x - \mu_o)(\mu_o - \mu) p_x \end{aligned} \quad (6.35)$$

Next we substitute the two sums on the left of each line in terms of σ_b^2 and σ_o^2 , using equations (6.32). We also notice that the two sums in the middle of each line can be expressed in terms of equations (6.28) and (6.29), since μ , μ_b and μ_o are constants and they do not depend on the summing variable x :

$$\begin{aligned} \sigma_T^2 &= \theta(t)\sigma_b^2 + (\mu_b - \mu)^2\theta(t) + 2(\mu_b - \mu) \sum_{x=1}^t (x - \mu_b) p_x \\ &\quad + (1 - \theta(t))\sigma_o^2 + (\mu_o - \mu)^2(1 - \theta(t)) + 2(\mu_o - \mu) \sum_{x=t+1}^L (x - \mu_b) p_x \end{aligned} \quad (6.36)$$

The two terms with the sums are zero, since, for example:

$$\sum_{x=1}^t (x - \mu_b) p_x = \sum_{x=1}^t x p_x - \sum_{x=1}^t \mu_b p_x = \mu_b \theta(t) - \mu_b \theta(t) = 0 \quad (6.37)$$

Then by rearranging the remaining terms,

$$\begin{aligned} \sigma_T^2 &= \underbrace{\theta(t)\sigma_b^2 + (1 - \theta(t))\sigma_o^2}_{\text{terms depending on the variance within each class}} + \underbrace{(\mu_b - \mu)^2\theta(t) + (\mu_o - \mu)^2(1 - \theta(t))}_{\text{terms depending on the variance between the two classes}} \\ &\equiv \sigma_W^2(t) + \sigma_B^2(t) \end{aligned} \quad (6.38)$$

where $\sigma_W^2(t)$ is defined to be the within-class variance and $\sigma_B^2(t)$ is defined to be the between-class variance. Clearly, the total image variance σ_T^2 is a constant. We want to specify t so that $\sigma_W^2(t)$ is as small as possible, ie the classes that are created are as

compact as possible, and $\sigma_B^2(t)$ is as large as possible. Let us choose to work with $\sigma_B^2(t)$, ie let us try to choose t so that $\sigma_B^2(t)$ is maximum. We substitute in the definition of $\sigma_B^2(t)$ the expressions for μ_b and μ_o , as given by equations (6.30):

$$\begin{aligned}
 \sigma_B^2(t) &= (\mu_b - \mu)^2 \theta(t) + (\mu_o - \mu)^2 (1 - \theta(t)) \\
 &= \left[\frac{\mu(t)}{\theta(t)} - \mu \right]^2 \theta(t) + \left[\frac{\mu - \mu(t)}{1 - \theta(t)} - \mu \right]^2 (1 - \theta(t)) \\
 &= \frac{[\mu(t) - \mu\theta(t)]^2}{\theta(t)} + \frac{[\mu - \mu(t) - \mu + \mu\theta(t)]^2}{1 - \theta(t)} \\
 &= \frac{[\mu(t) - \mu\theta(t)]^2 [1 - \theta(t)] + \theta(t)[- \mu(t) + \mu\theta(t)]^2}{\theta(t)[1 - \theta(t)]} \\
 &= \frac{[\mu(t) - \mu\theta(t)]^2}{\theta(t)[1 - \theta(t)]}
 \end{aligned} \tag{6.39}$$

This function expresses the *interclass* variance $\sigma_B^2(t)$, in terms of the mean grey value of the image μ and quantities that can be computed once we know the values of the image histogram up to the chosen threshold t .

Example 6.8

Calculate Otsu's threshold for the image of Figure 6.3a and use it to threshold the image.

Figure 6.11a shows how $\sigma_B^2(t)$ varies as t scans all possible grey values. The first maximum of this function is at $t = 84$ and we choose this threshold to produce the result shown in figure 6.11b. We can see that the result is not noticeably different from the result obtained with the empirical threshold (figure 6.3c) and a little worse than the optimal threshold result (figure 6.8a). It is worse than the result obtained by hysteresis thresholding, reinforcing again the conclusion that spatial and grey level characteristics used in thresholding is a powerful combination.

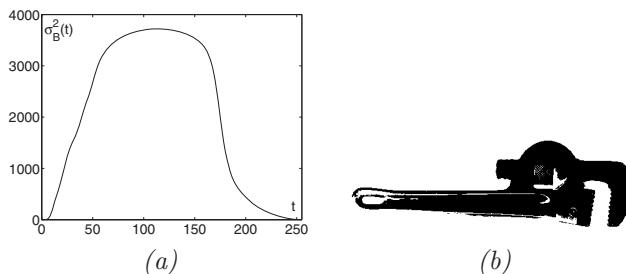


Figure 6.11: (a) The maximum of $\sigma_B^2(t)$ defines Otsu's threshold, which for this example is $t = 114$. (b) Resultant segmentation with Otsu's threshold.

Are there any drawbacks in Otsu's method?

Yes, a few:

1. Although the method does not make any assumption about the probability density functions $p_o(x)$ and $p_b(x)$, it describes them by using only their means and variances. Thus, it tacitly assumes that these two statistics are sufficient to represent them. This may not be true.
2. The method breaks down when the two populations are very unequal. When the two populations become very different in size from each other, $\sigma_B^2(t)$ may have two maxima and actually the correct maximum is not necessarily the global maximum. That is why in practice the correct maximum is selected from among all maxima of $\sigma_B^2(t)$ by checking that the value of the histogram at the selected threshold, p_t , is actually a valley (i.e. $p_t < p_{\mu_o}$ and $p_t < p_{\mu_b}$). Only if this is true, t should be accepted as the best threshold.
3. The method, as presented above, assumes that the histogram of the image is bimodal, ie that the image contains two classes. For more than two classes present in the image, the method has to be modified so that multiple thresholds are defined which maximise the *interclass* variance and minimise the *intraclass* variance.
4. The method will divide the image into two classes, even if this division does not make sense. A case when the method should not be directly applied is that of variable illumination.

How can we threshold images obtained under variable illumination?

In Chapter 4 (page 364), we saw that an image is essentially the *product* of a reflectance function $r(x, y)$, which is intrinsic to the viewed surfaces, and an illumination function $i(x, y)$:

$$f(x, y) = r(x, y)i(x, y) \quad (6.40)$$

Thus, any spatial variation of the illumination results in a multiplicative interference to the reflectance function that is recorded during the imaging process. We can convert the *multiplicative* interference into *additive*, if we take the logarithm of the image:

$$\ln f(x, y) = \ln r(x, y) + \ln i(x, y) \quad (6.41)$$

Then instead of forming the histogram of $f(x, y)$, we can form the histogram of $\ln f(x, y)$.

If we threshold the image according to the histogram of $\ln f(x, y)$, are we thresholding it according to the reflectance properties of the imaged surfaces?

No, under variable illumination. To be able to understand what is going on, we have to try to answer the following question. *How is the histogram of $\ln f(x, y)$ expressed in terms of the histograms of $\ln r(x, y)$ and $\ln i(x, y)$?* Ideally, we are interested in thresholding the histogram of $r(x, y)$, or $\ln r(x, y)$.

Let us define some new variables:

$$\begin{aligned} z(x, y) &\equiv \ln f(x, y) \\ \tilde{r}(x, y) &\equiv \ln r(x, y) \\ \tilde{i}(x, y) &\equiv \ln i(x, y) \end{aligned} \quad (6.42)$$

Therefore, equation (6.41) may be written as:

$$z(x, y) = \tilde{r}(x, y) + \tilde{i}(x, y) \quad (6.43)$$

If $f(x, y)$, $r(x, y)$ and $i(x, y)$ are thought of as random variables, then $z(x, y)$, $\tilde{r}(x, y)$ and $\tilde{i}(x, y)$ are also random variables. So, the question may be rephrased as follows. *What is the histogram of the sum of two random variables in terms of the histograms of the two variables?* A histogram may be thought of as a probability density function. Rephrasing the question again, we have the following. *What is the probability density function of the sum of two random variables in terms of the probability density functions of the two variables?*

In Box 6.2, we show that the probability density function of $z(x, y)$ is the convolution of the probability density function of $\tilde{r}(x, y)$ with the probability density function of $\tilde{i}(x, y)$. This means that any direct thresholding of the histogram of the logarithm of the grey values we observe will produce nonsense, as the histograms of the two effects we are trying to separate (that of the illumination field from that of the observed surface properties) are not simply added.

Box 6.2. The probability density function of the sum of two random variables

We have seen in Chapter 3 that the probability density function of a random variable is the derivative of the distribution function of the variable (see page 181). So, we may solve the problem by trying to answer the following question. *What is the distribution function of the sum of two random variables in terms of the probability density functions or the distribution functions of the two variables?* In the (\tilde{i}, \tilde{r}) space, equation (6.43) represents a line for a given value of z . By definition, we know that:

$$\text{Distribution function of } z = P_z(u) = \text{Probability of } z \leq u \equiv \mathcal{P}(z \leq u) \quad (6.44)$$

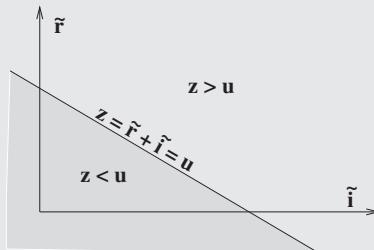


Figure 6.12: Sum z is less than u in the shadowed half plane.

Line $\tilde{r} + \tilde{i} = u$ divides the (\tilde{i}, \tilde{r}) plane into two half planes, one in which $z > u$ and one where $z < u$. The probability of $z < u$ is equal to the integral of the probability density function of pairs (\tilde{i}, \tilde{r}) , over the area of the half plane in which $z < u$ (see figure 6.12):

$$P_z(u) = \int_{\tilde{i}=-\infty}^{+\infty} \int_{\tilde{r}=-\infty}^{u-\tilde{i}} p_{\tilde{r}\tilde{i}}(\tilde{r}, \tilde{i}) d\tilde{r} d\tilde{i} \quad (6.45)$$

Here $p_{\tilde{r}\tilde{i}}(\tilde{r}, \tilde{i})$ is the joint probability density function of the two random variables \tilde{r} and \tilde{i} .

To work out the probability density function of z , we differentiate $P_z(u)$ with respect to u , using Leibniz's rule (see Box 4.9, on page 348), applied twice; once with

$$f(x; \lambda) \rightarrow \int_{\tilde{r}=-\infty}^{u-\tilde{i}} p_{\tilde{u}\tilde{i}}(\tilde{r}, \tilde{i}) d\tilde{r}$$

$$b(\lambda) \rightarrow +\infty$$

$$a(\lambda) \rightarrow -\infty \quad (6.46)$$

and once more when we need to differentiate $f(x; \lambda)$, which itself is an integral that depends on parameter u , with respect to which we differentiate:

$$\begin{aligned} p_z(u) &= \frac{dP_z u}{du} \int_{\tilde{i}=-\infty}^{+\infty} \frac{d}{du} \left[\int_{\tilde{r}=-\infty}^{u-\tilde{i}} p_{\tilde{r}\tilde{i}}(\tilde{r}, \tilde{i}) d\tilde{r} \right] d\tilde{i} \\ &= \int_{\tilde{i}=-\infty}^{+\infty} p_{\tilde{r}\tilde{i}}(u - \tilde{i}, \tilde{i}) d\tilde{i} \end{aligned} \quad (6.47)$$

The two random variables \tilde{r} and \tilde{i} , one associated with the imaged surface and one with the source of illumination, are independent, and therefore their joint probability density function may be written as the product of their two probability density functions:

$$p_{\tilde{r}\tilde{i}}(\tilde{r}, \tilde{i}) = p_{\tilde{r}}(\tilde{r}) p_{\tilde{i}}(\tilde{i}) \quad (6.48)$$

Upon substitution in (6.47), we obtain:

$$p_z(u) = \int_{-\infty}^{+\infty} p_{\tilde{r}}(u - \tilde{i}) p_{\tilde{i}}(\tilde{i}) d\tilde{i} \quad (6.49)$$

This shows that the histogram (= probability density function) of z is equal to the *convolution* of the two histograms of the two random variables \tilde{r} and \tilde{i} .

If the illumination is uniform, then:

$$i(x, y) = \text{constant} \Rightarrow \tilde{i} = \ln i(x, y) = \tilde{i}_o = \text{constant} \quad (6.50)$$

Then $p_{\tilde{i}}(\tilde{i}) = \delta(\tilde{i} - \tilde{i}_o)$, and after substitution in (6.49) and integration, we obtain $p_z(u) = p_{\tilde{r}}(u)$.

That is, under uniform illumination, the histogram of the reflectance function (intrinsic to the object) is the same as the histogram of the observed grey values. If, however, the illumination is not uniform, even if we had a perfectly distinguishable object, the histogram is badly distorted and the various thresholding methods break down.

Since straightforward thresholding methods break down under variable illumination, how can we cope with it?

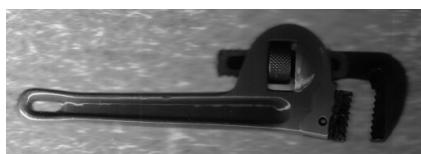
There are two ways in which we can circumvent the problem of variable illumination:

1. Divide the image into more or less uniformly illuminated patches and histogram and threshold each patch as if it were a separate image. Some adjustment may be needed when the patches are put together, as the threshold essentially will jump from one value in one patch to another value in a neighbouring patch.
2. Obtain an image of just the illumination field, using the image of a surface with uniform reflectance and divide the image $f(x, y)$ by $i(x, y)$, ie essentially subtract the illumination component $\tilde{i}(x, y)$ from $z(x, y)$. Then multiply $\frac{f(x,y)}{i(x,y)}$ with a reference value, say $i(0, 0)$, to bring the whole image under the same illumination and proceed using the corrected image. This is essentially the method of flatfielding (see page 366).

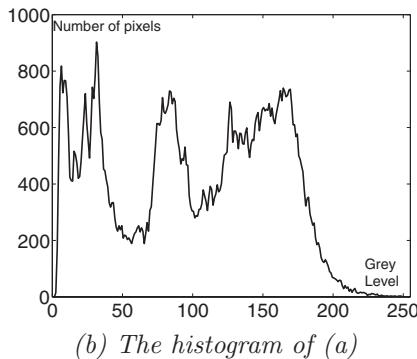
Example 6.9

Threshold the image of Figure 6.13a.

This image exhibits an illumination variation from left to right. Figure 6.13b shows the histogram of the image. Using Otsu's method, we identify threshold $t = 99$. The result of thresholding the image with this threshold is shown in Figure 6.13c. The result of dividing the image into three subimages from left to right and applying Otsu's method to each subimage separately is shown in Figure 6.13d. The three local thresholds, from left to right, were 114, 84 and 52.



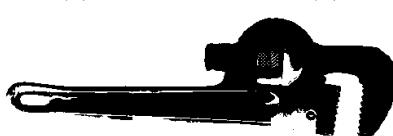
(a) Original image



(b) The histogram of (a)



(c) Global thresholding



(d) Local thresholding

Figure 6.13: Global versus local thresholding for an image with variable illumination.

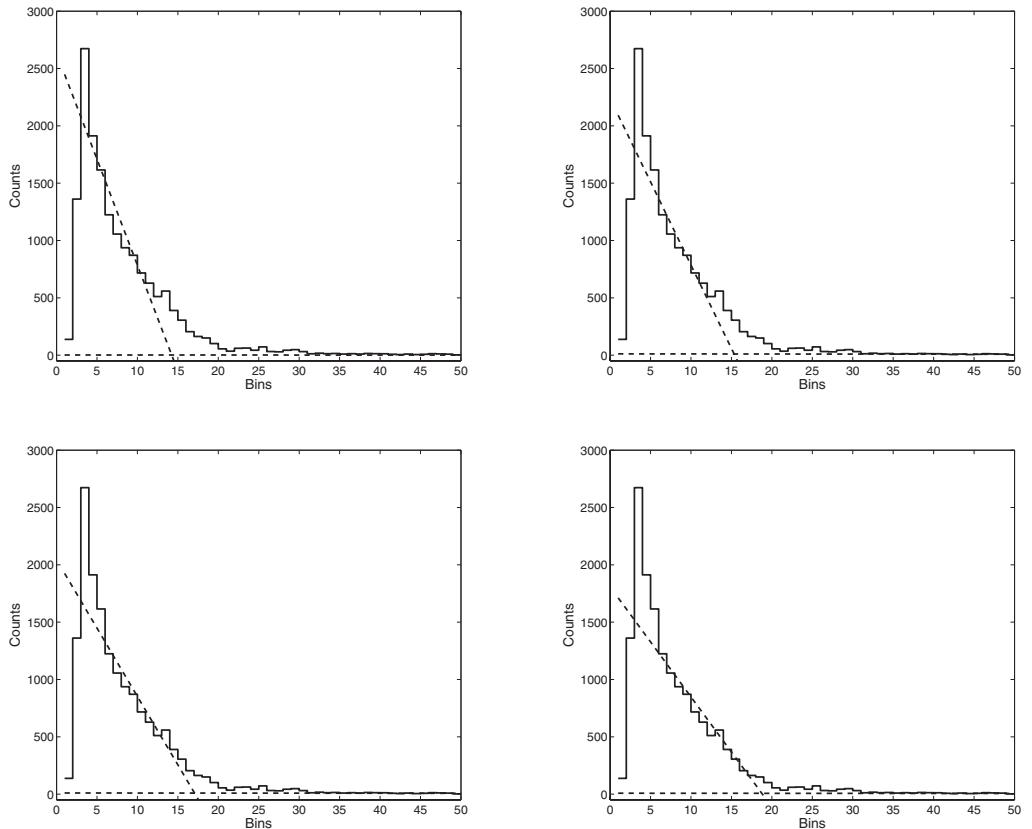


Figure 6.14: Identifying a threshold when the histogram has a long tail. In the first iteration, 5 bins on the right of the peak and 5 bins on the left of the last bin are fitted with straight lines. Their intersection defines a first estimate of the threshold. The points on the left of the threshold and up to the peak are fitted again and the outliers are removed. The inliers are refitted to produce the first line for the second iteration. The points on the right of the threshold are also fitted again and the outliers are removed. The inliers are refitted to produce the second line for the second iteration. Their intersection yields the second estimate of the threshold. In the third and fourth iterations, this process is repeated to refine the estimate of the threshold.

What do we do if the histogram has only one peak?

Such situation is often encountered in practice. It arises when one of the two populations has a roughly flat distribution that creates a long tail of the histogram. One way to identify a meaningful threshold in such a case is to fit with straight lines the descending part of the peak and the long tail, and select as threshold the coordinate where the two lines meet. The so called **knee** algorithm is as follows.

Step 1: Consider the peak of the histogram, with abscissa b_{peak} , and n bins on the right of the peak towards the long tail. Typical value for $n = 5$. In the (bin, count) space fit these

points with a straight line using least square error.

Step 2: Consider the last bin of the histogram, with abscissa b_{last} , and n bins on the left of this bin, towards the peak. In the (bin, count) space fit these points with a straight line, using least square error.

Step 3: Work out the point where the two lines meet. The abscissa of the point of intersection is the first estimate of the value of the threshold, t_1 .

Step 4: Consider all points with abscissa in the range $[b_{peak}, t_1]$ and fit them with a straight line in the least square error sense. If all points have residual error less than a tolerance, keep this line. If not, omit the points with error larger than the tolerance and refit the rest with a least square error line.

Step 5: Consider all points with abscissa in the range $[t_1, b_{last}]$ and fit them with a straight line in the least square error sense. If all points have residual error less than a tolerance, keep this line. If not, omit the points with error larger than the tolerance and refit the rest with a least square error line.

Step 6: Find the intersections of the lines constructed in Steps 4 and 5. The abscissa of the point of intersection is the new estimate of the value of the threshold, t_2 .

The process may be repeated as many times as we like. Every time we go to Step 4 and 5, we consider the pairs of points that are assumed to be represented by the straight line we have for them.

An example of using this algorithm to identify a threshold for a histogram with a long tail is shown in figure 6.14.

Are there any shortcomings of the grey value thresholding methods?

Yes. With the exception of hysteresis thresholding, which is of limited use, the spatial proximity of the pixels in the image is not considered at all in the segmentation process. Instead, only the grey values of the pixels are used.

For example, consider the two images in figure 6.15.

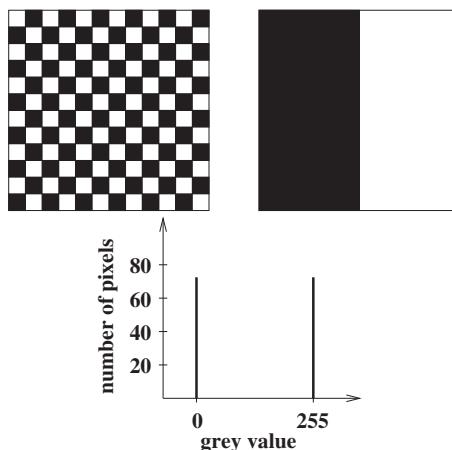


Figure 6.15: Two very different images with identical histograms.

Clearly, the first image is the image of a uniform region, while the second image contains two quite distinct regions. Even so, both images have identical histograms. Their histograms are bimodal and we can easily choose a threshold to split the pixels that make up the two peaks. However, if we use this threshold to segment the first image, we shall get nonsense.

How can we cope with images that contain regions that are not uniform but they are *perceived* as uniform?

Regions that are not uniform in terms of the grey values of their pixels, but are perceived as uniform, are called **textured regions**. For segmentation purposes then, each pixel is not characterised by its grey value, but by another number or numbers, which quantify the variation of the grey values in a small patch around that pixel. These numbers, which characterise the pixels, are called **features** or **attributes**. If only one attribute is used to characterise a pixel, any of the thresholding methods discussed so far may be used to segment the object (see example 6.10). Usually, however, more than one attribute are used to characterise the pixels of an image. So, we may envisage that each pixel is characterised not by a scalar, but by a vector, each component of which measures something different at or around the pixel position. Then each pixel is represented by a point in a multidimensional space, where we measure one component of the feature vector per axis. Pixels belonging to the same region will have similar or identical values in their attributes and, thus, they will cluster together. The problem then becomes one of identifying clusters of pixels in a multidimensional space. Essentially it is similar to histogramming, only now we deal with multidimensional histograms. There are several other **clustering** methods that may be used, but they are in the realm of **pattern recognition** and thus beyond the scope of this book¹.

Example 6.10

You are asked to segment the image of figure 6.16 into object and background. Black pixels in the image have value 0 and white pixels have value 1. Define a scalar that may be used to characterise each pixel and segment the image by thresholding the values of this scalar.

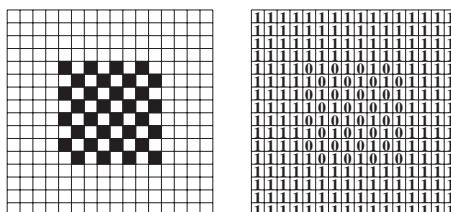


Figure 6.16: An object on a white background.

¹More on texture may be found in the book “Image Processing, dealing with Texture”, by Petrou and Garcia Sevilla, John Wiley & Sons, Ltd, ISBN-13-978-0-470-02628-1

We define as a characteristic value for each pixel the absolute difference between the grey value of the pixel and the average grey value of its four nearest neighbours. Figure 6.17a then shows the value this scalar has for all image pixels. Figure 6.17b shows the histogram of these values. We select, as an appropriate threshold, the value of 0.5. Any pixel with value greater or equal to this threshold is considered as belonging to the same image region. The two segments we identify this way are presented as grey and white in figure 6.18.

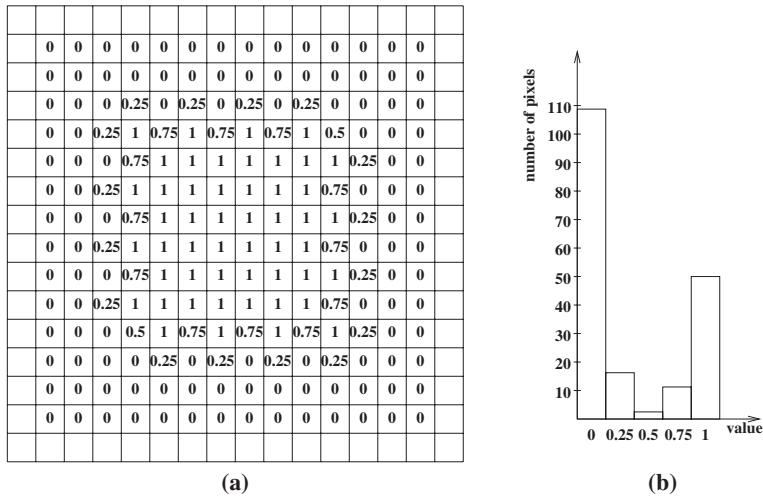


Figure 6.17: (a) The values of a feature that may be used to characterise the image of figure 6.16. (b) The histogram of these values.

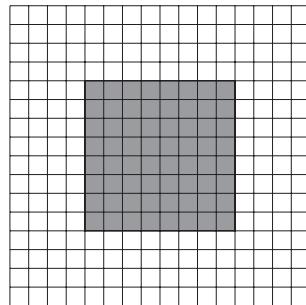


Figure 6.18: Segmentation result of the image of figure 6.16 using the values of figure 6.17.

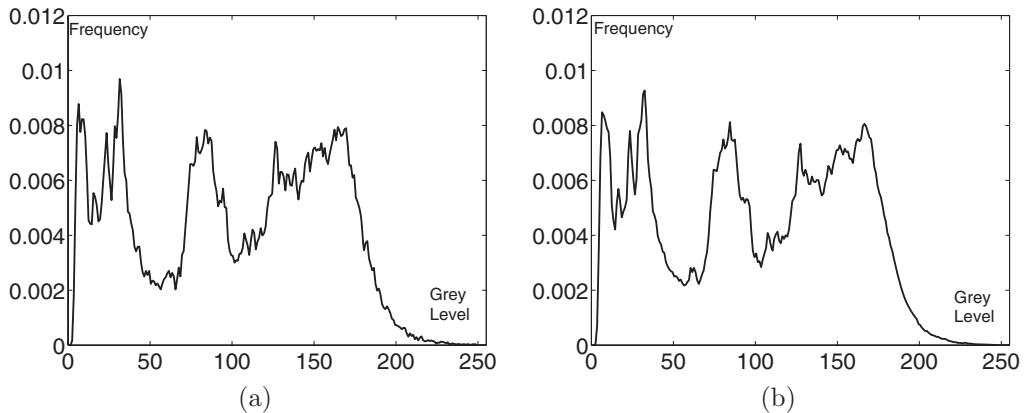


Figure 6.19: (a) The histogram of image 6.13a. (b) The histogram of the same image upsampled by a factor of 4 in each direction using linear interpolation.

Can we improve histogramming methods by taking into consideration the spatial proximity of pixels?

To a limited extent, yes. We may upsample the image by using bilinear (or some other form of) interpolation and then compute the histogram. This way, the histogram becomes smoother (because it is computed from many more samples) and, at the same time, the extra pixels created have values that depend on the spatial arrangement of the original pixels. Other more sophisticated methods of histogram estimation also exist, using the so called **kernel functions**. A pixel may be thought of as a delta function with which we sample the scene at a certain point. A kernel-based method replaces that delta function by a function with more spread, the kernel function. Then it treats the image as a collection of overlapping kernel functions.

Figure 6.19 shows the histogram of figure 6.13a, on page 548, and, next to it, the same histogram computed after upsampling the original image by a factor of 4 in each direction: three extra rows and three extra columns were inserted between any two adjacent rows and columns, and the values of the new pixels were computed by using bilinear interpolation as described in Chapter 5 (see page 518). We can see how much smoother the new histogram is. A more accurate estimation of the histogram allows the more accurate estimation of subsequent quantities from it, including, for example, the threshold for performing segmentation.

Are there any segmentation methods that take into consideration the spatial proximity of pixels?

Yes, they are called **region growing** methods. Examples of such methods are the **watershed** segmentation method and the less sophisticated but more straightforward **split and merge** algorithm. In general, one starts from some seed pixels and attaches neighbouring pixels to them, provided the attributes of the pixels in the region created in this way vary within a predefined range. So, each seed grows gradually by accumulating more and more neighbouring pixels, until all pixels in the image have been assigned to a region.

How can one choose the seed pixels?

There is no clear answer to this question, and this is the most important drawback of this type of method. In some applications, the choice of seeds is easy. For example, in target tracking in infrared images, the target will appear bright, and one can use as seeds the few brightest pixels. The split and merge method does not require any seeds. The watershed method identifies the seeds by **morphological image reconstruction**.

How does the split and merge method work?

Initially the whole image is considered as one region. If the range of attributes within this region is greater than a predetermined value, then the region is split into four quadrants and each quadrant is tested in the same way, until every square region created in this way contains pixels with range of attributes within the given value. At the end, all adjacent regions with attributes within the same range may be merged.

An example is shown in figure 6.20, where, for simplicity, a binary 8×8 image is considered. The tree structure shows the successive splitting of the image into quadrants. Such a tree is called **quad tree**.

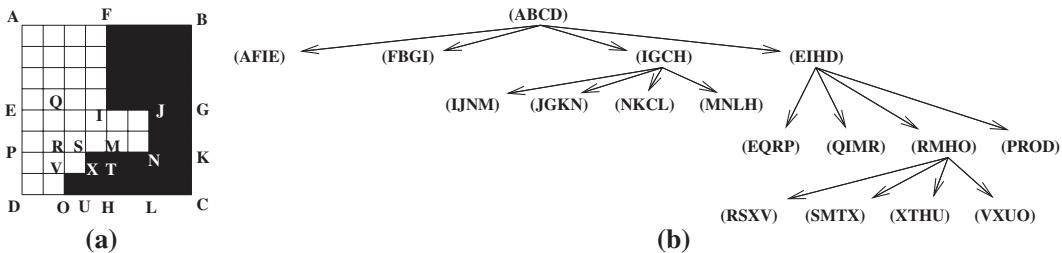


Figure 6.20: Image segmentation by splitting inhomogeneous quadrants.

We end up having the following regions:

$$\begin{aligned} & (\text{AFIE})(\text{FBGI})(\text{IJNM})(\text{JGKN})(\text{NKCL})(\text{MNLH}) \\ & (\text{EQRP})(\text{QIMR})(\text{PROD})(\text{RSXV})(\text{SMTX})(\text{XTHU})(\text{VXUO}) \end{aligned}$$

These are all the leaves of the quad tree. Any two adjacent regions then are checked for merging and eventually only the two main regions of irregular shape emerge. The above quad tree structure is clearly favoured when the image is square with $N = 2^n$ pixels in each side.

Split and merge algorithms often start at some intermediate level of the quad tree (ie some blocks of size $2^l \times 2^l$, where $l < n$) and check each block for further splitting into four square sub-blocks and any two adjacent blocks for merging. At the end, again we check for merging any two adjacent regions.

What is morphological image reconstruction?

Consider an image $f(i, j)$. Consider also another image $g(i, j)$, called **the mask**, such that $g(i, j) \leq f(i, j)$ for every pixel (i, j) . Define the **dilation** of image $g(i, j)$, by a **structuring**

element B , as follows: consider a neighbourhood of the shape and size of B around each pixel of $g(i, j)$; select the largest value inside this neighbourhood and assign it to the central pixel in the output image. This operation is denoted as $g(i, j) \oplus B$. The reconstruction of image f by g is obtained by iterating

$$g^{k+1}(i, j) = \min\{f(i, j), g^k(i, j) \oplus B\} \quad (6.51)$$

until image g^k does not change any further.

Consider also another mask image \tilde{g} , such that $\tilde{g}(i, j) \geq f(i, j)$ for every pixel (i, j) . Define the **erosion** of image \tilde{g} by a structuring element B , as follows: consider a neighbourhood of the shape and size of B around each pixel of \tilde{g} ; select the smallest value inside this neighbourhood and assign it to the central pixel in the output image. This operation is denoted as $g(i, j) \ominus B$. The reconstruction of image f by \tilde{g} is obtained by iterating

$$\tilde{g}^{k+1}(i, j) = \max\{f(i, j), \tilde{g}^k(i, j) \ominus B\} \quad (6.52)$$

Example 6.11

Reconstruct morphologically image f , shown in figure 6.21a, using a structuring element of size 3×3 . Start by creating mask g by subtracting 1 from all pixels of the original image f .

Figure 6.21b shows image g constructed from the original image.

15 16 16 15 14 14 15 14 15 15 15 15 14 14 15 15 16	14 15 15 14 13 13 14 13 14 14 14 14 13 13 14 14 15
15 15 16 16 15 14 14 14 14 15 15 16 16 16 16 15 15	14 14 15 15 14 13 13 13 13 14 14 14 15 15 15 14 14
14 16 16 15 14 14 15 14 15 15 16 14 16 15 14 14 14	13 15 15 14 13 13 14 13 14 14 15 13 15 14 13 13
14 15 16 16 17 16 16 18 17 16 13 14 15 16 15 14	13 14 15 15 16 15 15 17 16 15 12 13 14 15 14 13
13 15 15 15 26 26 27 29 28 17 15 14 14 15 15 16	12 14 14 14 25 25 26 28 27 16 14 13 13 14 14 15
13 16 15 16 27 27 27 26 26 16 14 15 13 13 14 16	12 15 14 15 26 26 26 25 25 15 13 14 12 12 13 15
14 15 14 15 26 28 31 28 27 17 14 15 13 14 13 14	13 14 13 14 25 27 30 27 26 16 13 14 12 13 12 13
14 14 16 15 28 29 26 27 27 18 16 15 14 14 15 16	13 13 15 14 27 28 25 26 26 17 15 14 13 13 14 15
15 16 15 16 28 27 27 28 29 17 16 16 14 15 15 15	14 15 14 15 27 26 27 28 16 15 15 13 14 14 14
15 16 15 14 15 14 15 15 14 13 16 15 15 15 14 14	14 15 14 13 14 13 14 14 14 13 12 15 14 14 13 13
16 17 18 15 16 18 17 17 14 23 22 23 14 15 13 14	15 16 17 14 15 17 16 16 13 22 21 22 13 14 12 13
17 17 18 29 25 23 18 16 15 24 24 22 13 14 13 13	16 16 17 28 24 22 17 15 14 23 23 21 12 13 12 12
17 18 17 18 31 22 17 16 16 23 21 22 16 15 14 13	16 17 16 17 30 21 16 15 15 22 20 21 15 14 13 12
16 17 17 18 19 20 16 16 15 16 15 14 14 15 13 15	15 16 16 17 18 19 15 15 14 15 14 13 14 12 14 13
16 18 18 17 17 18 17 17 16 16 14 14 14 13 14 15	15 17 17 16 16 17 16 16 15 15 13 13 13 12 13 14
17 17 16 16 16 17 16 16 15 16 15 15 14 14 15 15	16 16 15 15 15 16 15 15 14 15 14 13 13 13 14 14

(a)

(b)

Figure 6.21: (a) An original image f . (b) Image g constructed from image f by subtracting 1 from all pixel values.

Figure 6.22 shows the result of the first iteration of the algorithm. Figure 6.23 shows the results of the second and third (final) iterations. Figure 6.23b is the morphological reconstruction of f by g , obtained after three iterations. As iterations progress, the image shrinks as the border pixels cannot be processed.

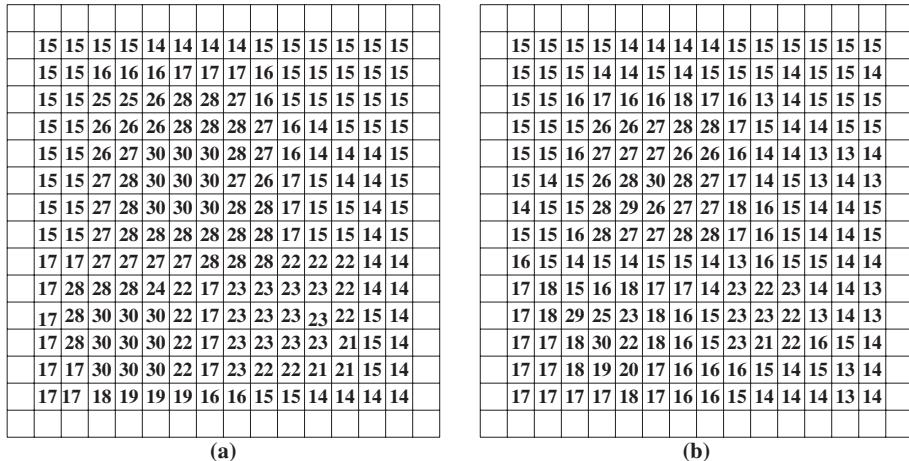


Figure 6.22: (a) The dilation of image g , by a structuring element of size 3×3 : it was created by taking the maximum of image 6.21b inside a sliding window of size 3×3 and placing it in the central position of the window in the output image. (b) The morphological reconstruction of f by g : it was created by taking the minimum between 6.21a and (a), pixel by pixel.

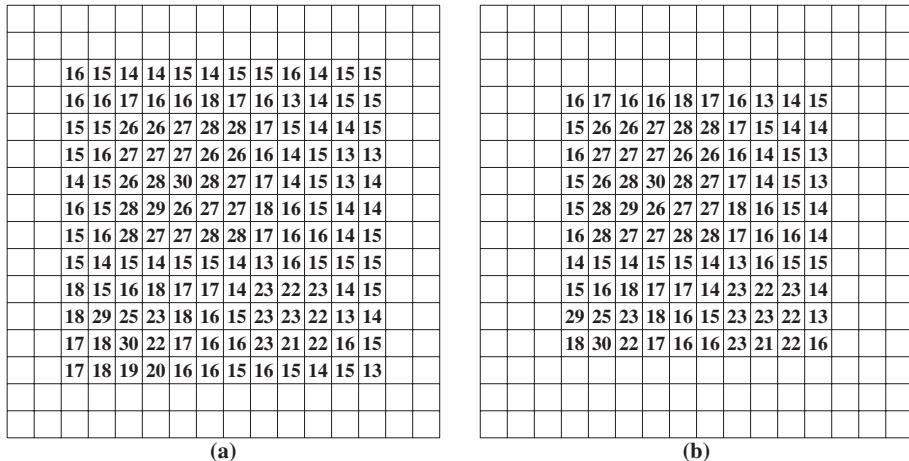


Figure 6.23: (a) The result of applying the process shown in figure 6.22 to image 6.22a. (b) The result of applying the process shown in figure 6.22 to (a). Further applications of the same process do not change the result.

How does morphological image reconstruction allow us to identify the seeds needed for the watershed algorithm?

If we subtract from image f its reconstruction by g , we are left with the significant maxima of f . If we subtract from image f its reconstruction by \tilde{g} , we are left with the significant minima of f . These may be used as seed patches which may be grown to segments of the image, by accumulating more and more pixels that surround them. Each region stops growing when it comes in contact with another region. To ensure that the meeting points between regions are at the correct places, we use the gradient magnitude image to perform the seed selection and region growing method.

How do we compute the gradient magnitude image?

The gradient vector of a 2D function $f(x, y)$ is $(\partial f(x, y)/\partial x, \partial f(x, y)/\partial y)$. Its magnitude is $\sqrt{\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2}$. In the discrete domain, the partial derivatives are replaced by the first differences of the function, $\Delta_i f(i, j)$ and $\Delta_j f(i, j)$:

$$\text{Gradient-Magnitude} \equiv \sqrt{(\Delta_i f(i, j))^2 + (\Delta_j f(i, j))^2} \quad (6.53)$$

Figure 6.24 shows an original image and the output obtained if the $\begin{bmatrix} -1 & +1 \end{bmatrix}$ and $\begin{bmatrix} +1 \\ -1 \end{bmatrix}$ convolution filters are applied along the horizontal and vertical directions, respectively, to compute the first image differences. The outputs of the two convolutions are squared, added and square rooted to produce the gradient magnitude associated with each pixel.

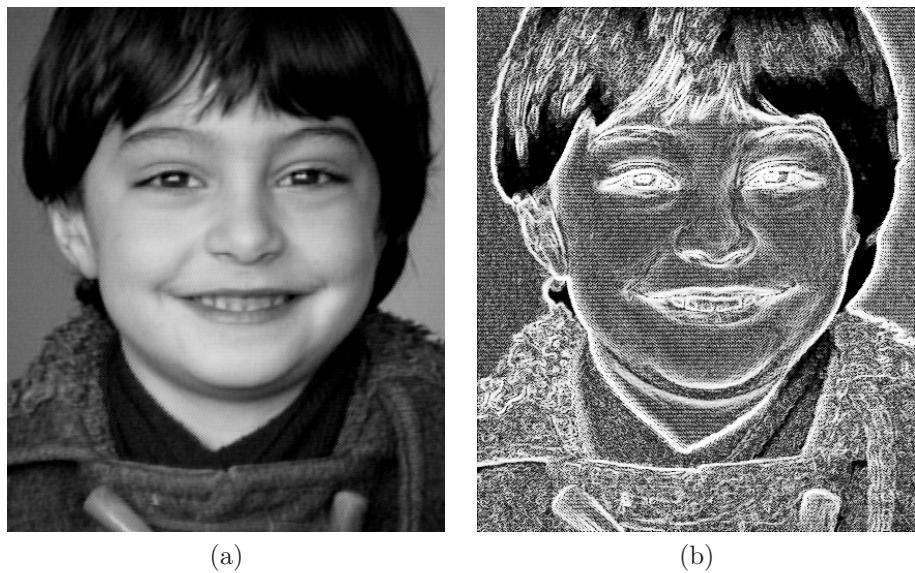


Figure 6.24: (a) Original image. (b) The image of the gradient magnitudes. For displaying purposes, the gradient image has been subjected to histogram equalisation.

Example 6.12

Use the morphological reconstruction of image f , shown in 6.22a, by g , created in example 6.11, to identify the maxima of image f .

The morphological reconstruction of f by g is shown in 6.23b. To identify the maxima of f , we subtract it from the original image f . The result is shown in 6.25b. The maxima are highlighted in grey in the result and in the original image.

15	16	16	15	14	14	15	14	15	15	15	14	14	15	15	16
15	15	16	16	15	14	14	14	14	15	15	16	16	16	15	15
14	16	16	15	14	14	15	14	15	16	14	16	15	14	14	14
14	15	16	16	17	16	16	18	17	16	13	14	15	16	15	14
13	15	15	15	26	26	27	29	28	17	15	14	14	15	15	16
13	16	15	16	27	27	27	26	26	16	14	15	13	13	14	16
14	15	14	15	26	28	31	28	27	17	14	15	13	14	13	14
14	14	16	15	28	29	26	27	27	18	16	15	14	14	15	16
15	16	15	16	28	27	27	28	29	17	16	16	14	15	15	15
15	16	15	14	15	14	15	15	14	13	16	15	15	15	14	14
16	17	18	15	16	18	17	17	14	23	22	23	14	15	13	14
17	17	18	29	25	23	18	16	15	24	24	22	13	14	13	13
17	18	17	18	31	22	17	16	16	23	21	22	16	15	14	13
16	17	17	18	19	20	16	16	15	16	15	14	15	13	15	14
16	18	18	17	17	18	17	17	16	16	14	14	14	13	14	15
17	17	16	16	16	17	16	16	15	15	15	14	14	15	15	15

(a)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b)

Figure 6.25: (a) The original image f . (b) After subtracting the morphological reconstruction of f by g . Grey indicates the identified maxima of f .

What is the role of the number we subtract from f to create mask g in the morphological reconstruction of f by g ?

If we create g by subtracting from f a number I_0 , we are identifying maxima in f that stick out from their surroundings by at most I_0 .

Example 6.13

Identify all maxima that are less significant than 3, ie they stick out from their surrounding pixels by at most 3 grey levels, in image f shown in figure 6.22a, using a structuring element of size 3×3 .

We start by creating image g by subtracting 3 from all pixels of the original image f . The result is shown in 6.26a. Figure 6.26b shows the result of reconstructing f by this

g , obtained after three iterations. Figure 6.27 shows the maxima identified this time. We note that they indeed are the maxima that differ from their surrounding pixels by at most 3 values.

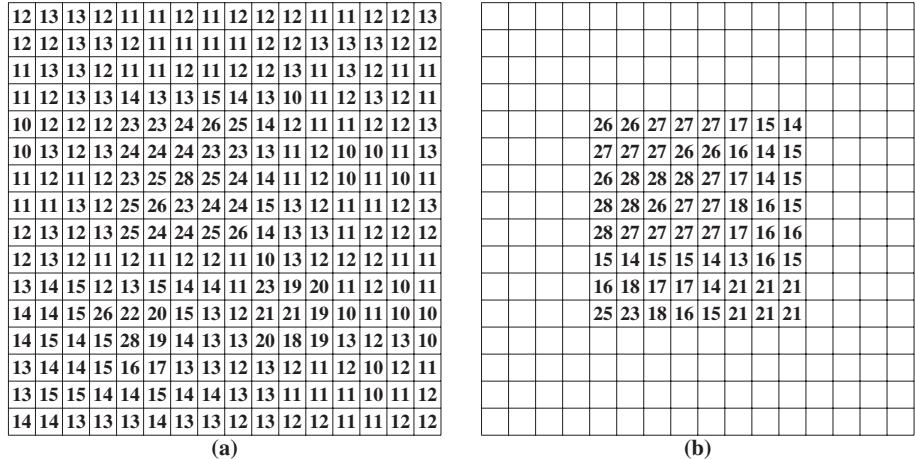


Figure 6.26: (a) Image g constructed from f by subtracting 3. (b) The morphological reconstruction of f by g , using a 3×3 structuring element.

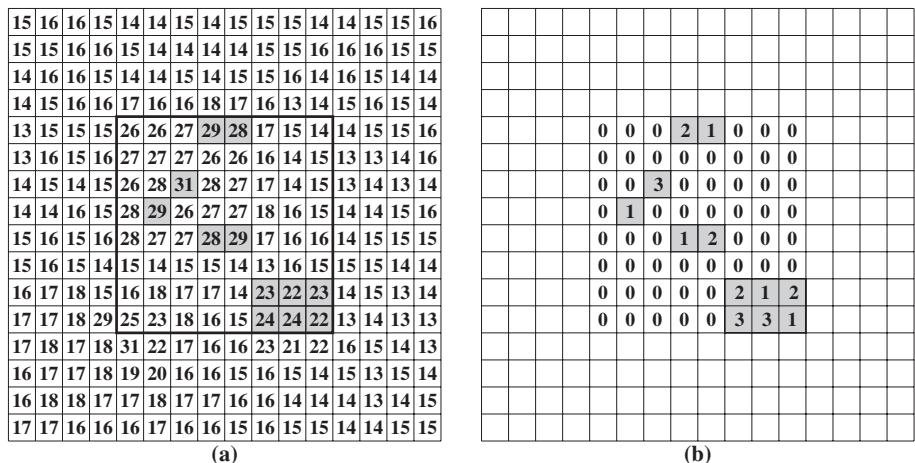


Figure 6.27: (a) The original image f . (b) After subtracting the morphological reconstruction of f by g . Grey indicates the identified maxima of f within the region that is not affected by border effects.

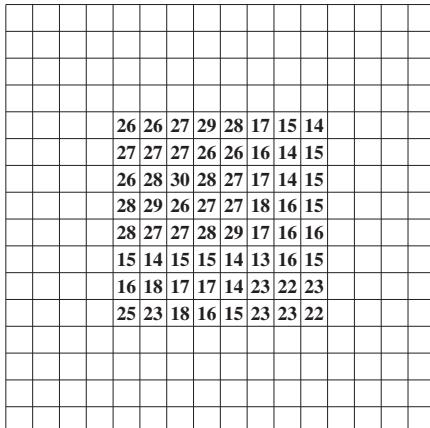
What is the role of the shape and size of the structuring element in the morphological reconstruction of f by g ?

If we create g by subtracting from f a number I_0 , we are identifying maxima in f that stick out from their surroundings by at most I_0 . The structuring element we use identifies the shape and size of the neighbourhood of a pixel over which it must stick out in order to be identified as a maximum.

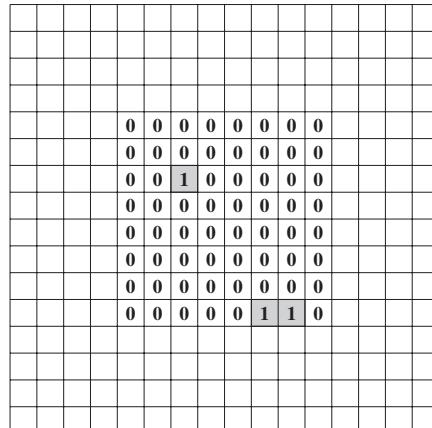
Example 6.14

Identify all maxima that stick out from their surrounding pixels, inside a local window of size 5×5 , by one grey level, in image f , shown in figure 6.22a.

We start by creating image g by subtracting 1 from all pixels of the original image f . The result is shown in 6.22b. Figure 6.28a shows the result of reconstructing f by this g using a 5×5 structuring element. Two iterations were necessary. Figure 6.28b shows the maxima identified this time. We note that they indeed are only the maxima that differ from their surrounding 5×5 pixels by more than 1 grey value.



(a)



(b)

Figure 6.28: (a) The morphological reconstruction of f by g using a 5×5 structuring element. (b) After subtracting the morphological reconstruction of f by g from f . Grey indicates the identified maxima of f .

Example 6.15

You are given the following sequence of numbers:

$$f = [6, 10, 10, 10, 7, 8, 8, 7, 4, 11, 6, 6, 6, 8, 8, 5, 5, 5, 5, 6, 4, 5, 4] \quad (6.54)$$

Identify all its local maxima by morphological reconstruction.

We must apply formula (6.51), on page 555. We first create sequence g by subtracting 1 from f . This is shown in figure 6.29a. Then we must dilate g . We select to do that by using a structuring element of length 3, ie by taking the maximum of every three successive elements of g . However, before we do that, we extend g by one sample on either side, so that all samples of interest have left and right neighbours. Finally, we take the minimum between the dilated version of g and f . The successive steps of iteratively applying this process, until convergence, are shown in figure 6.29.

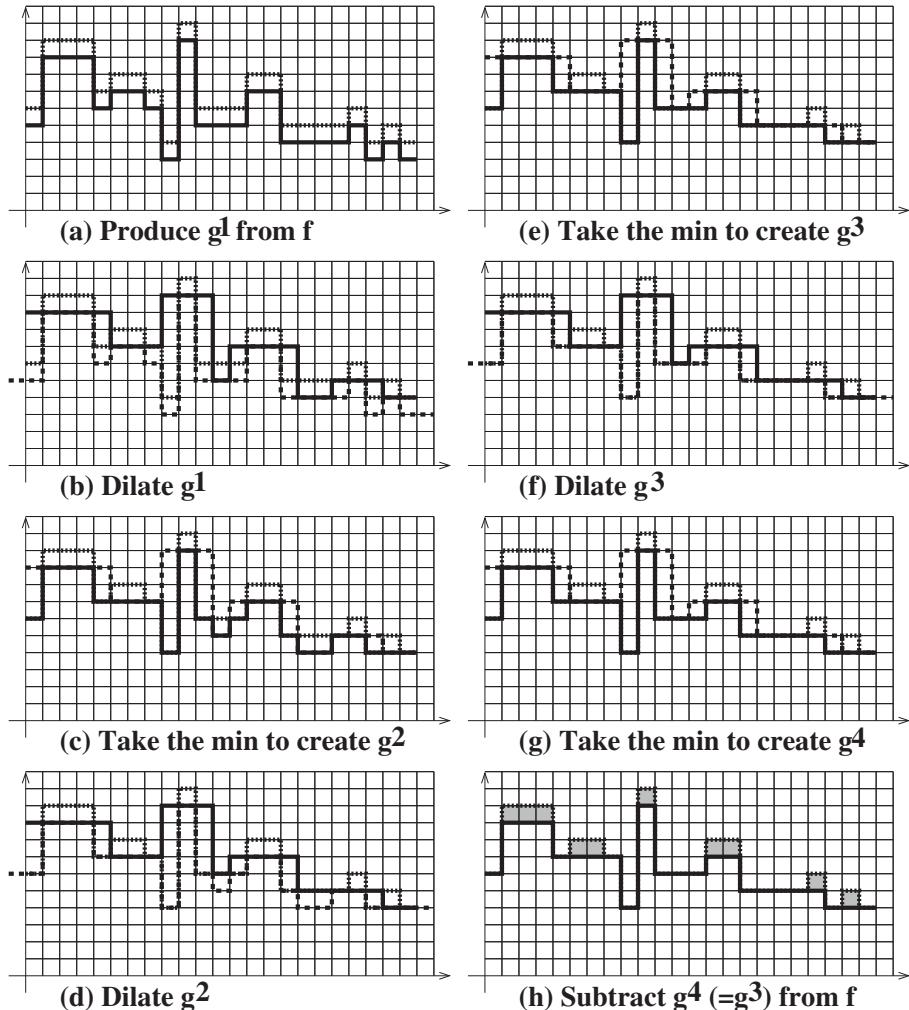


Figure 6.29: The sequence of steps required in order to apply formula (6.51) to identify the local maxima in sequence f . In each panel, sequence f is shown with a dotted line. In each panel, the sequence that is being processed is shown with a dashed line and the result obtained is shown with a continuous line.

Example 6.16

For the sequence of example 6.15, identify the local maxima that stick out from their surroundings by at most 2 levels.

We must apply formula (6.51). As we are interested in local maxima that stick above their surroundings by at most 2 levels, we create sequence g by subtracting 2 from f . We repeat then the process of example 6.15 until convergence. All steps needed are shown in figures 6.30 and 6.31. We observe two things.

(i) The more g differs from f , the more iteration steps are needed for convergence. This is expected, as g has to evolve more in order to “get stuck” underneath f . Note that, if we had created g by subtracting from f a much larger number than 2, g would never reach f to be stopped by it, but rather g would converge to a flat signal below any trough or peak of f . So, the maximum number we should use, to create g from f , should be the peak-to-peak amplitude of the largest fluctuation of the f values.

(ii) If i_0 is the value we subtract from f in order to create g , the process identifies all peaks that stick above their surroundings by at most i_0 . This means that if we want to identify only the peaks that stick above their surrounding by exactly i_0 , we must either threshold the result, to keep only the maxima that have value exactly i_0 , or we must apply a hierarchical approach, where we successively identify peaks of increasing maximum size, until no g converges to the flat signal, and then subtract each result from the next one. This way we can segment the image using successively lower and lower peaks in a multiresolution scheme, which allows us to stop at the desired resolution.

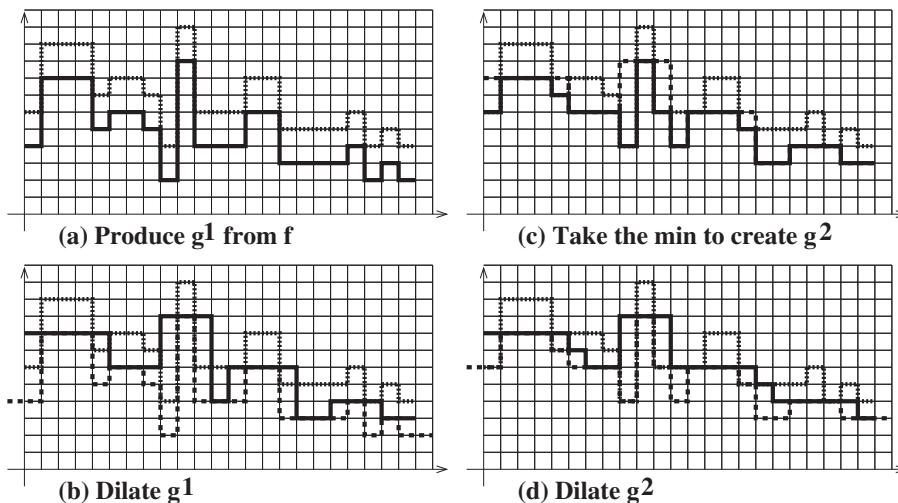


Figure 6.30: The sequence of steps required in order to apply formula (6.51) to identify the local maxima in sequence f , that stick out by at most 2 levels above their surroundings. In each panel, sequence f is shown with a dotted line. In each panel, the sequence that is being processed is shown with a dashed line and the result obtained is shown with a continuous line.

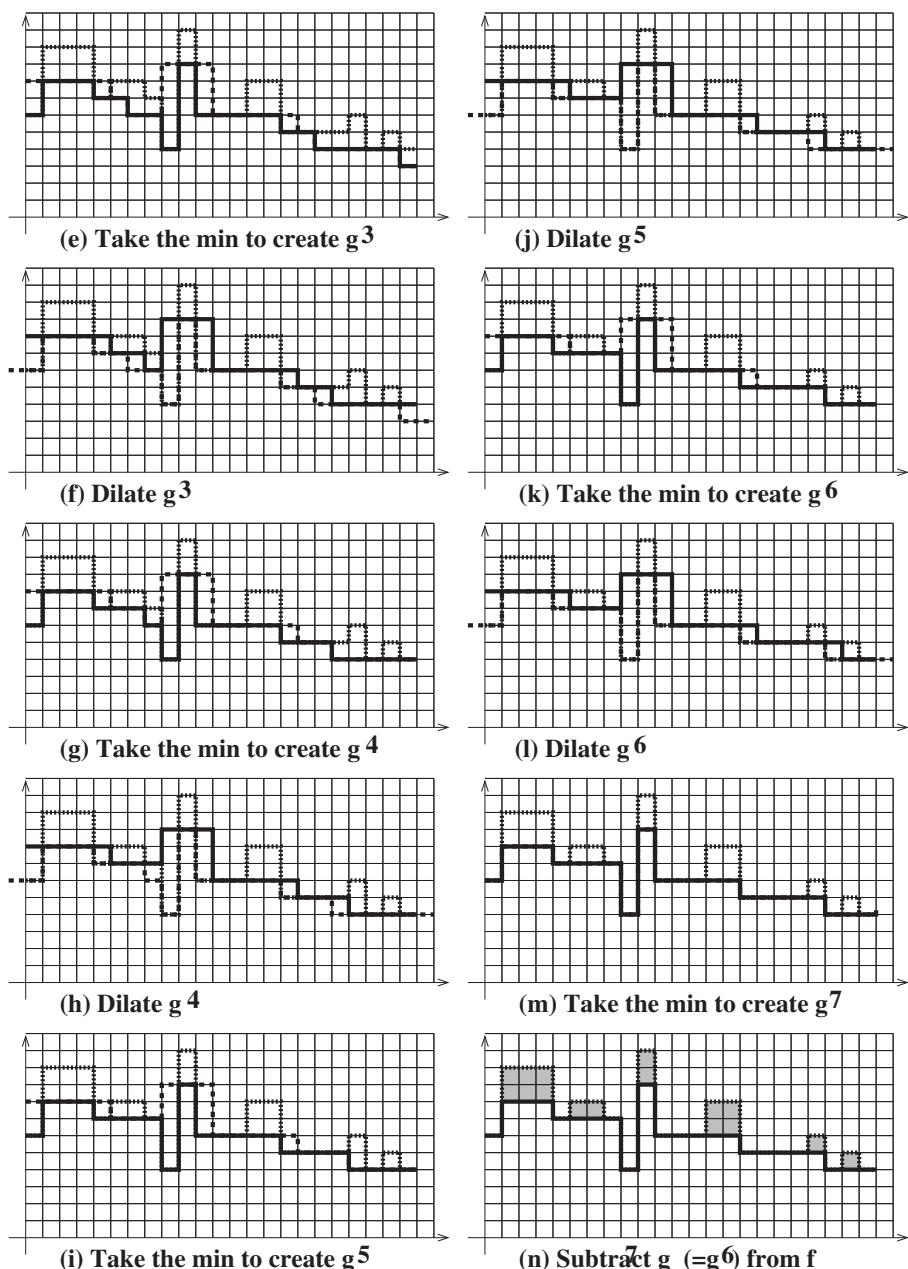


Figure 6.31: Continuation of the process of figure 6.30. Note that the identified maxima stick above their surroundings by less than or equal to 2.

Example 6.17

For the sequence of example 6.15, identify all local minima, by morphological reconstruction.

First, we construct sequence \tilde{g}^1 by adding 1 to every sample of the sequence. This is shown with the solid line in figure 6.32a. We then erode \tilde{g}^1 by taking the minimum of every three successive numbers. Note that before we do that, we extend the sequence by one sample in each direction, by repeating the first and the last sample, so that all samples have left and right neighbours. The result is shown in 6.32b, with the solid line. Next, take the maximum between the eroded version of \tilde{g}^1 and the original sequence, to produce \tilde{g}^2 . We repeat this process as shown in the panels of figure 6.32, until the new signal \tilde{g} we produce no longer changes. This happens at the fourth iteration. We then subtract the original sequence from \tilde{g}^3 . The samples that have nonzero values identify the local minima.

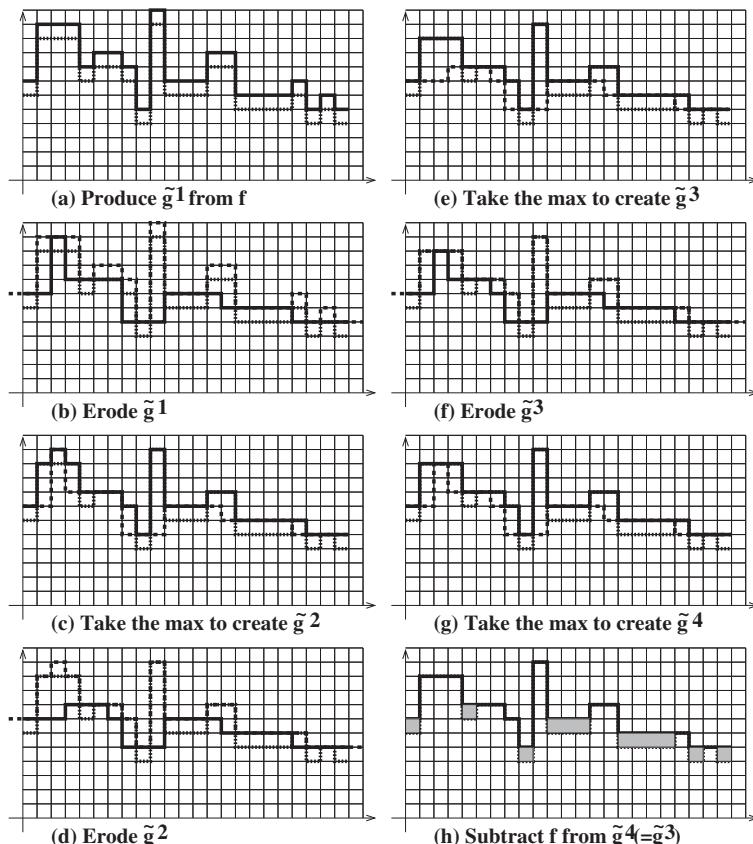


Figure 6.32: The sequence of steps required in order to apply formula (6.52) to identify the local minima in sequence f . In each panel, sequence f is shown with the dotted line. In each panel, the sequence that is being processed is shown with a dashed line and the result obtained with a continuous line.

Example 6.18

Use the local minima identified in example 6.17 to segment the sequence of example 6.15.

Figuratively speaking, we first assume that the local minima identified are “pierced” at their bottom. Then we start raising the level of water from the bottom and every time the water enters a minimum, a new “lake” is created, with its own name. The water level carries on increasing, until all lakes meet. In practice, we start from the value of the deepest minimum identified. All samples around the (equally deep) deepest minima are given some labels that identify these distinct minima. Let us say that this minimum value of the equally deep deepest minima is f_{min} . As shown in figure 6.33a we have for this sequence three deepest minima, with value 4. Each one is given its own label, identified in the figure with a different tone of grey. Then we raise the level of “water” by 1, and all samples with value 5 or less that are adjacent to an existing label inherit the value of this label (see figure 6.34). If a new local minimum is flooded this way, ie if a local minimum with value 5 exists, it is assigned a new label. Then we consider all samples with value 6: if they are adjacent to a labelled sample, they inherit the label of that sample; if not, a new label is created. This process carries on until no unlabelled samples are left. Note that samples that correspond to local maxima may be labelled on a first-come-first-serve basis, ie they may get the label of a neighbouring pixel at random. One, however, may be a little more sophisticated and leave those samples for the end. Then one may assign them labels so that they cause the minimum increase of the variance of the values of the region to which they are assigned. We sequentially consider assigning the pixel to each one of the possible regions it might belong to. For each case we estimate the change in the variance of the grey values of the region that will be created by incorporating the pixel. The region that will suffer the least disturbance by the incorporation of the pixel will bequest its label to the pixel. An alternative criterion would be simply to assign to a pixel the label of its neighbour that has the most similar grey value. Note that in this example, the sample with value 11 (the highest peak of the sequence) has been assigned the same label as the deepest trough of the sequence (the sample with value 4) by the ad-hoc approach. A more elaborate approach would clearly assign to this sample the label of its right neighbour.

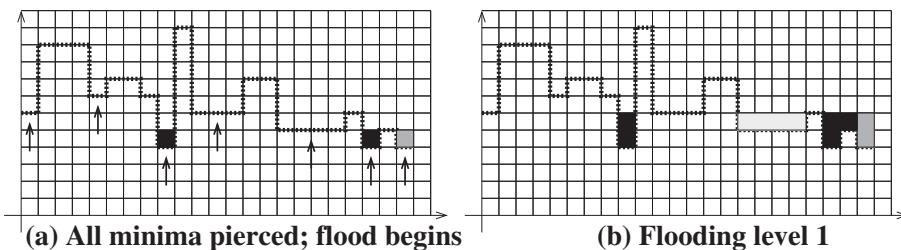


Figure 6.33: See caption of figure 6.34.

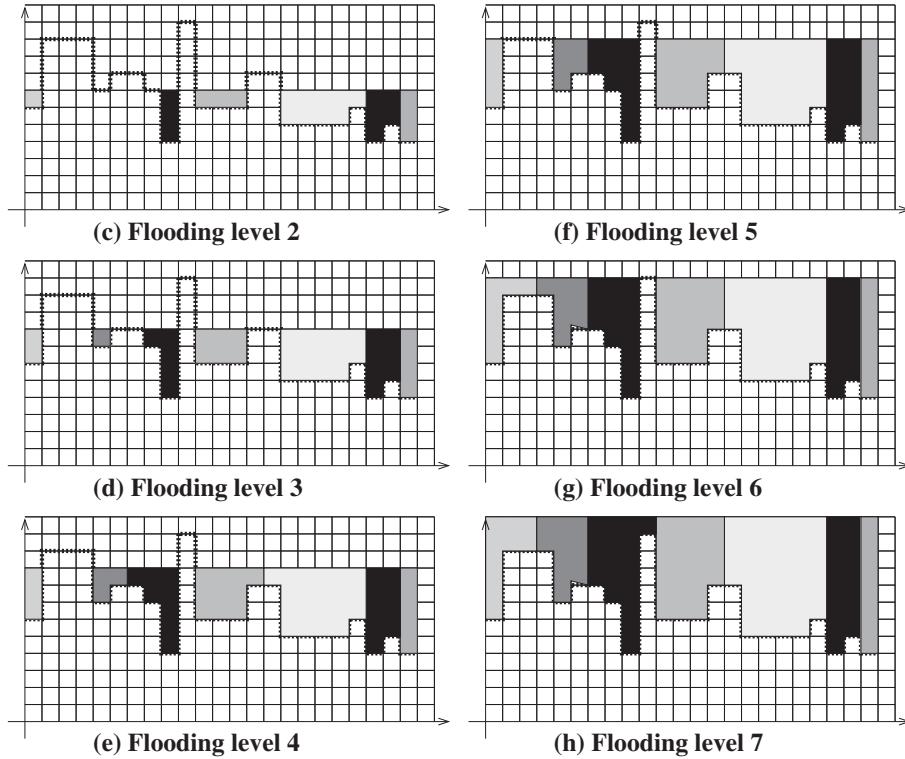


Figure 6.34: The sequence of steps required in order to segment the signal in 6.33a, with the watershed algorithm. The different tones of grey represent different labels.

How does the use of the gradient magnitude image help segment the image by the watershed algorithm?

The places where the magnitude of the gradient of the image is high are the places where we wish different regions of the image to meet. So, to prevent regions from growing inside the territory of neighbouring regions, we “erect” barriers at the places of high gradient magnitude, the so called **watersheds**. We also consider as seeds the minima of the gradient magnitude image, instead of the minima and maxima of the original grey image. The minima of the gradient magnitude usually correspond to flat patches of the image. These flat image patches could be either brighter or darker than their surroundings. Thus, working with the *gradient* minima as opposed to the *image* minima, we may segment simultaneously regions that are either brighter or darker than their surroundings. The algorithm is as follows.

Step 0: Create an empty output array o , the same size as the input image.

Step 1: Compute the gradient magnitude image w from the input image.

Step 2: By morphological reconstruction, identify the minima of the gradient magnitude image and use them as seeds from which regions will grow.

Step 4: In the output array, give a distinct label to each seed patch of pixels.

Step 5: Set the threshold at the minimum value of the gradient magnitude image, t_{min} .

Step 6: All pixels with value in w less or equal to the threshold, and which are adjacent to one of the labelled patches in o , and which have not yet been assigned a label, are given in o the label of the labelled patch to which they are adjacent.

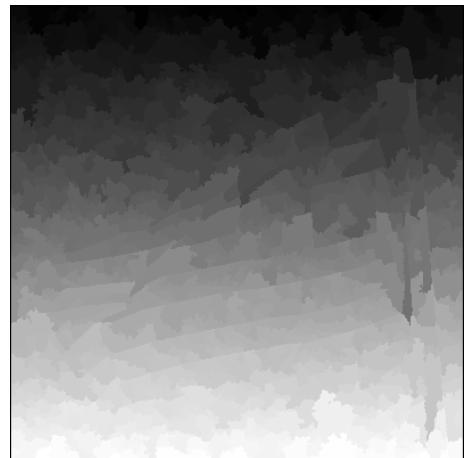
Step 7: Repeat Step 6 until no assignment of labels happens.

Step 8: If there are pixels still with no labels, increment the threshold by Δt and return to Step 6.

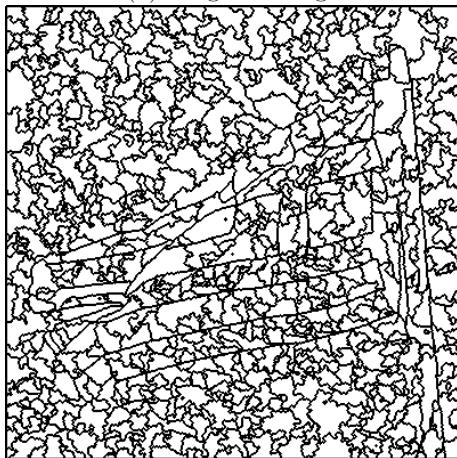
Figure 6.35 shows the result of applying this algorithm to an image. In general, the watershed algorithm creates image oversegmentation.



(a) Original image



(b) Labels of the segments shown as grey values



(c) Outlines of the segments



(d) Segments shown with their average grey value

Figure 6.35: The watershed algorithm applied to the Greek flag.

Are there any drawbacks in the watershed algorithm which works with the gradient magnitude image?

Yes. The gradient magnitude is usually very grossly estimated. This is particularly important for medical image processing, where small image structures may have to be segmented and minute image details preserved (see example 6.19). The situation may be improved if subpixel gradient estimation is performed (see example 6.20).

However, another issue is that the minima of the gradient magnitude do not necessarily correspond to peaks and troughs of the grey image (so that they can be used as seeds to allow regions to grow). The gradient magnitude becomes 0 at saddle points of the image too, ie at places where the image brightness obtains a maximum along one direction and an inflection along the orthogonal direction (see example 6.21).

A much better method for using the watershed algorithm is to identify the ridges and waterlines of the image first, with subpixel accuracy. The minima and maxima of the image are identified as the points where non-ascending and non-descending paths from neighbouring pixels end up, respectively. All pixels, that have a non-ascending path of a chain of pixels leading to a particular minimum, correspond to that minimum and form the so called **waterhole** of that minimum. Non-ascending path means that the path we follow, to move away from the pixel, either leads to a pixel with the same value, or moves to a pixel with a lower value. All pixels, that have a non-descending path of a chain of pixels leading to a particular maximum, correspond to that maximum and form the so called **hill** of that maximum. Non-descending path means that the path we follow, to move away from the pixel, either leads to a pixel with the same value, or moves to a pixel with a higher value. Methods that use these concepts to segment images are beyond the scope of this book.

Example 6.19

Figure 6.36a shows an image with a dark object on a bright background. Compute the first difference $\Delta_i f(i, j)$ at location (i, j) , by subtracting the value at location $(i - 1, j)$ from the value at location $(i + 1, j)$ and dividing by 2, and the first difference $\Delta_j f(i, j)$, by subtracting the value at location $(i, j - 1)$ from the value at location $(i, j + 1)$ and dividing by 2. Then, compute the gradient map of this image. Comment on whether the gradient map you create this way can allow you to erect watershed barriers to segment this image by using the watershed algorithm.

Figures 6.36c and 6.36d show the first differences computed for the pixels of this image, while figure 6.36b shows the gradient map computed from them, by using:

$$\text{Gradient_Magnitude} = \sqrt{(\Delta_i f(i, j))^2 + (\Delta_j f(i, j))^2} \quad (6.55)$$

The gradient values are high for several neighbouring pixels, and we cannot easily see how we may join the ridges of them to form the watersheds we need.

3	4	5	16	15	15	14	14
4	5	6	17	16	15	14	15
5	6	7	17	17	16	15	14
19	18	18	7	6	5	14	13
20	19	18	6	5	4	14	15
20	20	19	5	4	3	18	18
18	19	18	17	15	13	17	17
18	19	18	18	16	17	16	18

(a)

1.4	6.1	5.0	1.4	1.1	0.5	
6.6	8.1	7.1	5.0	5.1	1.0	
6.5	7.8	8.1	6.1	6.3	4.0	
1.4	6.5	6.6	1.4	4.6	5.9	
0.5	7.5	9.3	5.1	8.3	7.6	
0.5	1.1	6.7	6.3	7.1	2.2	

(b)

1	6	5	-1	-1	0	
1	5.5	5	-0.5	-1	-1	
-0.5	-5.5	-6	-1	4	4	
-1	-6.5	-6.5	-1	4.5	5.5	
-0.5	-7.5	-7.5	-1	7	7.5	
0	-1	-1.5	-2	1	2	

(c)

1	1	0.5	1	0.5	0.5	
6.5	6	-5	-5	-5	0	
6.5	5.5	-5.5	-6	-6	-0.5	
1	0.5	-1	-1	-1	2	
0	0	5.5	5	4.5	1.5	
-0.5	-0.5	6.5	6	7	-1	

(d)

Figure 6.36: (a) A grey image. The thick black line indicates the object depicted in this image. We would like to use the gradient information to identify a ridge that will allow us to separate the object from the background. (b) The gradient magnitude of the image. It is rather gross for the job we want to do. (c) The first differences along the vertical direction. (d) The first differences along the horizontal direction. The gradient magnitude shown in (b) is the square root of the sum of the squares of the first differences shown in (c) and (d).

Example 6.20

The first difference between two neighbouring pixels corresponds to the image derivative at the position between the two pixels. Create the so called dual grid of image 6.36a, by considering these positions, and estimate the gradient magnitude image for this image at the subpixel positions.

To create the dual grid of image 6.36a, we create cells between any two neighbouring pixels along the vertical and horizontal directions, where we assign interpolated grey image values between the adjacent pixels. Figure 6.37 shows this calculation. We then

use these values to compute the first differences of the image at the positions of the dual grid, that correspond to the empty cells in figure 6.37. The square root of the sum of the squares of these first differences gives the gradient magnitude at the points of the dual grid. This is shown in figure 6.38.

3	3.5	4	4.5	5	10.5	16	15.5	15	15	15	14.5	14	14	14
3.5		4.5		5.5		16.5		15.5		15		14		14.5
4	4.5	5	5.5	6	11.5	17	16.5	16	15.5	15	14.5	14	14.5	15
4.5		5.5		6.5		17		16.5		15.5		14.5		14.5
5	5.5	6	6.5	7	12	17	17	17	16.5	16	15.5	15	14.5	14
12		12		12.5		12		11.5		10.5		14.5		13.5
19	18.5	18	18	18	12.5	7	6.5	6	5.5	5	9.5	14	13.5	13
19.5		18.5		18		6.5		5.5		4.5		14		14
20	19.5	19	18.5	18	12	6	5.5	5	4.5	4	9	14	14.5	15
20		19.5		18.5		5.5		4.5		3.5		16		16.5
20	20	20	19.5	19	12	5	4.5	4	3.5	3	10.5	18	18	18
19		19.5		18.5		11		9.5		8		17.5		17.5
18	18.5	19	18.5	18	17.5	17	16	15	14	13	15	17	17	17
18		19		18		17.5		15.5		15		16.5		17.5
18	18.5	19	18.5	18	18	18	17	16	16.5	17	16.5	16	17	18

Figure 6.37: The interpolated values of image 6.36a at the interpixel positions.

3		4		5		16		15		15		14		14
	1.4		1.4		11		1.4		0.7		1		0.7	
4		5		6		17		16		15		14		15
	1.4		1.4		11.5		0.7		1.4		1.4		0	
5		6		7		17		17		16		15		14
	13		11.5		0.7		10.5		11.0		7.2		1.4	
19		18		18		7		6		5		14		13
	1.4		0.7		11.5		1.4		1.4		9.5		1	
20		19		18		6		5		4		14		15
	0.7		1.4		13		1.4		1.4		12.6		3.5	
20		20		19		5		4		3		18		18
	1.6		1.4		9.3		11.6		10.5		10.5		1	
18		19		18		17		15		13		17		17
	1		1		0.7		2.2		2.5		2.1		1	
18		19		18		18		16		17		16		18

Figure 6.38: The values of the gradient magnitude of image 6.36a at the positions of the dual grid. The pixels of the original image and their grey values are shown in the grey cells.

Example 6.21

Use the result of example 6.20 to identify the watersheds in image 6.36a. Identify also the minima of the gradient magnitude which may be used as seeds by the watershed algorithm.

By observing the gradient magnitude values in 6.38, we note that we have two distinct sets of values that may be identified by selecting thresholds 1 and 3. We consider values below or equal to 1 as corresponding to the minima of the gradient map and values above 3 as corresponding to the watersheds of the image.

The values that may be linked to form the watersheds are shown in figure 6.39 as black dots. The minima that may be used as seeds are shown as open circles. Note that one of these open circles, near the narrow neck of the object, does not correspond to either a minimum or a maximum of the image, but to a saddle point.

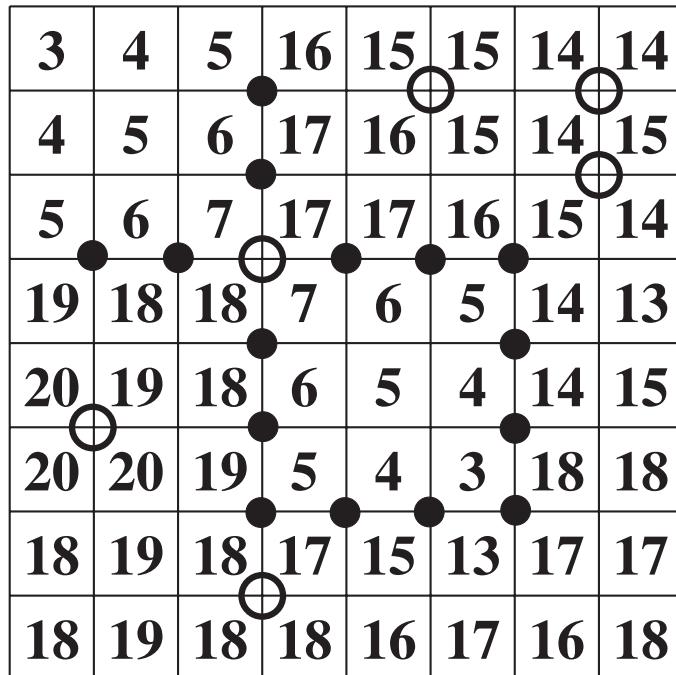


Figure 6.39: The dark dots correspond to interpixel positions with high gradient magnitude, appropriate for being joined to form watersheds that will stop one image region spilling into another. The open circles correspond to minima of the gradient magnitude and they may be used as seeds for the watershed algorithm.

Example B6.22

In image 6.36a, identify all non-descending paths starting from pixel (1, 1) that has grey value 3. Can you identify any non-ascending paths starting from this pixel?

We start from pixel (1, 1) and we examine all its neighbours in a 3×3 neighbourhood. We proceed to the pixel with the highest value. If more than one neighbours have the same value, we must consider all options. This creates the bifurcation of the path we follow. The result is shown in figure 6.40. Note that all branches of the non-descending path lead to the same patch of locally maximal values. If we construct similar paths from all pixels in the image, the pixels that have their paths leading to the same local maximum form the hill of that maximum.

There are no non-ascending paths starting from pixel (1, 1) as the value of this pixel is a local minimum. If we construct all non-ascending paths that start from each pixel in the image, the pixels that have their paths ending up to pixel (1, 1) constitute the water basin of this minimum.

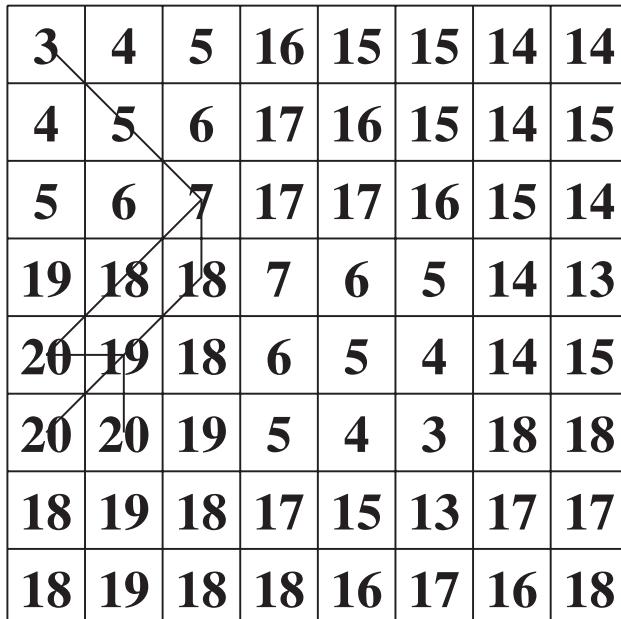


Figure 6.40: All non-descending paths that start from the top left corner pixel of this image.

Example B6.23

In image 6.36a identify all non-ascending paths starting from pixels (4, 1), (4, 2), (4, 3), (5, 1), (5, 2), (5, 3), (6, 1), (6, 2), (6, 3), (7, 1), (7, 2), (7, 3), (8, 1), (8, 2) and (8, 3). Pixels which have non-ascending paths to two different local minima form the ridges of the image. Identify which of these pixels belong to the ridges of the image.

All non-ascending paths, starting from the pixels mentioned, are marked in figure 6.41. The local minima, to which these paths end up, are pixels (1, 1), (6, 6) and (8, 1). Pixels which have non-ascending paths to more than one minima are highlighted in grey.

If we construct all non-descending paths, starting from all pixels in the image, the pixels, that have such paths ending to more than one local maxima, form the waterlines of the image.

3	4	5	16	15	15	14	14
4	5	6	17	16	15	14	15
5	6	7	17	17	16	15	14
19	18	18	7	6	5	14	13
20	19	18	6	5	4	14	15
20	20	19	5	4	3	18	18
18	19	18	17	15	13	17	17
18	19	18	18	16	17	16	18

Figure 6.41: The grey pixels have non-ascending paths that end up to two different local minima. These pixels form the ridges of the image. Note that neighbouring pixels (5, 1) and (6, 1) have non-ascending paths that end up to two different minima. One may postulate the existence of a ridge between these two pixels, at subpixel location. This subpixel ridge is marked with a grey horizontal box in the image.

Is it possible to segment an image by filtering?

Yes. The edge preserving mode filter we encountered in Chapter 4 (page 328), if applied repeatedly, produces a segmentation. It turns out that only the edge adaptive version of the filter gives acceptable results (page 385). An example is shown in figure 6.42, where the weighted mode filter with weights

$$\begin{pmatrix} 1 & 3 & 1 \\ 3 & 5 & 3 \\ 1 & 3 & 1 \end{pmatrix} \quad (6.56)$$

was used 134 times. This filter is very slow.



(a) Original image



(b) Edge adaptive mode filtering

Figure 6.42: The edge preserving mode algorithm used to segment “Leonidas”.

Another type of filtering is the **mean shift** filtering, we also saw in Chapter 4, used for smoothing (see pages 339 and 385). The smooth image produced by this algorithm may be segmented by agglomerating all segments which contain representative pixels that are within a certain distance from each other in the spatio-brightness domain.

How can we use the mean shift algorithm to segment an image?

The algorithm described on page 339 was shifting all pixels at each iteration, thus dynamically changing the feature space. It converges when all pixels belong to a single cluster. So, that version of the algorithm has to be run for a prespecified number of iterations. At the end the user has to select the desired result. However, one may also run the algorithm while keeping the feature space fixed: each time we concentrate on one pixel only, updating its values until they reach a fixed point. This fixed point is the nearest mode of the static feature space, ie the centre of the nearest agglomeration of pixels in the feature space. Once the final values of this pixel have been identified, the next pixel is considered, again working in the original unchanged feature space, where all pixels have their original values. In this version

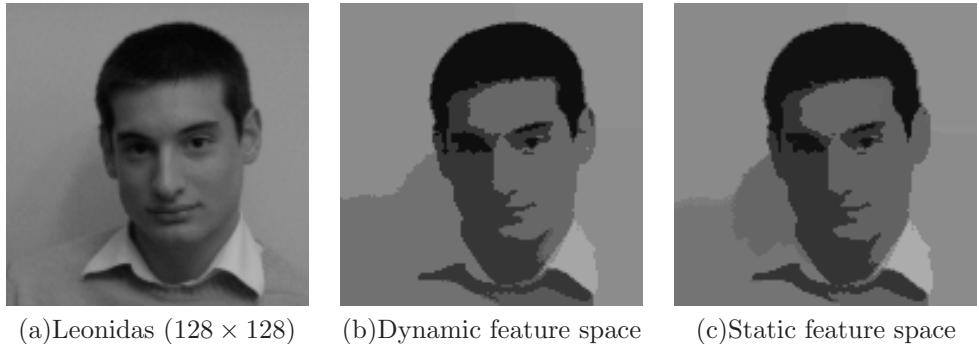


Figure 6.43: Leonidas segmented by mean shift. (b) This result was obtained after 80 iterations, using $h_x = 15$, $h_y = 15$ and $h_g = 0.05$. (The grey values were in the range $[0, 1]$.) The triplets of all pixels in the feature space were updated at each iteration. (c) As in (b), but now trajectories of pixels in the static feature space were computed. Each trajectory was considered as having reached its fixed point when the value of (6.57) was less than or equal to 0.0001.

of the algorithm, we compute trajectories of pixels that lead to the nearest cluster centre. Each trajectory is disregarded at the end, and what matters only is the final triplet of values associated with the pixel. The convergence of each trajectory is judged by comparing two successive triplets of values, using their distance, computed as

$$\frac{(x_{ij}^{m+1} - x_{ij}^m)^2}{h_x^2} + \frac{(y_{ij}^{m+1} - y_{ij}^m)^2}{h_y^2} + \frac{(g_{ij}^{m+1} - g_{ij}^m)^2}{h_g^2} \quad (6.57)$$

where $(x_{ij}^{m+1}, y_{ij}^{m+1}, g_{ij}^{m+1})$ and $(x_{ij}^m, y_{ij}^m, g_{ij}^m)$ are the triplets of values associated with pixel (i, j) at iterations $m+1$ and m , respectively. This distance value is compared with a threshold th to assess convergence of the values of pixel (i, j) .

After the mean shift algorithm has been run, either with a dynamic or a static feature space, every pixel is associated with a vector $(x_{ij}^{m_{final}}, y_{ij}^{m_{final}}, g_{ij}^{m_{final}})$. However, there are only a few such distinct vectors in comparison with the original number of pixels. Each such vector is represented by a point in the 3D space of position-grey value. Let us say that we have M such distinct vectors, each one linked to a large number of pixels. Let us call them $\mathbf{z}_m \equiv (x_m, y_m, g_m)$ for $i = 1, 2, \dots, M$. We may create a double entry table of size $M \times M$, where the element at position (m, n) is 1, if

$$\begin{aligned} |x_m - x_n| &\leq h_x \\ |y_m - y_n| &\leq h_y \\ |g_m - g_n| &\leq h_g \end{aligned} \quad (6.58)$$

and it is 0 if the above conditions do not hold simultaneously. Then, we look for connected regions made up of 1s in this array (“islands” of 1s). All pixels, that are linked to all vectors \mathbf{z}_m that make up the same island, receive the same label. This way the segmented image is formed. Figure 6.43 shows an example of applying this algorithm.

Treating pixels as triplets of values frees us from the rectangular grid and allows us to think of representing an image by a **graph**.

What is a graph?

A graph is a mathematical construction that consists of a set of nodes, called **vertices**, and a set of **edges**, ie lines that join vertices that are related according to some relationship we specify. If every node is connected to every other node, the graph is said to be **fully connected**. If the edges represent relationships that are symmetric, ie the order of the connected nodes does not matter, then the graph is called **undirected**. We often assign *weights* to the edges to quantify the relationship that exists between the joined nodes.

How can we use a graph to represent an image?

We can consider each pixel to be a node. We may connect every pixel to every other pixel and assign as weight to the connection the inverse of the absolute grey value difference of the two connected pixels. These weights then measure the *similarity* between the two pixels. This way the image is represented by an undirected **relational graph**. To be consistent with the mean shift algorithm, we may define the similarity to be the inverse of the weighted sum of the distance and the square grey value difference of the two pixels.

How can we use the graph representation of an image to segment it?

We may start by removing the weakest links of the graph, ie disconnect pixels with the least similarity in order to form disjoint sets of pixels. To do that in a systematic and unbiased way, we may use the **normalised cuts algorithm**.

What is the normalised cuts algorithm?

Let us assume that the set of vertices V of a graph is divided into two subsets, A and B . We define the **cut** of sets A and B as

$$\text{cut}(A, B) \equiv \sum_{u \in A, v \in B} w(u, v) \quad (6.59)$$

where $w(u, v)$ is the weight of the edge that joins nodes u and v . We then define the **normalised cut** between sets A and B as:

$$\text{ncut}(A, B) \equiv \frac{\text{cut}(A, B)}{\text{cut}(A, V)} + \frac{\text{cut}(A, B)}{\text{cut}(B, V)} \quad (6.60)$$

The normalised cuts algorithm divides the pixels of the image into the two subsets A and B that minimise $\text{ncut}(A, B)$.

Box 6.3. The normalised cuts algorithm as an eigenvalue problem

Let us define the total connection a node V_i of a graph has to all other nodes, as $d_i \equiv \sum_j w_{ij}$, where w_{ij} is the weight between nodes V_i and V_j . This is called the

degree of the node. Also, define a vector \mathbf{x} , such that its element x_i has value 1, if node V_i belongs to subset A , and it has value -1 , if V_i belongs to subset B . We may then express $ncut(A, B)$, defined by (6.60), as:

$$ncut(A, B) = \frac{\sum_{x_i=1, x_j=-1} w_{ij}}{\sum_{x_i=1} d_i} + \frac{\sum_{x_i=-1, x_j=1} w_{ij}}{\sum_{x_i=-1} d_i} \quad (6.61)$$

Note that the problem we are trying to solve is to identify vector \mathbf{x} , because if we know vector \mathbf{x} , we know which pixel (node of the graph) belongs to set A and which does not belong to set A . So, we may denote function $ncut(A, B)$ as $C(\mathbf{x})$, which expresses the cost function we wish to minimise, as a function of the unknown vector \mathbf{x} .

Let us denote by $\mathbf{1}$ a vector with all its elements equal to 1. Note that an element of vector $(\mathbf{1} + \mathbf{x})/2$ will be 1, if the corresponding node belongs to set A , and it will be 0, if it does not. Similarly, an element of vector $(\mathbf{1} - \mathbf{x})/2$ will be 1, if the corresponding node belongs to set B , and 0 otherwise. It can be shown then (see examples 6.24 and 6.25), that (6.61) may be written as

$$C(\mathbf{x}) = \frac{(\mathbf{1} + \mathbf{x})^T L (\mathbf{1} + \mathbf{x})}{4k \mathbf{1}^T D \mathbf{1}} + \frac{(\mathbf{1} - \mathbf{x})^T L (\mathbf{1} - \mathbf{x})}{4(1-k) \mathbf{1}^T D \mathbf{1}} \quad (6.62)$$

where D is a diagonal matrix with values d_i along its diagonal, $L \equiv D - W$ with W being the matrix made up from weights w_{ij} , and:

$$k \equiv \frac{\sum_{x_i=1} d_i}{\sum_i d_i} \quad (6.63)$$

Matrix L is known as the **Laplacian matrix** of the graph.

Obviously:

$$C(\mathbf{x}) = \frac{(1-k)(\mathbf{1} + \mathbf{x})^T L (\mathbf{1} + \mathbf{x}) + k(\mathbf{1} - \mathbf{x})^T L (\mathbf{1} - \mathbf{x})}{4k(1-k) \mathbf{1}^T D \mathbf{1}} \quad (6.64)$$

Let us divide numerator and denominator with $(1-k)^2$:

$$C(\mathbf{x}) = \frac{\frac{1}{1-k}(\mathbf{1} + \mathbf{x})^T L (\mathbf{1} + \mathbf{x}) + \frac{k}{(1-k)^2}(\mathbf{1} - \mathbf{x})^T L (\mathbf{1} - \mathbf{x})}{4 \frac{k}{1-k} \mathbf{1}^T D \mathbf{1}} \quad (6.65)$$

Define:

$$b \equiv \frac{k}{1-k} \Rightarrow k = \frac{b}{1+b} \Rightarrow \frac{1}{1-k} = 1+b \quad \text{and} \quad \frac{k}{(1-k)^2} = b(1+b) \quad (6.66)$$

If we substitute from (6.66) into (6.65), we obtain:

$$C(\mathbf{x}) = \frac{(1+b)(\mathbf{1} + \mathbf{x})^T L (\mathbf{1} + \mathbf{x}) + b(1+b)(\mathbf{1} - \mathbf{x})^T L (\mathbf{1} - \mathbf{x})}{4b \mathbf{1}^T D \mathbf{1}} \quad (6.67)$$

Term $\mathbf{1}^T L \mathbf{1}$ is 0. This is because $L \mathbf{1}$ means nothing else than summing all elements along the same row of matrix L . We know that the diagonal element in each row is the

sum of the weights of all the edges that start from the corresponding node, while all other elements are the weights from that node to each other node, with a negative sign (see example 6.26). Term $\mathbf{1}^T L \mathbf{1}$ then may change sign as convenient. This allows us to write (see example 6.27):

$$C(\mathbf{x}) = \frac{[\mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x})]^T L [\mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x})]}{4b\mathbf{1}^T D \mathbf{1}} \quad (6.68)$$

It can be shown (see example 6.32) that $4b\mathbf{1}^T D \mathbf{1} = \mathbf{y}^T D \mathbf{y}$, where \mathbf{y} is:

$$\mathbf{y} \equiv \mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x}) \quad (6.69)$$

Then cut $C(\mathbf{x})$ is given by:

$$C(\mathbf{x}) = \frac{\mathbf{y}^T L \mathbf{y}}{\mathbf{y}^T D \mathbf{y}} \quad (6.70)$$

This is known as the **Rayleigh quotient**. The problem is then to identify a vector \mathbf{x} , which determines which node of the graph belongs to set A and which not, so that the Rayleigh quotient $C(\mathbf{x})$ is minimised.

Example B6.24

Consider the fully connected graph of figure 6.44. It consists of nodes $V = \{V_1, V_2, V_3, V_4\}$. Set A consists of nodes $A = \{V_1, V_4\}$, while set B consists of nodes $B = \{V_2, V_3\}$. The weights associated with each edge are marked on the figure. Compute the normalised cut between sets A and B , using definition (6.61).

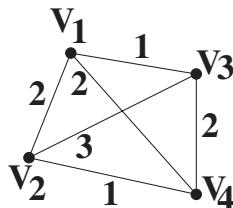


Figure 6.44: A fully connected graph.

Vector \mathbf{x} obviously is $\mathbf{x}^T = (1, -1, -1, 1)$. From (6.61) then:

$$\begin{aligned} ncut(A, B) &= \frac{w_{12} + w_{13} + w_{42} + w_{43}}{d_1 + d_4} + \frac{w_{21} + w_{24} + w_{31} + w_{34}}{d_2 + d_3} \\ &= \frac{6}{10} + \frac{6}{12} = \frac{11}{10} \end{aligned} \quad (6.71)$$

Example B6.25

Compute the normalised cut of sets A and B of example 6.24, using formula (6.62).

From figure 6.44 we work out matrices D and W , and from them matrix L :

$$D = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix} \quad W = \begin{pmatrix} 0 & 2 & 1 & 2 \\ 2 & 0 & 3 & 1 \\ 1 & 3 & 0 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix} \quad (6.72)$$

Then:

$$L = D - W = \begin{pmatrix} 5 & -2 & -1 & -2 \\ -2 & 6 & -3 & -1 \\ -1 & -3 & 6 & -2 \\ -2 & -1 & -2 & 5 \end{pmatrix} \quad (6.73)$$

Parameter k , defined by (6.63), is:

$$k = \frac{d_1 + d_4}{d_1 + d_2 + d_3 + d_4} = \frac{5 + 5}{5 + 6 + 6 + 5} = \frac{10}{22} = \frac{5}{11} \quad (6.74)$$

Vectors $\mathbf{1} + \mathbf{x}$ and $\mathbf{1} - \mathbf{x}$ are:

$$\begin{aligned} \mathbf{1} + \mathbf{x} &= \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 0 \\ 2 \end{pmatrix} \\ \mathbf{1} - \mathbf{x} &= \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 0 \end{pmatrix} \end{aligned} \quad (6.75)$$

Then we can compute the various quantities that appear in (6.62):

$$\begin{aligned} (\mathbf{1} + \mathbf{x})^T L (\mathbf{1} + \mathbf{x}) &= (2 \ 0 \ 0 \ 2) \begin{pmatrix} 5 & -2 & -1 & -2 \\ -2 & 6 & -3 & -1 \\ -1 & -3 & 6 & -2 \\ -2 & -1 & -2 & 5 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \\ 0 \\ 2 \end{pmatrix} \\ &= (2 \ 0 \ 0 \ 2) \begin{pmatrix} 6 \\ -6 \\ -6 \\ 6 \end{pmatrix} = 24 \end{aligned} \quad (6.76)$$

$$\begin{aligned}
 (\mathbf{1} - \mathbf{x})^T L (\mathbf{1} - \mathbf{x}) &= \begin{pmatrix} 0 & 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} 5 & -2 & -1 & -2 \\ -2 & 6 & -3 & -1 \\ -1 & -3 & 6 & -2 \\ -2 & -1 & -2 & 5 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ 2 \\ 0 \end{pmatrix} \\
 &= \begin{pmatrix} 0 & 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} -6 \\ 6 \\ 6 \\ -6 \end{pmatrix} = 24 \\
 \mathbf{1}^T D \mathbf{1} &= \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 6 \\ 5 \end{pmatrix} = 22 \tag{6.77}
 \end{aligned}$$

Substituting in (6.62), we obtain:

$$ncut(A, B) = \frac{24}{4 \times \frac{10}{22} \times 22} + \frac{24}{4 \times (1 - \frac{10}{22}) \times 22} = \frac{11}{10} \tag{6.78}$$

Example B6.26

For the graph of example 6.24 compute scalars $\mathbf{x}^T L \mathbf{1}$, $\mathbf{1}^T L \mathbf{x}$ and $\mathbf{1}^T L \mathbf{1}$.

For this graph, vector \mathbf{x} is $\mathbf{x}^T = (1, -1, -1, 1)$ and matrix L is given by (6.73). We have then:

$$\begin{aligned}
 \mathbf{x}^T L \mathbf{1} &= \begin{pmatrix} 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 5 & -2 & -1 & -2 \\ -2 & 6 & -3 & -1 \\ -1 & -3 & 6 & -2 \\ -2 & -1 & -2 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 0 \\
 \mathbf{1}^T L \mathbf{x} &= \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 5 & -2 & -1 & -2 \\ -2 & 6 & -3 & -1 \\ -1 & -3 & 6 & -2 \\ -2 & -1 & -2 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 6 \\ -6 \\ -6 \\ 6 \end{pmatrix} = 0 \\
 \mathbf{1}^T L \mathbf{1} &= \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 5 & -2 & -1 & -2 \\ -2 & 6 & -3 & -1 \\ -1 & -3 & 6 & -2 \\ -2 & -1 & -2 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 0 \tag{6.79}
 \end{aligned}$$

Example B6.27

Write the numerator of the right-hand side of equation (6.67) in the form $\mathbf{y}^T L \mathbf{y}$, where \mathbf{y} is some vector.

Let us expand the numerator of (6.67):

$$\begin{aligned}
 & (1+b)(\mathbf{1} + \mathbf{x})^T L(\mathbf{1} + \mathbf{x}) + b(1+b)(\mathbf{1} - \mathbf{x})^T L(\mathbf{1} - \mathbf{x}) = \\
 & (\mathbf{1} + \mathbf{x})^T L(\mathbf{1} + \mathbf{x}) + b(\mathbf{1} + \mathbf{x})^T L(\mathbf{1} + \mathbf{x}) + b(\mathbf{1} - \mathbf{x})^T L(\mathbf{1} - \mathbf{x}) + b^2(\mathbf{1} - \mathbf{x})^T L(\mathbf{1} - \mathbf{x}) = \\
 & \quad (\mathbf{1} + \mathbf{x})^T L(\mathbf{1} + \mathbf{x}) + b^2(\mathbf{1} - \mathbf{x})^T L(\mathbf{1} - \mathbf{x}) + \\
 & \quad b[\underbrace{\mathbf{1}^T L \mathbf{1} + \mathbf{1}^T L \mathbf{x} + \mathbf{x}^T L \mathbf{1} + \mathbf{x}^T L \mathbf{x}}_{\equiv S} + \mathbf{1}^T L \mathbf{x} - \mathbf{x}^T L \mathbf{1} + \mathbf{x}^T L \mathbf{x}]
 \end{aligned} \tag{6.80}$$

As $\mathbf{1}^T L \mathbf{1} = 0$, we can change the sign of these terms inside the square bracket. For those terms then, we have:

$$S = \underbrace{-\mathbf{1}^T L \mathbf{1}}_a + \underbrace{\mathbf{1}^T L \mathbf{x}}_b + \underbrace{\mathbf{x}^T L \mathbf{1}}_c + \underbrace{\mathbf{x}^T L \mathbf{x}}_c - \underbrace{\mathbf{1}^T L \mathbf{x}}_d - \underbrace{\mathbf{1}^T L \mathbf{x}}_a - \underbrace{\mathbf{x}^T L \mathbf{1}}_d + \underbrace{\mathbf{x}^T L \mathbf{x}}_b \tag{6.81}$$

We combine together terms marked with the same letter:

$$S = \underbrace{-\mathbf{1}^T L(\mathbf{1} + \mathbf{x})}_e + \underbrace{(\mathbf{1} + \mathbf{x})^T L \mathbf{x}}_f + \underbrace{\mathbf{x}^T L(\mathbf{1} + \mathbf{x})}_e - \underbrace{(\mathbf{1} + \mathbf{x})^T L \mathbf{1}}_f \tag{6.82}$$

We again combine terms marked with the same letter:

$$S = -(\mathbf{1} - \mathbf{x})^T L(\mathbf{1} + \mathbf{x}) - (\mathbf{1} + \mathbf{x})^T L(\mathbf{1} - \mathbf{x}) \tag{6.83}$$

When we substitute S in (6.80), we obtain the numerator of (6.67) as:

$$\begin{aligned}
 & \underbrace{(\mathbf{1} + \mathbf{x})^T L(\mathbf{1} + \mathbf{x})}_g + \underbrace{b^2(\mathbf{1} - \mathbf{x})^T L(\mathbf{1} - \mathbf{x})}_h - \underbrace{b(\mathbf{1} - \mathbf{x})^T L(\mathbf{1} + \mathbf{x})}_g - \underbrace{b(\mathbf{1} + \mathbf{x})^T L(\mathbf{1} - \mathbf{x})}_h = \\
 & [\mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x})]^T L(\mathbf{1} + \mathbf{x}) - b[\mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x})]^T L(\mathbf{1} - \mathbf{x}) = \\
 & [\mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x})]^T L[\mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x})]
 \end{aligned} \tag{6.84}$$

We note that the numerator of (6.67) may be written in the form $\mathbf{y}^T L \mathbf{y}$, if we define vector \mathbf{y} as:

$$\mathbf{y} \equiv \mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x}) \tag{6.85}$$

Example B6.28

For vector \mathbf{y} , defined by (6.85), and matrix D , defined in Box 6.3, show that $\mathbf{y}^T D \mathbf{1} = 0$.

First, we observe that $D\mathbf{1}$ is nothing other than a vector with the d_i values of the nodes of the graph. When we then multiply it from the left with vector $(\mathbf{1} + \mathbf{x})/2$, which has as nonzero elements those that identify the nodes that belong to set A , we get $\sum_{x_i=1} d_i$. When we multiply it from the left with vector $(\mathbf{1} - \mathbf{x})/2$, which has as nonzero elements those that identify the nodes that do not belong to set A , we get $\sum_{x_i=-1} d_i$.

Also, from the definition of b (equation (6.66), on page 577) and the definition of k (equation (6.63)), we obtain:

$$b \equiv \frac{k}{1-k} = \frac{\sum_{x_i=1} d_i}{\sum_{x_i=-1} d_i} \quad (6.86)$$

We make use of all these observations to compute $\mathbf{y}^T D \mathbf{1}$, as follows:

$$\begin{aligned} \mathbf{y}^T D \mathbf{1} &= [\mathbf{1} + \mathbf{x} - b(\mathbf{1} - \mathbf{x})]^T D \mathbf{1} \\ &= (\mathbf{1} + \mathbf{x})^T D \mathbf{1} - b(\mathbf{1} - \mathbf{x})^T D \mathbf{1} \\ &= 2 \sum_{x_i=1} d_i - 2b \sum_{x_i=-1} d_i \\ &= 2 \sum_{x_i=1} d_i - 2 \frac{\sum_{x_i=1} d_i}{\sum_{x_i=-1} d_i} \sum_{x_i=-1} d_i = 0 \end{aligned} \quad (6.87)$$

Example B6.29

For the graph of example 6.24, compute $\mathbf{y}^T D \mathbf{1}$.

To compute vector \mathbf{y} , as defined by (6.85), we need first the value of b . Parameter k in example 6.25 was computed as $5/11$. Then from definition (6.66) we deduce that $b = (5/11)/(6/11) = 5/6$. We may then compute vector \mathbf{y} , using definition (6.85), as:

$$\mathbf{y} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} - \frac{5}{6} \times \left[\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \right] = \begin{pmatrix} 2 \\ 0 \\ 0 \\ 2 \end{pmatrix} - \frac{5}{6} \times \begin{pmatrix} 0 \\ 2 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ -5/3 \\ -5/3 \\ 2 \end{pmatrix} \quad (6.88)$$

Using matrix D , as given by (6.73), we are then ready to compute $\mathbf{y}^T D \mathbf{1}$:

$$\mathbf{y}^T D \mathbf{1} = \begin{pmatrix} 2 & -\frac{5}{3} & -\frac{5}{3} & 2 \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 & -\frac{5}{3} & -\frac{5}{3} & 2 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 6 \\ 5 \end{pmatrix} = 0 \quad (6.89)$$

Example B6.30

For the graph of example 6.24 calculate vector $D\mathbf{x}$ and from that the values of $\mathbf{1}^T D\mathbf{x}$ and $\mathbf{x}^T D\mathbf{x}$. Generalise your results for any graph.

$$D\mathbf{x} = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ -6 \\ -6 \\ 5 \end{pmatrix} \quad (6.90)$$

Then:

$$\begin{aligned} \mathbf{1}^T D\mathbf{x} &= (1 \ 1 \ 1 \ 1) \begin{pmatrix} 5 \\ -6 \\ -6 \\ 5 \end{pmatrix} = 5 - 6 - 6 + 5 = -2 \\ \mathbf{x}^T D\mathbf{x} &= (1 \ -1 \ -1 \ 1) \begin{pmatrix} 5 \\ -6 \\ -6 \\ 5 \end{pmatrix} = 5 + 6 + 6 + 5 = 22 \end{aligned} \quad (6.91)$$

We notice that operation with vector $\mathbf{1}$ from the left on vector $D\mathbf{x}$ effectively sums the d_i values of all the nodes, that belong to set A , and subtracts from them the sum of the d_i values of all the nodes, that do not belong to set A . On the contrary, operation with vector \mathbf{x} from the left on vector $D\mathbf{x}$ effectively sums up all the d_i values. We conclude that:

$$\begin{aligned} \mathbf{1}^T D\mathbf{x} &= \sum_{x_i=1} d_i - \sum_{x_i=-1} d_i \\ \mathbf{x}^T D\mathbf{x} &= \sum_i d_i = \sum_{x_i=1} d_i + \sum_{x_i=-1} d_i \end{aligned} \quad (6.92)$$

Example B6.31

Show that:

$$\mathbf{1}^T D \mathbf{1} = \sum_i d_i \quad (6.93)$$

$$L \mathbf{1} = \mathbf{0} \quad (6.94)$$

Let us assume that the graph has N nodes.

$$\begin{aligned} \mathbf{1}^T D \mathbf{1} &= (1 \ 1 \ \dots \ 1) \begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & d_N \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ \dots \\ 1 \end{pmatrix} \\ &= (1 \ 1 \ \dots \ 1) \begin{pmatrix} d_1 \\ d_2 \\ \dots \\ d_N \end{pmatrix} = \sum_{i=1}^N d_i \end{aligned} \quad (6.95)$$

$$\begin{aligned} L \mathbf{1} &= \begin{pmatrix} d_1 & -w_{12} & \dots & -w_{1N} \\ -w_{21} & d_2 & \dots & -w_{2N} \\ \dots & \dots & \dots & \dots \\ -w_{N1} & -w_{N2} & \dots & d_N \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ \dots \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} d_1 - \sum_{i=2}^N w_{iN} \\ d_2 - w_{21} - \sum_{i=3}^N w_{iN} \\ \dots \\ d_N - \sum_{i=1}^{N-1} w_{iN} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \end{pmatrix} = \mathbf{0} \end{aligned} \quad (6.96)$$

Example B6.32

For vector \mathbf{y} , defined by (6.85), and matrix D , defined in Box 6.3, show that $\mathbf{y}^T D \mathbf{y} = 4b \mathbf{1}^T D \mathbf{1}$.

Let us calculate $\mathbf{y}^T D \mathbf{y}$, where we substitute only vector \mathbf{y} on the right, from its definition (6.85):

$$\mathbf{y}^T D \mathbf{y} = \underbrace{\mathbf{y}^T D \mathbf{1}}_{=0} + \underbrace{\mathbf{y}^T D \mathbf{x}}_{=0} - b \underbrace{\mathbf{y}^T D \mathbf{1}}_{=0} + b \mathbf{y}^T D \mathbf{x} \quad (6.97)$$

Here we made use of the result of example 6.28. Let us compute $\mathbf{y}^T D \mathbf{x}$:

$$\mathbf{y}^T D \mathbf{x} = \mathbf{1}^T D \mathbf{x} + \mathbf{x}^T D \mathbf{x} - b \mathbf{1}^T D \mathbf{x} + b \mathbf{x}^T D \mathbf{x} \quad (6.98)$$

If we make use of (6.92), we obtain:

$$\begin{aligned} \mathbf{y}^T D \mathbf{x} &= \sum_{x_i=1} d_i - \sum_{x_i=-1} d_i + \sum_i d_i - b \sum_{x_i=1} d_i + b \sum_{x_i=-1} d_i + b \sum_i d_i \\ &= 2 \sum_{x_i=1} d_i + 2b \sum_{x_i=-1} d_i \end{aligned} \quad (6.99)$$

Let us substitute now in (6.97):

$$\begin{aligned} \mathbf{y}^T D \mathbf{y} &= 2 \sum_{x_i=1} d_i + 2b \underbrace{\sum_{x_i=-1} d_i}_{=\sum_{x_i=1} d_i} + 2b \underbrace{\sum_{x_i=1} d_i}_{=b \sum_{x_i=-1} d_i} + 2b^2 \sum_{x_i=-1} d_i \\ &= 2 \sum_{x_i=1} d_i + 2 \sum_{x_i=1} d_i + 2b^2 \sum_{x_i=-1} d_i + 2b^2 \sum_{x_i=-1} d_i \\ &= 4 \underbrace{\sum_{x_i=1} d_i}_{=b \sum_{x_i=-1} d_i} + 4b^2 \sum_{x_i=-1} d_i \\ &= 4b \left[\underbrace{\sum_{x_i=-1} d_i}_{=\sum_{x_i=1} d_i} + b \underbrace{\sum_{x_i=-1} d_i}_{=\sum_{x_i=1} d_i} \right] \\ &= 4b \left[\sum_{x_i=-1} d_i + \sum_{x_i=1} d_i \right] = 4b \sum_i d_i \end{aligned} \quad (6.100)$$

According to (6.95), $\sum_i d_i = \mathbf{1}^T D \mathbf{1}$, so the proof is completed.

Box 6.4. How do we minimise the Rayleigh quotient?

It is obvious from (6.70) that if we manage to find a number λ such that

$$L \mathbf{y} = \lambda D \mathbf{y} \quad (6.101)$$

then by substituting in the numerator of (6.70), we shall have $C(\mathbf{x}) = \lambda$. If we select λ to be minimum, the corresponding vector \mathbf{y} (and by extension the corresponding vector \mathbf{x}) will be the solution of our minimisation problem. First of all, we must write (6.101) as a usual eigenvalue equation, where a matrix multiplies a vector that yields the vector

itself times a scalar. To do that, we multiply both sides of this expression with $D^{-\frac{1}{2}}$. This is possible because D is a diagonal matrix with positive values along its diagonal, and so taking its inverse square root is trivial (see also example 2.3, on page 51):

$$D^{-\frac{1}{2}}Ly = \lambda D^{\frac{1}{2}}y \quad (6.102)$$

Let us define vector $\mathbf{z} \equiv D^{\frac{1}{2}}y$. Then obviously $y = D^{-\frac{1}{2}}\mathbf{z}$ and we have:

$$D^{-\frac{1}{2}}LD^{-\frac{1}{2}}\mathbf{z} = \lambda\mathbf{z} \quad (6.103)$$

It can be shown that matrix $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ does not have negative eigenvalues (see examples 6.33, 6.34 and 6.35). Its smallest eigenvalue then must be 0. This eigenvalue, however, corresponds to a vector $\mathbf{x} = \mathbf{1}$, ie a vector that does not divide the set of nodes into two subsets (see example 6.36). Then it is obvious that the eigenvector we require is the smallest nonzero eigenvector of matrix $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$. This is known as the **Fiedler vector**.

Example B6.33

A symmetric matrix A is called positive semidefinite if $\mathbf{x}^T A \mathbf{x} \geq 0$ for every real vector \mathbf{x} . For a graph that consists of three nodes, show that its Laplacian matrix is positive semidefinite.

Remembering that the weights of the nodes are symmetric, ie that $w_{ij} = w_{ji}$, the Laplacian matrix of a 3-node graph has the form:

$$L = \begin{pmatrix} d_1 & -w_{12} & -w_{13} \\ -w_{12} & d_2 & -w_{23} \\ -w_{13} & -w_{23} & d_3 \end{pmatrix} \quad (6.104)$$

Let us consider a vector $\mathbf{x} \equiv (x_1, x_2, x_3)^T$ and let us compute $\mathbf{x}^T L \mathbf{x}$:

$$\begin{aligned} \mathbf{x}^T L \mathbf{x} &= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} d_1 & -w_{12} & -w_{13} \\ -w_{12} & d_2 & -w_{23} \\ -w_{13} & -w_{23} & d_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\ &= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} d_1 x_1 - w_{12} x_2 - w_{13} x_3 \\ -w_{12} x_1 + d_2 x_2 - w_{23} x_3 \\ -w_{13} x_1 - w_{23} x_2 + d_3 x_3 \end{pmatrix} \\ &= d_1 x_1^2 + d_2 x_2^2 + d_3 x_3^2 - 2w_{12} x_2 x_1 - 2w_{13} x_3 x_1 - 2w_{23} x_3 x_2 \end{aligned} \quad (6.105)$$

We remember that the degree of a node is equal to the sum of the weights it has with

the other nodes. So, $d_1 = w_{12} + w_{13}$, $d_2 = w_{21} + w_{23}$ and $d_3 = w_{31} + w_{32}$. Then:

$$\begin{aligned}\mathbf{x}^T L \mathbf{x} &= w_{12}x_1^2 + w_{13}x_1^2 + w_{21}x_2^2 + w_{23}x_2^2 + w_{31}x_3^2 + w_{32}x_3^2 \\ &\quad - 2w_{12}x_2x_1 - 2w_{13}x_3x_1 - 2w_{23}x_3x_2 \\ &= w_{12}(x_1 - x_2)^2 + w_{13}(x_1 - x_3)^2 + w_{23}(x_2 - x_3)^2\end{aligned}\tag{6.106}$$

The right-hand side of (6.106) is always greater than or equal to 0, for any vector \mathbf{x} . So, matrix L is positive semidefinite.

Example B6.34

Show that, if L is a positive semidefinite matrix and A is a diagonal matrix, ALA is also positive semidefinite.

To prove that ALA is positive semidefinite, we must show that $\mathbf{x}^T ALA \mathbf{x}$ is always greater than or equal to 0 for every real vector \mathbf{x} . We note that $A\mathbf{x}$ is another vector, which we might call \mathbf{y} . We also note that since A is a diagonal matrix, $\mathbf{y}^T = \mathbf{x}^T A$. So, we may write: $\mathbf{x}^T ALA \mathbf{x} = \mathbf{y}^T Ly$. Since L is positive semidefinite, $\mathbf{y}^T Ly \geq 0$ and so $\mathbf{x}^T ALA \mathbf{x} \geq 0$. Therefore, ALA is also positive semidefinite.

Example B6.35

Show that, if A is a positive semidefinite matrix, all its eigenvalues are non-negative.

Assume that \mathbf{x} is an eigenvector of A and λ is the corresponding eigenvalue. Then $A\mathbf{x} = \lambda\mathbf{x}$. Multiply both sides of this equation with \mathbf{x}^T to obtain:

$$\mathbf{x}^T A \mathbf{x} = \mathbf{x}^T \lambda \mathbf{x}\tag{6.107}$$

As λ is a scalar, it can come just after the equal sign on the right-hand side of (6.107). Also, as \mathbf{x} is an eigenvector, $\mathbf{x}^T \mathbf{x} = 1$. Then

$$\lambda = \mathbf{x}^T A \mathbf{x} \geq 0\tag{6.108}$$

since A is positive semidefinite.

Example B6.36

Show that the eigenvector that corresponds to the 0 eigenvalue of matrix $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ in Box 6.4 is vector **1**, ie a vector with all its elements equal to 1.

We observe that if we set $\mathbf{y} = \mathbf{1}$ in (6.101), $\lambda = 0$ because we have shown that $L\mathbf{1} = \mathbf{0}$, where **0** is a vector of zeros (see (6.96)). So, vector **1**, normalised so the sum of the squares of its elements is 1, is the eigenvector of eigenvalue $\lambda = 0$. This eigenvector does not partition the graph and so it is of no interest to the minimisation of (6.70).

Example B6.37

Show that all solutions of (6.103) satisfy $\mathbf{y}^T D \mathbf{1} = 0$.

Since all eigenvectors are orthogonal to each other, they will also be orthogonal to eigenvector $\mathbf{z}_0 = D^{\frac{1}{2}}\mathbf{1}$, the eigenvector that corresponds to the 0 eigenvalue. Then, eigenvector \mathbf{z}_i , which corresponds to eigenvalue λ_i , must satisfy:

$$\mathbf{z}_i^T \mathbf{z}_0 = 0 \Rightarrow \mathbf{y}_i^T D^{\frac{1}{2}} D^{\frac{1}{2}} \mathbf{1} = 0 \Rightarrow \mathbf{y}_i^T D \mathbf{1} = 0 \quad (6.109)$$

Example B6.38

Work out the values of the elements of vector \mathbf{y} , as defined by (6.85), that identify the nodes that belong to the two sets.

The i th element of vector \mathbf{y} is given by:

$$y_i = 1 + x_i - b(1 - x_i) \quad (6.110)$$

If node V_i belongs to set A , $x_i = 1$ and so $y_i = 2 > 0$. If node V_i does not belong to set A , $x_i = -1$, and so $y_i = -2b < 0$.

So, thresholding the y_i values into positive and negative allows us to identify the nodes that belong to the two classes.

How do we apply the normalised graph cuts algorithm in practice?

Step 1: Represent the image by a graph, where every pixel is represented by a node and the weights of the edges represent similarity between the pixels they join. As a measure of similarity, we may use the inverse of the distance computed by the mean shift algorithm:

$$w_{ij;kl} = \left[\frac{(i-k)^2}{h_x^2} + \frac{(j-l)^2}{h_y^2} + \frac{(g_{ij} - g_{kl})^2}{h_g^2} \right]^{-\frac{1}{2}} \quad (6.111)$$

Here $w_{ij;kl}$ is the weight between pixels (i, j) and (k, l) , with grey values g_{ij} and g_{kl} , respectively, and h_x , h_y and h_g are suitably chosen positive parameters. However, any other plausible measure of pixel similarity may be used instead.

Step 2: Construct matrices L , D and $E \equiv D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ as follows:

Matrix D is a diagonal matrix. Its i th element along the diagonal is the sum of the weights of the edges that connect node i with all other nodes.

Matrix W is made up from all the weights: its w_{ij} element is the weight between nodes V_i and V_j . This matrix has 0s along its diagonal.

Matrix L is $L = D - W$.

Note that if the image is $M \times N$ pixels in size, these matrices will be $MN \times MN$ in size.

Step 3: Find the smallest nonzero eigenvalue of E and the corresponding eigenvector \mathbf{z}_1 . It will be of size $MN \times 1$.

Step 4: Compute vector \mathbf{y} as $\mathbf{y}_1 = D^{-\frac{1}{2}}\mathbf{z}_1$.

This vector will have as many elements as there are pixels in the image. Depending on the way we indexed the pixels of the image in order to form its graph representation, there is a one-to-one correspondence between the elements of vector \mathbf{y}_1 and the image pixels.

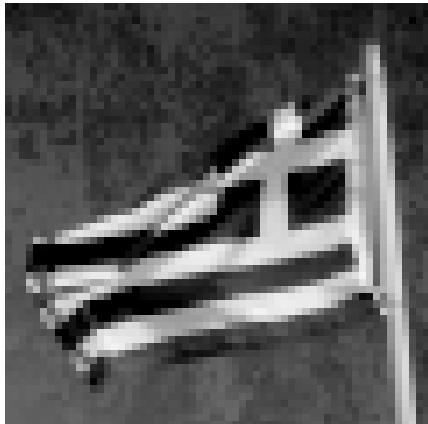
Step 5: Consider all pixels (nodes of the graph) that correspond to positive elements of vector \mathbf{y}_1 as belonging to one region and the ones that correspond to negative elements of \mathbf{y}_1 as belonging to another region.

In practice, it is not possible to calculate directly the eigenvalues of matrix E , as it could be very big. If we allow edges only between pixels that are relatively near to each other, then this matrix is sparse. In that case, there are special methods that can be used, like, for example, the **Lanczos method**. Such methods, however, are beyond the scope of this book.

Once the original image has been partitioned into two regions, the pixels of each region may be represented by separate graphs which may also be partitioned in two regions each, and so on. Figure 6.45 shows two examples of applying this algorithm.

Is it possible to segment an image by considering the *dissimilarities* between regions, as opposed to considering the similarities between pixels?

Yes, in such an approach we examine the differences between neighbouring pixels and say that pixels with different attribute values belong to different regions and, therefore, we postulate a boundary separating them. Such a boundary is called an **edge** and the process is called **edge detection**.

(a) Original images (64×64)

(b) First division



(c) Second division

Figure 6.45: Using the normalised graph cuts algorithm to segment the Greek flag and Leonidas. (b) Division in two regions. (c) Each region is further subdivided into two regions. Parameter values for the Greek flag $h_x = h_y = 5$ and $h_g = 0.01$ and for Leonidas $h_x = h_y = 4$ and $h_g = 0.02$.

6.2 Edge detection

How do we measure the dissimilarity between neighbouring pixels?

We may slide a window across the image, at each position calculate the statistical properties of the pixels within each half of the window and compare the two results. The places where these statistical properties differ most are where the boundaries of the regions are.

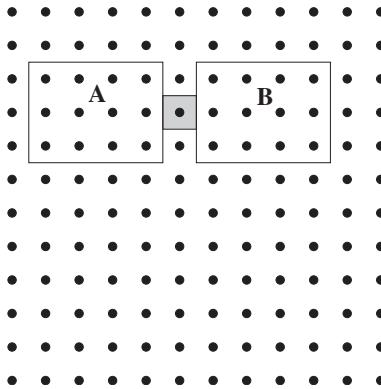


Figure 6.46: One may use a sliding window to measure the dissimilarity between two neighbouring patches of the image. A statistic is computed in each half of the window and the difference between the two values is assigned to the central pixel, highlighted in grey here.

For example, consider the 12×12 image of figure 6.46. Each dot represents a pixel. The rectangle drawn is a 3×9 window which could be placed so that its centre coincides with every pixel in the image, apart from those too close to the edge of the image. We can calculate the statistical properties of the twelve pixels inside the left part of the window (part A) and those of the twelve pixels inside the right part of the window (part B), and assign their difference to the central pixel. The simplest statistic we may compute is the mean grey value inside each subwindow. Other statistics may also (or alternatively) be computed. For example, we may calculate the standard deviation or even the skewness of the grey values of the pixels within each half of the window. However, one has to be careful as the number of pixels inside the window is rather small and high order statistics are not computed reliably from a small number of samples. Therefore, the size of the window plays a crucial role, as we need a large enough window, to calculate the statistics reliably, and a small enough window, to include within each half only part of a single region and avoid contamination from neighbouring regions.

We have to slide the window horizontally to scan the whole image. Local maxima of the assigned values are candidate positions for vertical boundaries. Local maxima where the value of the difference is greater than a certain threshold are accepted as vertical boundaries between adjacent regions. We can repeat the process by rotating the window by 90° and sliding it vertically to scan the whole image again.

In general, such statistical methods, that rely on local calculations, are not very reliable

boundary identifiers. However, if one has prior knowledge of the existence of a boundary, then one may slide the window perpendicularly to the suspected direction of the boundary, to locate its position very accurately. In such a case, statistical filters are very powerful tools of local boundary detection (see figure 6.47).

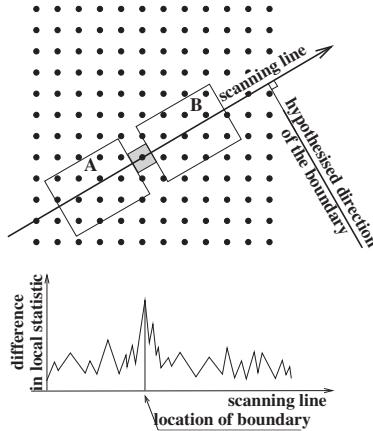


Figure 6.47: If we have some prior knowledge of the existence of a boundary, we may slide a bipolar window orthogonal to the hypothesised direction of the boundary and compute the exact location of the boundary, as the place where the difference in the value of the computed statistic is maximal.

What is the smallest possible window we can choose?

The smallest possible window we can choose consists of two adjacent pixels. The only “statistic” we can calculate from such a window is the difference of the grey values of the two pixels. When this difference is high, we say we have an **edge** passing between the two pixels. Of course, the difference of the grey values of the two pixels is not a statistic but rather an estimate of the first derivative of the intensity function, with respect to the spatial variable along the direction we compute the difference. This is because first derivatives of image function $f(i, j)$ are approximated by first differences in the discrete case:

$$\begin{aligned}\Delta f_x(i, j) &\equiv f(i+1, j) - f(i, j) \\ \Delta f_y(i, j) &\equiv f(i, j+1) - f(i, j)\end{aligned}\quad (6.112)$$

Calculating $\Delta f_x(i, j)$ at each pixel position is equivalent to convolving the image with a mask (filter) of the form $\boxed{+1 \ -1}$ in the x direction, and calculating $\Delta f_y(i, j)$ is equivalent to convolving the image with the filter $\boxed{\begin{matrix} +1 \\ -1 \end{matrix}}$ in the y direction.

The first and the simplest edge detection scheme then is to convolve the image with these two masks and produce two outputs. Note that these small masks have even lengths, so their centres are not associated with any particular pixel in the image as they slide across the image. So, the output of each calculation should be assigned to the position in between the

two adjacent pixels. These positions are said to constitute the **dual** grid of the image grid (see example 6.20, on page 569). In practice, we seldomly invoke the dual grid. We usually adopt a convention and try to be consistent. For example, we may always assign the output value to the first pixel of the mask. If necessary, we later may remember that this value actually measures the difference between the two adjacent pixels at the position half a pixel to the left or the bottom of the pixel to which it is assigned. So, with this understanding, and for simplicity from now and on, we shall be talking about **edge pixels** or **edgels**.

In the first output, produced by convolution with mask $\begin{bmatrix} +1 & -1 \end{bmatrix}$, any pixel, that has an absolute value larger than values of its left and right neighbours, is a candidate to be a vertical edge pixel. In the second output, produced by convolution with mask $\begin{bmatrix} +1 \\ -1 \end{bmatrix}$, any pixel, that has an absolute value larger than the values of its top and bottom neighbours, is a candidate to be a horizontal edge pixel. The process of identifying the local maxima as candidate edge pixels is called **non-maxima suppression**.

In the case of zero noise, this scheme will clearly pick up the discontinuities in image brightness.

What happens when the image has noise?

In the presence of noise every small and irrelevant fluctuation in the intensity value will be amplified by differentiating the image. It is common sense then, that one should smooth the image first, with a low pass filter, and then find the local differences.

Let us consider, for example, a 1D signal. Assume that one uses as a low pass filter a simple averaging procedure. We smooth the signal by replacing each intensity value with the average of three successive intensity values:

$$A_i \equiv \frac{I_{i-1} + I_i + I_{i+1}}{3} \quad (6.113)$$

Then, we estimate the derivative at position i by averaging the two differences between the value at position i and its left and right neighbours:

$$F_i \equiv \frac{(A_{i+1} - A_i) + (A_i - A_{i-1})}{2} = \frac{A_{i+1} - A_{i-1}}{2} \quad (6.114)$$

If we substitute from (6.113) into (6.114), we obtain:

$$F_i = \frac{1}{6}[I_{i+2} + I_{i+1} - I_{i-1} - I_{i-2}] \quad (6.115)$$

One may combine the two linear operations, of smoothing and finding differences, into one operation, if one uses large enough masks. In this case, the first difference at each position could be estimated by using a mask like $\begin{bmatrix} -\frac{1}{6} & -\frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} \end{bmatrix}$. It is clear that the larger the mask used, the more effective the smoothing. However, it is also clear that the more blurred the edge becomes too, so the more inaccurately its position will be specified (see figure 6.48).

For an image which is a 2D signal, one should use 2D masks. The smallest mask that one should use, to combine minimum smoothing with differencing, is a 3×3 mask. In this case, we also have the option to smooth in one direction and take the difference along the other.

This implies that the 2D mask may be the result of applying, in a cascaded way, first a 3×1 smoothing mask and then a 1×3 differencing masks, or vice versa. In general, however, a 2D 3×3 mask will have the form:

1	K	1
0	0	0
-1	K	-1

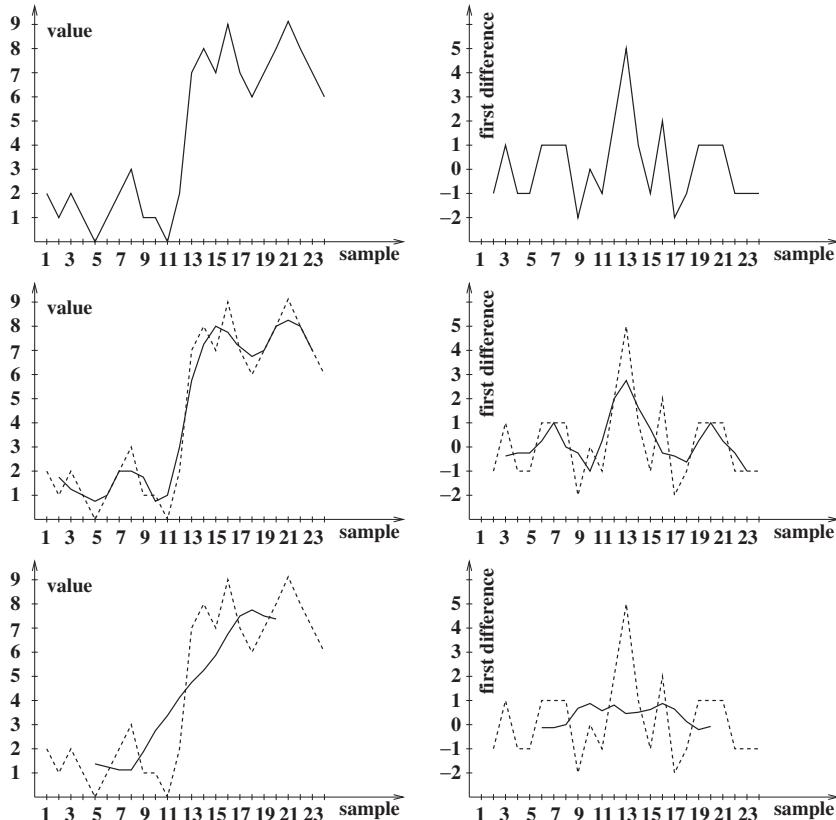


Figure 6.48: The left column shows a noisy signal and two smoothed versions of it, one produced by replacing each sample with the average of 3 successive samples (middle), and one produced by replacing each sample with the average of 9 successive samples (bottom). On the top right is the result of estimating the first difference of the original signal, by subtracting from the value of each sample the value of its previous sample. The edge manifests itself with a sharp peak in the unsmoothed signal. However, there are several secondary peaks present in this output. When the first difference is estimated from the smoothed version of the signal, the number of secondary peaks reduces, but at the same time, the main peak that corresponds to the true edge becomes blunter (right column, middle and bottom panels). The more severe the smoothing we applied was, the blunter the main peak. In the panels referring to the smoothed signal, the original input or its difference signal are shown by a dashed line to allow direct comparison.

Box 6.5. How can we choose the weights of a 3×3 mask for edge detection?

Let us denote the generic form of the 3×3 mask, we wish to define, as:

a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}
a_{31}	a_{32}	a_{33}

We are going to use one such mask to calculate Δf_x and another to calculate Δf_y . Such masks must obey the following conditions.

1. The mask which calculates Δf_x must be produced from the mask that calculates Δf_y by a 90° rotation. Let us consider from now and on the mask which will produce Δf_y only. The calculated value will be assigned to the central pixel.
2. We do not want to give any extra weight to the left or right neighbours of the central pixel, so we must have identical weights in the left and right columns. The 3×3 mask, therefore, must have the form:

a_{11}	a_{12}	a_{11}
a_{21}	a_{22}	a_{21}
a_{31}	a_{32}	a_{31}

3. Let us say that we want to subtract the signal “in front” of the central pixel from the signal “behind” it, in order to find local differences, and we want these two subtracted signals to have equal weights. The 3×3 mask, therefore, must have the form:

a_{11}	a_{12}	a_{11}
a_{21}	a_{22}	a_{21}
$-a_{11}$	$-a_{12}$	$-a_{11}$

4. If the image is absolutely smooth, we want to have zero response. So, the sum of all the weights must be zero. Therefore, $a_{22} = -2a_{21}$:

a_{11}	a_{12}	a_{11}
a_{21}	$-2a_{21}$	a_{21}
$-a_{11}$	$-a_{12}$	$-a_{11}$

5. In the case of a smooth signal, and as we differentiate in the direction of the columns, we expect each column to produce 0 output. Therefore, $a_{21} = 0$:

a_{11}	a_{12}	a_{11}
0	0	0
$-a_{11}$	$-a_{12}$	$-a_{11}$

We can divide these weights throughout by a_{11} so that, finally, this mask depends only on one parameter. Then the filter on page 594 follows.

What is the best value of parameter K ?

It can be shown that the orientations of edges which are almost aligned with the image axes are not affected by the differentiation, if we choose $K = 2$. We have then the **Sobel** masks for differentiating an image along two directions:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

Note that we have changed convention for the second mask and subtract the values of the pixels “behind” the central pixel from the values of the pixels “in front”. This is intentional, so that the orientation of the calculated edge, computed by using the components of the gradient vector derived using these masks, is measured from the horizontal axis anticlockwise (see Box 6.6).

Box 6.6. Derivation of the Sobel filters

Edges are characterised by their strength and orientation, defined as:

$$\text{Strength} \equiv E(i, j) \equiv \sqrt{[\Delta f_x(i, j)]^2 + [\Delta f_y(i, j)]^2}$$

$$\text{Orientation: } a(i, j) \equiv \tan^{-1} \frac{\Delta f_y(i, j)}{\Delta f_x(i, j)} \quad (6.116)$$

The idea is to try to specify parameter K of the filter on page 594, so that, the output of the operator is as faithful as possible to the true values of E and a which correspond to the non-discretised image:

$$E = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

$$a = \tan^{-1} \frac{\partial f / \partial y}{\partial f / \partial x} \quad (6.117)$$

Consider a straight step edge in the scene, passing through the middle of a pixel. Each pixel is assumed to be a tile of size 1×1 . Assume that the edge has orientation θ and assume that θ is small enough so that the edge cuts lines AB and CD as opposed to cutting lines AC and BD ($0 \leq \theta \leq \tan^{-1}(\frac{1}{3})$) (see figure 6.49).

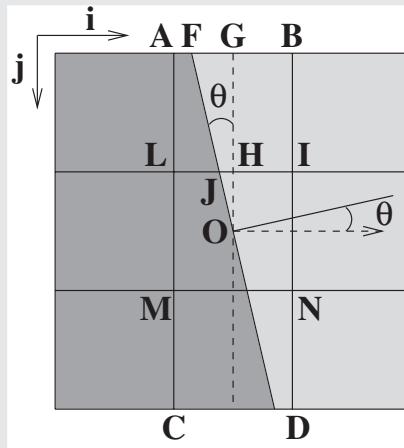


Figure 6.49: Zooming into a 3×3 patch of an image around pixel (i, j) .

Also, assume that on the left of the edge we have a dark region with grey value G_1 and on the right a bright region with grey value G_2 . Then, clearly, the pixel values inside the mask are:

$$\begin{aligned} f(i-1, j-1) &= f(i-1, j) = f(i-1, j+1) = G_1 \\ f(i+1, j-1) &= f(i+1, j) = f(i+1, j+1) = G_2 \end{aligned} \quad (6.118)$$

The pixels in the central column have mixed values. If we assume that each pixel is like a tile with dimensions 1×1 and denote the area of a polygon by the name of the polygon inside brackets, then pixel $ABIL$ will have value:

$$f(i, j-1) = G_1(AFJL) + G_2(FBIJ) = G_1 \left[\frac{1}{2} - (FGHJ) \right] + G_2 \left[\frac{1}{2} + (FGHJ) \right] \quad (6.119)$$

We must find the area of trapezium $FGHJ$. From the triangles OJH and OFG , we have:

$$JH = \frac{1}{2} \tan \theta, \quad FG = \frac{3}{2} \tan \theta \quad (6.120)$$

Therefore, $(FGHJ) = \frac{1}{2}(JH + FG) = \tan \theta$, and by substitution into equation (6.119), we obtain:

$$f(i, j-1) = G_1 \left(\frac{1}{2} - \tan \theta \right) + G_2 \left(\frac{1}{2} + \tan \theta \right) \quad (6.121)$$

By symmetry:

$$f(i, j+1) = G_2 \left(\frac{1}{2} - \tan \theta \right) + G_1 \left(\frac{1}{2} + \tan \theta \right) \quad (6.122)$$

Clearly:

$$f(i, j) = \frac{G_1 + G_2}{2} \quad (6.123)$$

Let us see now what the filter on page 594 will calculate in this case:

$$\begin{aligned} \Delta f_x &= f(i+1, j+1) + f(i+1, j-1) + Kf(i+1, j) \\ &\quad - [f(i-1, j+1) + f(i-1, j-1) + Kf(i-1, j)] \\ &= (G_2 + G_2 + KG_2) - (G_1 + G_1 + KG_1) = (G_2 - G_1)(2 + K) \end{aligned}$$

$$\begin{aligned} \Delta f_y &= -[f(i-1, j+1) + f(i+1, j+1) + Kf(i, j+1)] \\ &\quad + f(i-1, j-1) + f(i+1, j-1) + Kf(i, j-1) \\ &= -G_1 - G_2 - KG_2 \left(\frac{1}{2} - \tan \theta \right) - KG_1 \left(\frac{1}{2} + \tan \theta \right) + G_1 + G_2 \\ &\quad + KG_1 \left(\frac{1}{2} - \tan \theta \right) + KG_2 \left(\frac{1}{2} + \tan \theta \right) \\ &= -K(G_2 - G_1) \left(\frac{1}{2} - \tan \theta \right) + K(G_2 - G_1) \left(\frac{1}{2} + \tan \theta \right) \\ &= K(G_2 - G_1) \left(-\frac{1}{2} + \tan \theta + \frac{1}{2} + \tan \theta \right) \\ &= 2K(G_2 - G_1) \tan \theta \end{aligned} \quad (6.124)$$

The magnitude of the edge will then be

$$E = \sqrt{(G_2 - G_1)^2(2 + K)^2 + (2K)^2(G_2 - G_1)^2 \tan^2 \theta} \quad (6.125)$$

$$= (G_2 - G_1)(2 + K) \sqrt{1 + \left(\frac{2K}{2 + K} \right)^2 \tan^2 \theta} \quad (6.126)$$

and the orientation of the edge:

$$\tan \alpha = \frac{\Delta f_y}{\Delta f_x} = \frac{2K \tan \theta}{2 + K} \quad (6.127)$$

Note that if we choose $K = 2$,

$$\tan \alpha = \tan \theta \quad (6.128)$$

i.e. the calculated orientation of the edge will be equal to the true orientation.

One can perform a similar calculation for the case $\tan^{-1} \frac{1}{3} \leq \theta \leq 45^\circ$. In that case, we have error introduced in the calculation of the orientation of the edge.

Example 6.39

Write down the formula, which expresses the output $O(i, j)$ at position (i, j) of the convolution of the image with the Sobel mask, that differentiates along the i axis, as a function of the input values $I(i, j)$. (Note: Ignore boundary effects, ie assume that (i, j) is sufficiently far from the image border.)

$$\begin{aligned} O(i, j) = & -I(i-1, j-1) - 2I(i-1, j) - I(i-1, j+1) \\ & + I(i+1, j-1) + 2I(i+1, j) + I(i+1, j+1) \end{aligned} \quad (6.129)$$

Example 6.40

You have a 3×3 image which can be represented by a 9×1 vector. Construct a 9×9 matrix, which, when it operates on the image vector, produces another vector, each element of which is the estimate of the gradient component of the image along the i axis. (To deal with the boundary pixels, assume that the image is repeated periodically in all directions.)

Consider a 3×3 image:

$$j \downarrow \begin{pmatrix} i \rightarrow \\ f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \quad (6.130)$$

Periodic repetition of this image implies that we have:

$$\begin{matrix} f_{33} & f_{31} & f_{32} & f_{33} & f_{31} \end{matrix} \quad (6.131)$$

$$\begin{matrix} f_{13} & \left(\begin{matrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{matrix} \right) & f_{11} \\ f_{23} & & f_{21} \\ f_{33} & & f_{31} \end{matrix} \quad (6.132)$$

$$\begin{matrix} f_{13} & f_{11} & f_{12} & f_{13} & f_{11} \end{matrix} \quad (6.133)$$

Using the result of example 6.39, we notice that the first derivative at position $(1, 1)$ is given by:

$$f_{32} + 2f_{12} + f_{22} - f_{33} - 2f_{13} - f_{23} \quad (6.134)$$

The vector representation of the image is:

$$\begin{pmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{pmatrix} \quad (6.135)$$

If we operate with a 9×9 matrix on this image, we notice that in order to get the desired output, the first row of the matrix should be:

$$0 \ 0 \ 0 \ 2 \ 1 \ 1 \ -2 \ -1 \ -1$$

The derivative at position (1,2), ie at pixel with value f_{21} , is given by:

$$f_{12} + 2f_{22} + f_{32} - f_{13} - 2f_{23} - f_{33} \quad (6.136)$$

Therefore, the second row of the 9×9 matrix should be:

$$0 \ 0 \ 0 \ 1 \ 2 \ 1 \ -1 \ -2 \ -1$$

The derivative at position (1,3) is given by:

$$f_{22} + 2f_{32} + f_{12} - f_{23} - 2f_{33} - f_{13}$$

Therefore, the third row of the 9×9 matrix should be:

$$0 \ 0 \ 0 \ 1 \ 1 \ 2 \ -1 \ -1 \ -2$$

Reasoning this way, we conclude that the matrix we require must be:

$$\begin{pmatrix} 0 & 0 & 0 & 2 & 1 & 1 & -2 & -1 & -1 \\ 0 & 0 & 0 & 1 & 2 & 1 & -1 & -2 & -1 \\ 0 & 0 & 0 & 1 & 1 & 2 & -1 & -1 & -2 \\ -2 & -1 & -1 & 0 & 0 & 0 & 2 & 1 & 1 \\ -1 & -2 & -1 & 0 & 0 & 0 & 1 & 2 & 1 \\ -1 & -1 & -2 & 0 & 0 & 0 & 1 & 1 & 2 \\ 2 & 1 & 1 & -2 & -1 & -1 & 0 & 0 & 0 \\ 1 & 2 & 1 & -1 & -2 & -1 & 0 & 0 & 0 \\ 1 & 1 & 2 & -1 & -1 & -2 & 0 & 0 & 0 \end{pmatrix} \quad (6.137)$$

This is a block circulant matrix.

Example 6.41

Using the matrix derived in example 6.40, calculate the first derivative along the i axis of the following image:

$$\begin{pmatrix} 3 & 1 & 0 \\ 3 & 1 & 0 \\ 3 & 1 & 0 \end{pmatrix} \quad (6.138)$$

What is the component of the gradient along the i axis at the centre of the image?

$$\begin{pmatrix} 0 & 0 & 0 & 2 & 1 & 1 & -2 & -1 & -1 \\ 0 & 0 & 0 & 1 & 2 & 1 & -1 & -2 & -1 \\ 0 & 0 & 0 & 1 & 1 & 2 & -1 & -1 & -2 \\ -2 & -1 & -1 & 0 & 0 & 0 & 2 & 1 & 1 \\ -1 & -2 & -1 & 0 & 0 & 0 & 1 & 2 & 1 \\ -1 & -1 & -2 & 0 & 0 & 0 & 1 & 1 & 2 \\ 2 & 1 & 1 & -2 & -1 & -1 & 0 & 0 & 0 \\ 1 & 2 & 1 & -1 & -2 & -1 & 0 & 0 & 0 \\ 1 & 1 & 2 & -1 & -1 & -2 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 3 \\ 3 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \\ 4 \\ -12 \\ -12 \\ -12 \\ 8 \\ 8 \\ 8 \end{pmatrix} \quad (6.139)$$

At the centre of the image the component of the gradient along the i axis is -12 .

In the general case, how do we decide whether a pixel is an edge pixel or not?

Edges are positions in the image where the image function changes. As the image is a 2D function, to find these positions we have to calculate the gradient of the function $\nabla f(x, y)$. The gradient of a 2D function is a 2D vector, with the partial derivatives of the function along two orthogonal directions as its components. In the discrete case, these partial derivatives are the partial differences computed along two orthogonal directions, by using masks like, for example, the Sobel masks. If we convolve an image with these masks, we have a gradient vector associated with each pixel. Edges are the places where the magnitude of the gradient vector is a local maximum along the direction of the gradient vector (ie the orientation of the gradient at that pixel position). For this purpose, the local value of the gradient magnitude will have to be compared with the values of the gradient estimated along this orientation and at unit distance on either side away from the pixel. In general, these gradient values will not be known, because they will happen to be at positions in between the pixels. Then, either a local surface is fitted to the image and used for the estimation of the gradient magnitude at any interpixel position required, or the value of the gradient magnitude is calculated by interpolating the values of the gradient magnitudes at the known integer positions.

After this process of **non-maxima suppression** takes place, the values of the gradient

vectors that remain are thresholded and only pixels with gradient values above the threshold are considered as edge pixels identified in the **edge map** (see figure 6.50).

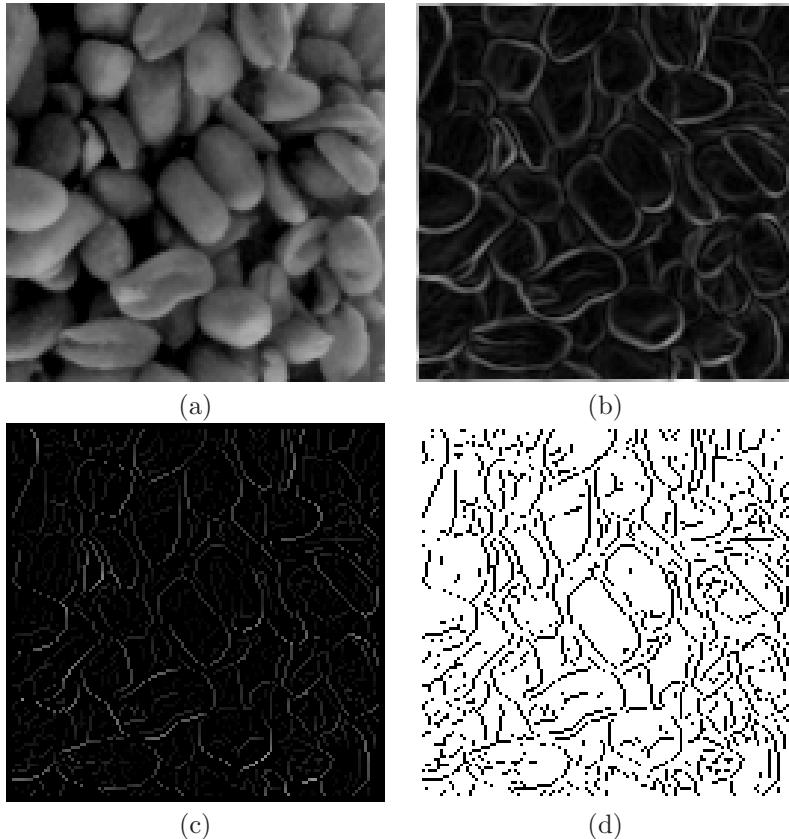


Figure 6.50: (a) Original image. (b) The gradient magnitude “image”: the brighter the pixel, the higher the value of the gradient of the image at that point. (c) After non-maxima suppression: only pixels with locally maximal values of the gradient magnitude retain their values. The gradient values of all other pixels are set to zero. (d) The edge map: the gradient values in (c) are thresholded: all pixels with values above the threshold are labelled 0 (black) and all pixels with values below the threshold are labelled 255 (white).

How do we perform linear edge detection in practice?

Step 0: Create an output array the same size as the input image and assign to every pixel value 255 (white).

Step 1: Convolve the input image with a mask that estimates its horizontal derivative $\Delta f_x(i, j)$. This mask may be such that, in order to perform the estimation, smooths the image in the vertical direction.

Step 2: Convolve the input image with a mask that estimates its vertical derivative $\Delta f_y(i, j)$.

This mask may be such that, in order to perform the estimation, smooths the image in the horizontal direction.

Step 3: Compute the gradient magnitude at each pixel position:

$$G(i, j) \equiv \sqrt{\Delta f_x(i, j)^2 + \Delta f_y(i, j)^2} \quad (6.140)$$

Step 4: Pixels that have $G(i, j)$ above a predefined threshold have their gradient orientation estimated. Let us say that the orientation of the gradient vector of such a pixel (i, j) is $\theta(i, j)$, with respect to the horizontal (the i) axis. Angle $\theta(i, j)$ is allowed to vary in the range $[-90^\circ, 90^\circ]$.

Step 5: For each pixel with gradient orientation $\theta(i, j)$, compute the gradient magnitude at neighbouring positions:

$$(i + \cos \theta(i, j), j - \sin \theta(i, j)) \quad \text{and} \quad (i - \cos \theta(i, j), j + \sin \theta(i, j)) \quad (6.141)$$

You may use bilinear interpolation for that, as these positions are most likely in between pixels (see page 518). If $G(i, j)$ is greater than both these estimated gradient magnitude values, mark pixel (i, j) as an edge pixel in the output array, by giving it value 0 (black).

Example 6.42

The following image is given

0	1	2	0	4	5
0	0	1	1	4	5
0	2	0	4	5	4
0	0	5	4	6	6
0	0	6	6	5	6
5	4	6	5	4	5

Use the following masks to estimate the magnitude and orientation of the local gradient at all pixel positions of the image, except the boundary pixels:

-1	0	1
-3	0	3
-1	0	1

-1	-3	-1
0	0	0
1	3	1

Then, indicate which pixels represent horizontal edge pixels (edgels) and which represent vertical.

If we convolve the image with the first mask, we shall have an estimate of the gradient component along the x (horizontal) axis:

5	4	16	17		
6	11	19	6		
21	20	7	6		
24	23	-4	2		

ΔI_x

If we convolve the image with the second mask, we shall have an estimate of the gradient component along the y (vertical) axis:

1	-1	11	6		
4	15	15	10		
0	18	12	4		
18	8	2	-6		

 ΔI_y

The gradient magnitude is given by: $|G| = \sqrt{(\Delta I_x)^2 + (\Delta I_y)^2}$:

	$\sqrt{26}$	$\sqrt{17}$	$\sqrt{377}$	$\sqrt{325}$	
	$\sqrt{52}$	$\sqrt{346}$	$\sqrt{586}$	$\sqrt{136}$	
	$\sqrt{441}$	$\sqrt{724}$	$\sqrt{193}$	$\sqrt{52}$	
	$\sqrt{900}$	$\sqrt{593}$	$\sqrt{20}$	$\sqrt{40}$	

The gradient orientation is given by:

$$\theta = \tan^{-1} \frac{\Delta I_y}{\Delta I_x} \quad (6.142)$$

	$\tan^{-1}(1/5)$	$-\tan^{-1}(1/4)$	$\tan^{-1}(11/16)$	$\tan^{-1}(6/17)$	
	$\tan^{-1}(2/3)$	$\tan^{-1}(15/11)$	$\tan^{-1}(15/19)$	$\tan^{-1}(5/3)$	
	$\tan^{-1} 0$	$\tan^{-1}(9/10)$	$\tan^{-1}(12/7)$	$\tan^{-1}(2/3)$	
	$\tan^{-1}(3/4)$	$\tan^{-1}(8/23)$	$-\tan^{-1}(1/2)$	$-\tan^{-1} 3$	

We know that for an angle θ to be in the range -45° to 45° ($0^\circ \leq |\theta| \leq 45^\circ$) its tangent must satisfy $0 \leq |\tan \theta| \leq 1$. Also, an angle θ is in the range 45° to 90° , or -90° to -45° ($45^\circ < |\theta| \leq 90^\circ$), if $1 < |\tan \theta| \leq +\infty$.

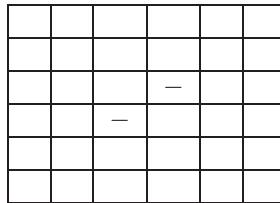
As we want to quantise the gradient orientations to vertical and horizontal ones only, we shall make all orientations $0^\circ \leq |\theta| \leq 45^\circ$ horizontal, ie set them to 0, and all orientations with $45^\circ < |\theta| \leq 90^\circ$ vertical, ie we shall set them to 90° .

By inspecting the orientation array above, we infer the following gradient orientations:

	0	0	0	0	
	0	90°	0	90°	
	0	0	90°	0	
	0	0	0	90°	

A pixel is a horizontal edge pixel if the magnitude of its gradient is a local maximum, when compared with its vertical neighbours, and its orientation is 0° . A pixel is a vertical edge if its orientation is 90° and its magnitude is a local maximum in the horizontal direction.

By inspecting the gradient magnitudes and the quantised orientations we derived, we identify the following horizontal (-) edgels only:



Example 6.43

Are the masks used in example 6.42 separable? How can we take advantage of the separability of a 2D mask to reduce the computational cost of convolution?

The masks used in example 6.42 are separable because they may be implemented as a sequence of two 1D filters, applied in a cascaded way one after the other:

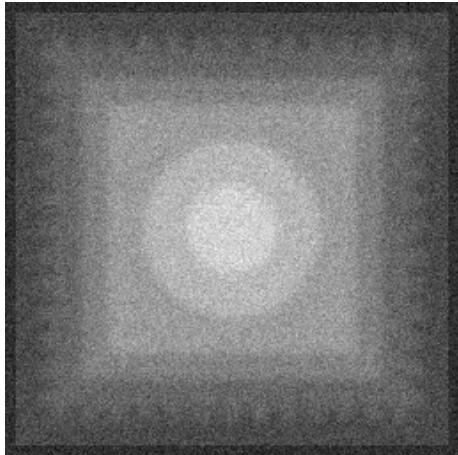
$$\begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -3 & 0 & 3 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \Rightarrow \begin{array}{|c|} \hline 1 \\ \hline 3 \\ \hline 1 \\ \hline \end{array} \text{ followed by } \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline -1 & -3 & -1 \\ \hline 0 & 0 & 0 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|} \hline 1 & 3 & 1 \\ \hline \end{array} \text{ followed by } \begin{array}{|c|} \hline -1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}$$

Any $2D N \times N$ separable mask may be implemented as a cascade of two 1D masks of size N . This implementation replaces N^2 multiplications and additions per pixel by $2N$ such operations per pixel.

Are Sobel masks appropriate for all images?

Sobel masks are appropriate for images with low levels of noise. They are inadequate for noisy images. See, for example, figure 6.51, which shows the results of edge detection using Sobel masks for a very noisy image.



(a) A very noisy image

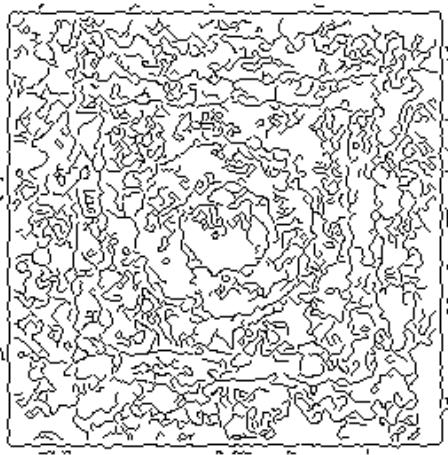
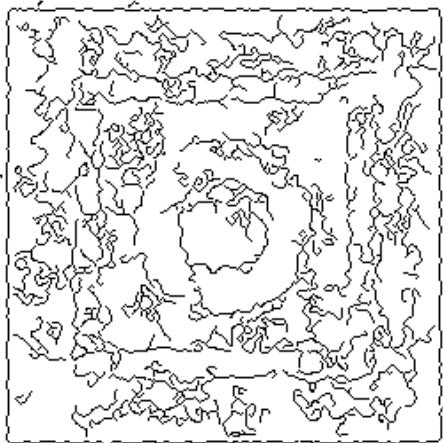
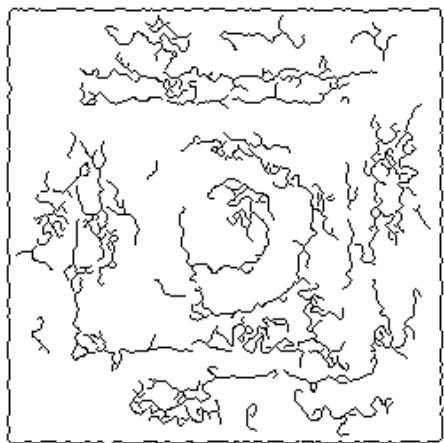
(b) Edge map (Sobel, threshold th_1)(c) Edge map (Sobel, threshold th_2)(d) Edge map (Sobel, threshold th_3)

Figure 6.51: Trying to detect edges in a blurred and noisy image using the Sobel edge detector may be very tricky. One may experiment with different threshold values but the result is not satisfactory. The three results shown were obtained by using different thresholds within the same edge detection framework, and they were the best among many others obtained for different threshold values.

How can we choose the weights of the mask if we need a larger mask owing to the presence of significant noise in the image?

We shall consider the problem of detecting abrupt changes in the value of a 1D signal, like the one shown in figure 6.52.

Let us assume that the feature we want to detect is $u(x)$ and it is immersed in additive white Gaussian noise $n(x)$. The mask we want to use for edge detection should have certain desirable characteristics, called **Canny's criteria**, as follows.

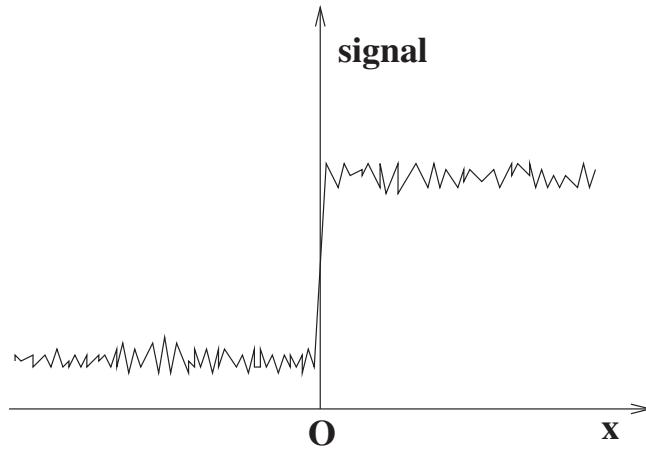


Figure 6.52: A noisy 1D edge.

1. Good signal to noise ratio.
2. Good locality, ie the edge should be detected where it actually is.
3. Small number of false alarms, ie the maxima of the filter response should be mainly due to the presence of the true edges in the image, rather than due to noise.

Canny showed that a filter function $f(x)$ has maximal signal to noise ratio, if it is chosen in such a way that it maximises the quantity

$$SNR \equiv \frac{\int_{-\infty}^{+\infty} f(x)u(-x)dx}{n_0 \sqrt{\int_{-\infty}^{+\infty} f^2(x)dx}} \quad (6.143)$$

where n_0 is the standard deviation of the additive Gaussian noise. As this is a constant for a particular image, when we try to decide what function $f(x)$ should be, we omit it from the computation. So, we try to maximise the quantity:

$$S \equiv \frac{\int_{-\infty}^{+\infty} f(x)u(-x)dx}{\sqrt{\int_{-\infty}^{+\infty} f^2(x)dx}} \quad (6.144)$$

Canny also showed that the filter function $f(x)$ detects an edge with the minimum shifting away from its true location, if it is chosen in such a way that its first and second derivatives maximise the quantity:

$$L \equiv \frac{\int_{-\infty}^{+\infty} f''(x)u(-x)dx}{\sqrt{\int_{-\infty}^{+\infty} [f'(x)]^2 dx}} \quad (6.145)$$

Finally, he showed that the output of the convolution of the signal with filter $f(x)$ will contain minimal number of false responses, if function $f(x)$ is chosen in such a way that its first and second derivatives maximise the quantity:

$$C \equiv \sqrt{\frac{\int_{-\infty}^{+\infty} (f'(x))^2 dx}{\int_{-\infty}^{+\infty} (f''(x))^2 dx}} \quad (6.146)$$

One, therefore, may design an optimal edge enhancing filter by trying to maximise the above three quantities S , L and C .

Canny combined the first two into a single performance measure \tilde{P} , which he maximised under the constraint that C is constant. He derived a convolution filter that way, which he did not use, because he noticed that it could be approximated by the derivative of a Gaussian. So, he adopted the derivative of a Gaussian as a good enough image derivative estimation filter (see page 352 and example 4.22).

Alternatively, one may combine all three quantities S , L , and C into a single performance measure P

$$P \equiv (S \times L \times C)^2 = \frac{\left[\int_{-\infty}^{+\infty} f(x)u(-x)dx \right]^2 \left[\int_{-\infty}^{+\infty} f''(x)u(-x)dx \right]^2}{\int_{-\infty}^{+\infty} f^2(x)dx \int_{-\infty}^{+\infty} (f''(x))^2 dx} \quad (6.147)$$

and try to choose $f(x)$ so that P is maximum. The free parameters which will appear in the functional expression of $f(x)$ can be calculated from the boundary conditions imposed on $f(x)$ and by substitution into the expression for P and selection of their numerical values so that P is maximum.

Once function $f(x)$ has been worked out, it must be sampled to produce a digital filter. The number of samples used defines the size of the filter. The higher the level of noise in the image, ie the higher the value of n_0 in (6.143), the more densely the filter should be sampled: dense sampling makes the filter longer (consisting of more taps). This implies a higher degree of smoothing imposed to the data, while estimating its first difference.

The filter defined in the above way is a filter that will respond maximally to a discontinuity of the signal that has the form of a step function, ie it is a filter that estimates, somehow, the local first difference of the signal. If we integrate it, we may obtain a filter that may be used to smooth the signal. So, to move from 1D to 2D, we work out the differentiating filter, then we integrate it to form the corresponding smoothing filter, then we sample both filters to make them usable for digital signals, and then we apply first the smoothing filter along one direction, followed by the differencing filter along the orthogonal direction, to estimate the first derivative along the differencing direction. We then repeat the process by exchanging the directions of smoothing and differentiation, to estimate the derivative along the other direction. Finally, we use these two directional differences to estimate the local gradient magnitude and orientation, and proceed as we did earlier to work out the edge map of the image, applying non-maxima suppression and thresholding.

It must be stressed that the filters defined in this way are appropriate for the detection of *antisymmetric features*, ie *step* or *ramp edges*, when the noise in the signal is *additive, white* and *Gaussian*.

Can we use the optimal filters for edges to detect lines in an image in an optimal way?

No. Edge detection filters respond badly to lines. The theory for optimal edge detector filters has been developed under the assumption that there is a single isolated step edge. We can see that, from the limits used in the integrals in equation (6.143). For example, the limits are from $-\infty$ to $+\infty$, assuming that in the whole infinite length of the signal, there is only a single step edge. If we have a line, its profile looks like the one shown in figure 6.53.

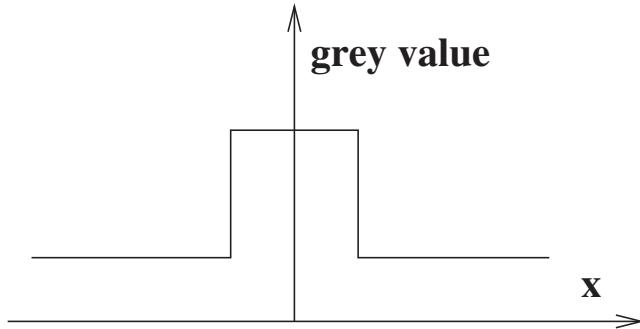


Figure 6.53: The profile of a line.

It looks like two step edges back to back. The responses of the filter to the two step edges interfere and the result is not satisfactory: the two steps may or may not be detected. If they are detected, they tend to be detected as two edges noticeably misplaced from their true positions and shifted away from each other.

Apart from this, there is another more fundamental difference between step edges and lines.

What is the fundamental difference between step edges and lines?

A step edge is scale invariant: it looks the same whether we stretch it or shrink it. A line has an intrinsic length-scale: this is its width. From the moment the feature we wish to detect has a characteristic “size”, the size of the filter we must use becomes important. Similar considerations apply also if one wishes to develop appropriate filters for ramp edges as opposed to step edges: the length over which the ramp rises (or drops) is characteristic of the feature and it cannot be ignored, when designing the optimal filter. The criterion expressed by equation (6.147) may be appropriately modified to develop optimal filters for the detection of ramp edges of various slopes. Figure 6.54 shows attempts to detect edges in the image of figure 6.51a, using the wrong filter. The edges are blurred, so they actually have ramp-like profiles, but the filter used is the optimal filter for step edges. Figure 6.55 shows the results of using the optimal filter for ramps on the same image, and the effect the size of the filter has on the result. Figure 6.56 shows an example of a relatively “clean” and “unblurred” image, for which one does not need to worry too much about which filter one uses for edge detection.

Canny’s criteria have also been modified for the case of lines and used to derive optimal filters for line detection that depend on the width and sharpness of the line.

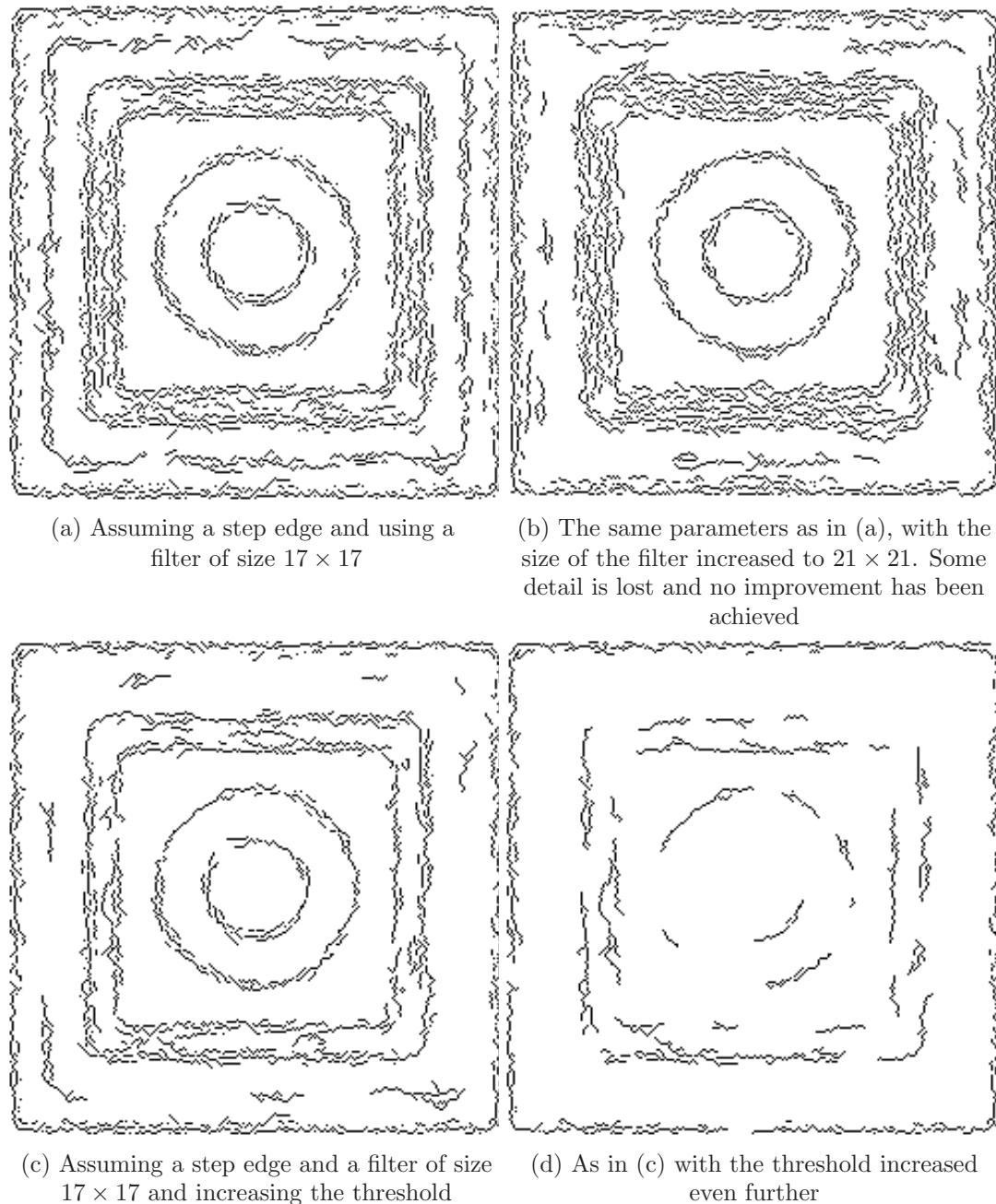


Figure 6.54: Trying to deal with the high level of noise by using an optimal filter of large size and experimenting with the thresholds may not help, if the edges are blurred and resemble ramps, and the “optimal” filter we use has been developed to be optimal for step edges and not for ramp edges.

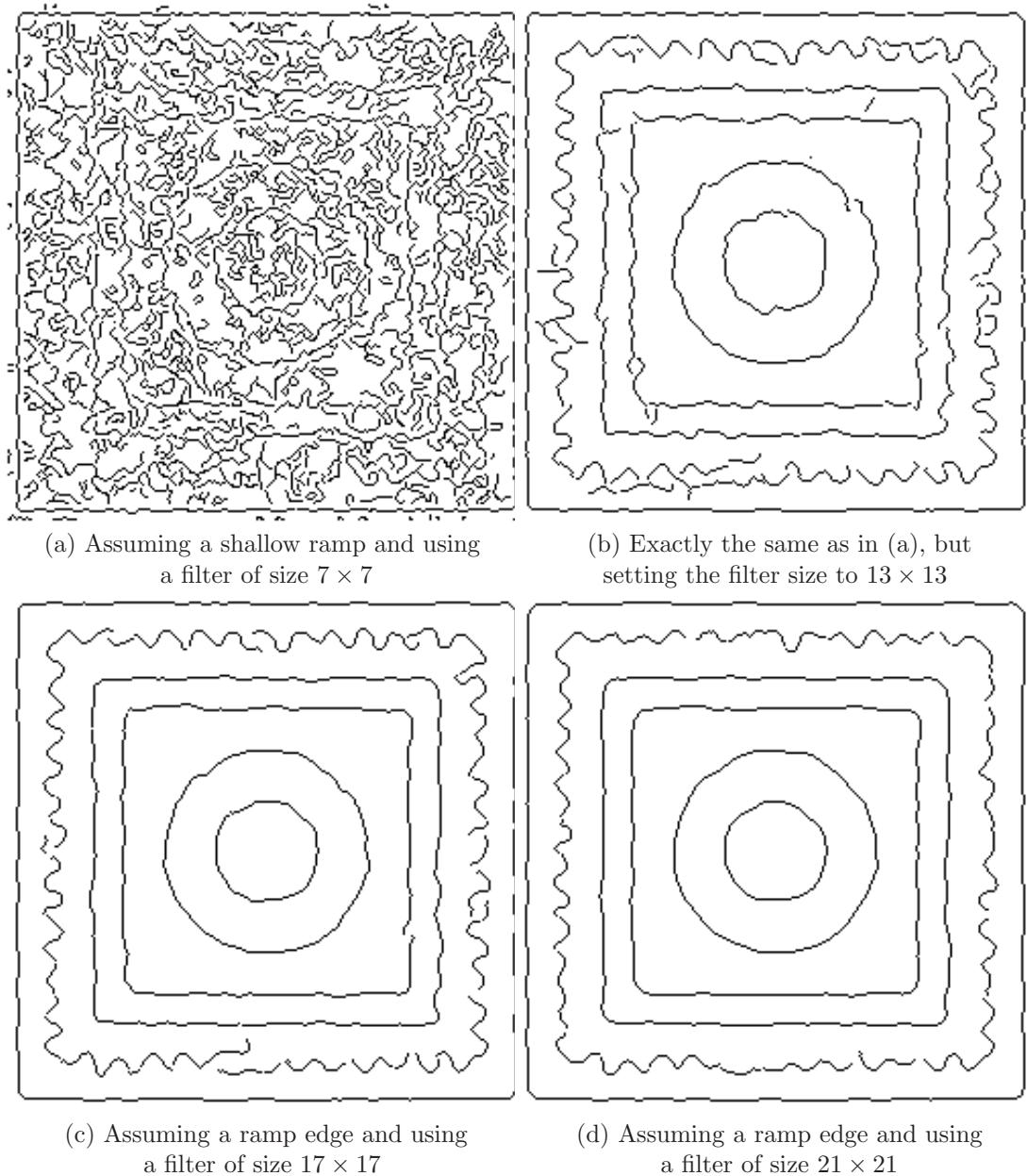


Figure 6.55: Having chosen the right model for the features we wish to detect, using the right filter size for the level of noise may prove crucial. Spectacularly good results may be obtained if the right filter shape and size are used. The filter shape determines the type of feature that will be enhanced, while the filter size determines the level of noise this filter can tolerate. These results should be compared with those of figure 6.54, where the optimal filter for step edges was used, while here we used the optimal filter of ramp edges.

Example B6.44

Assume that $u(x)$ is defined for positive and negative values of x and that we want to enhance a feature like $u(x)$, at position $x = 0$, in a noisy signal. Use equation (6.144) to show that if the feature we want to enhance is an even function, we must choose an even filter, and if it is an odd function, we must choose an odd filter.

Any function $f(x)$ may be written as the sum of an odd and an even part:

$$f(x) \equiv \underbrace{\frac{1}{2}[f(x) - f(-x)]}_{f_o(x)(\text{odd})} + \underbrace{\frac{1}{2}[f(x) + f(-x)]}_{f_e(x)(\text{even})} \quad (6.148)$$

In general, therefore, $f(x)$ may be written as:

$$f(x) = f_e(x) + f_o(x) \quad (6.149)$$

Assume that $u(x)$ is even. Then, the integral in the numerator of S is:

$$\begin{aligned} \int_{-\infty}^{+\infty} f(x)u(-x)dx &= \int_{-\infty}^{+\infty} f_e(x)u(-x)dx + \underbrace{\int_{-\infty}^{+\infty} f_o(x)u(-x)dx}_{\substack{\text{odd integrand integrated} \\ \text{over a symmetric interval:} \\ \text{it vanishes}}} \\ &= \int_{-\infty}^{+\infty} f_e(x)u(-x)dx \end{aligned} \quad (6.150)$$

The integral in the denominator in the expression for S is:

$$\begin{aligned} \int_{-\infty}^{+\infty} f^2(x)dx &= \int_{-\infty}^{+\infty} f_e^2(x)dx + \int_{-\infty}^{+\infty} f_o^2(x)dx + \underbrace{2 \int_{-\infty}^{+\infty} f_e(x)f_o(x)dx}_{\substack{\text{odd integrand integrated} \\ \text{over a symmetric interval:} \\ \text{it vanishes}}} \\ &= \int_{-\infty}^{+\infty} f_e^2(x)dx + \int_{-\infty}^{+\infty} f_o^2(x)dx \end{aligned} \quad (6.151)$$

So, we see that the odd part of the filter does not contribute at all to the signal response, while it contributes to the noise response. That is, it reduces the signal to noise ratio. Thus, to enhance an even feature we must use an even filter. Similarly, to enhance an odd feature, we must use an odd filter.

Example 6.45

You are asked to use the following filter masks:

$$\begin{array}{|c|c|c|} \hline -1 & 2 & -1 \\ \hline -1 & 2 & -1 \\ \hline -1 & 2 & -1 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline 2 & 2 & 2 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

What type of feature are these masks appropriate for enhancing? (The word “feature” here means structural feature, eg an edge, a line, a corner, etc in an image. This appears to be different from the previous use of the word “feature” we encountered in this chapter, where we defined it to mean “attribute”, eg a number that characterises a pixel. If we consider that masks like these measure somehow how strong a local edge or a local line is, then we may relate the two uses of the word feature: the output of applying such a mask produces a number that characterises a pixel by telling us how much edge-like or line-like the pixel is. This number is an attribute of the pixel, ie a feature.)

The first filter enhances vertical lines and the second horizontal lines, in both cases brighter than the background.

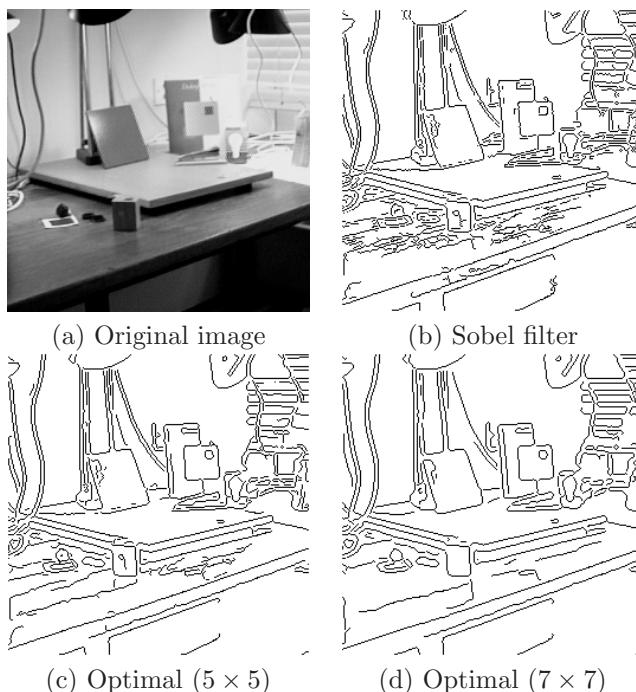


Figure 6.56: Everyday life images are relatively clean and good results can be obtained by using relatively small filters. Note that the smaller the filter, the more details of the image are preserved.

Example 6.46

Use the masks of example 6.45 to process the image below. (Do not process the border pixels.)

1	1	5	3	0
0	1	4	1	0
1	1	3	2	1
0	2	5	3	0
1	0	4	2	0

Then, by choosing an appropriate threshold for the output images calculated, produce the feature maps of the input image.

The result of the convolution of the image with the first mask is:

-8	15	-1		
-5	14	-1		
-8	14	1		

The result of the convolution of the image with the second mask is:

-2	-3	-4		
-2	-4	-1		
4	8	4		

Note that the outputs contain values that may easily be divided into two classes: positive and negative. We choose, therefore, as our threshold the zero value, $t = 0$. This threshold seems to be in the largest gap between the two populations of the output values (ie one population that presumably represents the background and one that represents the features we want to detect). When thresholding, we shall give one label to one class (say all numbers above the threshold will be labelled 1) and another label to the other class (all numbers below the threshold will be labelled 0). The two outputs then become:

0	1	0	0	0	0
0	1	0	0	0	0
0	1	0	1	1	1

These are the feature maps of the given image.

Box 6.7. Convolving a random noise signal with a filter

Assume that our noise signal is $n(x)$ and we convolve it with a filter $f(x)$. The result of the convolution will be:

$$g(x_0) = \int_{-\infty}^{+\infty} f(x)n(x_0 - x)dx \quad (6.152)$$

As the input noise is a random variable, this quantity is a random variable too. If we take its expectation value, it will be 0 as long as the expectation value of the input noise is 0. We may also try to characterise it by calculating its variance, that is, its mean square value. This is given by:

$$E\{[g(x_0)]^2\} = E\{g(x_0)g(x_0)\} \quad (6.153)$$

This is the definition of the autocorrelation function of the output, calculated at argument 0. So, we must calculate first the autocorrelation function of g , $R_{gg}(\tau)$.

We multiply both sides of equation (6.152) with $g(x_0 + \tau)$ and take expectation values:

$$\begin{aligned} E\{g(x_0)g(x_0 + \tau)\} &= E\left\{\int_{-\infty}^{+\infty} f(x)g(x_0 + \tau)n(x_0 - x)dx\right\} \\ \Rightarrow R_{gg}(\tau) &= \int_{-\infty}^{+\infty} f(x)E\{g(x_0 + \tau)n(x_0 - x)\}dx \\ \Rightarrow R_{gg}(\tau) &= \int_{-\infty}^{+\infty} f(x)R_{ng}(x_0 + \tau - x_0 + x)dx \\ \Rightarrow R_{gg}(\tau) &= \int_{-\infty}^{+\infty} f(x)R_{ng}(\tau + x)dx \end{aligned} \quad (6.154)$$

Here $R_{ng}(a)$ is the cross-correlation function between the input noise signal and the output noise signal, at relative shift a . We must calculate R_{ng} . We multiply both sides of (6.152) with $n(x_0 - \tau)$ and then take expectation values. (We multiply with $n(x_0 - \tau)$ instead of $n(x_0 + \tau)$ because in (6.154) we defined the argument of R_{ng} as the difference of the argument of g minus the argument of n .)

$$\begin{aligned} E\{g(x_0)n(x_0 - \tau)\} &= \int_{-\infty}^{+\infty} f(x)E\{n(x_0 - x)n(x_0 - \tau)\}dx \\ \Rightarrow R_{ng}(\tau) &= \int_{-\infty}^{+\infty} f(x)R_{nn}(x_0 - x - x_0 + \tau)dx \\ \Rightarrow R_{ng}(\tau) &= \int_{-\infty}^{+\infty} f(x)R_{nn}(\tau - x)dx \end{aligned} \quad (6.155)$$

Here $R_{nn}(a)$ is the autocorrelation function of the input noise signal at relative shift a . However, $n(x)$ is assumed to be white Gaussian noise, so its autocorrelation function

is a delta function given by $R_{nn}(\tau) = n_0^2\delta(\tau)$ where n_0^2 is the variance of the noise. Therefore:

$$R_{ng}(\tau) = \int_{-\infty}^{+\infty} f(x)n_0^2\delta(\tau - x)dx = n_0^2f(\tau) \quad (6.156)$$

So, R_{ng} with argument $\tau + x$, as it appears in (6.154), is:

$$R_{ng}(\tau + x) = n_0^2f(\tau + x) \quad (6.157)$$

If we substitute this into (6.154), we have:

$$R_{gg}(\tau) = n_0^2 \int_{-\infty}^{+\infty} f(x)f(\tau + x)dx \Rightarrow R_{gg}(0) = n_0^2 \int_{-\infty}^{+\infty} f^2(x)dx \quad (6.158)$$

If we substitute this into equation (6.153), we obtain:

$$E\left\{[g(x_0)]^2\right\} = n_0^2 \int_{-\infty}^{+\infty} f^2(x)dx \quad (6.159)$$

The mean filter response to noise is given by the square root of the above expression.

Box 6.8. Calculation of the signal to noise ratio after convolution of a noisy edge signal with a filter

Assume that $f(x)$ is the filter we want to develop, so that it enhances an edge in a noisy signal. The signal consists of two components: the deterministic signal of interest $u(x)$ and the random noise component $n(x)$:

$$I(x) = u(x) + n(x) \quad (6.160)$$

The response of the filter due to the deterministic component of the noisy signal, when the latter is convolved with the filter, is:

$$s(x_0) = \int_{-\infty}^{+\infty} f(x)u(x_0 - x)dx \quad (6.161)$$

Let us assume that the edge we wish to detect is actually at position $x_0 = 0$. Then:

$$s(0) = \int_{-\infty}^{+\infty} f(x)u(-x)dx \quad (6.162)$$

The mean response of the filter due to the noise component (see Box 6.7) is

$n_0 \sqrt{\int_{-\infty}^{+\infty} f^2(x) dx}$. The signal to noise ratio, therefore, is:

$$SNR = \frac{\int_{-\infty}^{+\infty} f(x)u(-x)dx}{n_0 \sqrt{\int_{-\infty}^{+\infty} f^2(x) dx}} \quad (6.163)$$

Box 6.9. Derivation of the good locality measure

The result of the convolution of signal (6.160) with filter $f(x)$ is

$$\begin{aligned} O(x_0) &\equiv \int_{-\infty}^{+\infty} f(x)I(x_0 - x)dx = \int_{-\infty}^{+\infty} f(x_0 - x)I(x)dx \\ &= \int_{-\infty}^{+\infty} f(x_0 - x)u(x)dx + \int_{-\infty}^{+\infty} f(x_0 - x)n(x)dx \equiv s(x_0) + g(x_0) \end{aligned} \quad (6.164)$$

where $s(x_0)$ is the output due to the deterministic signal of interest and $g(x_0)$ is the output due to noise.

The edge is detected at the local maximum of this output, that is at the place where the first derivative of $O(x_0)$ with respect to x_0 becomes 0:

$$\frac{dO(x_0)}{dx_0} = \frac{ds(x_0)}{dx_0} + \frac{dg(x_0)}{dx_0} = \int_{-\infty}^{+\infty} f'(x_0 - x)u(x)dx + \int_{-\infty}^{+\infty} f'(x_0 - x)n(x)dx \quad (6.165)$$

We expand the derivative of the filter about point $x_0 = 0$, the true position of the edge, and keep only the first two terms of the expansion:

$$f'(x_0 - x) \simeq f'(-x) + x_0 f''(-x) \quad (6.166)$$

Upon substitution into $\frac{ds(x_0)}{dx_0}$, we obtain:

$$\frac{ds(x_0)}{dx_0} \simeq \int_{-\infty}^{+\infty} f'(-x)u(x)dx + x_0 \int_{-\infty}^{+\infty} f''(-x)u(x)dx \quad (6.167)$$

The feature we want to detect is an antisymmetric feature, ie an edge that has a shape like .

According to the result proven in example 6.44, $f(x)$ should also be an antisymmetric function. This means that its first derivative will be symmetric and, when it is multiplied with the antisymmetric function $u(x)$, it will produce an antisymmetric integrand, which, upon integration over a symmetric interval, will make the first term in equation (6.167) vanish. Then:

$$\frac{ds(x_0)}{dx_0} \simeq \underbrace{x_0 \int_{-\infty}^{+\infty} f''(-x)u(x)dx}_{\text{set } \tilde{x} \equiv -x} = x_0 \int_{-\infty}^{+\infty} f''(\tilde{x})u(-\tilde{x})d\tilde{x} \quad (6.168)$$

The derivative of the filter response to noise may be written in a more convenient way:

$$\frac{dg(x_0)}{dx_0} = \int_{-\infty}^{+\infty} f'(x_0 - x)n(x)dx = \int_{-\infty}^{+\infty} f'(x)n(x_0 - x)dx \quad (6.169)$$

The position of the edge will be marked at the value of x that makes the sum of the right-hand sides of equations (6.168) and (6.169) zero:

$$\begin{aligned} \frac{ds(x_0)}{dx_0} + \frac{dg(x_0)}{dx_0} &= 0 \Rightarrow \\ x_0 \int_{-\infty}^{+\infty} f''(x)u(-x)dx + \int_{-\infty}^{+\infty} f'(x)n(x_0 - x)dx &= 0 \Rightarrow \\ x_0 \int_{-\infty}^{+\infty} f''(x)u(-x)dx &= - \int_{-\infty}^{+\infty} f'(x)n(x_0 - x)dx \end{aligned} \quad (6.170)$$

The right-hand side of this expression is a random variable, indicating that the location of the edge will be marked at various randomly distributed positions around the true position, which is at $x_0 = 0$. We can calculate the mean shifting away from the true position as the expectation value of x_0 . This, however, is expected to be 0. So, we calculate instead the variance of the x_0 values. We square both sides of (6.170) and take their expectation value:

$$E\{x_0^2\} \left[\int_{-\infty}^{+\infty} f''(x)u(-x)dx \right]^2 = E \left\{ \left[\int_{-\infty}^{+\infty} f'(x)n(x_0 - x)dx \right]^2 \right\} \quad (6.171)$$

Note that the expectation value operator applies only to the random components.

In Box 6.7 we saw that if a noise signal with variance n_0^2 is convolved with a filter $f(x)$, the mean square value of the output signal is given by:

$$n_0^2 \int_{-\infty}^{+\infty} f^2(x)dx \quad (6.172)$$

(see equation (6.159)). The right-hand side of equation (6.170) here indicates the convolution of the noise component with filter $f'(x)$. Its mean square value, therefore, is $n_0^2 \int_{-\infty}^{+\infty} [f'(x)]^2 dx$. Then:

$$E\{x_0^2\} = \frac{n_0^2 \int_{-\infty}^{+\infty} [f'(x)]^2 dx}{\left[\int_{-\infty}^{+\infty} f''(x)u(-x)dx \right]^2} \quad (6.173)$$

The smaller this expectation value is, the better the localisation of the edge. We may define, therefore, the good locality measure as the inverse of the square root of the above quantity. We may also ignore factor n_0 , as it is the standard deviation of the noise during the imaging process, over which we do not have control. So, a filter is optimal with respect to good locality, if it maximises the quantity:

$$L \equiv \frac{\int_{-\infty}^{+\infty} f''(x)u(-x)dx}{\sqrt{\int_{-\infty}^{+\infty} [f'(x)]^2 dx}} \quad (6.174)$$

Example B6.47

Show that the differentiation of the output of a convolution, of a signal $u(x)$ with a filter $f(x)$, may be achieved by convolving the signal with the derivative of the filter.

The output of the convolution is

$$s(x_0) = \int_{-\infty}^{+\infty} f(x)u(x_0 - x)dx \quad (6.175)$$

or

$$s(x_0) = \int_{-\infty}^{+\infty} f(x_0 - x)u(x)dx \quad (6.176)$$

Applying Leibniz's rule for differentiating an integral with respect to a parameter (see Box 4.9, on page 348), we obtain from equation (6.176)

$$\frac{ds(x_0)}{dx_0} = \int_{-\infty}^{+\infty} f'(x_0 - x)u(x)dx \quad (6.177)$$

which upon changing variables may be written as:

$$\frac{ds(x_0)}{dx_0} = \int_{-\infty}^{+\infty} f'(x)u(x_0 - x)dx \quad (6.178)$$

Box 6.10. Derivation of the count of false maxima

It has been shown by Rice, that the average distance of any two successive zero crossings of the convolution of a function h with Gaussian noise is given by

$$x_{av} = \pi \sqrt{\frac{-R_{hh}(0)}{R''_{hh}(0)}} \quad (6.179)$$

where $R_{hh}(\tau)$ is the spatial autocorrelation function of function $h(x)$, ie:

$$R_{hh}(\tau) = \int_{-\infty}^{+\infty} h(x)h(x + \tau)dx \quad (6.180)$$

Therefore:

$$R_{hh}(0) = \int_{-\infty}^{+\infty} (h(x))^2 dx \quad (6.181)$$

Using Leibniz's rule (see Box 4.9, on page 348) we can differentiate (6.180) with respect to τ , to obtain:

$$R'_{hh}(\tau) = \int_{-\infty}^{+\infty} h(x)h'(x + \tau)dx \quad (6.182)$$

We define a new variable of integration $\tilde{x} \equiv x + \tau \Rightarrow x = \tilde{x} - \tau$ and $dx = d\tilde{x}$ to obtain:

$$R'_{hh}(\tau) = \int_{-\infty}^{+\infty} h(\tilde{x} - \tau)h'(\tilde{x})d\tilde{x} \quad (6.183)$$

We differentiate (6.183) once more:

$$R''_{hh}(\tau) = - \int_{-\infty}^{+\infty} h'(\tilde{x} - \tau)h'(\tilde{x})d\tilde{x} \Rightarrow R''_{hh}(0) = - \int_{-\infty}^{+\infty} (h'(x))^2 dx \quad (6.184)$$

Therefore, the average distance of zero crossings of the output signal, when a noise signal is filtered with function $h(x)$, is given by:

$$x_{av} = \pi \sqrt{\frac{\int_{-\infty}^{+\infty} (h(x))^2 dx}{\int_{-\infty}^{+\infty} (h'(x))^2 dx}} \quad (6.185)$$

From the analysis in Box 6.9, we can see that the false maxima in our case will come from equation $\int_{-\infty}^{+\infty} f'(x)n(x_0 - x)dx = 0$, which will give the false alarms in the absence of any signal ($u(x) = 0$). This is equivalent to saying that the false maxima coincide with the zeros in the output signal when the noise signal is filtered with function $f'(x)$. So, if we want to reduce the number of false local maxima, we should make the average distance between the zero crossings as large as possible, for filter function $h(x) \equiv f'(x)$. Therefore, we define the good measure of scarcity of false alarms as:

$$C \equiv \sqrt{\frac{\int_{-\infty}^{+\infty} (f'(x))^2 dx}{\int_{-\infty}^{+\infty} (f''(x))^2 dx}} \quad (6.186)$$

Can edge detection lead to image segmentation?

Not directly. The detected edges are usually fragmented and have gaps, so, in general, they do not divide the image into regions. To use edge detection to segment an image, we must

follow one of two routes:

- use **hysteresis edge linking**, as proposed by Canny, in combination with some further postprocessing;
- use a **Laplacian of Gaussian** filter to detect edges.

What is hysteresis edge linking?

It is a way of postprocessing the output of the non-maxima suppression stage of an edge detector, so that more complete edges may be extracted. Instead of specifying one threshold to threshold the gradient magnitude values, we specify two thresholds. Edgels with gradient magnitude weaker than the lower threshold are considered as noise and discarded. All remaining edgels, with gradient magnitude above the lower threshold, are linked to form chains of linked pixels. If at least one of the pixels in such a chain has gradient magnitude stronger than the higher threshold, all pixels of the chain are kept as edgels.

So, the algorithm on page 602 has to be modified as follows. In Step 4, the word “threshold” refers to the *low* threshold. Then the following steps have to be added.

Step 6: In the output image identify edgels with three or more neighbouring edgels. Such edgels are junctions. Remove them from the edge map and mark them in some other array.

Step 7: After the junction edgels have been removed, the edge map consists of disconnected strings of edgels. Examine each string in turn. If at least one of its edgels has gradient magnitude above the *high* threshold, keep the whole string. If none of the edgels of the string has gradient magnitude above the *high* threshold, remove the whole string from the edge map.

Step 8: Re-insert the junction pixels you removed in Step 6, if they are adjacent to at least one edgel in the edge map.

Does hysteresis edge linking lead to closed edge contours?

No. It simply leads to longer and better edge strings. To extract closed contours, we must combine the edges we extract with some other algorithm, like **snakes** or **level set methods**, which fit flexible curves to the grey values of the image. These methods are beyond the scope of this book. Alternatively, edge information is used in the watershed algorithm, which we discussed in the previous section, and which is effectively a hybrid method that combines region and gradient information to segment the image.

Example B6.48

A realistic ramp edge at position $x = 0$ may be modelled by a sigmoid function

$$u(x) = \frac{1}{1 + e^{-sx}} \quad (6.187)$$

where s is a parameter that controls the slope of the edge. Show that the second derivative of this function is 0 at the position of the edge.

The first derivative of the model edge is:

$$\begin{aligned}
u'(x) &= s \frac{e^{-sx}}{(1 + e^{-sx})^2} \\
&= s \frac{e^{-sx}}{1 + e^{-2sx} + 2e^{-sx}} \\
&= s \frac{1}{e^{sx} + e^{-sx} + 2} \\
&= s \frac{1}{2 \cosh(sx) + 2}
\end{aligned} \tag{6.188}$$

The second derivative of $u(x)$ is:

$$u''(x) = -s^2 \frac{\sinh(sx)}{2(\cosh(sx) + 1)^2} \tag{6.189}$$

For $x = 0$ this function is 0.

Example B6.49

Show that the second derivative of a Gaussianly smoothed signal may be obtained by convolving the signal with the second derivative of the Gaussian function.

Let us call the signal $f(x)$, the smoothing Gaussian filter $g(x)$ and the smoothed signal $o(x)$. We have:

$$o(x) = \int_{-\infty}^{+\infty} f(x - \tilde{x}) g(\tilde{x}) d\tilde{x} \tag{6.190}$$

In example 6.47 we saw that the derivative of $o(x)$ may be obtained by convolving $f(x)$ with the derivative of the filter:

$$o'(x) = \int_{-\infty}^{+\infty} g'(\tilde{x}) f(x - \tilde{x}) d\tilde{x} \tag{6.191}$$

This may be written as:

$$o'(x) = \int_{-\infty}^{+\infty} g'(\tilde{x}) f(\tilde{x}) d\tilde{x} \tag{6.192}$$

Then application again of the same rule (ie derivative of output=convolution with the derivative of the filter) yields:

$$o''(x) = \int_{-\infty}^{+\infty} g''(\tilde{x}) f(x - \tilde{x}) d\tilde{x} \tag{6.193}$$

What is the Laplacian of Gaussian edge detection method?

The second derivative of the image becomes 0 at the positions of edges (see example 6.48). The second derivative of the image may be computed by convolving it with the second derivative of a smoothing filter, like for example the Gaussian filter. This filter may be used along two orthogonal image directions, or we may try to implement it as a radially symmetric filter, like filter (4.122), on page 352, which was used to enhance blob-like structures in the image in Chapter 4.

This filter effectively estimates the sum of the second differences of the image at each pixel. To identify then the positions of the edges, we must identify the places where this output becomes 0. Due to digital effects, there may not be many such places. However, the output changes sign at a zero crossing, so all we have to do is to identify the places where the output changes sign, even if these places are in between two adjacent pixels. These places correspond to edges. Because of this, this filter allows the detection of edges as closed contours which delineate image regions.

Figure 6.57 shows the edges of an image produced by considering the zero crossings of the output of the image convolved with filter (4.122), of various sizes, identified by checking the sign of the output signal between any two horizontally and vertically adjacent pixels.

The algorithm used to produce these outputs is as follows.

Step 0: Create an output array the same size as your input image and set all its pixels equal to 255 (white).

Step 1: Select the size of filter (4.122) you will use and compute its weights using the procedure described in example 4.22, on page 352.

Step 2: Convolve the input image g with this filter to produce result c .

Step 3: Scan the image line by line and at each pixel (i, j) , examine the sign of the product $c(i, j)c(i + 1, j)$. If this product is less or equal to 0, mark pixel (i, j) in the output array as an edge, by setting its value equal to 0.

Step 4: Scan the image column by column and at each pixel (i, j) , examine the sign of the product $c(i, j)c(i, j + 1)$. If this product is less or equal to 0, mark pixel (i, j) in the output array as an edge, by setting its value equal to 0.

If you wish to create “thick” edges, every time $c(i, j)c(i + 1, j)$ or $c(i, j)c(i, j + 1)$ is less or equal to 0, mark both pixels as edges, ie turn to 0 the values of both (i, j) and $(i + 1, j)$, or (i, j) and $(i, j + 1)$, respectively.

Is it possible to detect edges and lines simultaneously?

Yes. There is a method based on the so called **phase congruency** that allows the simultaneous detection of edges and lines. However, this method is much more general and it allows the simultaneous detection of pairs of symmetric-antisymmetric image features of various types.

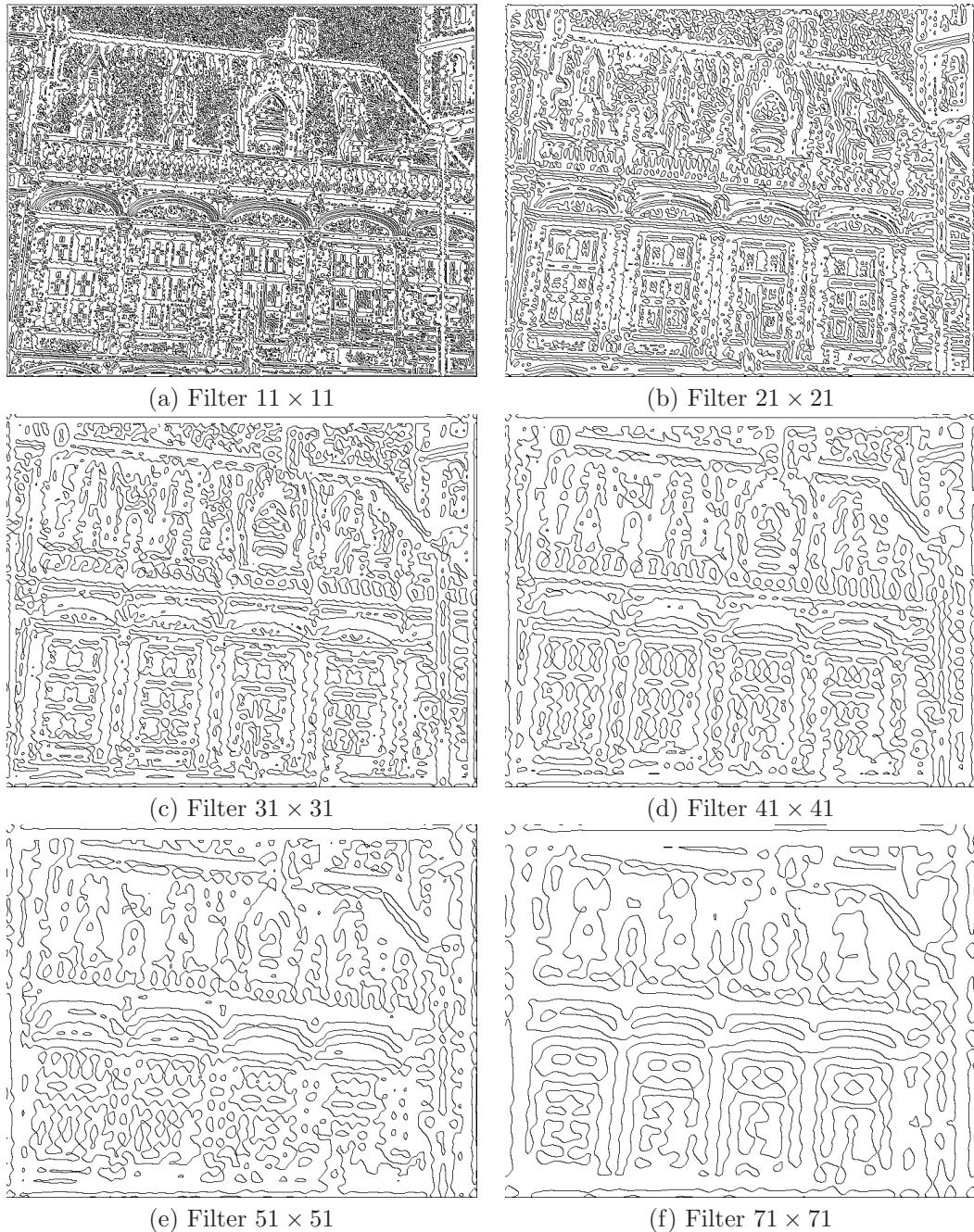


Figure 6.57: Edge detection for image 4.25, on page 352, (of size 573×723) by using filter (4.122), on page 352, in various sizes. All contours identified are closed. The larger the filter, the grosser the edges identified, due to the smoothing component of the filter.

6.3 Phase congruency and the monogenic signal

What is phase congruency?

Phase congruency is the situation when all harmonics of the Fourier series expansion of a periodic signal have the same phase. Consider a 1D continuous signal $f(t)$, defined for $-T \leq t \leq T$, and its representation as a Fourier series:

$$\begin{aligned} f(t) &= \frac{a_0}{2} + \sum_{n=1}^{+\infty} a_n \cos \frac{\pi n t}{T} + \sum_{n=1}^{+\infty} b_n \sin \frac{\pi n t}{T} \\ &= \frac{a_0}{2} + \sum_{n=1}^{+\infty} \sqrt{a_n^2 + b_n^2} \left[\frac{a_n}{\sqrt{a_n^2 + b_n^2}} \cos \frac{\pi n t}{T} + \frac{b_n}{\sqrt{a_n^2 + b_n^2}} \sin \frac{\pi n t}{T} \right] \\ &\equiv \frac{a_0}{2} + \sum_{n=1}^{+\infty} A_n \cos \left(\frac{\pi n t}{T} - \phi_n \right) \end{aligned} \quad (6.194)$$

The coefficients of the series expansion are given by:

$$\begin{aligned} a_n &= \frac{1}{T} \int_{-T}^T f(t) \cos \frac{\pi n t}{T} dt \\ b_n &= \frac{1}{T} \int_{-T}^T f(t) \sin \frac{\pi n t}{T} dt \end{aligned} \quad (6.195)$$

In (6.194) the amplitude A_n of the expansion was defined as

$$A_n \equiv \sqrt{a_n^2 + b_n^2} \quad (6.196)$$

and the phase ϕ_n such that:

$$\cos \phi_n \equiv \frac{a_n}{\sqrt{a_n^2 + b_n^2}} \quad \text{and} \quad \sin \phi_n \equiv \frac{b_n}{\sqrt{a_n^2 + b_n^2}} \quad (6.197)$$

We also made use of $\cos \alpha \cos \beta + \sin \alpha \sin \beta = \cos(\alpha - \beta)$.

We have phase congruency at a point t when $\frac{\pi n t}{T} - \phi_n$ has the same value, modulo 2π , for all indices n .

What is phase congruency for a 1D digital signal?

A $(2N + 1)$ -sample long digital signal may be represented by its DFT, which treats the signal as if it were periodic with period N . So, for a digital real signal $f(i)$, with $i = -N, -N + 1, \dots, 0, \dots, N - 1, N$, we have

$$f(i) = \sum_{k=-N}^N F(k) e^{j \frac{2\pi k i}{2N+1}} \quad (6.198)$$

where $F(k)$ is usually complex and given by:

$$F(k) = \frac{1}{2N+1} \sum_{i=-N}^N f(i)e^{-j\frac{2\pi ki}{2N+1}} \quad (6.199)$$

We may write $F(k) \equiv A_k e^{j\phi_k}$, so that A_k is a non-negative real number and ϕ_k is such that $A_k \cos \phi_k = \text{Real}\{F(k)\}$ and $A_k \sin \phi_k = \text{Imaginary}\{F(k)\}$. Then (6.198) may be written as:

$$\begin{aligned} f(i) &= \sum_{k=-N}^N A_k e^{j(\frac{2\pi ki}{2N+1} + \phi_k)} \\ &= \sum_{k=-N}^N A_k \cos\left(\frac{2\pi ki}{2N+1} + \phi_k\right) + j \sum_{k=-N}^N A_k \sin\left(\frac{2\pi ki}{2N+1} + \phi_k\right) \\ &= \sum_{k=-N}^N A_k \cos\left(\frac{2\pi ki}{2N+1} + \phi_k\right) \end{aligned} \quad (6.200)$$

The last equality follows because $f(i)$ is a real signal and so the imaginary component of its expansion must be 0. This signal exhibits phase congruency at a sample i , when angles $\frac{2\pi ki}{2N+1} + \phi_k$ for all values of k have the same value, modulo 2π .

How does phase congruency allow us to detect lines and edges?

It has been observed that phase congruency coincides with high local energy in a zero mean signal and that when there is an edge or a line, the signal exhibits high local energy.

Why does phase congruency coincide with the maximum of the local energy of the signal?

Let us consider what the DFT does to a signal that has zero mean: it maps a 1D signal to a 2D space, where along one axis we measure the real part of a “vector” and along the other its imaginary part. The mapping is such that the value of the original signal at a point is written as a sum of many such vectors. The energy of the signal at that point is defined as the magnitude of the sum vector: expansion (6.200) for signal $f(i)$ may be written as

$$f(i) = \sum_{k=-N}^N \left[A_k \cos\left(\frac{2\pi ki}{2N+1} + \phi_k\right) + j A_k \sin\left(\frac{2\pi ki}{2N+1} + \phi_k\right) \right] \quad (6.201)$$

So, the local energy of the signal at point i , being the magnitude of the sum vector, is:

$$E(i) \equiv \sqrt{\left[\sum_{k=-N}^N A_k \cos\left(\frac{2\pi ki}{2N+1} + \phi_k\right) \right]^2 + \left[\sum_{k=-N}^N A_k \sin\left(\frac{2\pi ki}{2N+1} + \phi_k\right) \right]^2} \quad (6.202)$$

Figure 6.58 shows schematically how the constituent vectors are added to produce the sum vector. Note that the sum vector we obtain will have the largest possible length if

all constituent vectors point towards the same direction. This happens when the ratio of the two components of each vector is constant for all vectors. The vectors we add are: $\left(A_k \cos \left(\frac{2\pi k i}{2N+1} + \phi_k \right), A_k \sin \left(\frac{2\pi k i}{2N+1} + \phi_k \right) \right)$. The ratio of the two components of such a vector is

$$\frac{A_k \sin \left(\frac{2\pi k i}{2N+1} + \phi_k \right)}{A_k \cos \left(\frac{2\pi k i}{2N+1} + \phi_k \right)} = \tan \left(\frac{2\pi k i}{2N+1} + \phi_k \right) \quad (6.203)$$

This ratio is constant when angle $\frac{2\pi k i}{2N+1} + \phi_k$ is the same (modulo π) for all k , ie when we have phase congruency.

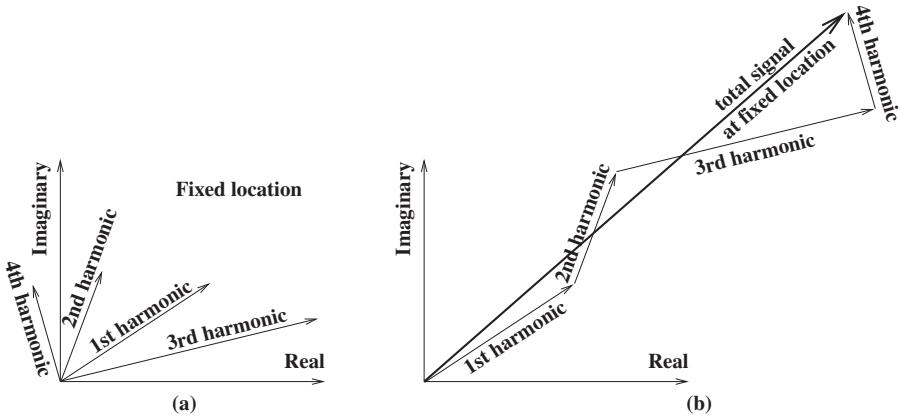


Figure 6.58: For a fixed location (signal sample), the signal is made up from the addition of several harmonic vectors.

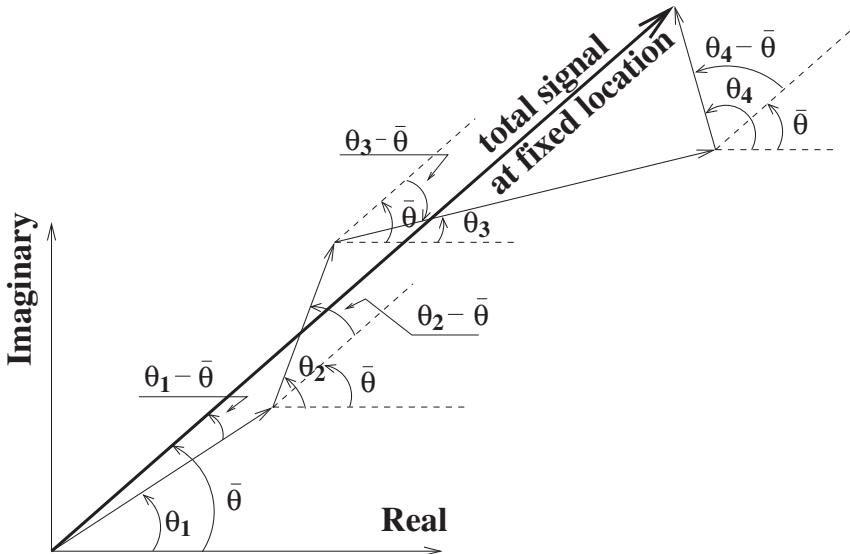
How can we measure phase congruency?

Let us denote the phase of harmonic k as $\theta_k(i) \equiv \frac{2\pi k i}{2N+1} + \phi_k$, for i fixed. Let us define the orientation of the sum vector in figure 6.58 as $\bar{\theta}(i)$. This is shown pictorially in figure 6.59a. So, the projection of each added vector on the sum vector, that represents the signal, is actually $A_k \cos(\theta_k(i) - \bar{\theta}(i))$, as shown in figure 6.59b. If all $\theta_k(i)$ point in the same direction, we have perfect phase congruency. In all other situations, we may quantify the degree of phase congruency by using the quantity:

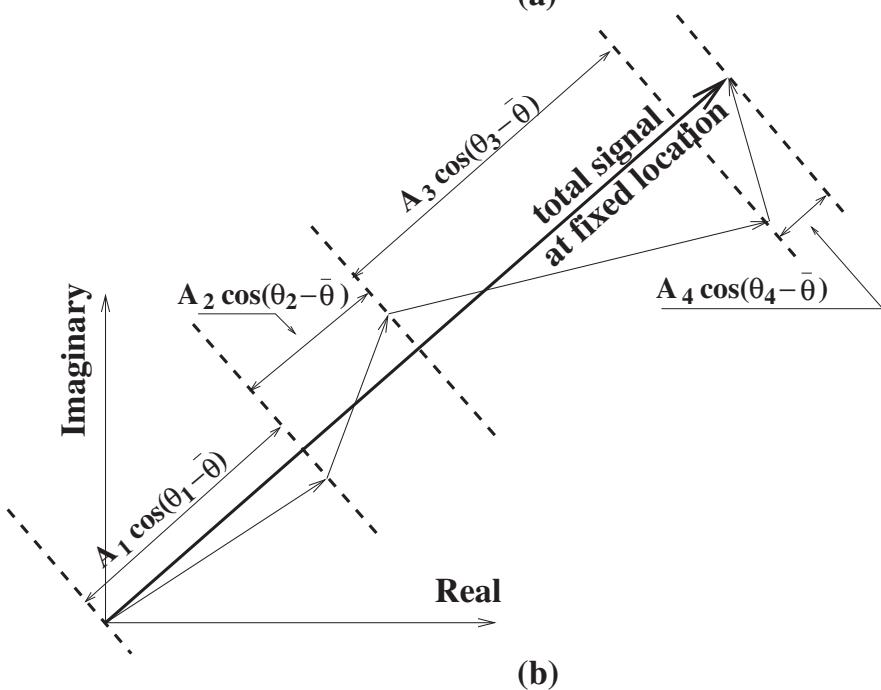
$$P_c(i) \equiv \frac{\sum_k A_k \cos(\theta_k(i) - \bar{\theta}(i))}{\sum_k A_k} \quad (6.204)$$

Couldn't we measure phase congruency by simply averaging the phases of the harmonic components?

Yes, but not in a straightforward way. Angles are measured modulo 360° . An angle of 1° and an angle of 358° point almost in the same direction, but their average is an angle of 179.5°



(a)



(b)

Figure 6.59: For a fixed location (signal sample), the signal is made up from the projections of the harmonic amplitudes on the signal direction. In (a) the relative orientations of the various harmonics in relation to the orientation of the total signal are marked. In (b) the projections of the various harmonics along the direction of the signal are added to make up the signal.

pointing in the opposite direction. There is a small trick one can use to average such numbers. Let us assume that we have to average five angles, $\theta_1, \dots, \theta_5$. We add to each angle 360° and, thus, create a set of ten angles (the original five ones plus the new five ones we created). We rank them in increasing order and we take the standard deviation of every successive five angles. The set of five angles that has the smallest standard deviation is the desired set. Its average is the average of the original angles.

This process is not very useful for large sets of angles to be averaged, so we prefer to use definition (6.204) to quantify phase congruency.

Example 6.50

**Calculate the average and the median angle of angles:
 $20^\circ, 40^\circ, 290^\circ, 130^\circ$ and 320° .**

We add 360° to each given angle and, thus, we obtain angles $380^\circ, 400^\circ, 650^\circ, 490^\circ$ and 680° . We rank in increasing order the set of ten angles:

$$20^\circ, 40^\circ, 130^\circ, 290^\circ, 320^\circ, 380^\circ, 400^\circ, 490^\circ, 650^\circ, 680^\circ$$

We consider every successive five and compute their mean and variance:

$$\begin{aligned} \mu_1 &= \frac{20 + 40 + 130 + 290 + 320}{5} = 160 \\ \sigma_1^2 &= \frac{(20 - 160)^2 + (40 - 160)^2 + (130 - 160)^2 + (290 - 160)^2 + (320 - 160)^2}{5} \\ &= 15480 \end{aligned} \quad (6.205)$$

We do the same for the remaining four sets of successive five angles (the last set is expected to be identical in terms of variance with the first set of numbers we started with):

$$\begin{aligned} \mu_2 &= \frac{40 + 130 + 290 + 320 + 380}{5} = 232 & \sigma_2^2 &= 16056 \\ \mu_3 &= \frac{130 + 290 + 320 + 380 + 400}{5} = 304 & \sigma_3^2 &= 9144 \\ \mu_4 &= \frac{290 + 320 + 380 + 400 + 490}{5} = 376 & \sigma_4^2 &= 4824 \\ \mu_5 &= \frac{320 + 380 + 400 + 490 + 650}{5} = 448 & \sigma_5^2 &= 13176 \end{aligned} \quad (6.206)$$

The set with the minimum variance is set 4 and its mean is 376. Therefore, the average angle of the original angles is $376 - 360 = 16^\circ$ and the median is $380 - 360 = 20^\circ$.

Example 6.51

Calculate the value of $\bar{\theta}$ in definition (6.204), on page 627.

This is the orientation of the sum vector (the total signal) given by equation (6.201). So, angle $\bar{\theta}$ is:

$$\bar{\theta} = \tan^{-1} \frac{\sum_{k=-N}^N A_k \sin\left(\frac{2\pi k i}{2N+1} + \phi_k\right)}{\sum_{k=-N}^N A_k \cos\left(\frac{2\pi k i}{2N+1} + \phi_k\right)} \quad (6.207)$$

Care should be taken so that the angle is in the range $[0, 360^\circ]$. To make sure that we define the angle correctly, in the right quadrant of the trigonometric circle, it is better to define it using its sine and cosine (or simply check the signs of the sine and cosine functions):

$$\begin{aligned} \cos \bar{\theta} &= \frac{\sum_{k=-N}^N A_k \cos\left(\frac{2\pi k i}{2N+1} + \phi_k\right)}{\sqrt{\left[\sum_{k=-N}^N A_k \sin\left(\frac{2\pi k i}{2N+1} + \phi_k\right)\right]^2 + \left[\sum_{k=-N}^N A_k \cos\left(\frac{2\pi k i}{2N+1} + \phi_k\right)\right]^2}} \\ \sin \bar{\theta} &= \frac{\sum_{k=-N}^N A_k \sin\left(\frac{2\pi k i}{2N+1} + \phi_k\right)}{\sqrt{\left[\sum_{k=-N}^N A_k \sin\left(\frac{2\pi k i}{2N+1} + \phi_k\right)\right]^2 + \left[\sum_{k=-N}^N A_k \cos\left(\frac{2\pi k i}{2N+1} + \phi_k\right)\right]^2}} \end{aligned} \quad (6.208)$$

How do we measure phase congruency in practice?

Since the maxima of the phase congruency coincide with the maxima of the local energy of the signal, we measure the local signal energy and identify its local maxima, in order to identify the places where signal features (eg edges or lines) are present.

How do we measure the local energy of the signal?

We note that in order to measure the local signal energy, we analyse the signal into its harmonic components, that are defined in a 2D space (see figure 6.58a). All we need then to do is to define an appropriate basis for this space and identify our signal components with respect to this basis. We note that a zero-mean real symmetric signal will have all its harmonic components coincide with the real axis of figure 6.58a. (This is the basis of the cosine transform we saw in Chapter 2.) We also note that a zero-mean real antisymmetric signal will have all its harmonic components coincide with the imaginary axis of figure 6.58a. (This is the basis of the sine transform we saw in Chapter 2.) So, if we define two signals that are real and with zero-mean, and one is symmetric and the other antisymmetric, and they are orthogonal to each other *in the frequency domain*, we shall have an orthogonal basis for this space. If the two signals are also of unit length each, we shall have an orthonormal basis.

One way to make sure that two signals are orthogonal in the frequency domain is to define them so that their phases are in quadrature, ie at 90° from each other. In particular,

starting from one basis function, we may construct the other, by rotating its positive frequency components by $+90^\circ$, and its negative frequency components by -90° . This is explained schematically in figure 6.60. Functions that have Fourier transforms that differ only by a phase shift of $+90^\circ$ in the positive frequencies and by -90° in the negative frequencies are said to constitute a **Hilbert transform** pair. These two basis signals, in the time domain, they should be of the same size as the segment of the signal we wish to use in order to define the local energy, ie the size of the basis signals in the time domain is defined by what we mean with the word “local”. Then all we have to do is to project the signal on these two basis signals, square and add the two components and thus obtain the local energy of the signal.

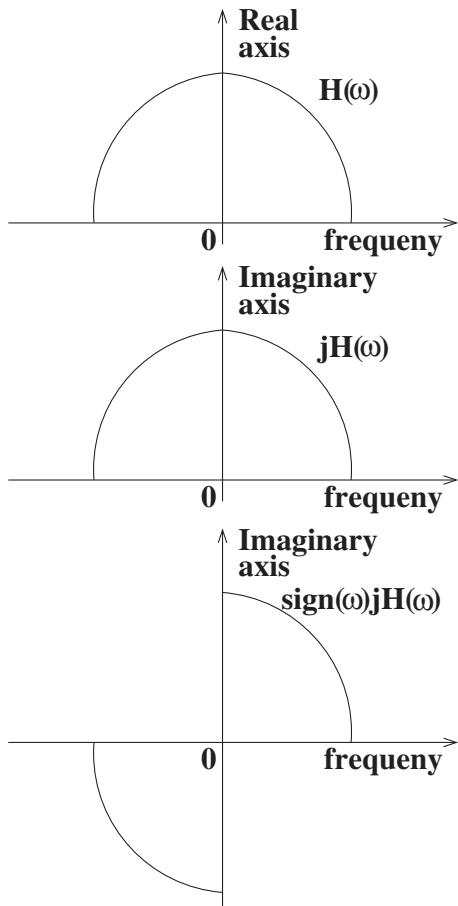


Figure 6.60: A symmetric filter has a real Fourier transform (top). As the filter is real in the time domain, its Fourier transform is symmetric about the 0 frequency. If we multiply this Fourier transform with j , it will refer to the imaginary axis (middle). If we multiply it with $-j$ for the negative frequencies and with $+j$ for the positive ones (bottom), we create a filter that is orthogonal to the original one, in the frequency domain: if we multiply point by point the function at the top with the function at the bottom and sum up the results, we get 0.

For example, signal $(-1, 2, -1)$ is a symmetric signal that may be used to represent the imaginary axis in the 2D Fourier space. Signal $(-1, 0, +1)$ an antisymmetric signal that may be used to represent the imaginary axis in the 2D Fourier space. The two signals are orthogonal to each other in the time domain too, as their dot product is $(-1) \times (-1) + 2 \times 0 + (-1) \times (+1) = 0$. So, they constitute an orthogonal basis for the space of interest. We can make the basis orthonormal if we divide the elements of the symmetric signal with $\sqrt{(-1)^2 + 2^2 + (-1)^2} = \sqrt{6}$ and the elements of the antisymmetric signal with $\sqrt{(-1)^2 + (+1)^2} = \sqrt{2}$. Then all we have to do, in order to work out the local energy of the signal at *every* point, is to convolve the signal with these two basis signals (=filters), get the two convolution outputs, square the value of each sample of each output, and sum the two squared results.

Why should we perform convolution with the two basis signals in order to get the projection of the local signal on the basis signals?

Projection of the signal on the two unit signals (basis signals) involves taking every sample of the signal and its local neighbourhood, the same size as the unit signals, and multiplying them point by point and adding the products. This is effectively convolution if we ignore the reversal of the convolving signal we have to do to perform real convolution. We may call such a convolution **pseudo-convolution**. This is shown schematically in figure 6.61. The pseudo-convolution is what we usually do when we deal with images: the pseudo-convolution is equivalent to the projection of one signal on the other. The fact that we do not perform the reversal is irrelevant when the unit (or basis, or filter) signal is symmetric and it results in sign reversal if the unit (or basis, or filter) signal is antisymmetric. However, as we get the square of each resultant value, a change in sign is irrelevant.

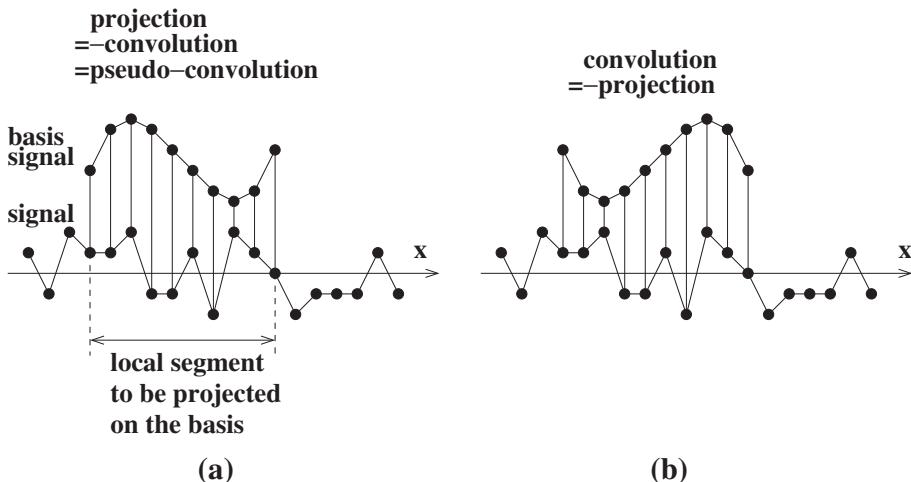


Figure 6.61: (a) The projection of a segment of the signal on a basis signal involves the multiplication of the segment with the basis signal point by point and the addition of the results. (b) Convolution, according to its algebraic definition, involves the reversal of the basis signal before we perform the point by point multiplication and addition. Usually, in image processing, we “forget” this reversal (because the sign of the result is usually ignored) and we actually perform a pseudo-convolution.

Example 6.52

Consider the filter $(-1/\sqrt{6}, 2/\sqrt{6}, -1/\sqrt{6})$. Compute its DFT and define a filter with which it can be combined to form a basis for the complex frequency space.

We shall use (6.199) with $N = 1$ to compute the DFT of the filter:

$$F(k) = \frac{1}{3} \frac{1}{\sqrt{6}} \left[-e^{j\frac{2\pi k}{3}} + 2 - e^{-j\frac{2\pi k}{3}} \right] \quad (6.209)$$

Next, we work out the DFT components by allowing k to take values $-1, 0$ and 1 .

$$\begin{aligned} F(-1) &= \frac{1}{3\sqrt{6}} \left[-e^{-j\frac{2\pi}{3}} + 2 - e^{j\frac{2\pi}{3}} \right] \\ &= \frac{1}{3\sqrt{6}} \left[-\cos \frac{2\pi}{3} + j \sin \frac{2\pi}{3} + 2 - \cos \frac{2\pi}{3} - j \sin \frac{2\pi}{3} \right] \\ &= \frac{1}{3\sqrt{6}} \left[-2 \cos \frac{2\pi}{3} + 2 \right] \\ &= \frac{1}{3\sqrt{6}} \left[-2 \times \left(-\frac{1}{2} \right) + 2 \right] = \frac{1}{\sqrt{6}} \\ F(0) &= 0 \\ F(1) &= \frac{1}{3\sqrt{6}} \left[-e^{j\frac{2\pi}{3}} + 2 - e^{-j\frac{2\pi}{3}} \right] \\ &= \frac{1}{3\sqrt{6}} \left[-\cos \frac{2\pi}{3} - j \sin \frac{2\pi}{3} + 2 - \cos \frac{2\pi}{3} + j \sin \frac{2\pi}{3} \right] \\ &= \frac{1}{\sqrt{6}} \end{aligned} \quad (6.210)$$

The positive frequency component (the $k = 1$ component) of the DFT of the orthogonal filter should be at $+90^\circ$ from $F(1)$ and the negative frequency component (the $k = -1$ component) of the DFT of the orthogonal filter should be at -90° from $F(-1)$. Therefore, each of the above $F(k)$ has to be multiplied either with $e^{j\frac{\pi}{2}}$ to change its phase by $+90^\circ$, or with $e^{-j\frac{\pi}{2}}$ to change its phase by -90° , in order to yield the DFT components of the orthogonal filter. So, the components of the orthogonal filter are:

$$\begin{aligned} \tilde{F}(-1) &= \frac{1}{\sqrt{6}} e^{-j\frac{\pi}{2}} \\ \tilde{F}(0) &= 0 \\ \tilde{F}(1) &= \frac{1}{\sqrt{6}} e^{j\frac{\pi}{2}} \end{aligned} \quad (6.211)$$

We use these values then in (6.198) to construct the orthogonal filter:

$$\begin{aligned}
\tilde{f}(i) &= \sum_{k=-1}^1 \tilde{F}(k) e^{j \frac{2\pi k i}{3}} \\
&= \frac{1}{\sqrt{6}} \left[e^{-j \frac{\pi}{2}} e^{-j \frac{2\pi i}{3}} + e^{j \frac{\pi}{2}} e^{j \frac{2\pi i}{3}} \right] \\
&= \frac{1}{\sqrt{6}} \left[e^{-j \frac{\pi}{6}(3+4i)} + e^{j \frac{\pi}{6}(3+4i)} \right] \\
&= \frac{1}{\sqrt{6}} 2 \cos \frac{\pi(3+4i)}{6}
\end{aligned} \tag{6.212}$$

To work out the filter weights now, we allow i to take values $-1, 0$ and 1 :

$$\begin{aligned}
\tilde{f}(-1) &= \frac{1}{\sqrt{6}} \sqrt{3} \\
&= \frac{1}{\sqrt{2}} \\
\tilde{f}(0) &= 0 \\
\tilde{f}(1) &= \frac{1}{\sqrt{6}} 2 \cos \frac{7\pi}{6} \\
&= -\frac{1}{\sqrt{2}}
\end{aligned} \tag{6.213}$$

So, filters

$$\left(-\frac{1}{\sqrt{6}}, \frac{2}{\sqrt{6}}, -\frac{1}{\sqrt{6}} \right) \quad \text{and} \quad \left(\frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}} \right) \tag{6.214}$$

constitute a Hilbert transform pair.

Example 6.53

Consider the filter $(-\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$. Compute its DFT and define a filter with which it can be combined to form a basis for the complex frequency space.

We shall use (6.199) with $N = 1$ to compute the DFT of the filter:

$$F(k) = \frac{1}{3} \frac{1}{\sqrt{2}} \left[-e^{j \frac{2\pi k}{3}} + e^{-j \frac{2\pi k}{3}} \right] \tag{6.215}$$

Next, we work out the DFT components by allowing k to take values $-1, 0$ and 1 .

$$\begin{aligned}
F(-1) &= \frac{1}{3\sqrt{2}} \left[-e^{-j\frac{2\pi}{3}} + e^{j\frac{2\pi}{3}} \right] \\
&= \frac{1}{3\sqrt{2}} \left[-\cos \frac{2\pi}{3} + j \sin \frac{2\pi}{3} + \cos \frac{2\pi}{3} + j \sin \frac{2\pi}{3} \right] \\
&= \frac{1}{3\sqrt{2}} 2j \sin \frac{2\pi}{3} \\
&= j \frac{1}{3\sqrt{2}} \sqrt{3} = j \frac{1}{\sqrt{6}} \\
F(0) &= 0 \\
F(1) &= \frac{1}{3\sqrt{2}} \left[-e^{j\frac{2\pi}{3}} + e^{-j\frac{2\pi}{3}} \right] \\
&= \frac{1}{3\sqrt{2}} \left[-\cos \frac{2\pi}{3} - j \sin \frac{2\pi}{3} + \cos \frac{2\pi}{3} - j \sin \frac{2\pi}{3} \right] \\
&= -j \frac{1}{\sqrt{6}}
\end{aligned} \tag{6.216}$$

The $k = 1$ component of the DFT of the orthogonal filter should be at $+90^\circ$ from $F(1)$ and the $k = -1$ component should be at -90° from $F(-1)$. So, the components of the orthogonal filter are:

$$\begin{aligned}
\tilde{F}(-1) &= j \frac{1}{\sqrt{6}} e^{-j\frac{\pi}{2}} = \frac{1}{\sqrt{6}} \\
\tilde{F}(0) &= 0 \\
\tilde{F}(1) &= -j \frac{1}{\sqrt{6}} e^{j\frac{\pi}{2}} = \frac{1}{\sqrt{6}}
\end{aligned} \tag{6.217}$$

We use these values then in (6.198) to construct the orthogonal filter:

$$\begin{aligned}
\tilde{f}(i) &= \sum_{k=-1}^1 \tilde{F}(k) e^{j\frac{2\pi k i}{3}} \\
&= \frac{1}{\sqrt{6}} \left[e^{-j\frac{2\pi i}{3}} + e^{j\frac{2\pi i}{3}} \right] \\
&= \frac{1}{\sqrt{6}} 2 \cos \frac{2\pi i}{3}
\end{aligned} \tag{6.218}$$

To work out the filter weights now, we allow i to take values $-1, 0$ and 1 :

$$\begin{aligned}
\tilde{f}(-1) &= \frac{1}{\sqrt{6}} 2 \cos \frac{2\pi}{3} = -\frac{1}{\sqrt{6}} \\
\tilde{f}(0) &= \frac{2}{\sqrt{6}} \\
\tilde{f}(1) &= -\frac{1}{\sqrt{6}}
\end{aligned} \tag{6.219}$$

Example 6.54

Construct the Hilbert transform pair of filter $(-1, 1, 1, -1)$.

First compute the DFT of the given filter. The centre of the filter is between its two central samples, ie its samples are at half integer positions. Its DFT then is given by

$$\begin{aligned} F(k) &= \frac{1}{4} \sum_{i=-2}^1 f(i) e^{-j \frac{2\pi k(i+0.5)}{4}} \\ &= \frac{1}{4} \left[-e^{j \frac{2\pi k 3}{8}} + e^{j \frac{2\pi k}{8}} + e^{-j \frac{2\pi k}{8}} - e^{-j \frac{2\pi k 3}{8}} \right] \\ &= \frac{1}{2} \left[-\cos \frac{\pi k 3}{4} + \cos \frac{\pi k}{4} \right] \end{aligned} \quad (6.220)$$

$$\begin{aligned} F(0) &= 0 \\ F(1) &= \frac{1}{2} \left[-\cos \frac{3\pi}{4} + \cos \frac{\pi}{4} \right] = \frac{1}{\sqrt{2}} \\ F(2) &= \frac{1}{2} \left[-\cos \frac{3\pi}{2} + \cos \frac{\pi}{2} \right] = 0 \\ F(3) &= \frac{1}{2} \left[-\cos \frac{\pi}{4} + \cos \frac{3\pi}{4} \right] = -\frac{1}{\sqrt{2}} \end{aligned} \quad (6.221)$$

The elements that correspond to the negative frequencies are elements $F(2)$ and $F(3)$. We multiply the positive frequency component (namely $F(1)$) with $e^{j\frac{\pi}{2}}$ and the nonzero negative frequency component, namely $F(3)$, with $e^{-j\frac{\pi}{2}}$ to produce the DFT of the desired filter:

$$\begin{aligned} \tilde{F}(0) &= 0 \\ \tilde{F}(1) &= \frac{1}{\sqrt{2}} e^{j \frac{\pi}{2}} \\ \tilde{F}(2) &= 0 \\ \tilde{F}(3) &= -\frac{1}{\sqrt{2}} e^{-j \frac{\pi}{2}} \end{aligned} \quad (6.222)$$

We then take the inverse DFT to produce the filter:

$$\begin{aligned} \tilde{f}(i) &= \sum_{k=0}^3 \tilde{F}(k) e^{j \frac{2\pi k(i+0.5)}{4}} \\ &= \frac{1}{\sqrt{2}} e^{j \frac{\pi(i+1.5)}{2}} - \frac{1}{\sqrt{2}} e^{j \frac{\pi(3i+0.5)}{2}} \end{aligned} \quad (6.223)$$

$$\begin{aligned}
\tilde{f}(-2) &= \frac{1}{\sqrt{2}} \left[e^{-j\frac{\pi}{4}} - e^{-j\frac{11\pi}{4}} \right] \\
&= \frac{1}{\sqrt{2}} \left[\cos \frac{\pi}{4} - j \sin \frac{\pi}{4} - \cos \frac{11\pi}{4} + j \sin \frac{11\pi}{4} \right] = 1 \\
\tilde{f}(-1) &= \frac{1}{\sqrt{2}} \left[e^{j\frac{\pi}{4}} - e^{-j\frac{5\pi}{4}} \right] = 1 \\
\tilde{f}(0) &= \frac{1}{\sqrt{2}} \left[e^{j\frac{3\pi}{4}} - e^{j\frac{\pi}{4}} \right] = -1 \\
\tilde{f}(1) &= \frac{1}{\sqrt{2}} \left[e^{j\frac{5\pi}{4}} - e^{j\frac{7\pi}{4}} \right] = -1
\end{aligned} \tag{6.224}$$

So, filters $(-1, 1, 1, -1)$ and $(1, 1, -1, -1)$ constitute a Hilbert transform pair.

Box 6.11. Some properties of the continuous Fourier transform

The Fourier transform of a function $f(x)$ is $F(\omega)$, defined as:

$$F(\omega) \equiv \int_{-\infty}^{+\infty} f(x) e^{-j\omega x} dx \tag{6.225}$$

The inverse transform may be worked out as:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega) e^{j\omega x} d\omega \tag{6.226}$$

Some important properties of the Fourier transform are tabulated below.

Real domain	Frequency domain	Name of the property
$f(x)$	$F(\omega)$	Definition of the Fourier transform
$F(x)$	$2\pi f(-\omega)$	Duality
$\frac{d^n f(x)}{dx^n}$	$(j\omega)^n F(\omega)$	FT of the n th derivative of a function
$\int_{-\infty}^x f(y) dy$	$-\frac{j}{\omega} F(\omega)$	FT of the integral of a function
$f(\alpha x)$	$\frac{1}{\alpha} F\left(\frac{\omega}{\alpha}\right)$	Scaling property
$f(x + \alpha)$	$e^{j\alpha\omega} F(\omega)$	Shifting property (real domain)
$f(x)e^{-j\alpha x}$	$F(\omega + \alpha)$	Shifting property (frequency domain)
$\int_{-\infty}^{+\infty} f(x) ^2 dx$	$= \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega) ^2 d\omega$	Parseval's theorem
1	$2\pi\delta(\omega)$	Fourier transform of a constant
$\delta(x)$	1	Fourier transform of the delta function

Example B6.55

Prove the duality property of the Fourier transform.

Let us consider the Fourier transform of function $F(x)$, as defined by equation (6.225):

$$\begin{aligned}
 \hat{F}(\omega) &= \int_{-\infty}^{+\infty} F(x)e^{-j\omega x}dx \\
 &= \int_{-\infty}^{+\infty} \left[\int_{-\infty}^{+\infty} f(y)e^{-jxy}dy \right] e^{-j\omega x}dx \\
 &= \int_{-\infty}^{+\infty} f(y) \left[\int_{-\infty}^{+\infty} e^{j(-y-\omega)x}dx \right] dy \\
 &= \int_{-\infty}^{+\infty} f(y)2\pi\delta(-y-\omega)dy \\
 &= \int_{-\infty}^{+\infty} f(y)2\pi\delta(y+\omega)dy \\
 &= 2\pi f(-\omega)
 \end{aligned} \tag{6.227}$$

Here we made use of the fact that the Fourier transform of the delta function is 1, which, by considering the inverse Fourier transform, immediately leads to the identity:

$$\delta(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{j\omega x}d\omega \tag{6.228}$$

We also used the property of delta function $\delta(-x) = \delta(x)$.

Example B6.56

The Fourier transform of a signal $f(x)$ is $F(\omega)$. We multiply $F(\omega)$ with function $H(\omega) \equiv \text{sgn}(\omega)j$ to obtain the Fourier transform $\tilde{F}(\omega)$, of a signal $\tilde{f}(x)$. Note that

$$\text{sgn}(\omega) = \begin{cases} -1 & \text{if } \omega < 0 \\ 0 & \text{if } \omega = 0 \\ +1 & \text{if } \omega > 0 \end{cases} \tag{6.229}$$

How could you obtain signal $\tilde{f}(x)$ directly from $f(x)$, without involving the Fourier transform?

Multiplication in the frequency domain is equivalent to convolution of the corresponding

functions in the real domain. So, to obtain $\tilde{f}(x)$ directly from $f(x)$, we shall have to convolve $f(x)$ with the inverse Fourier transform of function $H(\omega)$.

Let us call it function $h(x)$. Let us consider function $\text{sgn}(\omega)$ first. This is a step function. Let us call the Fourier transform of step function $\text{sgn}(x)$, $U(\omega)$. The derivative of function $\text{sgn}(x)$ will have Fourier transform $-j\omega U(\omega)$ (see Box 6.11). However, the derivative of a step function is a delta function $\delta(x)$ and the Fourier transform of the delta function is constant 1. So, we have that $-j\omega U(\omega) = 1$ and this implies that $U(\omega) = -1/(j\omega) = j/\omega$. So, this means that step function $\text{sgn}(x)$ and function j/ω constitute a Fourier transform pair. According to the duality property of the Fourier transform (see Box 6.11), the Fourier transform of function j/x then is $2\pi \text{sgn}(-\omega) = -2\pi \text{sgn}(\omega)$. When we multiply a function with a constant, its Fourier transform is also multiplied with the same constant, ie the Fourier transform of $-1/(2\pi x)$ is $\text{sgn}(\omega)j$. Therefore, the inverse Fourier transform of $H(\omega)$ is function $-1/(2\pi x)$. So, if we want to obtain signal $\tilde{f}(x)$ directly from signal $f(x)$, we must convolve it with function $-1/(2\pi x)$:

$$\tilde{f}(x) = -\frac{1}{2\pi} \int_{-\infty}^{+\infty} \frac{f(t)}{x-t} dt \quad (6.230)$$

This equation expresses the **Hilbert transform** of function $f(x)$.

Example 6.57

Show that

$$\begin{aligned} \int x \cos(\alpha x) dx &= \frac{1}{\alpha} x \sin(\alpha x) + \frac{1}{\alpha^2} \cos(\alpha x) \\ \int x \sin(\alpha x) dx &= -\frac{1}{\alpha} x \cos(\alpha x) + \frac{1}{\alpha^2} \sin(\alpha x) \end{aligned} \quad (6.231)$$

where α is a positive constant.

We use integration by parts and remember that the integral of $\cos x$ is $\sin x$ and the integral of $\sin x$ is $-\cos x$:

$$\begin{aligned} \int x \cos(\alpha x) dx &= \int x d[\sin(\alpha x)] \frac{1}{\alpha} \\ &= \frac{1}{\alpha} x \sin(\alpha x) - \frac{1}{\alpha} \int \sin(\alpha x) dx \\ &= \frac{1}{\alpha} x \sin(\alpha x) + \frac{1}{\alpha^2} \cos(\alpha x) \end{aligned} \quad (6.232)$$

$$\begin{aligned}
 \int x \sin(\alpha x) dx &= \int x d[\cos(\alpha x)] \frac{-1}{\alpha} \\
 &= -\frac{1}{\alpha} x \cos(\alpha x) + \frac{1}{\alpha} \int \cos(\alpha x) dx \\
 &= -\frac{1}{\alpha} x \cos(\alpha x) + \frac{1}{\alpha^2} \sin(\alpha x)
 \end{aligned} \tag{6.233}$$

Example 6.58

Compute the Fourier series of the periodic signal defined as follows:

$$f(x) = \begin{cases} -3 - \frac{x}{2} & \text{for } -6 \leq x < -4 \\ -1 & \text{for } -4 \leq x \leq -2 \\ \frac{1}{2}x & \text{for } -2 < x < 2 \\ +1 & \text{for } 2 \leq x \leq 4 \\ +3 - \frac{x}{2} & \text{for } 4 < x < 6 \end{cases} \tag{6.234}$$

Then plot this function and its first four nonzero harmonics on the same axes. At which points do you observe phase congruency?

We must use formulae (6.194) and (6.195), on page 625, with $T = 6$.

Since $f(x)$ is an odd function, its product with the cosine function is also odd. The integrals of such products over a symmetric interval of integration, vanish. So, all expansion coefficients a_n are 0.

Further, the integrands of the b_n coefficients in (6.195) are symmetric functions, so instead of integrating from $-T$ to $+T$, we may integrate only from 0 to $+T$ and double the result.

In the following calculation, we make use of (6.231):

$$\begin{aligned}
 b_n &= \frac{2}{6} \int_0^6 f(x) \sin \frac{\pi n x}{6} dx \\
 &= \frac{1}{3} \left[\int_0^2 \frac{x}{2} \sin \frac{\pi n x}{6} dx + \int_2^4 \sin \frac{\pi n x}{6} dx + \int_4^6 \left(3 - \frac{x}{2}\right) \sin \frac{\pi n x}{6} dx \right]
 \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{3} \left\{ \frac{1}{2} \left[-\frac{6}{\pi n} x \cos \frac{\pi n x}{6} + \frac{36}{\pi^2 n^2} \sin \frac{\pi n x}{6} \right]_0^2 + \left[-\frac{6}{\pi n} \cos \frac{\pi n x}{6} \right]_2^4 \right. \\
&\quad \left. + 3 \left[-\frac{6}{\pi n} \cos \frac{\pi n x}{6} \right]_4^6 - \frac{1}{2} \left[-\frac{6}{\pi n} x \cos \frac{\pi n x}{6} + \frac{36}{\pi^2 n^2} \sin \frac{\pi n x}{6} \right]_4^6 \right\} \\
&= \frac{1}{3} \left\{ \frac{1}{2} \left[-\frac{6}{\pi n} 2 \cos \frac{\pi n}{3} + \frac{36}{\pi^2 n^2} \sin \frac{\pi n}{3} \right] - \frac{6}{\pi n} \cos \frac{\pi n 2}{3} + \frac{6}{\pi n} \cos \frac{\pi n}{3} \right. \\
&\quad \left. + 3 \left[-\frac{6}{\pi n} \cos(\pi n) + \frac{6}{\pi n} \cos \frac{\pi n 2}{3} \right] \right. \\
&\quad \left. - \frac{1}{2} \left[-\frac{36}{\pi n} \cos(\pi n) + \frac{24}{\pi n} \cos \frac{\pi n 2}{3} - \frac{36}{\pi^2 n^2} \sin \frac{\pi n 2}{3} \right] \right\} \\
&= -\frac{2}{\pi n} \cos \frac{\pi n}{3} + \frac{6}{\pi^2 n^2} \sin \frac{\pi n}{3} - \frac{2}{\pi n} \cos \frac{\pi n 2}{3} + \frac{2}{\pi n} \cos \frac{\pi n}{3} - \frac{6}{\pi n} \cos(\pi n) \\
&\quad + \frac{6}{\pi n} \cos \frac{\pi n 2}{3} + \frac{6}{\pi n} \cos(\pi n) - \frac{4}{\pi n} \cos \frac{\pi n 2}{3} + \frac{6}{\pi^2 n^2} \sin \frac{\pi n 2}{3} \\
&= \frac{6}{\pi^2 n^2} \left[\sin \frac{\pi n}{3} + \sin \frac{2\pi n}{3} \right]
\end{aligned} \tag{6.235}$$

We may now work out the values of b_n for $n = 1, 2, \dots$. We remember that $\sin(\pi/3) = \sqrt{3}/2$. Then we obtain:

$$\begin{aligned}
b_1 &= \frac{6\sqrt{3}}{\pi^2} & b_5 &= -\frac{6\sqrt{3}}{\pi^2 25} \\
b_7 &= \frac{6\sqrt{3}}{\pi^2 49} & b_{11} &= -\frac{6\sqrt{3}}{\pi^2 121}
\end{aligned} \tag{6.236}$$

So, the harmonic expansion of the function is:

$$f(x) \simeq \underbrace{\frac{6\sqrt{3}}{\pi^2} \sin \frac{\pi x}{6}}_{1st harmonic} + \underbrace{\frac{6\sqrt{3}}{\pi^2 25} \sin \left(\frac{\pi 5x}{6} + \pi \right)}_{2nd harmonic} + \underbrace{\frac{6\sqrt{3}}{\pi^2 49} \sin \frac{\pi 7x}{6}}_{3rd harmonic} + \underbrace{\frac{6\sqrt{3}}{\pi^2 121} \sin \left(\frac{\pi 11x}{6} + \pi \right)}_{4th harmonic} \tag{6.237}$$

Note that the negative coefficients were made positive and a π was introduced in the phase of the sinusoid to account for the negative sign (remember that $\sin(\phi + \pi) = -\sin \phi$).

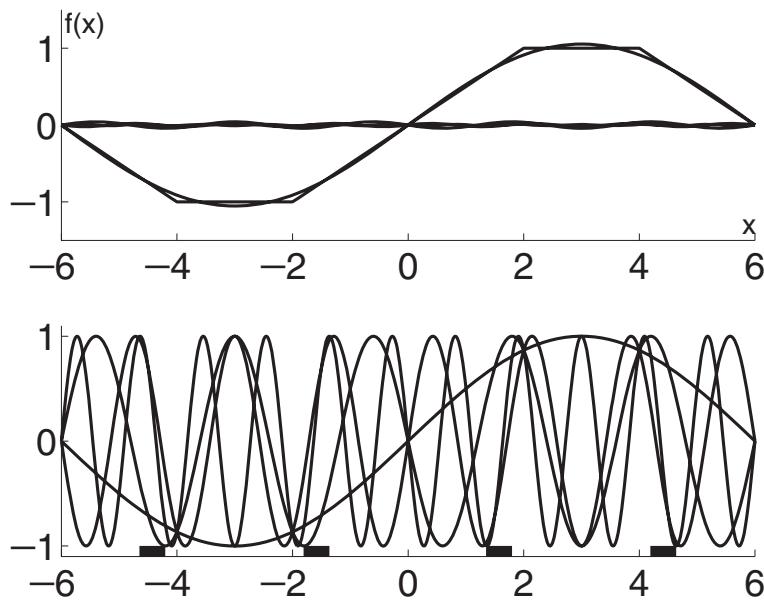


Figure 6.62: At the top the original signal and its first four harmonics. At the bottom, all harmonics plotted with the same amplitude. The thick dashes at the bottom of the figure mark the ranges of the abscissa where all harmonics are “in phase”, in the sense that they are all increasing, or all decreasing. These are the places where the signal changes abruptly.

Figure 6.62 shows the plot of the original function and each of the four harmonics identified above. It is difficult to distinguish the 3rd and 4th harmonics in the top panel. In the bottom panel, we omit the coefficient of the harmonic in order to make clearer the phase of each sinusoid at each point. We observe that the phases coincide only at the points where the flat part of the trapezoid function starts or ends. These are the points of maximum phase congruency and they are expected to be the places of maximum local signal energy.

Example 6.59

Sample the continuous signal given by (6.234) at integer and half integer values of x . Then compute the DFT of the sampled signal and plot its first five harmonics on the same axes as the digital signal. At which points do you observe phase congruency?

The sampled signal is:

$$(0, -1/4, -1/2, -3/4, -1, -1, -1, -1, -1, -3/4, -1/2, -1/4, 0, \\ 1/4, 1/2, 3/4, 1, 1, 1, 1, 1, 3/4, 1/2, 1/4) \quad (6.238)$$

Note that we do not include the last 0 value at point $x = 6$, because in DFT the signal is assumed to be repeated in both directions, so if we had a 0 at the end of this sampled sequence, it would be as if the signal in two successive samples had value 0, given that the next sample after the end of the sequence is the first sample of the sequence. These samples correspond to indices $i: (-12, -11, \dots, 0, \dots, 11)$.

The DFT of this signal is:

$$F(k) = \frac{1}{24} \sum_{i=-12}^{11} f(i) e^{-j \frac{2\pi k i}{24}} \quad (6.239)$$

As $f(-12) = f(0) = 0$ and $f(i) = -f(-i)$, and since $-e^{j\phi} + e^{-j\phi} = -2j \sin \phi$, this expression may be written as:

$$\begin{aligned} F(k) &= -\frac{2j}{24} \sum_{i=1}^{11} f(i) \sin \frac{2\pi k i}{24} \\ &= -\frac{2j}{24} \left[\frac{1}{4} \sin \frac{2\pi k}{24} + \frac{1}{2} \sin \frac{4\pi k}{24} + \frac{3}{4} \sin \frac{6\pi k}{24} + \sin \frac{8\pi k}{24} + \sin \frac{10\pi k}{24} + \sin \frac{12\pi k}{24} \right. \\ &\quad \left. + \sin \frac{14\pi k}{24} + \sin \frac{16\pi k}{24} + \frac{3}{4} \sin \frac{18\pi k}{24} + \frac{1}{2} \sin \frac{20\pi k}{24} + \frac{1}{4} \sin \frac{22\pi k}{24} \right] \end{aligned} \quad (6.240)$$

Note that due to the antisymmetry of the sine function, $F(-k) = -F(k)$.

$F(1) = -0.5294j$, $F(2) = F(3) = F(4) = F(6) = F(8) = F(9) = F(10) = 0$, $F(5) = 0.0244j$, $F(7) = -0.0144j$ and $F(11) = 0.0092j$.

Now we have the values of $F(k)$, we may use (6.198), on page 625, to work out the DFT expansion of $f(i)$:

$$\begin{aligned} f(i) &= \sum_{k=-12}^{11} F(k) e^{j \frac{2\pi k i}{24}} \\ &= 2j \sum_{k=1}^{11} F(k) \sin \frac{2\pi k i}{24} \end{aligned} \quad (6.241)$$

Note that $F(-12) = F(0) = 0$ and that is why we could simplify the above formula. Also, as $F(k)$ is an imaginary number, factor $2j$ in the above formula makes the result real. The individual terms of this sample are the harmonics of function $f(i)$. As there are only four nonzero values of $F(k)$, there are only four harmonics. The first one is $1.0588 \sin \frac{\pi i}{12}$. Variable i is the index of the sampled function $f(i)$. If we want to plot these harmonics as continuous functions, we must use as independent variable, variable x in the range $[-6, 6]$ and replace i in these formulae with $2x$, since x was sampled with step 0.5.

In figure 6.63, we plot the original function and its four harmonics as continuous functions. To identify the points of phase congruency we also plot all harmonics with the same amplitude. The places where all harmonics act in symphony, ie they are all increasing or all decreasing, are marked with the thick black dashes at the bottom of the figure. We observe that phase congruency coincides again with the places where the signal changes suddenly.

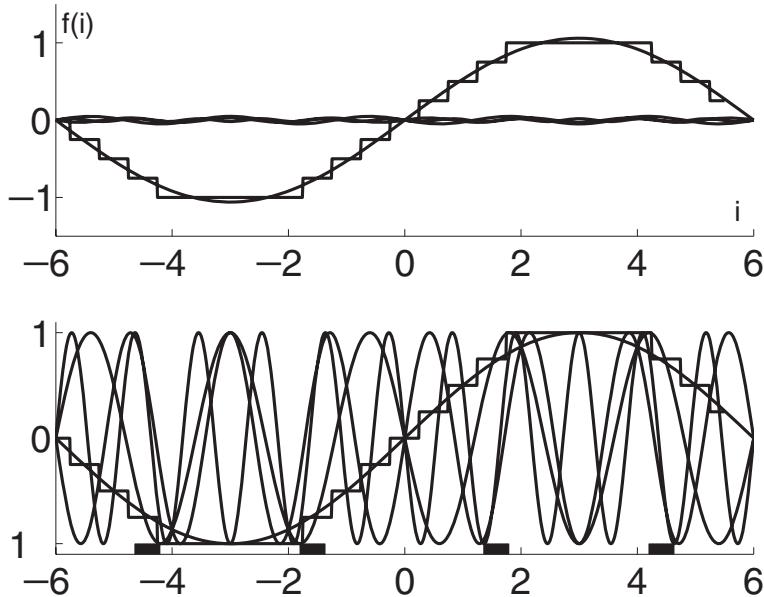


Figure 6.63: At the top, the original digital signal with its harmonics. At the bottom, the harmonics plotted with the same amplitude, in order to assess the places of phase congruency. Places where the harmonics all increase or all decrease together are marked with the thick dashes at the bottom of the graph.

Example 6.60

Compute the local energy of the digital signal of example 6.59, by using the filters developed in example 6.54 and identify the points where the local energy is maximum. To avoid border effects, do not process the first two and the last two samples.

We pseudo-convolve sequence (6.238) with filters $(-1, 1, 1, -1)$ and $(1, 1, -1, -1)$ and square the output at each sample and add the two results to deduce the square of the local energy. The results are shown in table 6.1. The bold entries in the final column represent the local maxima of the energy. Note that these do not coincide very well with the places of phase congruency we identified in example 6.59. This is because the filters we use are not very compact in the time domain, ie they do not have very good localisation properties.

<i>signal</i>	<i>conv1</i>	$(conv1)^2$	<i>conv2</i>	$(conv2)^2$	<i>loc.energy</i>
0					
$-1/4$	0	0	1	1	1
$-1/2$	0	0	1	1	1
$-3/4$	$-1/4$	$1/16$	$3/4$	$9/16$	$10/16$
-1	$-1/4$	$1/16$	$1/4$	$1/16$	$1/8$
-1	0	0	0	0	0
-1	0	0	0	0	0
-1	$-1/4$	$1/16$	$-1/4$	$1/16$	$1/8$
-1	$-1/4$	$1/16$	$-3/4$	$9/16$	$10/16$
$-3/4$	0	0	-1	1	1
$-1/2$	0	0	-1	1	1
$-1/4$	0	0	-1	1	1
0	0	0	-1	1	1
$1/4$	0	0	-1	1	1
$1/2$	0	0	-1	1	1
$3/4$	$1/4$	$1/16$	$-3/4$	$9/16$	$10/16$
1	$1/4$	$1/16$	$-1/4$	$1/16$	$1/8$
1	0	0	0	0	0
1	0	0	0	0	0
1	$1/4$	$1/16$	$1/4$	$1/16$	$1/8$
1	$1/4$	$1/16$	$3/4$	$9/16$	$10/16$
$3/4$	0	0	1	1	1
$1/2$	0	0	1	1	1
$1/4$	0	0			
0					

Table 6.1: The first column is the signal. The second column is the result of convolving the signal with filter $(-1, 1, 1, -1)$. The third column is the square of this output. The fourth column is the result of convolving the signal with filter $(1, 1, -1, -1)$. The fifth column is the square of this result. The final column is the square of the local energy, computed as the sum of the two squares. Note that as the filters are even, the output value is assigned between two samples. In bold, the local maxima of the local energy.

Example 6.61

Compute the local energy of the digital signal of example 6.59 by using the filters developed in example 6.52 and identify the points where the local energy is maximum. To avoid border effects, do not process the first and the last sample.

We pseudo-convolve sequence (6.238) with filters $(-1, 2, -1)$ and $(1, 0, -1)$ and square the output at each sample. Before we add the two outputs, we multiply the output of the first filter with $1/6$ and that of the second filter with $1/2$ to take into consideration the scaling factors of the filters. Then we add the two results to deduce the square of the local energy. The results are shown in table 6.2. The entries of the last column in bold represent the local maxima of the energy. Note that these are closer to the maxima of the phase congruency than the points identified in example 6.60, as the filters we use here are more compact in the time domain.

signal	conv1	$(conv1)^2$	conv2	$(conv2)^2$	loc.energy
0	0	0	$1/2$	$1/4$	$1/8$
$-1/4$	0	0	$1/2$	$1/4$	$1/8$
$-1/2$	0	0	$1/2$	$1/4$	$1/8$
$-3/4$	0	0	$1/2$	$1/4$	$1/8$
-1	$-1/4$	$1/16$	$1/4$	$1/16$	$1/24$
-1	0	0	0	0	0
-1	0	0	0	0	0
-1	0	0	0	0	0
-1	$-1/4$	$1/16$	$-1/4$	$1/16$	$1/24$
$-3/4$	0	0	$-1/2$	$1/4$	$1/8$
$-1/2$	0	0	$-1/2$	$1/4$	$1/8$
$-1/4$	0	0	$-1/2$	$1/4$	$1/8$
0	0	0	$-1/2$	$1/4$	$1/8$
$1/4$	0	0	$-1/2$	$1/4$	$1/8$
$1/2$	0	0	$-1/2$	$1/4$	$1/8$
$3/4$	0	0	$-1/2$	$1/4$	$1/8$
1	$1/4$	$1/16$	$-1/4$	$1/16$	$1/24$
1	0	0	0	0	0
1	0	0	0	0	0
1	0	0	0	0	0
1	$1/4$	$1/16$	$1/4$	$1/16$	$1/24$
$3/4$	0	0	$1/2$	$1/4$	$1/8$
$1/2$	0	0	$1/2$	$1/4$	$1/8$
$1/4$	0	0	$1/2$	$1/4$	$1/8$
0					

Table 6.2: The first column is the signal. The second column is the result of convolving the signal with filter $(-1, 2, -1)$. The third column is the square of this output. The fourth column is the result of pseudo-convolving the signal with filter $(1, 0, -1)$. The fifth column is the square of this result. The final column is the square of the local energy, computed as the sum of the two squares, after they are scaled by the squared values of the normalising factors of the two filters. In bold, the local maxima of the local energy.

If all we need to compute is the local energy of the signal, why don't we use Parseval's theorem to compute it in the real domain inside a local window?

Indeed, one may do that. Parseval's theorem (see page 637) says that the energy computed in the frequency domain is the same as the energy computed in the real domain. Given that we are not interested in the dc component of the signal, all we have to do is to remove the local mean and take the sum of the squares of the remaining components. This is a simple operation, which might be quite effective for clean signals. However, the use of the Fourier domain allows one to omit some frequencies, that may correspond to noise, and also develop filters that preferentially detect features of specific frequencies.

Example 6.62

Compute the local energy of the digital signal of example 6.59 by using a scanning 1×3 window. You must remove the local mean of the windowed signal and then work out the sum of the squares of the remainders and assign it to the central sample of the window. Thus, you may identify the points where the local energy is maximum. To avoid border effects, do not process the first and the last sample.

The result is shown in table 6.3. The entries of the last column in bold represent the local maxima of the energy. Note that the energy values we computed in example 6.61 and here are the same. This is due to Parseval's theorem.

signal	local mean	local energy
0		
-1/4	-1/4	1/8
-1/2	-1/2	1/8
-3/4	-3/4	1/8
-1	-11/12	1/24
-1	-1	0
-1	-1	0
-1	-1	0
-1	-11/12	1/24
-3/4	-3/4	1/8
-1/2	-1/2	1/8
-1/4	-1/4	1/8
0	0	1/8
1/4	1/4	1/8
1/2	1/2	1/8
3/4	3/4	1/8
1	11/12	1/24
1	1	0
1	1	0
1	1	0
1	11/12	1/24
3/4	3/4	1/8
1/2	1/2	1/8
1/4	1/4	1/8
0		

Table 6.3: The first column is the signal. The second column is the local average of the signal, inside a 3-samples long window. The last column is the sum of the squares of the residuals inside each window, after the local mean has been removed.

How do we decide which filters to use for the calculation of the local energy?

Apart from the obvious requirement that the two filters should constitute a Hilbert transform pair, have 0 dc component and the sum of squares of their elements should be 1, it is desirable for the filters to have also the following properties.

- 1) They should be compact in the real domain in order to have good locality. The filters used in example 6.60 lacked this property.
- 2) If we wish to process the image in a single scale only, the filters should be narrow in the real domain so that they have broad spectrum in the frequency domain and, thus, capture the local structural characteristics of the signal at any scale they occur. The filters used in example 6.61 had this property. However, such filters are not immune to noise.
- 3) It is better if we consider filters that are also compact in the frequency domain, so that they help us estimate the local signal energy within a certain frequency band. Such filters have the advantage that they will pick up signal features which will look like even and odd ripples of different frequencies, ie they will go beyond the mere detection of lines and edges.
- 4) If we decide to follow the multiband/multiscale approach, the filter should be easily scaled to allow the user to translate it into a different band.
- 5) Filters that are defined with finite bandwidth should have maximal concentration in the real domain with minimal side lobes, so that they do not create artifacts and they can be easily truncated to be used as convolution filters. Their exact shape will depend on the filter parameters.

A filter that exhibits most of the desirable properties is:

$$H(\omega) = \log \left(\frac{\cos^2(\omega - \omega_0)}{\cos^2 \omega_1} \right) \quad \text{for } |\omega - \omega_0| < \omega_1 \quad (6.242)$$

Here it is assumed that the total bandwidth of the signal is scaled to be in the range $[0, 1]$ and ω_0 is a number within this range. The bandwidth of the filter is determined by ω_1 . This filter should be used as the symmetric filter. It should be combined with its Hilbert transform pair to extract the local signal energy. Note that this filter will respond best to symmetric features, which, however, may not be just simple lines.

Example 6.63

Assume that the central frequency response of the filter defined by equation (6.242) is $\omega_0 = 0.5$ and the bandwidth parameter is $\omega_1 = 0.2$. Calculate the frequency response of the filter and its weights in the real domain.

Figure 6.64a shows the plot of filter (6.242) as a function of ω . It was constructed by allowing ω to take values in the range $[0.3, 0.7]$ in steps of 0.01. The logarithm used was base 10. To obtain the filter in the real domain, we sampled the range of ω $[0, 1]$ in steps of 0.01. We also considered the same values of $H(\omega)$ for negative frequencies, ie in the range $[-1, 0]$, with the same sampling step. This yielded 201 points in total,

which were treated as the DFT of the filter. The inverse DFT yielded the real filter. In 6.64b we plot only the central 61 values of the filter. As the filter is of finite bandwidth in the frequency domain, it is of infinite extent in the real domain, with gradually dying down side lobes. Note that the filter is not a simple line detection filter, but rather one that will respond maximally to patterns in the signal that are similar to its appearance. The significant nonzero values of the filter are:

$$(-0.132, 0.011, 0.466, -0.015, -0.795, 0.007, 0.933, \\ 0.007, -0.795, -0.015, 0.466, 0.011, -0.132)$$

One may approximate this filter as a 9-tap filter with weights:

$$(0.249, -0.012, -0.488, 0.005, 0.492, 0.005, -0.488, -0.012, 0.249)$$

These values were obtained by keeping only the 9 central values of the filter and normalising them, so that the positive weights sum up to +1 and the negative weights sum up to -1. A simplified version of this filter might be:

$$0.25, 0.00, -0.50, 0.00, 0.50, 0.00, -0.50, 0.00, 0.25.$$

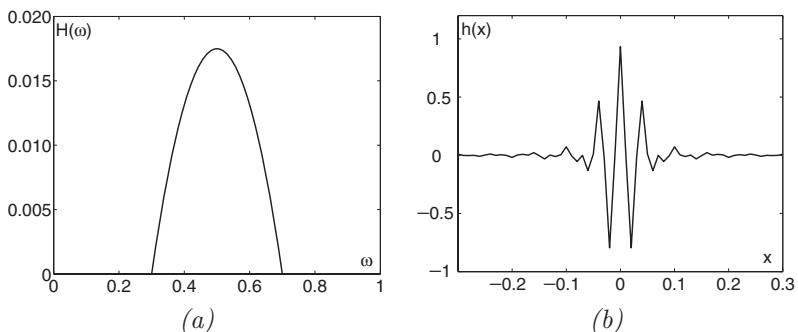


Figure 6.64: (a) Filter (6.242) in the frequency domain and (b) in the real domain.

Example 6.64

Compute the Hilbert transform pair of filter (6.242).

The Fourier transform of the Hilbert pair of filter (6.242) should be:

$$\tilde{H}(\omega) \equiv \begin{cases} -jH(\omega) & \text{for } \omega < 0 \\ 0 & \text{for } \omega = 0 \\ jH(\omega) & \text{for } \omega > 0 \end{cases} \quad (6.243)$$

We sample the range $[-1, 1]$ of ω values with steps of 0.01, to produce 201 sampling points. We then take the inverse DFT of these samples to produce the filter in the real domain. In figure 6.65 we plot the central 61 values of this filter. This is an antisymmetric filter. However, it is not an edge detection filter. It is designed to respond maximally to antisymmetric multiple fluctuations of the signal. So, if used

in conjunction with the filter in 6.64b, they will detect features in the signal that are of these two types. The phase computed from their two outputs will tell us how much one or the other of the two filters the signal looks like, in the locality of each energy maximum.

The most significant central values of the filter in figure 6.65 are:

$(-0.0057, -0.0805, 0.0034, -0.0160, 0.0062, 0.2877, -0.0146, -0.6437, 0.0124, 0.8974, 0, -0.8974, -0.0124, 0.6437, 0.0146, -0.2877, -0.0062, 0.0160, -0.0034, 0.0805, 0.0057)$.

By keeping only the central 9 values, we may construct a 9-tap long filter, making sure that the positive weights sum up to +1 and the negative weights sum up to -1:

$(-0.0093, -0.4105, 0.0079, 0.5723, 0.0000, -0.5723, -0.0079, 0.4105, 0.0093)$

One may simplify such a filter to the form:

$(-0.20, -0.40, 0.60, 0.00, -0.60, 0.40, 0.20)$.

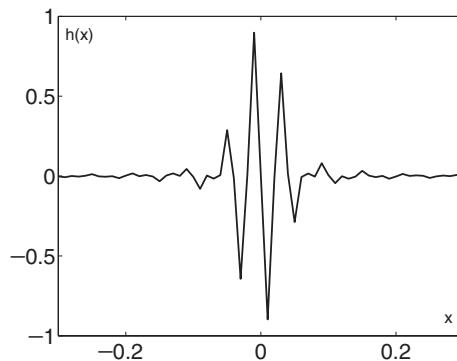


Figure 6.65: The Hilbert pair of the filter of figure 6.64b.

Example B6.65

We project a signal $f(x)$ on two filters that constitute a Hilbert transform pair, by pseudo-convolving the signal with these filters. Work out the relationship of the Fourier transforms of the two outputs of these pseudo-convolutions.

Let us call the even filter $e(x)$ and the odd filter $o(x)$. Let us denote their Fourier transforms by $E(\omega)$ and $O(\omega)$, respectively. We shall denote the components of the DFTs that correspond to positive frequencies with a plus (+) superscript and the components that correspond to negative frequencies with a minus (-) superscript. We shall denote the real and the imaginary parts of the DFTs with indices R and I , respectively. We may then write:

$$E(\omega) = \begin{cases} E_R^-(\omega) + jE_I^-(\omega) & \text{for } \omega < 0 \\ E_R^+(\omega) + jE_I^+(\omega) & \text{for } \omega > 0 \end{cases} \quad (6.244)$$

Since filter $o(x)$ is the Hilbert transform pair of $e(x)$, its DFT is produced from the DFT of $e(x)$, by multiplying the components of the negative frequencies with $-j$ and the components of the positive frequencies with $+j$:

$$O(\omega) = \begin{cases} E_I^-(\omega) - jE_R^-(\omega) & \text{for } \omega < 0 \\ -E_I^+(\omega) + jE_R^+(\omega) & \text{for } \omega > 0 \end{cases} \quad (6.245)$$

Let us denote by $F(\omega)$ the DFT of the signal. The local projection of the signal on $e(x)$ is identical to its convolution with $e(x)$, given that $e(x)$ is symmetric. The DFT of the output will be $F(\omega)E(\omega)$. The local projection of the signal on $o(x)$ is minus its convolution with $o(x)$, given that $o(x)$ is antisymmetric. The DFT of this output will be $-F(\omega)O(\omega)$. If we were to perform the two pseudo-convolutions in one go, by using as a single filter, the complex filter $c(x) \equiv e(x) + jo(x)$, the DFT of the result would be $F(\omega)E(\omega) - jF(\omega)O(\omega) \equiv C(\omega)$:

$$\begin{aligned} C(\omega) &= \begin{cases} F(\omega)E_R^-(\omega) + jF(\omega)E_I^-(\omega) - F(\omega)E_R^-(\omega) - jF(\omega)E_I^-(\omega) & \text{for } \omega < 0 \\ F(\omega)E_R^+(\omega) + jF(\omega)E_I^+(\omega) + F(\omega)E_R^+(\omega) + jF(\omega)E_I^+(\omega) & \text{for } \omega > 0 \end{cases} \\ &= \begin{cases} 0 & \text{for } \omega < 0 \\ 2F(\omega)E_R^+(\omega) + j2F(\omega)E_I^+(\omega) & \text{for } \omega > 0 \end{cases} \end{aligned} \quad (6.246)$$

We note that, only the DFT components of positive phase of the even filter appear in the DFT of the projection onto the complex filter. This indicates that we may obtain the convolution with the two original filters, if we take the DFT of the signal, multiply it with the components of the DFT of the original even filter, that correspond to positive frequencies only, double the result and take the inverse DFT: the real part of the inverse DFT will be the result of the local projection of the signal on the even filter and the imaginary part of the result will be the local projection of the signal on the odd filter.

How do we compute the local energy of a 1D signal in practice?

In practice we do not need to use two filters to estimate the local energy (see example 6.65). Let us assume that we have the DFT values of the even filter for positive frequencies and the DFT of the signal. The algorithm we have to use then is:

- Step 0:** Set to 0 all the DFT amplitudes that correspond to the negative frequencies of the even filter. Call this result $E^+(\omega)$.
- Step 1:** Take the DFT of the signal $F(\omega)$.
- Step 2:** Multiply $F(\omega)$ with $2E^+(\omega)$, point by point.
- Step 3:** Take the inverse DFT of this product.
- Step 4:** Take the real part of the result and square it sample by sample.
- Step 5:** Take the imaginary part of the result and square it sample by sample.
- Step 6:** Sum the two squared results: this is the local energy of the signal.
- Step 7:** Identify the local maxima of the local energy as the places where the signal has either a symmetric or an antisymmetric feature.

How can we tell whether the maximum of the local energy corresponds to a symmetric or an antisymmetric feature?

This is determined by the phase of the complex output produced by the above algorithm. Let us call it $r(x) \equiv e(x) + jo(x)$. The real part of this result corresponds to the local projection of the signal on the even filter and the imaginary part to the projection of the signal on the odd filter. The phase of the result is given by $\tan^{-1}(o(x)/e(x))$. The closer to $\pi/2$ this phase is, the more similar the feature is to the antisymmetric filter. The closer to 0 this phase is, the more similar the feature is to the symmetric filter.

Example 6.66

Use the filters developed in example 6.52, on page 633, to work out the location and type of the features of the following signal:

$$f(x) = \begin{cases} 7 & \text{for } -10 \leq x < -8 \\ 15 & \text{for } -8 \leq x < -6 \\ 9 & \text{for } -6 \leq x < -4 \\ 2 & \text{for } -4 \leq x < -3.9 \\ 9 & \text{for } -3.9 \leq x < -2 \\ 18 & \text{for } -2 \leq x < -1.9 \\ 9 & \text{for } -1.9 \leq x < 0 \\ 16 & \text{for } 0 \leq x < 4 \\ 10 & \text{for } 4 \leq x < 7.3 \\ 2 & \text{for } 7.3 \leq x < 7.5 \\ 10 & \text{for } 7.5 \leq x < 8 \\ 7 & \text{for } 8 \leq x < 10 \end{cases} \quad (6.247)$$

Sample this signal with step 0.1 to produce the digital signal you will use in your calculations.

Sampling the signal in steps of 0.1 produces a digital signal $f(i)$, where i ranges from 0 to 200. We convolve the signal with filters $(-1, 2, -1)/\sqrt{6}$ and $(1, 0, -1)/\sqrt{2}$, to produce outputs $e(i)$ and $o(i)$, respectively. To avoid boundary effects and in order to be consistent with the treatment in the frequency domain, we assume that the signal is repeated ad infinitum in both directions. We then square and add the two outputs to produce the local energy. To identify the locations of the features, we have to identify the local energy maxima. We define as maxima the locations at which the local energy is larger than one of its neighbours and larger or equal than the other of its neighbours. Because we are dealing with real numbers, several insignificant maxima may be identified. As we know that this signal is clean (no noise), we use a pretty low threshold, accepting as local maxima only those that are larger than 10^{-3} , from at least one of their neighbours. Then we compute the phase for each index i as $\tan^{-1}(o(i)/e(i))$. We threshold the phase so that when its absolute value is above $\pi/4$, we mark it as 1 and when it is below $\pi/4$ as 0. So, a local energy maximum with the corresponding phase above the threshold is marked as a spike (a line) and a local energy maximum with

the corresponding phase below the threshold is marked as an edge. In figure 6.66 we plot the original signal, the local energy, and the local phase. The identified edges are marked with an \times on the original signal and the identified lines with an open circle.

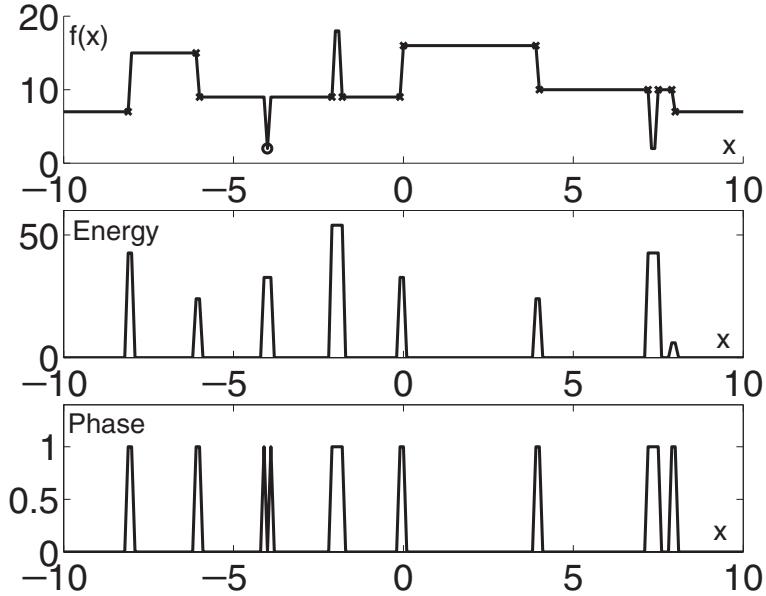


Figure 6.66: At the top, the original signal sampled with 201 points. The \times mark the locations of edges, while the open circles mark the locations of spikes. Note that as the filters are very small, lines wider than a single sample are identified as two back-to-back edges. In the middle, the local energy. We can see that its maxima coincide well with the signal features. At the bottom the local phase thresholded: it is 1 when its value at a local energy maximum is greater than $\pi/4$, and 0 otherwise.

Example 6.67

Repeat example 6.66 by working in the frequency domain.

The convolution of the signal with a filter is equivalent to the multiplication of their DFTs. As the signal is 201-sample long, its DFT is also 201-sample long. The filter, however, is 3-tap long. To compute its DFT in the form that it can be multiplied with the DFT of the signal point by point, it also has to be 201-tap long. Given that we assume the signal to be repeated ad infinitum in both directions, the left neighbour of the first sample of the signal is the last sample. So, we may consider that the signal is convolved with the following filters:

$$\left(\frac{2}{\sqrt{6}}, -\frac{1}{\sqrt{6}}, \underbrace{0, 0, 0, \dots, 0, 0, 0}_{198 \text{ zeros}}, -\frac{1}{\sqrt{6}} \right) \quad (6.248)$$

$$\left(0, -\frac{1}{\sqrt{2}}, \underbrace{0, 0, 0, \dots, 0, 0, 0}_{198 \text{ zeros}}, \frac{1}{\sqrt{2}}\right) \quad (6.249)$$

The DFTs of these two filters are plotted in figure 6.67. Note that the DFT of the first filter is purely real and the DFT of the second filter is purely imaginary. The dot product of these two DFTs is 0, confirming that these two DFTs form an orthogonal basis in the Fourier domain.

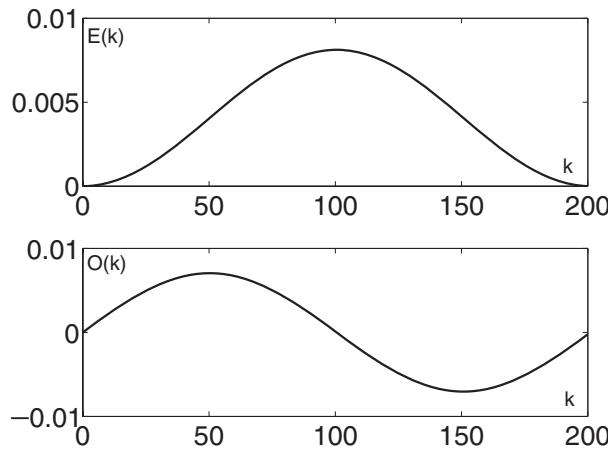


Figure 6.67: The DFT of filter (6.248) at the top, and of filter (6.249) at the bottom. The horizontal axis is sampled with 201 points. $O(k)$ is purely imaginary.

We then take the DFT of the signal and multiply it point by point with the DFTs of the two filters. Let us call the two outputs $O_e(k)$ and $O_o(k)$, respectively, with k taking values in the range $[0, 200]$. We take the inverse DFT of each one of these outputs to produce $e(i)$ and $o(i)$, respectively. The signal features then are computed in exactly the same way as in example 6.66. The values of $e(i)$ and $o(i)$ turn out to agree, to several decimal places, with the values obtained by direct convolutions in example 6.66. On the basis of that, one would expect the result of feature detection to be exactly identical with that in example 6.66. The result we obtain here is shown in figure 6.68, which is very different from that shown in figure 6.66. Upon closer inspection, it turns out that the non-maxima suppression we perform by taking the local maxima of the local energy is an unstable process. On either side of an edge or a spike, the values of the local energy tend to be the same, with differences only at the level of rounding error. When something changes in the run, these differences sometimes flip the choice of the maximum from one to the other, with consequences also on the computed phase of the feature. That is why, the negative spike of the signal in example 6.66 was correctly identified as such, while here one of its sides only is identified as an edge.

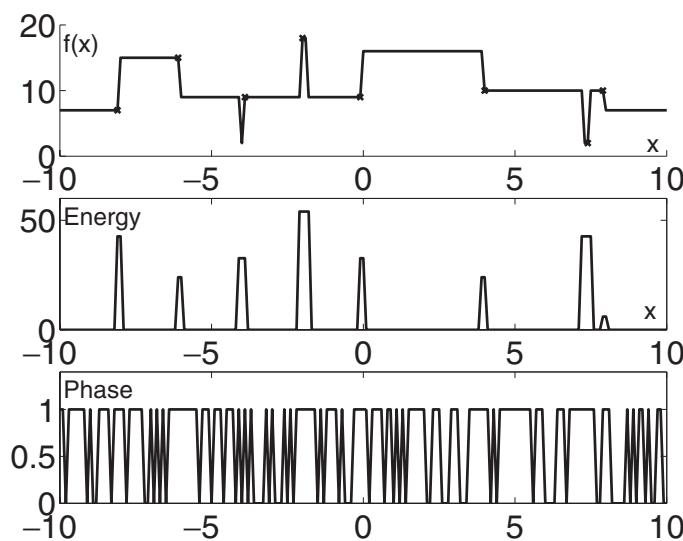


Figure 6.68: At the top, the original signal sampled with 201 points. The \times mark the locations of identified edges. In the middle, the local energy. At the bottom, the local phase thresholded: it is 1 when its value is greater than $\pi/4$, and 0 otherwise.

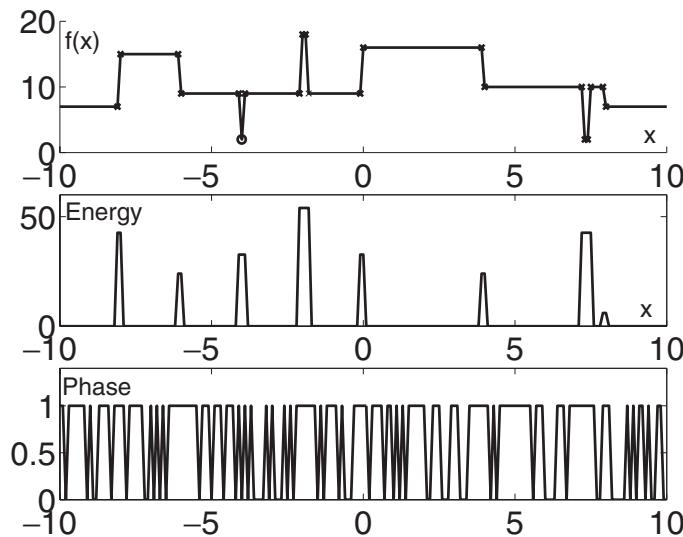


Figure 6.69: At the top, the original signal sampled with 201 points. The \times mark the locations of the identified edges and the open circle marks the location of the identified spike. These results were produced by thresholding the local energy and keeping all its values above the threshold as marking the locations of features (no non-maxima suppression). In the middle, the local energy. At the bottom, the local phase thresholded: it is 1 when its value is greater than $\pi/4$, and 0 otherwise.

One way to avoid having such unstable results is not to apply non-maxima suppression, but simply to threshold the local energy values. This will lead to thick features, with their locations marked by multiple points, but it is a much more robust approach than the previous one. The result of keeping all energy maxima higher than 1, is shown in figure 6.69. We can see that the spike and the edges have been identified correctly.

Example B6.68

For the filters used in examples 6.66 and 6.67, confirm the relationships between their DFTs, as discussed in example 6.65.

Let us consider the even filter $f_e(i)$ as given by (6.248) and its DFT, plotted at the top of figure 6.67. Given that this DFT is real, the DFT of its Hilbert transform pair should be imaginary.

The values of its indices that correspond to positive frequencies should be the same as the values of the DFT of $f_e(i)$ for the same indices, and the values of its indices that correspond to negative frequencies should be the negative of the values of the DFT of $f_e(i)$ for the same indices. If the indices of the DFT go from 0 to 200, the indices that correspond to negative frequencies are from 101 to 200, inclusive.

So, at the top of figure 6.70, we plot the DFT, $E(\omega)$, of filter $f_e(i)$ and below it, the DFT, $O(\omega)$, of its Hilbert pair. We then take the inverse DFT of $O(\omega)$ to recover the corresponding filter $f_o(i)$ in the real domain. The filter we obtain is shown at the bottom of figure 6.70.

We can see that it does not resemble at all filter (6.249), which we know to be the Hilbert pair of $f_e(i)$. However, it does start from 0, and its second value is its largest negative value, while its last value is exactly the opposite of that, both values having absolute value 0.5198, which is different from $1/\sqrt{2} = 0.707$. The reason for this discrepancy is quite simple: a filter cannot be compact in both the real and the frequency domain. Once we specified the frequency response of the filter to be of finite bandwidth, the filter in the real domain will have strong side lobes, which manifest themselves as the strong undesirable fluctuations we see at the bottom of figure 6.70.

In spite of the inaccuracy of its values, if we calculate the energy of this filter, it turns out to be exactly 1, just like it should be. In a sense, the energy of the filter is distributed in its side lobes and weakens its two main peaks.

Insisting on the filter being compact in the real domain, (filter (6.249)), results in a DFT that appears not to obey the expected relationship with the DFT of its Hilbert pair. Indeed, the DFT of filter (6.249), shown at the bottom of figure 6.67, is different from that shown in figure 6.70. However, we can recognise that the DFT shown in 6.67 is a “smoothed” version of the DFT in figure 6.70. We also remember that the DFT at the bottom of figure 6.67 was found to be exactly orthogonal to that of the $f_e(i)$ filter.

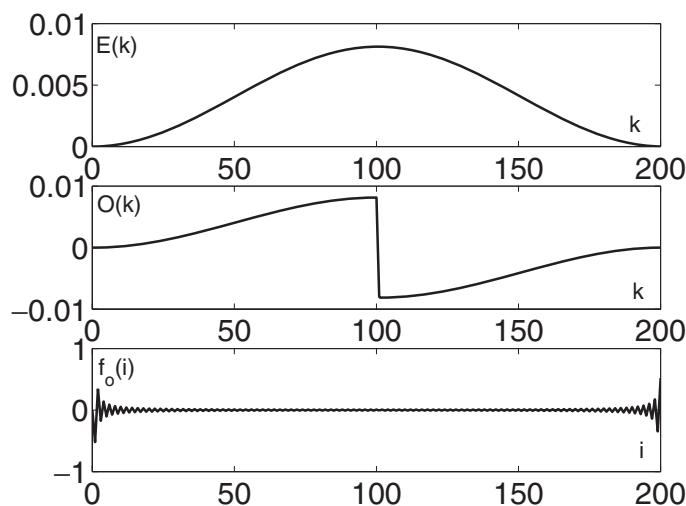


Figure 6.70: At the top, the DFT of filter 6.248. It consists of 201 samples. In the middle, the values that correspond to negative frequencies are negated, and all values are multiplied with j to form the DFT of the corresponding odd filter. This is a purely imaginary DFT and taking its inverse yields the filter plotted in the bottom panel. This filter is a 201-sample long filter.

Example 6.69

Use filter (6.242) to work out the location and type of the features of the following signal:

$$f(x) = \begin{cases} 7 & \text{for } -10 \leq x < -8 \\ 15 & \text{for } -8 \leq x < -6 \\ 9 & \text{for } -6 \leq x < -4 \\ 2 & \text{for } -4 \leq x < -3.9 \\ 12 & \text{for } -3.9 \leq x < -2 \\ 18 & \text{for } -2 \leq x < -1.9 \\ 10 & \text{for } -1.9 \leq x < 0 \\ 10 + 3x & \text{for } 0 \leq x < 2 \\ 16 & \text{for } 2 \leq x < 4 \\ 24 - 2x & \text{for } 4 \leq x < 7 \\ 10 & \text{for } 7 \leq x < 7.3 \\ 2 & \text{for } 7.3 \leq x < 7.5 \\ 18 & \text{for } 7.5 \leq x < 8 \\ 7 & \text{for } 8 \leq x < 10 \end{cases} \quad (6.250)$$

Sample this signal with step 0.1 to produce the digital signal you will use in your calculations.

First of all, given the type of signal we have, it is best if we select the parameters of the filter so that it is tuned to select features of the type we expect to have, namely edges and lines. So, we must select its parameters so that its frequency response is broad and in low frequencies, rather than in high frequencies like in examples 6.63 and 6.64, where its parameters tuned it to detect multiripple features. We select $\omega_0 = 0.2$ and $\omega_1 = 0.2$. At the top of figure 6.71, we plot $H(\omega)$ for these values of its parameters. The frequency range $[-1, 1]$ is sampled with 201 points. The indices that correspond to negative frequencies are those above 101. Indices [101, 200] correspond to values of ω in the range $[-1, 0)$. To produce the odd filter, the values of $H(k)$ for $k \in [0, 100]$ are multiplied with j and the values of $H(k)$ for $k \in [101, 200]$ are multiplied with $-j$. Taking the inverse DFTs yields the filters plotted in figure 6.71 as $f_e(i)$ and $f_o(i)$. These are filters appropriate for processing a 201-sample signal. Note that filter $f_e(i)$ is essentially a filter with one positive weight surrounded by two negative weights, exactly the type of filter appropriate for detecting spikes. These weights are:

$(-0.005, -0.109, -0.288, -0.374, -0.193, 0.251, 0.728, 0.933,$
 $0.728, 0.251, -0.193, -0.374, -0.288, -0.109, -0.005,)$

The $f_o(i)$ filter is a first difference filter with weights:

$(0.005, 0.279, 0.614, 0.755, 0.525, 0, -0.525, -0.755, -0.614, -0.279, -0.005).$

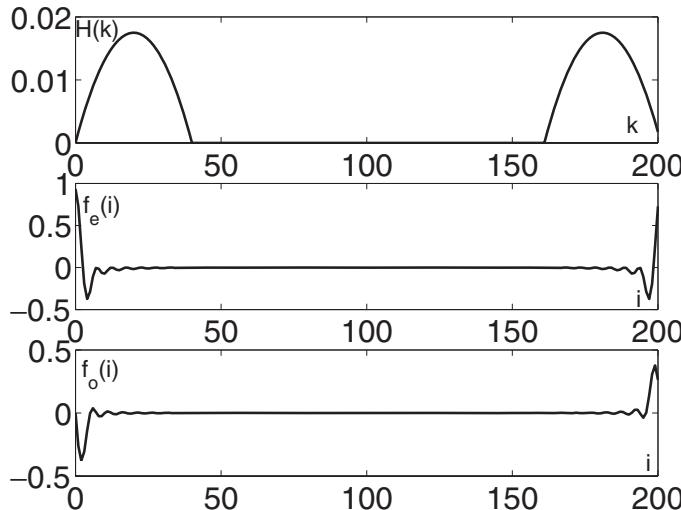


Figure 6.71: At the top, filter $H(k)$. In the middle, the real part of the inverse of $H(k)$. It is a line/spike detection filter. At the bottom, the real part of the inverse of $\text{sign}(\omega)jH(k)$. It is an edge detection filter.

In order to process the signal, however, we do not use these filters. Instead, we use the algorithm described on page 651: we multiply the DFT of the signal with $2H(k)$, where

for negative frequencies, ie for $k > 100$, we have set $H(k) = 0$, take the inverse DFT and separate the real from the imaginary part to extract results $e(i)$ and $o(i)$. From them we compute the local energy. Before we take its local maxima, we first smooth it with a low pass Gaussian filter of size 11. Figure 6.72 shows the original signal with the identified features and below it the local energy and local phase, thresholded to be 1 for phase above $\pi/4$ and 0 otherwise.

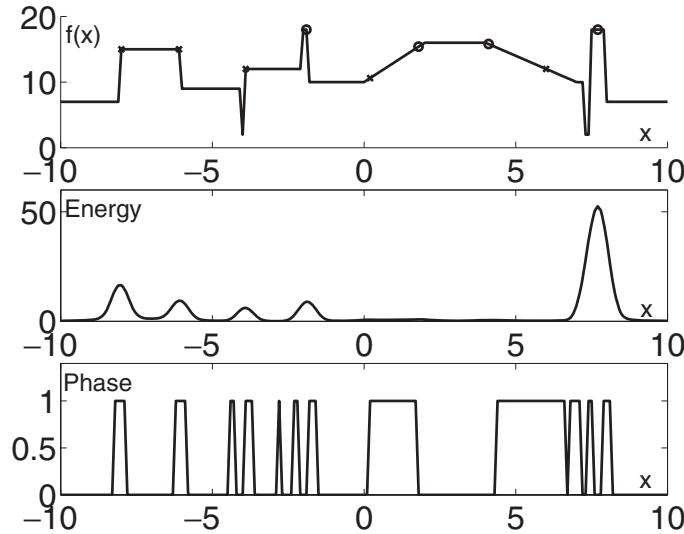


Figure 6.72: At the top, the original signal with the features identified: an open circle indicates a symmetric feature and a cross indicates an antisymmetric feature. In the middle, the smoothed local energy function and at the bottom, the thresholded local phase.

How can we compute phase congruency and local energy in 2D?

The problem in 2D is that the features may have a particular orientation. Thus, one requires three quantities in order to identify features in an image: local energy, local phase and local orientation. First attempts to generalise the 1D local energy calculation to 2D were based on the use of several orientations. Along each orientation, the 1D approach was applied and the orientation that yielded the maximum energy response was considered as the dominant local orientation. This ad-hoc approach had several drawbacks, concerning bias and isotropy. The correct way to proceed is based on the generalisation of the Hilbert transform to 2D, known as the **Riesz transform**. Application of the Riesz transform to a 2D signal produces the **monogenic signal**, which corresponds to the **analytic signal** in 1D.

What is the analytic signal?

If $f(x)$ is a signal and $f_H(x)$ is its Hilbert transform, the analytic signal is $f(x) + jf_H(x)$. If we know the real and the imaginary part of an analytic signal, we can compute the local

energy and phase of the original signal. The whole process we developed so far was aimed at computing the local components of the analytic signal of a given real signal.

How can we generalise the Hilbert transform to 2D?

Let us recall what the Hilbert transform does. It first considers an even filter and processes the signal with that. An even filter can be easily generalised to 2D, as it is isotropic. So, we may consider a 2D even filter and process with that the 2D image. Then the Hilbert transform creates from the even filter an odd filter, by multiplying its DFT with $\text{sign}(\omega)j$. Now, an odd filter has a preferred direction, as it is a differentiating filter. Given that an image has two directions, we must be able to generate two such filters, one for each direction. Their outputs have to be combined with the output of the original even filter, to produce the local energy. However, these filters being two, would dominate the sum. So, we must find a way to combine their outputs proportionally. If the frequency along one direction is ω_x and along the other is ω_y , we weigh these filters using $\omega_x/\sqrt{\omega_x^2 + \omega_y^2}$ and $\omega_y/\sqrt{\omega_x^2 + \omega_y^2}$. When we take the inverse DFTs of the image processed by these three filters, the ratio of the outputs of the two odd filters will tell us the dominant orientation of the detected feature. In addition, the higher the output of the even filter is, the more likely the detected feature to be a symmetric feature, and the higher the combined output of the two odd filters is, the more likely the detected feature to be an antisymmetric feature. This intuitive understanding of what we have to do leads to the definition of the Riesz transform.

How do we compute the Riesz transform of an image?

Step 0: Take the DFT of the image, $F(\omega_x, \omega_y)$.

Step 1: Consider an even filter with frequency response $H(\omega_x, \omega_y)$.

Step 2: Multiply the DFT of the image with the DFT of the filter point by point, and take the inverse DFT of the product, to produce $f_H(x, y)$.

Step 3: Define functions:

$$H_1(\omega_x, \omega_y) \equiv j \frac{\omega_x}{\sqrt{\omega_x^2 + \omega_y^2}} \quad H_2(\omega_x, \omega_y) \equiv j \frac{\omega_y}{\sqrt{\omega_x^2 + \omega_y^2}} \quad (6.251)$$

These functions will help produce the Hilbert pair of the even filter along each axis and at the same time, “share” the odd energy component between the two filters, with weights that, when squared and summed, yield 1.

Step 4: Take the inverse DFT of $F(\omega_x, \omega_y)H(\omega_x, \omega_y)H_1(\omega_x, \omega_y)$ to produce $f_{H_1}(x, y)$.

Step 5: Take the inverse DFT of $F(\omega_x, \omega_y)H(\omega_x, \omega_y)H_2(\omega_x, \omega_y)$ to produce $f_{H_2}(x, y)$.

The output $(f_H(x, y), f_{H_1}(x, y), f_{H_2}(x, y))$ is the monogenic signal of the image.

How can the monogenic signal be used?

The monogenic signal $(f_H(x, y), f_{H_1}(x, y), f_{H_2}(x, y))$, at a pixel position (x, y) , may be used to compute the local image energy as:

$$E(x, y) \equiv \sqrt{f_H(x, y)^2 + f_{H_1}(x, y)^2 + f_{H_2}(x, y)^2} \quad (6.252)$$

Local maxima of this quantity identify the locations of image features.

The local feature symmetry (ie how symmetric or antisymmetric the feature is) may be measured as:

$$\Phi(x, y) \equiv \left| \tan^{-1} \frac{f_H(x, y)}{\sqrt{f_{H_1}(x, y)^2 + f_{H_2}(x, y)^2}} \right| \quad (6.253)$$

Here we assume that function \tan^{-1} yields values in the range $[-\pi/2, \pi/2]$. If the feature is purely symmetric, the numerator of this fraction is maximal and the denominator is minimal. So the calculated angle will tend to be either close to 90° or to -90° . If we do not want to distinguish the two, we take the absolute value of the result. If the feature is mostly antisymmetric, the numerator approaches 0 while the denominator is large, and $\Phi(x, y)$ is close to 0. So, $\Phi(x, y)$ is a measure of local symmetry, taking values in the range $[0, \pi/2]$.

The local feature orientation may be computed as:

$$\Theta(x, y) \equiv \tan^{-1} \frac{f_{H_2}(x, y)}{f_{H_1}(x, y)} + \delta \left(1 + \text{sign} \left[\tan^{-1} \frac{f_{H_2}(x, y)}{f_{H_1}(x, y)} \right] \right) \pi \quad (6.254)$$

This number varies between 0 and π , as factor π is added only to negative angles computed by \tan^{-1} , to put them in the range $[\pi/2, \pi]$.

How do we select the even filter we use?

We usually wish to detect features that manifest themselves in one or more frequency bands. So, we place the nonzero frequency response of the filter in the band of interest. Some features, which are more prominent, manifest themselves in several bands. These features are considered more stable than others and one way of selecting them is to check their detectability in several frequency bands. The broader the band we use, the narrower the filter will be in the real domain and, therefore, the better it will respond to image details. The narrower the band, the broader the filter in the real domain and the grosser the features it will pick. So, using a selection of band widths allows us to detect features of different scales.

Figure 6.73 shows the frequency space of an image, with four different cases of band selection. In the first case, the band is such that the 2D even filter placed there will act as a low pass filter along the y axis and as a symmetric feature detector along the x axis. In the second case, the selected band contains the dc component along the x axis, but it does not contain it along the y axis. So, this filter will smooth along the x axis and enhance symmetric features along the y axis. In the third case, the filter does not contain the dc component along any of the axis. This filter, therefore, will respond to features that show local symmetry along both axis, namely diagonal features. Finally, the last case corresponds to a composite filter that will respond to features either along the main or the secondary diagonal of the axes.

Often, a systematic way is used to sample the bands. A Gaussian function defined with one of its axes along the radial direction is used to identify the band of interest. This type of frequency band creation leads to the so called **Gabor functions**². The bands that are identified in such cases are schematically shown in figure 6.74.

It has to be clarified that for each band we select, we can measure a different local energy, local phase and local orientation. So, from each selected band we can estimate how much the image locally resembles the feature that corresponds to that band, what is the local orientation of this feature, and whether the feature is mostly symmetric or mostly antisymmetric. The use of all frequency bands characterises the image and all its local structure fully.

²Gabor functions are fully covered in “Image Processing, dealing with Texture”, by Petrou and Garcia Sevilla.

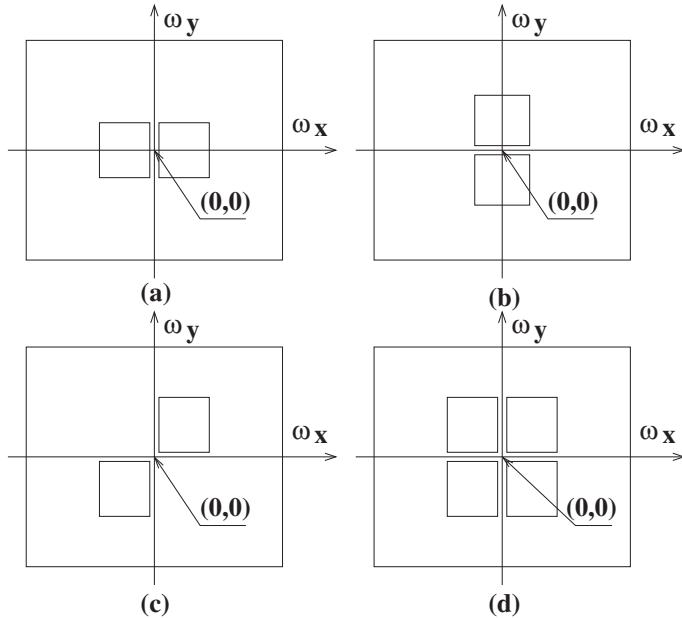


Figure 6.73: The 2D frequency domain of an image. Positive and negative frequencies are shown. The large rectangle indicates the frequency band of the digital image. The small rectangles indicate the bands where the selected even filter has nonzero response. When the filter does not include in its band the $(0,0)$ frequency (ie it has 0 dc component), the rectangles are slightly displaced from the axis to indicate this. (a) A filter in this band will smooth along the y axis (since it will have a nonzero dc component along this axis) and it will enhance a symmetric feature along the x axis. (b) A filter in this band will smooth along the x axis and enhance a symmetric feature along the y axis. (c) A filter in this band will enhance symmetric features along the main diagonal of the axes. (d) We may use a composite filter to enhance features along the two diagonal directions of the axes.

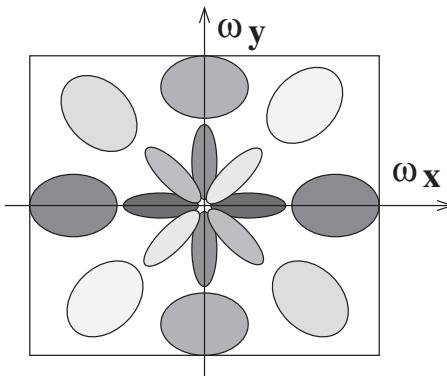


Figure 6.74: Usually the bands used to study the frequency content of an image are selected by considering Gaussian filters with elliptic cross-sections, with one of their axes aligned with the radial direction in the frequency domain. A pair of identical Gaussians has to be considered in corresponding bands of the positive and negative frequencies, for the corresponding filters in the spatial domain to be real.

Example 6.70

Apply the Riesz transform to the 201×201 image of figure 6.75 and compute its local energy and local phase. Identify the features of the image by considering the local maxima of the energy along directions orthogonal to the local orientation. Identify the type of local feature by considering the local phase.

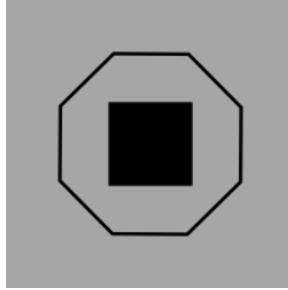


Figure 6.75: An original image containing edges and lines.

First we have to select the even filter we shall use. We are going to use filter (6.242), extended to 2D:

$$H(\omega_x, \omega_y) = \log \left(\frac{\cos^2(\omega_x - \omega_{0x})}{\cos^2 \omega_{1x}} \right) \log \left(\frac{\cos^2(\omega_y - \omega_{0y})}{\cos^2 \omega_{1y}} \right) \quad (6.255)$$

for $|\omega_x - \omega_{0x}| < \omega_{1x}$ and $|\omega_y - \omega_{0y}| < \omega_{1y}$

In the digital domain, we shall replace ω_x with $2\pi k/N$ and ω_y with $2\pi l/M$, for an $N \times M$ image:

$$H(k, l) = \log \left(\frac{\cos^2 \left(\frac{2\pi(k-k_0)}{N} \right)}{\cos^2 \left(\frac{2\pi k_1}{N} \right)} \right) \log \left(\frac{\cos^2 \left(\frac{2\pi(l-l_0)}{M} \right)}{\cos^2 \left(\frac{2\pi l_1}{M} \right)} \right) \quad (6.256)$$

for $|k - k_0| < k_1$ and $|l - l_0| < l_1$

We shall chose the parameters of this filter so it has nonzero response in the bands defined in figure 6.73a,b,d, so that, in the first case, we shall detect horizontal features, in the second case vertical features and in the third case diagonal features.

As the DFT we compute for the image has the dc component at its corner, we must be careful how we shall represent each filter. In the Fourier domain, a function is assumed to be repeated ad infinitum in all directions. So, we may think of the domain of interest repeated as shown in figure 6.76. Each panel in figure 6.76 was produced by repeating four times the corresponding panel of figure 6.73. Then we may shift the filter domain so that the dc component is at the corner. In practice, this means that we compute the filter values using formula (6.256) for $k = 0, 1, \dots, \lfloor \frac{N}{2} \rfloor$ and $l = 0, 1, \dots, \lfloor \frac{M}{2} \rfloor$. For

$N = M = 201$, this means that we compute the filter values for $k, l = 0, 1, \dots, 100$. For indices $k = 101, \dots, 200$ and $l = 0, 1, \dots, \lfloor \frac{M}{2} \rfloor$, we define the filter values using

$$H(k, l) = H(\alpha, l) \quad \text{for } k = \left\lfloor \frac{N}{2} \right\rfloor + 1, \dots, N - 1 \text{ and } l = 0, 1, \dots, \left\lfloor \frac{M}{2} \right\rfloor \quad (6.257)$$

where $\alpha \equiv N - k$. This reflects the filter values for the indices that correspond to negative frequencies, ie for indices $k = 101, \dots, 200$. Finally, we reflect again to create values for the l indices that correspond to negative frequencies

$$H(k, l) = H(k, \beta) \quad \text{for } k = 0, 1, \dots, N - 1 \text{ and } l = \left\lfloor \frac{M}{2} \right\rfloor + 1, \dots, M - 1 \quad (6.258)$$

where $\beta \equiv M - l$.

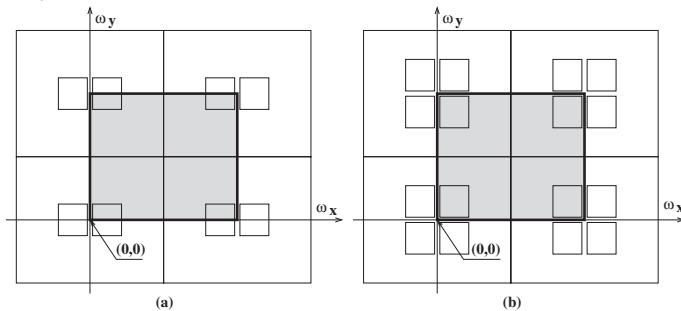


Figure 6.76: The frequency domain of a function is repeated in both directions. We may then select to work inside the domain defined by the shaded area, for which the dc component is at the corner.

Two of the three filters we construct this way are shown in figure 6.77. The first filter, designed to smooth along the k axis and respond to symmetric features along the l axis, was constructed with $k_0 = 0$, $k_1 = 20$, $l_0 = 20$ and $l_1 = 20$. The second one, designed to respond to features along either of the two diagonal directions, was constructed with $k_0 = k_1 = l_0 = l_1 = 20$.

After the filter we wish to use has been constructed, we construct functions $H_1(k, l)$ and $H_2(k, l)$, using (6.251), on page 660. Note that these formulae do not change, either we use ω or indices (k, l) . These functions are plotted in figure 6.78.

Finally, we are ready to produce the monogenic signal of the input image, $(f_H(x, y), f_{H_1}(x, y), f_{H_2}(x, y))$, with $f_H(x, y)$ being the inverse DFT of the point by point multiplication of the DFT of the input image with $H(k, l)$, $f_{H_1}(x, y)$ being the inverse DFT of the product of the DFT of the image with $H(k, l)$ and $H_1(k, l)$, and $f_{H_2}(x, y)$ being the inverse DFT of the product of the DFT of the image with $H(k, l)$ and $H_2(k, l)$.

The local image energy is then computed using (6.252). The local maxima of the energy are found by searching along the horizontal and the vertical axis, row by row and column by column, for places where the value of a pixel is larger than one of its

neighbours and larger or equal than the other neighbour, along the horizontal or the vertical direction, respectively. Figures 6.79a, 6.80a and 6.82a show the local energies of the input image in the three frequency bands we chose to work with, and figures 6.79b, 6.80b and 6.82b show the local maxima of these energies. The local maxima were computed after the energy function was smoothed with a Gaussian filter of size 21×21 (page 329).

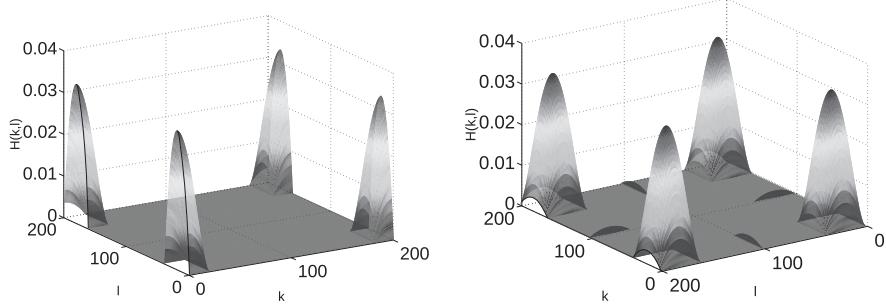


Figure 6.77: The filters that correspond to figures 6.73b and 6.73d.

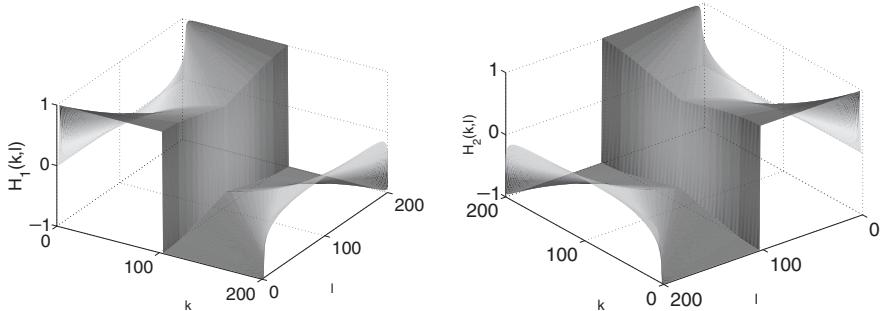
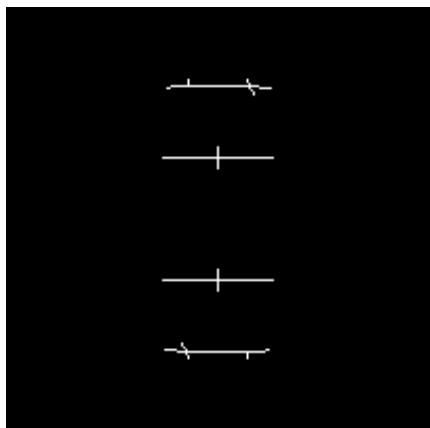


Figure 6.78: Functions $H_1(k,l)$ and $H_2(k,l)$, that are required to create the odd filters from the even one.

Before computing the orientation and symmetry of each feature, outputs $f_{H_1}(x,y)$ and $f_{H_2}(x,y)$ were also smoothed with the same filter. The orientation of each feature was computed using (6.254). This value varies between 0 and π . We scale it so that the range $[0, \pi/2]$ maps to the range $[0, 255]$, and the range $[\pi, \pi/2]$ maps also to the range $[0, 255]$, so that we do not distinguish between directions π and 0. These values are plotted in figures 6.79c, 6.81a and 6.82c. In all cases, black indicates horizontal local structure and white indicates vertical structure, while grey indicates diagonal structures. Note that the orientation of pixels that belong to flat regions is totally unreliable, as all the signal we get is due to noise (the antisymmetric filters have 0 dc components). The symmetry of a feature is computed using (6.253). It varies between 0 and $\pi/2$, and it is scaled to the range $[0, 255]$. These results are shown in figures 6.79d, 6.81b and 6.82d. In all cases, black indicates local antisymmetry, while white indicates local symmetry.



(a) Energy



(b) Local maxima

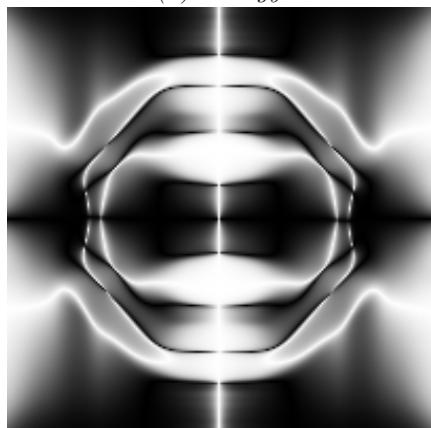
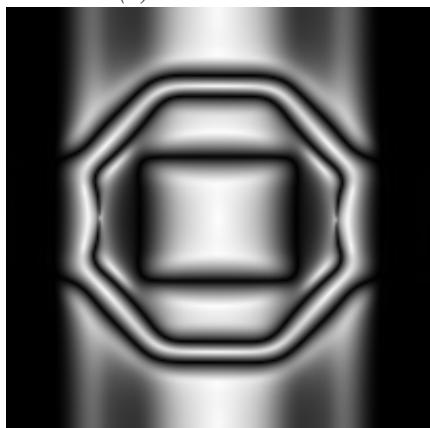
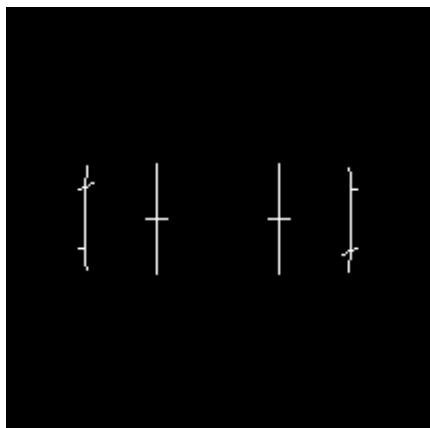
(c) Orientation ($\Theta(x, y)$)(d) Symmetry ($\Phi(x, y)$)

Figure 6.79: Filter designed to respond to horizontal features.



(a) Energy



(b) Local maxima

Figure 6.80: Filter designed to respond to vertical features.

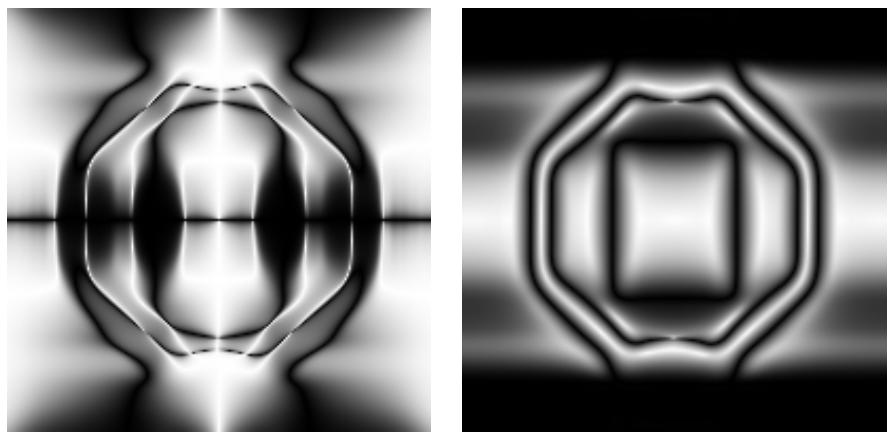


Figure 6.81: Filter designed to respond to vertical features.

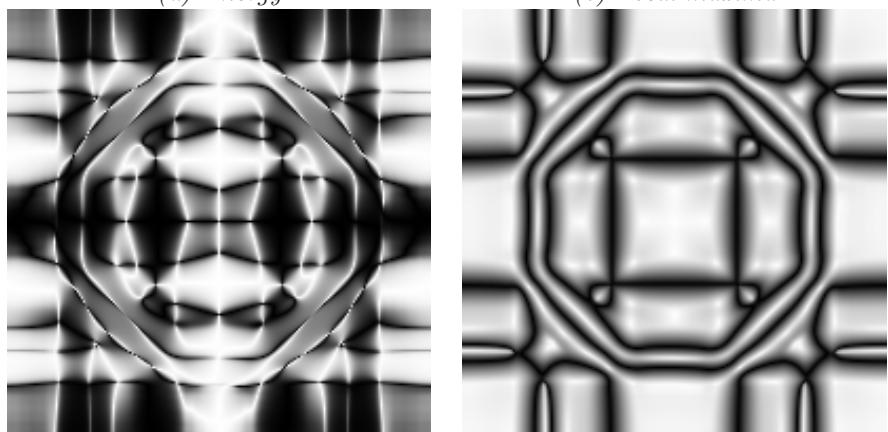
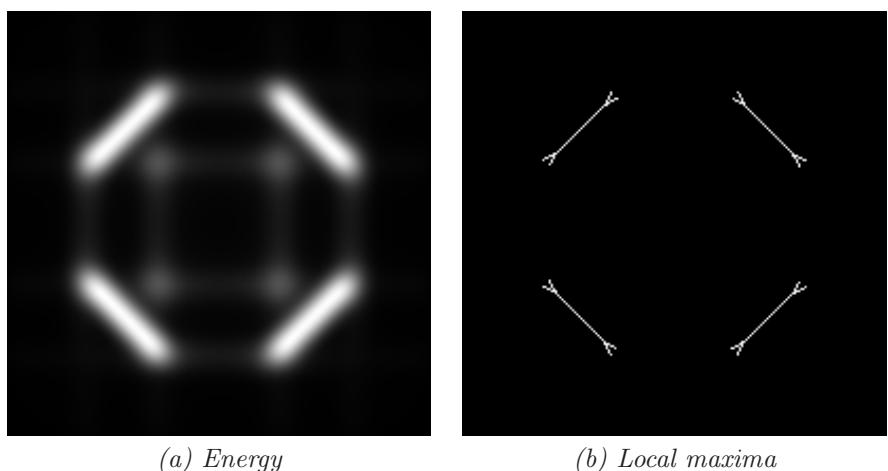


Figure 6.82: Filter designed to respond to diagonal features.

What is the “take home” message of this chapter?

This chapter dealt with the reduction of the information content of an image, so that it can be processed more easily by a computer vision system. It surveyed the two basic approaches for this purpose: region segmentation and edge detection. Region segmentation tries to identify spatially coherent sets of pixels that appear to constitute uniform patches in the image. These patches may represent surfaces or parts of surfaces of objects depicted in the image. Edge detection seeks to identify boundaries between such uniform patches. The most common approach for this is based on the estimate of the first derivative of the image. For images with low levels of noise, the Sobel masks are used to enhance the edges. For noisy images, the Canny filters should be used. Canny filters can be approximated by the derivative of a Gaussian, ie they have the form $x \exp\{-x^2/(2\sigma^2)\}$. Although parameter σ in this expression has to have a *specific* value for the filters to be optimal, according to Canny’s criteria, often people treat σ as a free parameter and experiment with various values of it. Care must be taken in this case when discretising the filter and truncating the Gaussian, not to create a filter with sharp ends. Either the Sobel masks are used or the derivatives of the Gaussian, the result is the enhancement of edges in the image. The output produced has to be further processed by non-maxima suppression (ie the identification of the local maxima in the output array) and thresholding (ie the retention of only the significant local maxima).

Edge detectors consist of all three stages described above and produce fragmented edges. Often, a further step is involved of linking the fragments together to create closed contours that identify uniform regions. If, however, the edges are identified as the zero crossings of the output of the Laplacian of a Gaussian filter, then closed edge contours are produced.

Alternatively, people may bypass this process by performing region segmentation directly and, if needed, extract the boundaries of the regions afterwards. Region-based methods are much more powerful when both attribute similarity and spatial proximity are taken into consideration, when deciding which pixels form which region. A powerful hybrid method, that uses gradient information and region growing, is that of the watershed transform. This method treats an image as a landscape: the areas of roughly uniform greyness are treated as the planes and plateaus of the landscape that have to be delineated. The lines that delineate them are the waterlines (bottoms of valleys that separate plateaus and hills) and the ridges (crescents of mountains that separate planes and slopes). Watershed approaches tend to oversegment the image and some postprocessing is necessary to agglomerate small fragments of regions to form larger regions.

Finally, we saw how edges and lines in an image may be extracted in one go, if we consider the analytic signal (for 1D) or its generalisation in 2D, namely the monogenic signal. The extraction of edges and lines is a special case of a very powerful method that allows the identification of local image structure and the degree of its symmetry, with the help of filters defined in the frequency domain. The monogenic signal is computed with the help of the Riesz transform, and it consists of three components: these three components may be used to extract the local energy of the image, the local phase (ie how symmetric or antisymmetric the image is at that location) and the local orientation of the identified feature. It can be used to extract features of multiple scales and features that persist over many scales, and so are more prominent image characteristics.

Chapter 7

Image Processing for Multispectral Images

What is a multispectral image?

It is an image that consists of many bands, sometimes only three (in which case it may be a **colour image**) and sometimes many more, even hundreds. Each band is a grey image, that represents the brightness of the scene according to the sensitivity of the sensor used to create that band. In such an image, every pixel is associated with a string of numbers, ie a vector, made up from the values of the pixel in the various bands. This string of numbers is the so called **spectral signature** of the pixel.

What are the problems that are special to multispectral images?

1. Replacing the bands with other bands that are either uncorrelated or independent; this problem is particularly relevant to remote sensing applications, but also to ordinary image processing when one wishes to create a single band grey image from the multispectral image.
2. Using the spectral signature of a pixel to recognise the type of object the pixel represents. This is a pattern recognition problem, that rests on the solution of the following image processing problem: remove the dependence of the spectral signature of a pixel on the spectrum of the light under which the image was captured. This is the problem of **spectral constancy**.
3. Dealing with a special subset of multispectral images, which consist of three bands only in the optical part of the electromagnetic spectrum, and which have to be processed in a way that either replaces or imitates the way humans perceive colours.
4. Using multispectral images in practical applications and performing routine operations with them. One of the issues here is that each image now is a vector field. So, for example, some filtering methods we have seen in previous chapters have to be adapted for vector-valued pixels.

Figure 7.1 shows schematically these problems.

What is this chapter about?

This chapter deals with all the above mentioned problems. The first two of them simply involve the generalisation of some of the methods we encountered in previous chapters, or even the development of some new methods specific for these tasks. We may collectively refer to them as “image preprocessing”, as these methods may be used to prepare an image for further image processing. The third problem will be discussed in association with the human visual system and in particular in relation to the meaning of the word “colour” in psychophysics. The last set of problems will be discussed after we have understood the psychophysics of colour vision, as, in some algorithms, the choices we have to make for various options rely on the target pattern recognition system the image is aimed for: if it is the human pattern recognition system, the way the human brain perceives colour matters.

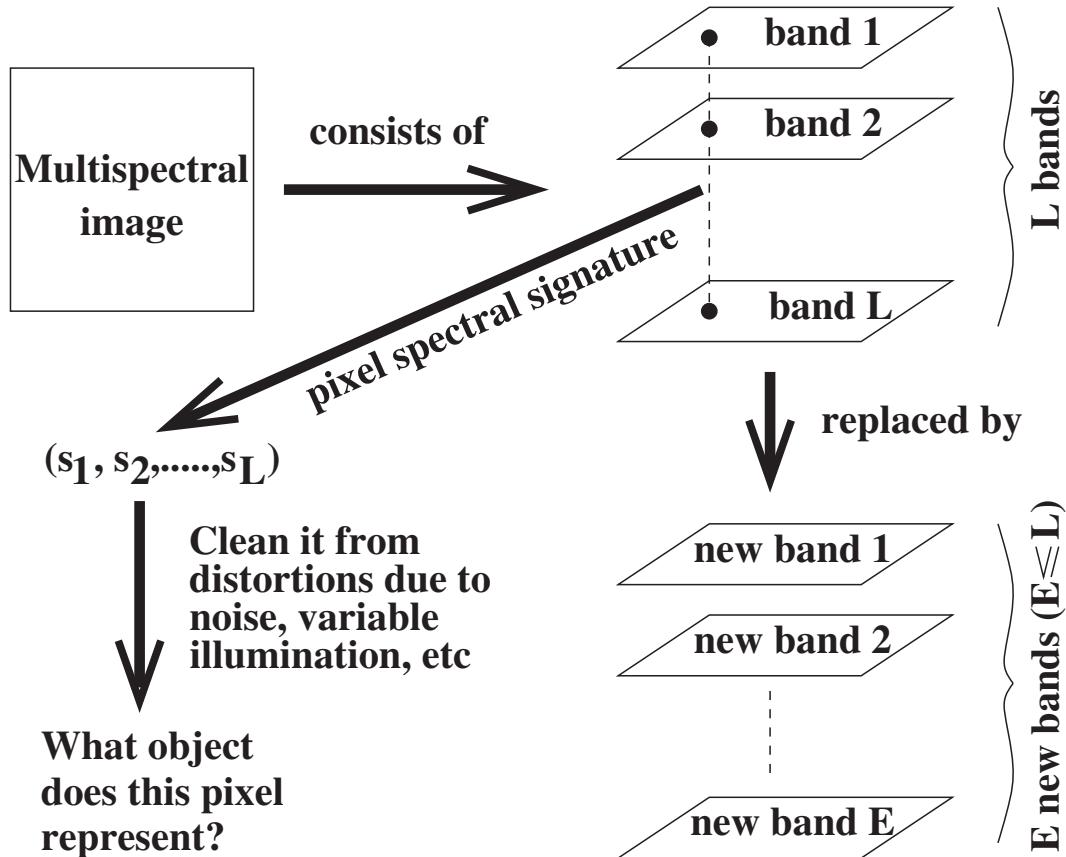


Figure 7.1: Schematic representation of the problems we try to solve using multispectral images. We may wish to replace the number of bands of the image with (usually fewer) new ones, which, for example, may be arranged according to contrast, and/or we may wish to use the values of pixels in the different bands to identify what type of object each pixel represents.

7.1 Image preprocessing for multispectral images

Why may one wish to replace the bands of a multispectral image with other bands?

1. In order to create bands that contain uncorrelated information. This process is similar to the Karhunen-Loeve transform we saw in Chapter 3, where basis images were created for an ensemble of images that shared some statistical properties. Here the ensemble of images consists of the bands of the same image. The uncorrelated bands may be used to reduce the number of bands of the image, by retaining, for example, only the bands with the maximum information.
2. A lot of image processing techniques do not require the spectral information; they may be perfectly adequately performed using a grey version of the multispectral image. Then, we have to define from the multispectral image a corresponding single grey image, that may be used for these tasks, and which will be somehow an optimal choice.
3. Remote sensing images have pixels that often correspond to patches on the ground of sizes several metres by several metres. Each such patch on the ground may contain several different objects, the spectral signatures of which will be recorded together by the corresponding pixel. The result is that the spectral signatures of the multispectral image may be mixed and one may wish to replace the mixed bands with other bands, in which the fractions of the spectral signatures of the objects present in the corresponding ground patch, are given. This process is known as **spectral unmixing**.

How do we usually construct a grey image from a multispectral image?

In the simplest approach, we select one of the bands of the image and use that as the grey image. However, this way significant image information may be lost. For example, some edges may be more prominent in one of the bands and others in another band. Selecting one of the bands means we may miss some of the image edges. One way around this is to average the grey values of all the bands. Indeed, this is quite commonly used by various image processing packages. However, the grey image we create this way is not optimal in the sense that it does not contain the maximum amount of image information (see Box 7.1).

How can we construct a single band from a multispectral image that contains the maximum amount of image information?

A grey image conveys information by the relative grey values of its pixels. The more contrast an image has, the more information it conveys. If one replaces the bands of the image with its uncorrelated components, the first uncorrelated band, which corresponds to maximum spread of pixel values, contains more information than any other band. Thus, the creation of a band

with maximum contrast is a byproduct of the creation of uncorrelated image bands. This can be achieved by using **principal component analysis**, or **PCA**, for short.

What is principal component analysis?

Principal component analysis is the methodology that allows one to identify the uncorrelated components of an ensemble of data. In Chapter 4 we saw how PCA (called Karhunen-Loeve transform there) allowed us to identify the uncorrelated components of an ensemble of *images*, with each image in the ensemble being considered as a version of a random field. Here we consider each *pixel* of the image to be a version of a random vector.

Let us consider for simplicity that we have three spectral bands, B_1 , B_2 and B_3 . Then, each image consists of three grey images. Alternatively, we may say that each pixel carries three values, one for each spectral band. We can plot these triplets in a 3D coordinate space, called $B_1B_2B_3$, because we measure the grey value of a pixel in each of the three bands along the three axes. The pixels of the multispectral image plotted in this space form a cloud of points. By identifying the principal axes of this cloud of points we can define a new coordinate system, such that the components of the original 3D vectors along these axes are uncorrelated. The principal axes are identified by the eigenvectors of the covariance matrix of the cloud of points. The spread of the components along each of the axes is given by the eigenvalue of the corresponding eigenvector. By selecting the eigenvector with the largest eigenvalue, and projecting all pixels (3D vectors) on it, we can construct from the original multispectral image three new bands and select from them the band with the maximum contrast.

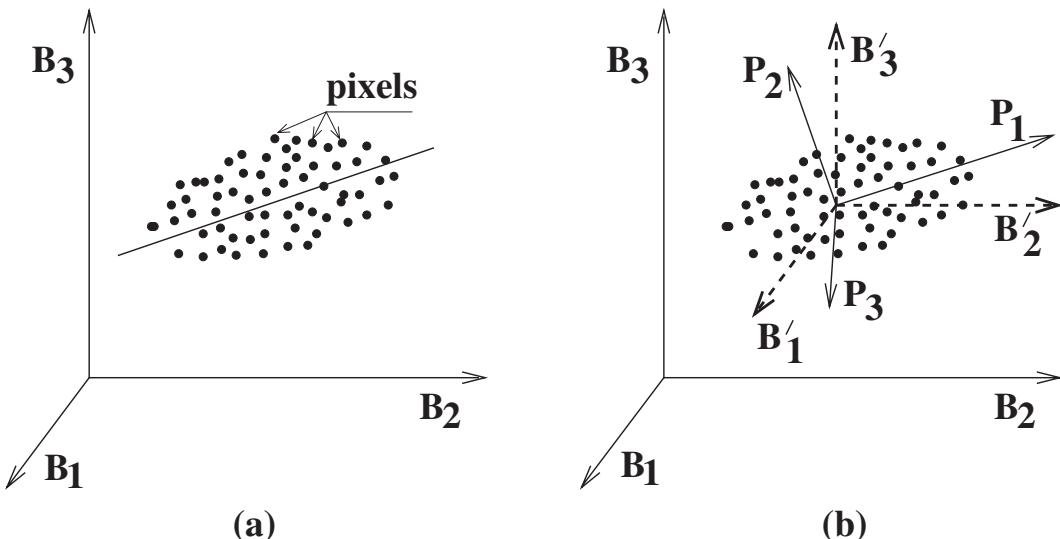


Figure 7.2: The pixels of a 3-band image form a cluster in the spectral space. (a) The maximum spread of this cluster may be along a line not parallel with any of the band axes. (b) The coordinate system may be translated so that its origin coincides with the cloud centre and then rotated, so that its first axis coincides with the direction of maximum spread.

Box 7.1. How do we measure information?

The information conveyed by a symbol is more significant when the probability of the symbol to arise is low. By “symbol” we mean a sentence, an event, or a code that is transmitted to us and has some meaning we can interpret. For example, the symbol “the sun will rise tomorrow from the east” has 0 information content, as the probability for this to happen is 1. It is plausible, therefore, to measure the information content of a symbol by the inverse of its probability to arise. In order to make this measure go to 0 when the probability is 1 and in order to have a measure that changes relatively slowly, instead of using the inverse of the probability, we use the logarithm of the inverse of the probability. We may consider an image to be a collection of symbols, namely the pixel values. Each pixel tells us something about the scene. To quantify how informative the image is, we may use the expectation value of the information content of its symbols, ie the average of $\log(1/p)$, with p being the probability for a particular grey value to arise, over the image:

$$H \equiv E \left\{ p \log \frac{1}{p} \right\} = - \sum_{k=1}^G p_k \log p_k \quad (7.1)$$

Here G is the number of distinct values the pixels can take (the number of grey levels in the image), and p_k is the frequency for a particular grey level in the image (the value of the k th bin of the normalised histogram of the image), when we use G bins to construct it. The information measure H is called the **entropy of the image**. The larger its value, the more information the image contains.

Example B7.1

Show that the histogram equalised version of an image conveys the maximum possible information the image may convey. (Perform the analysis in the continuous domain.)

The image conveys maximum information when its entropy is maximum. If we differentiate H , given by equation (7.1), with respect to p_k , we obtain:

$$\begin{aligned} \frac{\partial H}{\partial p_k} &= -\log p_k - p_k \frac{1}{p_k} \\ &= -\log p_k - 1 \end{aligned} \quad (7.2)$$

The second derivative of H with respect to p_k is:

$$\frac{\partial^2 H}{\partial p_k^2} = -\frac{1}{p_k} < 0 \quad (7.3)$$

This shows that at the root of the first derivative, H takes its maximum. The first derivatives of H with respect to the p_k s are 0 when $\log p_k = -1$ for all k . The important point here is that all p_k s have to take the same value. As they have to sum up to 1, this value is $1/G$ and this concludes the proof.

Example B7.2

Show that when the range of grey values of an image increases, its information content also increases.

Let us consider two versions of the same image, one with $G + 1$ grey values and one with G grey values. Let us also assume that we have applied histogram equalisation to each image to maximise the information it conveys, as shown in example 7.1. Then, in the first image, every grey value has probability $1/(G + 1)$ to arise. Substituting in (7.1), we work out that the information content of this image is

$$H_1 = - \sum_{k=1}^{G+1} \frac{1}{G+1} \log \left(\frac{1}{G+1} \right) = \log(G+1) \quad (7.4)$$

For the image with G grey levels, the information content is $H_2 = \log G$. Obviously $H_2 < H_1$. So, to maximise the information conveyed by a single band of a multispectral image, we must maximise the range of grey values of the band.

How do we perform principal component analysis in practice?

To perform principal component analysis we must diagonalise the covariance matrix of the data. The autocovariance function of the outputs of the assumed random experiment is

$$C(i, j) \equiv E\{(x_i(m, n) - x_{i0})(x_j(m, n) - x_{j0})\} \quad (7.5)$$

where $x_i(m, n)$ is the value of pixel (m, n) in band i , x_{i0} is the mean of band i , $x_j(m, n)$ is the value of the same pixel in band j , x_{j0} is the mean of band j and the expectation value is computed over all outcomes of the random experiment, ie over all pixels of the image. For an $M \times N$ image:

$$C(i, j) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N (x_i(m, n) - x_{i0})(x_j(m, n) - x_{j0}) \quad (7.6)$$

For a 3-band image, variables i and j take only three values, so the covariance matrix is a 3×3 matrix. For data that are uncorrelated, C is diagonal, ie $C(i, j) = 0$ for $i \neq j$. To achieve this, we must transform the data using the transformation matrix A made up from

the eigenvectors of the covariance matrix of the untransformed data. The process is as follows.

Step 1: Find the mean of the distribution of points in the spectral space, point $(\bar{B}_1, \bar{B}_2, \bar{B}_3)$, by computing the average grey value of each band.

Step 2: Subtract the mean grey value from the corresponding band. This is equivalent to translating the original coordinate system to be centred at the centre of the cloud of pixels (see axes $B'_1 B'_2 B'_3$ in figure 7.2b).

Step 3: Compute the autocorrelation matrix $C(i, j)$ of the initial cloud, using (7.6), where i and j identify the different bands.

Step 4: Compute the eigenvalues of $C(i, j)$ and arrange them in *decreasing* order. Form the eigenvector matrix A , having the eigenvectors as rows.

Step 5: Transform the cloud of pixels using matrix A . For a 3-band image, each triplet $\mathbf{x} = \begin{pmatrix} B_1 - \bar{B}_1 \\ B_2 - \bar{B}_2 \\ B_3 - \bar{B}_3 \end{pmatrix}$ is transformed into $\mathbf{y} = \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix}$ by: $\mathbf{y} = A\mathbf{x}$. In other words, the new values a pixel will carry will be given by $y_k = \sum_i a_{ki} x_i$, where k indexes the new bands, while i indexes the old bands.

This is a linear transformation. The new bands are linear combinations of the intensity values of the initial bands, arranged so that the first principal component contains most of the information for the image (see figure 7.2b). This is ensured by Step 4, where we use the largest eigenvalue for the first principal component. (The eigenvalue represents the spread of the data along the corresponding eigenvector.)

What are the advantages of using the principal components of an image, instead of the original bands?

1. The information conveyed by each principal band is maximal for the number of bits used, because the bands are uncorrelated and no information contained in one band can be predicted by the knowledge of the other bands.
2. If we want to use a grey version of the image, we can restrict ourselves to the first principal component only, and be sure that it has the maximum contrast and contains the maximum possible information conveyed by a single band of the image.

An example of principal component analysis is shown in figure 7.3. Although at first glance not much difference is observed between figures 7.3a, 7.3b, 7.3c and 7.3d, with a more careful examination, we can see that the first principal component combines the best parts of all three original bands: for example, the sky has maximum contrast in the third band and minimum contrast in the first band. In the first principal component, it has an intermediate contrast. The roof has maximum contrast in the first band, while it has much less contrast in the other two bands. In the first principal component, the roof has an intermediate contrast.

What are the disadvantages of using the principal components of an image instead of the original bands?

The grey values in the bands created from principal component analysis have no physical meaning, as they do not correspond to any physical bands. As a result, the grey value of a

(a) Band B_1 (d) 1st PC, P_1 (b) Band B_2 (e) 2nd PC, P_2 (c) Band B_3 (f) 3rd PC, P_3 Figure 7.3: Principal component analysis of the “Mount Athos border” (384×512 pixels).

pixel cannot be used for the classification of the pixel. This is particularly relevant to remote sensing applications, where, often, pixels are classified according to their grey values. In a principal component band, pixels that represent water, for example, may appear darker or brighter than other pixels in the image, depending on the image content, while the degree of greyness of water pixels in the various spectral bands is always consistent, well understood by remote sensing scientists, and often used to identify them.

Example 7.3

Is it possible for matrix C to represent the autocovariance matrix of a three-band image?

$$C = \begin{pmatrix} -1 & 0 & 1 \\ 0 & 1 & -2 \\ -2 & 2 & 0 \end{pmatrix} \quad (7.7)$$

This matrix cannot represent the autocovariance matrix of an image, because, from equation (7.6), it is obvious that C must be symmetric with positive elements along its diagonal.

Example 7.4

A 3-band image consists of bands with mean 3, 2 and 3, respectively. The autocovariance matrix of this image is given by:

$$C = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix} \quad (7.8)$$

A pixel has intensity values 5, 3 and 4 in the three bands, respectively. What will be the transformed values of the same pixel in the three principal component bands?

First, we must compute the eigenvalues of matrix C :

$$\begin{aligned} & \begin{vmatrix} 2-\lambda & 0 & 1 \\ 0 & 2-\lambda & 0 \\ 1 & 0 & 2-\lambda \end{vmatrix} = 0 \\ \Rightarrow & (2-\lambda)^3 - (2-\lambda) = 0 \Rightarrow (2-\lambda) [(2-\lambda)^2 - 1] = 0 \\ \Rightarrow & (2-\lambda)(1-\lambda)(3-\lambda) = 0 \end{aligned} \quad (7.9)$$

Therefore, $\lambda_1 = 3$, $\lambda_2 = 2$, $\lambda_3 = 1$. The corresponding eigenvectors are:

$$\begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 3 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \Rightarrow \begin{cases} 2x_1 + x_3 = 3x_1 \\ 2x_2 = 3x_2 \\ x_1 + 2x_3 = 3x_3 \end{cases} \Rightarrow \begin{cases} x_1 = x_3 \\ x_2 = 0 \\ x_1 + 2x_3 = 3x_3 \end{cases} \Rightarrow \mathbf{u}_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\begin{aligned}
 \left(\begin{array}{ccc} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{array} \right) \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 2 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \Rightarrow \begin{array}{l} 2x_1 + x_3 = 2x_1 \\ 2x_2 = 2x_2 \\ x_1 + 2x_3 = 2x_3 \end{array} \Rightarrow \begin{array}{l} x_2 \text{ anything} \\ x_1 = x_3 = 0 \end{array} \Rightarrow \mathbf{u}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\
 \left(\begin{array}{ccc} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{array} \right) \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \Rightarrow \begin{array}{l} 2x_1 + x_3 = x_1 \\ 2x_2 = x_2 \\ x_1 + 2x_3 = x_3 \end{array} \Rightarrow \begin{array}{l} x_1 = -x_3 \\ x_2 = 0 \end{array} \Rightarrow \mathbf{u}_3 = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}
 \end{aligned} \tag{7.10}$$

The transformation matrix A is:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \tag{7.11}$$

We first subtract the mean from pixel $\begin{pmatrix} 5 \\ 3 \\ 4 \end{pmatrix}$, to obtain $\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$ and then perform the transformation:

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{3}{\sqrt{2}} \\ 1 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{7.12}$$

Example 7.5

You are given the following 4×4 3-band image:

$$B_1 = \begin{pmatrix} 3 & 3 & 5 & 6 \\ 3 & 4 & 4 & 5 \\ 4 & 5 & 5 & 6 \\ 4 & 5 & 5 & 6 \end{pmatrix} \quad B_2 = \begin{pmatrix} 3 & 2 & 3 & 4 \\ 1 & 5 & 3 & 6 \\ 4 & 5 & 3 & 6 \\ 2 & 4 & 4 & 5 \end{pmatrix} \quad B_3 = \begin{pmatrix} 4 & 2 & 3 & 4 \\ 1 & 4 & 2 & 4 \\ 4 & 3 & 3 & 5 \\ 2 & 3 & 5 & 5 \end{pmatrix} \tag{7.13}$$

Calculate its three principal components and verify that they are uncorrelated.

First, we calculate the mean of each band:

$$\begin{aligned}
\bar{B}_1 &= \frac{1}{16}(3 + 3 + 5 + 6 + 3 + 4 + 4 + 5 + 4 + 5 + 5 + 6 + 4 + 5 + 5 + 6) \\
&= \frac{73}{16} = 4.5625 \\
\bar{B}_2 &= \frac{1}{16}(3 + 2 + 3 + 4 + 1 + 5 + 3 + 6 + 4 + 5 + 3 + 6 + 2 + 4 + 4 + 5) \\
&= \frac{60}{16} = 3.75 \\
\bar{B}_3 &= \frac{1}{16}(4 + 2 + 3 + 4 + 1 + 4 + 2 + 4 + 4 + 3 + 3 + 5 + 2 + 3 + 5 + 5) \\
&= \frac{54}{16} = 3.375
\end{aligned} \tag{7.14}$$

Next, we calculate the elements of the covariance matrix as:

$$\begin{aligned}
C_{B_1 B_1} &= \frac{1}{16} \sum_{k=1}^4 \sum_{l=1}^4 (B_1(k, l) - \bar{B}_1)^2 = 0.996094 \\
C_{B_1 B_2} &= \frac{1}{16} \sum_{k=1}^4 \sum_{l=1}^4 (B_1(k, l) - \bar{B}_1)(B_2(k, l) - \bar{B}_2) = 0.953125 \\
C_{B_1 B_3} &= \frac{1}{16} \sum_{k=1}^4 \sum_{l=1}^4 (B_1(k, l) - \bar{B}_1)(B_3(k, l) - \bar{B}_3) = 0.726563 \\
C_{B_2 B_2} &= \frac{1}{16} \sum_{k=1}^4 \sum_{l=1}^4 (B_2(k, l) - \bar{B}_2)^2 = 1.9375 \\
C_{B_2 B_3} &= \frac{1}{16} \sum_{k=1}^4 \sum_{l=1}^4 (B_2(k, l) - \bar{B}_2)(B_3(k, l) - \bar{B}_3) = 1.28125 \\
C_{B_3 B_3} &= \frac{1}{16} \sum_{k=1}^4 \sum_{l=1}^4 (B_3(k, l) - \bar{B}_3)^2 = 1.359375
\end{aligned} \tag{7.15}$$

Therefore, the covariance matrix is:

$$C = \begin{pmatrix} 0.996094 & 0.953125 & 0.726563 \\ 0.953125 & 1.937500 & 1.281250 \\ 0.726563 & 1.28125 & 1.359375 \end{pmatrix} \tag{7.16}$$

The eigenvalues of this matrix are:

$$\lambda_1 = 3.528765 \quad \lambda_2 = 0.435504 \quad \lambda_3 = 0.328700 \tag{7.17}$$

The corresponding eigenvectors are:

$$\mathbf{u}_1 = \begin{pmatrix} 0.427670 \\ 0.708330 \\ 0.561576 \end{pmatrix} \quad \mathbf{u}_2 = \begin{pmatrix} 0.876742 \\ -0.173808 \\ -0.448457 \end{pmatrix} \quad \mathbf{u}_3 = \begin{pmatrix} -0.220050 \\ 0.684149 \\ -0.695355 \end{pmatrix} \quad (7.18)$$

The transformation matrix, therefore, is:

$$A = \begin{pmatrix} 0.427670 & 0.708330 & 0.561576 \\ 0.876742 & -0.173808 & -0.448457 \\ -0.220050 & 0.684149 & -0.695355 \end{pmatrix} \quad (7.19)$$

We can find the principal components by using this matrix to transform the values of every pixel. For example, for the first few pixels we find:

$$\begin{aligned} \begin{pmatrix} -0.8485 \\ -1.5198 \\ -0.6039 \end{pmatrix} &= \begin{pmatrix} 0.427670 & 0.708330 & 0.561576 \\ 0.876742 & -0.173808 & -0.448457 \\ -0.220050 & 0.684149 & -0.695355 \end{pmatrix} \begin{pmatrix} 3 - 4.5625 \\ 3 - 3.75 \\ 4 - 3.375 \end{pmatrix} \\ \begin{pmatrix} -2.6800 \\ -0.4491 \\ 0.1027 \end{pmatrix} &= \begin{pmatrix} 0.427670 & 0.708330 & 0.561576 \\ 0.876742 & -0.173808 & -0.448457 \\ -0.220050 & 0.684149 & -0.695355 \end{pmatrix} \begin{pmatrix} 3 - 4.5625 \\ 2 - 3.75 \\ 2 - 3.375 \end{pmatrix} \\ \begin{pmatrix} -0.5547 \\ 0.6821 \\ -0.3486 \end{pmatrix} &= \begin{pmatrix} 0.427670 & 0.708330 & 0.561576 \\ 0.876742 & -0.173808 & -0.448457 \\ -0.220050 & 0.684149 & -0.695355 \end{pmatrix} \begin{pmatrix} 5 - 4.5625 \\ 3 - 3.75 \\ 3 - 3.375 \end{pmatrix} \end{aligned} \quad (7.20)$$

We use the first element of each transformed triplet to form the first principal component of the image, the second element for the second principal component and the third for the third one. In this way, we derive:

$$\begin{aligned} P_1 &= \begin{pmatrix} -0.8485 & -2.6800 & -0.5547 & 1.1428 \\ -3.9499 & 0.9958 & -1.5440 & 2.1318 \\ 0.2875 & 0.8619 & -0.5547 & 3.1211 \\ -2.2523 & 0.1536 & 1.2768 & 2.4128 \end{pmatrix} \\ P_2 &= \begin{pmatrix} -1.5198 & -0.4491 & 0.6821 & 0.9366 \\ 0.1731 & -0.9907 & 0.2538 & -0.2878 \\ -0.8169 & 0.3345 & 0.6821 & 0.1405 \\ 0.4276 & 0.5083 & -0.3886 & 0.3143 \end{pmatrix} \\ P_3 &= \begin{pmatrix} -0.6034 & 0.1027 & -0.3486 & -0.5799 \\ 0.1139 & 0.5444 & 0.5668 & 1.0085 \\ -0.1398 & 1.0197 & -0.3486 & 0.0931 \\ -0.1174 & 0.3355 & -1.0552 & -0.5911 \end{pmatrix} \end{aligned} \quad (7.21)$$

To confirm that these new bands contain uncorrelated data, we calculate their autocovariance matrix. First, we find the mean of each band: $\bar{P}_1, \bar{P}_2, \bar{P}_3$. Then we compute:

$$\begin{aligned}
 C_{P_1 P_1} &= \frac{1}{16} \sum_{i=1}^4 \sum_{j=1}^4 (P_1(i,j) - \bar{P}_1)^2 = 3.528765 \\
 C_{P_1 P_2} &= \frac{1}{16} \sum_{i=1}^4 \sum_{j=1}^4 (P_1(i,j) - \bar{P}_1)(P_2(i,j) - \bar{P}_2) = 0.0 \\
 C_{P_1 P_3} &= \frac{1}{16} \sum_{i=1}^4 \sum_{j=1}^4 (P_1(i,j) - \bar{P}_1)(P_3(i,j) - \bar{P}_3) = 0.0 \\
 C_{P_2 P_2} &= \frac{1}{16} \sum_{i=1}^4 \sum_{j=1}^4 (P_2(i,j) - \bar{P}_2)^2 = 0.435504 \\
 C_{P_2 P_3} &= \frac{1}{16} \sum_{i=1}^4 \sum_{j=1}^4 (P_2(i,j) - \bar{P}_2)(P_3(i,j) - \bar{P}_3) = 0.0 \\
 C_{P_3 P_3} &= \frac{1}{16} \sum_{i=1}^4 \sum_{j=1}^4 (P_3(i,j) - \bar{P}_3)^2 = 0.328700
 \end{aligned} \tag{7.22}$$

We see that this covariance matrix is diagonal, so it refers to uncorrelated data. To visualise these new bands, we have to map their values in the range [0, 255], with the same transformation formula. The minimum value we observe is -3.9499 and the maximum is 3.1211. The mapping then should be done according to

$$g_{new} = \left\lfloor \frac{g_{old} + 3.9499}{3.1211 + 3.9499} \times 255 + 0.5 \right\rfloor \tag{7.23}$$

where g_{old} is one of the values in matrices P_1, P_2 or P_3 and g_{new} is the corresponding new value.

Example 7.6

For the image of example 7.5 show that the first principal component has more contrast than any of the original bands.

The contrast of an image may be characterised by the range of grey values it has. We can see that the contrast of the original image was 3 in the first band, 5 in the second and 4 in the third band. The range of values in the first principal component is $3.1211 - (-3.9499) = 7.0710$. This is larger than any of the previous ranges.

Is it possible to work out only the first principal component of a multispectral image if we are not interested in the other components?

Yes. We may use the so called **power method**, that allows us to calculate only the most significant eigenvalue of a matrix. This, however, is possible only if the covariance matrix C has a single dominant eigenvalue (as opposed to two or more eigenvalues with the same absolute value). (See Box 7.2.)

Box 7.2. The power method for estimating the largest eigenvalue of a matrix

If matrix A is diagonalisable and has a single dominant eigenvalue, ie a single eigenvalue that has the maximum absolute value, this eigenvalue and the corresponding eigenvector may be estimated using the following algorithm.

Step 1: Select a vector of unit length, that is not parallel to the dominant eigenvalue. Let us call it \mathbf{x}'_0 . A vector chosen at random, most likely will not coincide with the dominant eigenvector of the matrix. Set $k = 0$.

Step 2: Compute $\mathbf{x}'_{k+1} = A\mathbf{x}'_k$.

Step 3: Normalise \mathbf{x}'_{k+1} to have unit length,

$$\mathbf{x}'_{k+1} \equiv \frac{\mathbf{x}'_{k+1}}{\sqrt{x_{k+1,1}^2 + x_{k+1,2}^2 + \dots + x_{k+1,N}^2}} \quad (7.24)$$

where $x_{k+1,i}$ is the i th element of \mathbf{x}'_{k+1} , which has N elements.

Step 4: If \mathbf{x}'_{k+1} is not the same as \mathbf{x}'_k , within a certain tolerance, set $k = k + 1$ and go to Step 2.

The more dissimilar the eigenvalues of the matrix are, the faster this algorithm converges.

Once convergence has been achieved, the dominant eigenvalue may be computed as the **Rayleigh quotient** of the estimated eigenvector \mathbf{x} :

$$\lambda_{dominant} = \frac{\mathbf{x}^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}} \quad (7.25)$$

This method may also be used to compute the minimum nonzero eigenvalue of a matrix (see example 7.8).

Example B7.7

Use the power method to work out an approximation of the dominant eigenvector and the corresponding eigenvalue of matrix C , given by (7.8).

We start by making a random choice $\mathbf{x}'_0 = (1, 1, 1)^T$. We compute \mathbf{x}_1 as:

$$\mathbf{x}_1 = C\mathbf{x}'_0 = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 3 \end{pmatrix} \quad (7.26)$$

We normalise \mathbf{x}_1 to produce $\mathbf{x}'_1 = (3/\sqrt{22}, 2/\sqrt{22}, 3/\sqrt{22})^T$. We then work out \mathbf{x}_2 :

$$\mathbf{x}_2 = C\mathbf{x}'_1 = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} \frac{3}{\sqrt{22}} \\ \frac{2}{\sqrt{22}} \\ \frac{3}{\sqrt{22}} \end{pmatrix} = \begin{pmatrix} \frac{9}{\sqrt{22}} \\ \frac{4}{\sqrt{22}} \\ \frac{9}{\sqrt{22}} \end{pmatrix} \quad (7.27)$$

We normalise \mathbf{x}_2 to produce $\mathbf{x}'_2 = (9/\sqrt{178}, 4/\sqrt{178}, 9/\sqrt{178})^T$. From this, we produce in the same way, $\mathbf{x}_3 = (27/\sqrt{178}, 8/\sqrt{178}, 27/\sqrt{178})^T$, which, when normalised, becomes $\mathbf{x}'_3 = (27/\sqrt{1522}, 8/\sqrt{1522}, 27/\sqrt{1522})^T$. Then:

$$\begin{aligned} \mathbf{x}_4 &= (81/\sqrt{1522}, 16/\sqrt{1522}, 81/\sqrt{1522})^T \\ \mathbf{x}'_4 &= (81/\sqrt{13378}, 16/\sqrt{13378}, 81/\sqrt{13378})^T \\ \mathbf{x}_5 &= (243/\sqrt{13378}, 32/\sqrt{13378}, 243/\sqrt{13378})^T \\ \mathbf{x}'_5 &= (243/\sqrt{119122}, 32/\sqrt{119122}, 243/\sqrt{119122})^T \\ \mathbf{x}_6 &= (729/\sqrt{119122}, 64/\sqrt{119122}, 729/\sqrt{119122})^T \\ \mathbf{x}'_6 &= (729/1032.95, 64/1032.95, 729/1032.95)^T \\ \mathbf{x}'_7 &= (0.7065, 0.0410, 0.7065)^T \\ \mathbf{x}'_8 &= (0.7068, 0.0275, 0.7068)^T \\ \mathbf{x}'_9 &= (0.7070, 0.0184, 0.7070)^T \end{aligned}$$

The corresponding eigenvalue is given by:

$$\lambda = \frac{(0.7070, 0.0184, 0.7070)^T \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} 0.7070 \\ 0.0184 \\ 0.7070 \end{pmatrix}}{0.7070^2 + 0.0184^2 + 0.7070^2} = 2.9997 \quad (7.28)$$

These results compare very well with the correct dominant eigenvalue that was computed to be 3 in example 7.4 and the corresponding eigenvector as $(1/\sqrt{2}, 0, 1/\sqrt{2})^T = (0.7071, 0, 0.7071)^T$. The other eigenvalues of this matrix were found to be 2 and 1 in example 7.4. These are not very different from 3 and that is why the power method for this matrix converges rather slowly.

Example B7.8

Show that the largest eigenvalue of a matrix is the smallest eigenvalue of its inverse and the corresponding eigenvector is the same.

Let us consider matrix A and its eigenpair (λ, \mathbf{x}) , where λ is the largest eigenvalue. By definition:

$$A\mathbf{x} = \lambda\mathbf{x} \quad (7.29)$$

Let us multiply both sides of this equation with A^{-1} , and remember that λ , being a scalar, can change position in the expression on the right-hand side:

$$\begin{aligned} A^{-1}A\mathbf{x} &= A^{-1}\lambda\mathbf{x} \Rightarrow \\ \mathbf{x} &= \lambda A^{-1}\mathbf{x} \Rightarrow \\ \frac{1}{\lambda}\mathbf{x} &= A^{-1}\mathbf{x} \end{aligned} \quad (7.30)$$

This shows that \mathbf{x} is an eigenvector of A^{-1} , with the corresponding eigenvalue being $1/\lambda$. Since λ is the largest eigenvalue of A , its inverse is obviously the smallest eigenvalue of A^{-1} . So, if we want to compute the smallest eigenvalue of a matrix, we can use the power method to compute the largest eigenvalue of its inverse.

What is the problem of spectral constancy?

The solution of this problem concentrates on making sure that the same physical surface patch, or different but spectrally identical surface patches, when imaged under two different illumination conditions and imaging geometries, are recognised by the computer as having the same spectrum. To deal with this problem, we have to think carefully on the process of image formation and correct for the recorded pixel values to eliminate the dependence on illumination.

What influences the spectral signature of a pixel?

The spectrum of the source that illuminates the imaged surface, the properties of the material the surface is made from and the sensitivity function of the sensor we use. The properties of the material of the surface are expressed by the **reflectance function**.

What is the reflectance function?

The reflectance function expresses the fraction of incident light that is reflected (as opposed to being absorbed) by the surface, as a function of the wavelength of the incident light.

Does the imaging geometry influence the spectral signature of a pixel?

No. The relative orientation of the imaged surface with respect to the illumination direction and the viewing direction of the camera influences only the total amount of light energy the

surface receives, but it does not influence the way this energy is distributed across different wavelengths. So, the *relative* values of the spectral signature components do not change with the imaging geometry.

How does the imaging geometry influence the light energy a pixel receives?

For a large fraction of materials, the light that is reflected by the surface is reflected with the same intensity in all directions. This then means that the orientation of the surface with respect of the camera is irrelevant to the process of image formation: in whichever direction the camera is, it will receive the same light intensity from the surface. Such materials are known as **Lambertian**. The orientation of the surface in relation to the illuminating source, however, is very important: if the surface is turned away from the light source, the surface will receive no light and it will be black. This dependence is expressed by the cosine of the angle between the normal vector of the surface and the direction of illumination. This is shown schematically in figure 7.4.

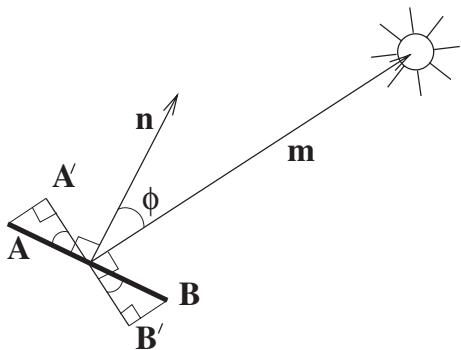


Figure 7.4: A surface AB catches light proportional to the area it presents towards the illuminating source. This is area $A'B'$, which is the projection of the surface orthogonal to the direction of the source. This is controlled by angle ϕ between the normal vector of the surface (vector \mathbf{n}) and the light source direction (vector \mathbf{m}). It is obvious that $A'B' = AB \cos \phi$ and, therefore, the light a surface catches per unit area is proportional to $\cos \phi$.

How do we model the process of image formation for Lambertian surfaces?

For Lambertian surfaces, we may model the process of image formation as

$$Q = \mathbf{m} \cdot \mathbf{n} \int_0^{+\infty} S(\lambda) I(\lambda) R(\lambda) d\lambda \quad (7.31)$$

where Q is the recording of a sensor with sensitivity function $S(\lambda)$, when it sees a surface with normal vector \mathbf{n} and reflectance function $R(\lambda)$, illuminated by a source with spectrum $I(\lambda)$, in direction \mathbf{m} . We see that the spectrum of a pixel, made up from the different Q values that correspond to the different camera sensors, depends on the imaging geometry through a simple scaling factor $\mathbf{m} \cdot \mathbf{n}$. The dependence on the imaging geometry and on the illumination

spectrum are interferences, since what we are interested in is to be able to reason about the materials identified by their function $R(\lambda)$, using the information conveyed by the observed spectral values Q .

How can we eliminate the dependence of the spectrum of a pixel on the imaging geometry?

Let us define

$$\alpha_{ij} \equiv \mathbf{m} \cdot \mathbf{n}_{ij} \quad (7.32)$$

which is a number expressing the imaging geometry for surface patch (i, j) , with normal vector \mathbf{n}_{ij} .

For a multispectral camera, with L bands, we have:

$$Q_l(i, j) = \alpha_{ij} \int_0^{+\infty} S_l(\lambda) I_0(\lambda) R_{ij}(\lambda) d\lambda \quad \text{for } l = 1, 2, \dots, L \quad (7.33)$$

To remove the dependence on the imaging geometry, define:

$$q_l(i, j) \equiv \frac{Q_l(i, j)}{\sum_{k=1}^L Q_k(i, j)} \quad \text{for } l = 1, 2, \dots, L \quad (7.34)$$

These $q_l(i, j)$ values constitute the **normalised spectrum** of pixel (i, j) , which is independent from the imaging geometry.

How can we eliminate the dependence of the spectrum of a pixel on the spectrum of the illuminating source?

We may assume, to a good approximation, that the camera responses are delta functions, ie:

$$S_l(\lambda) = \delta(\lambda - \lambda_l) \quad (7.35)$$

This assumption is nearly true for multispectral and hyperspectral cameras. Then:

$$Q_l(i, j) = \alpha_{ij} I_0(\lambda_l) R_{ij}(\lambda_l) \quad (7.36)$$

If the same surface patch is seen under a different illuminant, with different spectral characteristics, the values recorded for it will be:

$$Q'_l(i, j) = \alpha_{ij} I'_0(\lambda_l) R_{ij}(\lambda_l) \quad (7.37)$$

Note that:

$$\frac{Q_l(i, j)}{Q'_l(i, j)} = \frac{I_0(\lambda_l)}{I'_0(\lambda_l)} \quad \text{for } l = 1, 2, \dots, L \quad (7.38)$$

This expression means that under a new illuminant, the spectral values of a pixel become

$$\beta_l Q_l(i, j) \quad \text{for } l = 1, 2, \dots, L \quad (7.39)$$

where $\beta_l \equiv I'_0(\lambda_l)/I_0(\lambda_l)$ is a parameter that depends only on the band and is the same for all pixels. This means that all pixels in the image will change in the same way and so, the

average value of the image will change in the same way too. To remove dependence on the spectrum of the illuminant, we may, therefore, define

$$\tilde{q}_l(i, j) \equiv \frac{Q_l(i, j)}{\frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N Q_l(m, n)} \quad \text{for } l = 1, 2, \dots, L \quad (7.40)$$

where $M \times N$ is the size of the image.

What happens if we have more than one illuminating sources?

For K illuminating sources, in directions $\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_K$, the total illumination received by the surface patch will be

$$\sum_{k=1}^K \mathbf{m}_k \cdot \mathbf{n} I_{0k}(\lambda) \quad (7.41)$$

or:

$$\mathbf{n} \cdot \sum_{k=1}^K \mathbf{m}_k I_{0k}(\lambda) \quad (7.42)$$

We may always define an *effective* light source with intensity I_0 at \mathbf{m}_0 , such that:

$$I_0 \mathbf{m}_0 \equiv \sum_{k=1}^K \mathbf{m}_k I_{0k}(\lambda) \quad (7.43)$$

So, the analysis done for the use of a single illumination source may also be transferred to the case of several sources.

How can we remove the dependence of the spectral signature of a pixel on the imaging geometry and on the spectrum of the illuminant?

Some researchers have proposed an algorithm that may be used to remove the dependence of the image values on the imaging geometry and the spectrum of the illuminant, under three very important conditions:

- the surface is Lambertian;
- the camera sensitivities are delta functions;
- the surface has uniform spectrum, ie the surface is made up from the same material.

The steps of the algorithm are as follows.

Step 0: Set

$$\tilde{q}_l^0(i, j) = \frac{MN q_l(i, j)}{L \sum_{m=1}^M \sum_{n=1}^N q_l(m, n)} \quad (7.44)$$

where $q_l(i, j)$ is the brightness of pixel (i, j) in band l , L is the number of bands and $M \times N$ is the size of the image.

Step 1: At iteration step t and for $t \geq 1$, for each pixel (i, j) compute $q_l^t(i, j)$, using:

$$q_l^t(i, j) \equiv \frac{\tilde{q}_l^{t-1}(i, j)}{\sum_{k=1}^L \tilde{q}_k^{t-1}(i, j)} \quad \text{for } l = 1, 2, \dots, L \quad (7.45)$$

Step 2: For each pixel (i, j) compute $\tilde{q}_l(i, j)$, using

$$\tilde{q}_l^t(i, j) \equiv \frac{MNq_l^t(i, j)}{L \sum_{m=1}^M \sum_{n=1}^N q_l^t(m, n)} \quad (7.46)$$

Step 3: If the image changed, go to Step 1. If not, exit.

The factor of L in the denominator of (7.46) is to restore the correct range of values of $q_l(i, j)$, which is divided at each iteration step with the sum of L numbers of the same order of magnitude as itself (see equation (7.45)).

What do we have to do if the imaged surface is not made up from the same material?

In that case we have to segment the image first into regions of uniform pixel spectra and correct for the illumination interference for each region separately. Step 2 of the above algorithm then has to be replaced with:

Step 2(multimaterials): For each pixel (i, j) compute $\tilde{q}_l(i, j)$, using

$$\tilde{q}_l^t(i, j) \equiv \frac{N_R q_l^t(i, j)}{L \sum_{(m,n) \in R} q_l^t(m, n)} \quad (7.47)$$

where R is the region to which pixel (i, j) belongs and N_R is the number of pixels in that region.

What is the spectral unmixing problem?

This problem arises in remote sensing where low resolution sensors may capture images of the surface of the Earth, where an image pixel corresponds to a patch on the surface that contains several materials. The word “low” here is a relative term: it means that the resolution of the camera is low in relation to the typical size of the objects we wish to observe. For example, if we wish to identify cities, a pixel corresponding to $100m^2$ on the ground is adequate; if, however, we wish to identify cars, such a pixel size is too big.

Sensors on board Earth observation satellites may capture images with several bands. For example, a hyperspectral camera may capture images with a few hundred bands (instead of the 3-band images we commonly use). Let us say that we have data from a camera with L bands. This means that for each pixel we have L different values, one for each band. These values constitute the spectral signature of the pixel. If the pixel corresponds to a relatively large patch on the ground, several different materials may have contributed to its spectral signature. For example, if the pixel corresponds to 2×2 square metres on the ground, there might be some plants as well as some concrete around the flower bed of these plants. The spectral signature of the corresponding pixel then will be a combination of the spectral signatures of the plants and of the concrete. The *linear* mixing model assumes that this combination is linear and the mixing proportions correspond to the fractional covers of the patch on the ground. Thus, working out, for example, that the spectral signature of the pixel we observe can be created by adding 0.6 of the spectrum of the plants and 0.4 of the spectrum of the concrete, we may

infer that 60% of the $4m^2$ patch is covered by plants and 40% by concrete. That is why the mixing proportions sometimes are referred to as **cover proportions** as well.

How do we solve the linear spectral unmixing problem?

The problem may be solved by incorporating some prior knowledge concerning the spectra of the K different pure materials that are present in the scene. If \mathbf{s}_i is the spectrum of pixel i , and \mathbf{p}_k is the spectrum of pure material k , we may write

$$\mathbf{s}_i = \sum_{k=1}^K a_{ik} \mathbf{p}_k + \epsilon_i \quad (7.48)$$

where a_{ik} is the fraction of spectrum \mathbf{p}_k present in the spectrum of pixel i and ϵ_i is the residual error of the linear model used. For every pixel, we have K unknowns, the mixing proportions a_{ik} . We have at the same time MN such equations, one for each pixel, assuming that we are dealing with an $M \times N$ image. As these are vector equations, and as mixing proportions a_{ik} are the same for all bands, if we have L bands, we have L linear equations per pixel, for the K unknowns. If $K \leq L$ the system may be solved in the least square error sense to yield the unknown mixing proportions.

To guarantee that the a_{ik} values computed correspond indeed to mixing proportions, one usually imposes the constraints that $\sum_{k=1}^K a_{ik} = 1$ and that $0 \leq a_{ik} \leq 1$. Of course, if the linear mixing model were correct and if there were no intraclass variability and measurement noise, these constraints would have not been necessary, as the solution of the problem would automatically have obeyed them. However, natural materials exhibit quite a wide variation in their spectra, even if they belong to the same class (eg there are many different spectra that belong to class “grass”). So, to derive an acceptable solution, these constraints are necessary in practice.

Some variations of this algorithm have been proposed, where these constraints are not used, in order to allow for the case when the basic spectra \mathbf{p}_k are mixtures themselves. In some other methods, the statistical properties of the intraclass variations of the spectra of the various pure classes are used to enhance the system of equations that have to be solved. However, these advanced techniques are beyond the scope of this book.

Can we use library spectra for the pure materials?

In some cases, yes. The spectra we observe with the multispectral camera consist of the values of the true spectrum (the reflectance function) integrated with the spectral sensitivity function of the sensor. Library spectra usually have been obtained with instruments with much narrower sensitivity curves, amounting to being sampled versions of the continuous spectrum of the material. If the sensitivity curves of the cameras we use are very near delta functions, we may use library spectra for the pure materials. We have, however, to take into consideration the different conditions under which the spectra have been captured.

The spectra captured by Earth observation satellites, for example, may have to be corrected for atmospheric and other effects. Such correction is known as **radiometric correction**. The way it is performed is beyond the scope of this book and it can be found in books on remote sensing. Figure 7.5 shows schematically the relationship between camera based spectra and library spectra.

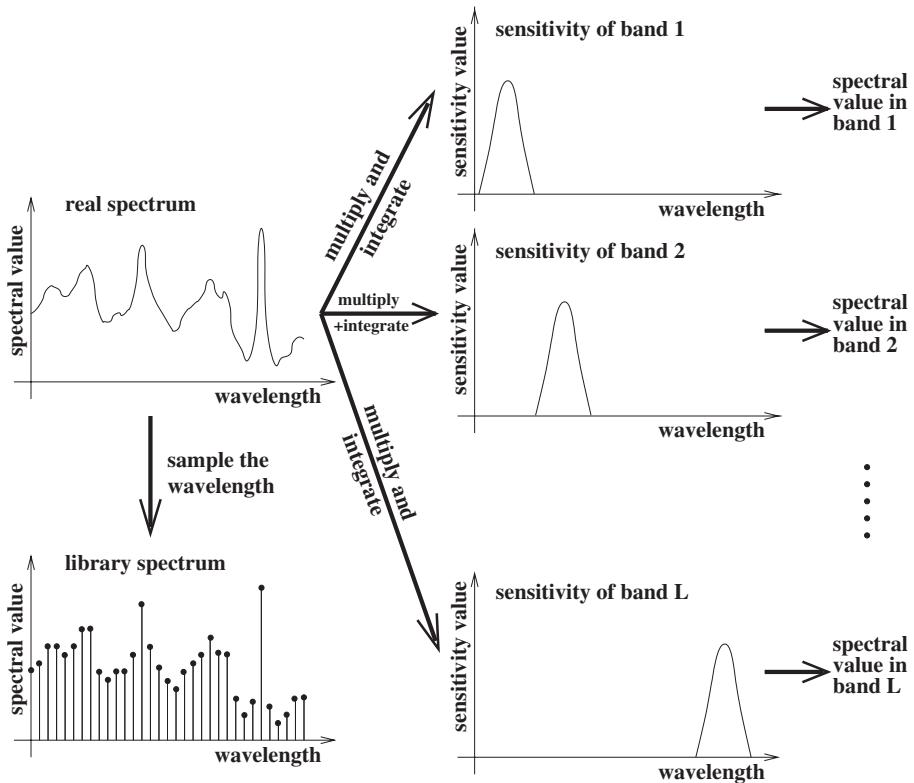


Figure 7.5: The library spectra may not be the same as the spectral signatures we obtain from a multispectral image.

How do we solve the linear spectral unmixing problem when we know the spectra of the pure components?

Let us consider the case where we have two pure spectra recorded together, as a mixed spectrum $s(\lambda)$, where λ is the wavelength of the electromagnetic spectrum. Let us assume that one of these pure spectra is $p_1(\lambda)$ and the other is $p_2(\lambda)$. As wavelength λ is discretised, we can consider these three functions, namely $s(\lambda)$, $p_1(\lambda)$ and $p_2(\lambda)$, as being represented by vectors \mathbf{s} , \mathbf{p}_1 and \mathbf{p}_2 , respectively, in an L -dimensional space, where L is the number of samples we use to sample the range of wavelengths. We shall use bold letters to indicate the discretised versions of the spectra, that are defined for specific values of λ , while keeping the non-bold version for the continuous spectrum.

In order to understand how to solve the problem, we shall start first from the case where $L = 2$, that is, each spectrum consists of only two values. Figure 7.6 shows this case schematically. Several points may be noted from this figure:

- (i) the pure spectra do not form an orthogonal system of axes;
- (ii) each spectrum has been captured possibly by different instrument and different calibration, so the length of each vector is irrelevant.

This means that:

- (i) we cannot use orthogonal projections to identify the components of \mathbf{s} along the two pure spectra directions;
- (ii) we cannot use the raw values of the pure spectra. We must normalise each one of them to become a unit vector in this space.

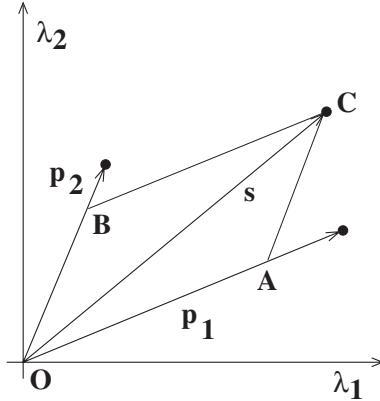


Figure 7.6: The pure spectra and the mixed spectrum can be thought of as vectors in a space that has as many axes as samples we use. We may consider the points where these vectors point as representing the spectra. Vectors \mathbf{p}_1 and \mathbf{p}_2 are the pure spectra. Length OA represents the amount of the first pure spectrum that has to be mixed with the amount of the second pure spectrum, represented by length OB , in order to produce the observed spectrum OC .

Let us denote by $\tilde{\mathbf{p}}_1$ and by $\tilde{\mathbf{p}}_2$ the two normalised pure spectra. Let us also denote by $(\tilde{p}_{11}, \tilde{p}_{12})$ the values of the first pure spectrum for the sampling wavelengths λ_1 and λ_2 . In a similar way, the two sample values of \mathbf{p}_2 are $(\tilde{p}_{21}, \tilde{p}_{22})$. Normalisation here means that

$$\tilde{p}_{11}^2 + \tilde{p}_{12}^2 = 1 \quad \tilde{p}_{21}^2 + \tilde{p}_{22}^2 = 1 \quad (7.49)$$

Note that the observed mixed spectrum \mathbf{s} does not have to be normalised. Let us say that the samples of \mathbf{s} are (s_1, s_2) . Once the pure spectra are normalised, we may write the equations

$$\begin{aligned} s_1 &= \alpha_1 \tilde{p}_{11} + \alpha_2 \tilde{p}_{21} \\ s_2 &= \alpha_1 \tilde{p}_{12} + \alpha_2 \tilde{p}_{22} \end{aligned} \quad (7.50)$$

where α_1 and α_2 are the mixing factors of the two pure spectra $p_1(\lambda)$ and $p_2(\lambda)$, respectively. This is a system of two equations with two unknowns, that can be solved to find values for the unknowns α_1 and α_2 . Once these values have been specified, they can be normalised so they sum up to 1, in order to become the sought mixing proportions:

$$\tilde{\alpha}_1 \equiv \frac{\alpha_1}{\alpha_1 + \alpha_2} \quad \tilde{\alpha}_2 \equiv \frac{\alpha_2}{\alpha_1 + \alpha_2} \quad (7.51)$$

Let us consider now the case where we use three samples to sample the range of wavelengths. Then the linear system of equations (7.50) will consist of three equations, one for

each sampling wavelength:

$$\begin{aligned}s_1 &= \alpha_1 \tilde{p}_{11} + \alpha_2 \tilde{p}_{21} \\ s_2 &= \alpha_1 \tilde{p}_{12} + \alpha_2 \tilde{p}_{22} \\ s_3 &= \alpha_1 \tilde{p}_{13} + \alpha_2 \tilde{p}_{23}\end{aligned}\quad (7.52)$$

Now we have more equations than unknowns. This system will then have to be solved by using the least squares error solution. This becomes very easy if we write the system of equations in a matrix form. Let us define vectors \mathbf{s} and \mathbf{a} , and matrix P :

$$\mathbf{s} \equiv \begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} \quad \mathbf{a} \equiv \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} \quad P \equiv \begin{pmatrix} \tilde{p}_{11} & \tilde{p}_{21} \\ \tilde{p}_{12} & \tilde{p}_{22} \\ \tilde{p}_{13} & \tilde{p}_{23} \end{pmatrix} \quad (7.53)$$

Note that vector \mathbf{s} is actually none other than the mixed spectrum. Matrix P is made up from the normalised pure spectra, written as columns next to each other. Vector \mathbf{a} is actually made up from the unknown mixing coefficients. System (7.52) then in matrix form may be written as:

$$\mathbf{s} = P\mathbf{a} \quad (7.54)$$

As matrix P is not square, we cannot take its inverse to solve the system. We may, however, multiply both sides of (7.54) with P^T :

$$P^T \mathbf{s} = P^T P \mathbf{a} \quad (7.55)$$

Matrix $P^T P$ now is a 2×2 matrix and we can take its inverse, in order to solve for \mathbf{a} :

$$\mathbf{a} = (P^T P)^{-1} P^T \mathbf{s} \quad (7.56)$$

This is the least square error solution of the system.

From this example, it should be noted that the maximum number of pure components we may recover is equal to the number of sampling points we use for λ (or the number of bands). As hyperspectral data are expected to have many more bands than components we wish to recover, this constraint does not pose any problem.

In the general case, we assume that we have spectra that are made up from L sampling points or bands, and that we wish to isolate K components. The mixed spectrum $s(\lambda)$ may be thought of as a vector \mathbf{s} of dimensions $L \times 1$. The vector of the unknown proportions is a $K \times 1$ vector. Matrix P is made up from the normalised pure spectra, written one next to the other as columns, and it has dimensions $L \times K$. We understand that $K < L$. The least squares error solution will be given by (7.56). Matrix $P^T P$ is a $K \times K$ matrix. We shall have to find its inverse. So, the algorithm is as follows.

Step 1: Identify the spectra of the pure substances you wish to identify in your mixture. Say they are K .

Step 2: Normalise the spectra of the pure substances by dividing each one with the square root of the sum of the squares of its values.

Step 3: Arrange the normalised spectra next to each other to form the columns of matrix P . This matrix will be $L \times K$ in size, where L is the numbers of samples we use to sample the electromagnetic spectrum.

Step 4: Compute matrix $Q \equiv P^T P$. This matrix will be $K \times K$ in size.

Step 5: Compute the inverse of matrix Q .

Step 6: Compute matrix $R \equiv Q^{-1} P^T$.

Step 7: Compute the $K \times 1$ vector $\mathbf{a} \equiv R\mathbf{s}$, where \mathbf{s} is the mixed spectrum.

Step 8: Divide each element of \mathbf{a} with the sum of all its elements. These are the mixing proportions that correspond to the mixed substances.

Is it possible that the inverse of matrix Q cannot be computed?

This happens when matrix Q is singular or nearly singular. This will happen if two of the rows or two of the columns of the matrix are almost identical. This is very unlikely, unless some of the pure substances we want to disambiguate have very similar spectra. To check that, we should not try to subtract one spectrum from the other in order to see whether we get a residual near 0, because the two spectra may be calibrated differently. Instead, we must compute their so called **spectral angular distance**, or **SAD**, for short. Let us consider again figure 7.6. All points along line OA have the same spectrum. However, two such points may have very large Euclidean distance from each other, although they lie on the same line OA . So, to estimate how different spectra \mathbf{p}_1 and \mathbf{p}_2 are, we must not measure the distance between points A and B , but rather the angle between lines OA and OB . This angle is computed as

$$SAD(\mathbf{p}_1, \mathbf{p}_2) \equiv \cos^{-1} \left\{ \frac{\mathbf{p}_1 \cdot \mathbf{p}_2}{\|\mathbf{p}_1\| \|\mathbf{p}_2\|} \right\} \quad (7.57)$$

where $\mathbf{p}_1 \cdot \mathbf{p}_2$ is the dot product of the two vectors and $\|\dots\|$ means the square root of the sum of squares of the elements of the vector it refers to. If this angle is near 0° or 180° , the two spectra are very similar, and only one of the two should be used. The two substances cannot be disambiguated. In practice, we do not need to take the inverse cosine to check. Just check the fraction:

$$Sim \equiv \left| \frac{\mathbf{p}_1 \cdot \mathbf{p}_2}{\|\mathbf{p}_1\| \|\mathbf{p}_2\|} \right| \quad (7.58)$$

If this is near +1, the two spectra cannot be disambiguated. The threshold of similarity has to be checked by trial and error. For example, we might say that spectra with SAD smaller than 5° or larger than 175° , might cause problems. These values correspond to values of Sim larger than 0.996. So, if Sim is larger than 0.996, we may not be able to separate the two substances.

What happens if the library spectra have been sampled at different wavelengths from the mixed spectrum?

It is important that all the spectra we use have been sampled at the same wavelengths with sensors that have comparable resolutions. If that is not the case, we must resample the library spectra, by using, for example, linear interpolation, so that we have their sample values at the same wavelengths as the mixed spectrum. An idea is to resample everything in the coarsest resolution, rather than upsampling everything in the finest resolution. This is because:

- (i) we have usually many more samples than substances we wish to disambiguate and so this will not pose a problem to the solution of the system of equations, and
- (ii) by upsampling, we assume that the sparsely sampled spectrum did not have any special

important feature at the positions of the missed samples, which were left out by the sparse sampling; an assumption that might not be true.

What happens if we do not know which pure substances might be present in the mixed substance?

Referring again to figure 7.6, any substance that is a mixture of pure spectra \mathbf{p}_1 and \mathbf{p}_2 will have its spectrum along a line starting from the origin O and lying between lines OA and OB . If line OC were outside the cone of directions defined by lines OA and OB , then it would be obvious that spectrum $s(\lambda)$ is not a linear mixture of spectra $p_1(\lambda)$ and $p_2(\lambda)$. This situation is depicted in figure 7.7. However, it might also be the case that, if we were to have an extra sampling wavelength, λ_3 , line OC would not even be on the plane defined by lines OA and OB . This situation is depicted in figure 7.8.

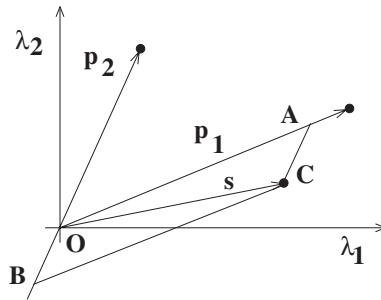


Figure 7.7: When the mixed spectrum \mathbf{s} is not a mixture of the pure spectra \mathbf{p}_1 and \mathbf{p}_2 , it may still be expressed as a linear combination of them, but at least one of the weights will be negative. In this example, the weight expressed by length OB is negative. A negative weight, therefore, indicates that we have not selected the pure spectra correctly.

Let us consider first the case depicted in figure 7.7. In this case, the situation may be interpreted as if spectrum \mathbf{p}_1 is a mixture of spectra \mathbf{s} and \mathbf{p}_2 . The algorithm will still yield a solution, but coefficient α_2 will be negative. This is because the relative values of the spectra imply that spectrum \mathbf{p}_1 may be expressed as a linear combination of the other two spectra with positive coefficients. However, if we solve such a mixture equation in terms of \mathbf{s} , at least one of the coefficients that multiplies one of the other spectra will be negative. In the algorithm of page 692, therefore, if we observe negative values in \mathbf{a} , we must realise that the pure spectra we assumed are not the right ones, or not the only ones.

Let us consider next the case depicted in figure 7.8. In this case, the fact that point C does not lie on the plane defined by points OAB , implies that there is at least one more pure substance present in the mixture. The least square error algorithm, described above, will find the components of the projection OD of vector OC on the plane defined by the two pure substances we consider. In this case, when we synthesise the observed mixed spectrum using the components of the pure substances we have estimated, we shall have a large residual error, equal to the distance CD of point C from the plane defined by the pure substances. This error can be estimated as follows:

$$e \equiv \| \mathbf{s} - (\alpha_1 \mathbf{p}_1 + \alpha_2 \mathbf{p}_2) \| \quad (7.59)$$

If this error is large, we may infer that we did not consider the right pure spectra. The algorithm then should introduce another pure spectrum and be executed again, in order to make sure that the observed mixed spectrum is synthesised with a satisfactorily low error. This may be repeated as many times as we wish, or as many times as we have pure spectra to add, until the error is acceptable. An acceptable error may be determined empirically, from several sets of training data, but perhaps it can be specified as a fraction of the total magnitude of spectrum \mathbf{s} . For example, if the square root of the sum of the squares of the values of \mathbf{s} is $\|\mathbf{s}\|$, an acceptable error might be $e_{threshold} = 0.05\|\mathbf{s}\|$.

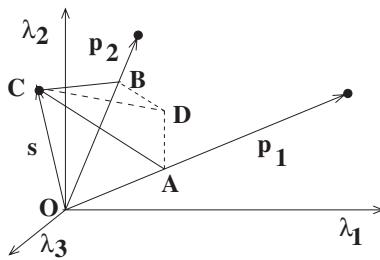


Figure 7.8: When the mixed spectrum \mathbf{s} is not a mixture of the pure spectra \mathbf{p}_1 and \mathbf{p}_2 , it may be lying outside the plane defined by the pure spectra. Then, the algorithm will yield a solution which will be actually synthesising the spectrum represented by point D , ie the projection of point C on the plane defined by the two spectra. The residual error, however, represented by length CD , will be high.

How do we solve the linear spectral unmixing problem if we do not know the spectra of the pure materials?

In this case the problem may be solved by applying, for example, independent component analysis (ICA). Note that the idea here is to identify some basis spectra in terms of which all observed spectra in the image can be expressed as linear combinations. The coefficients of the linear combinations are considered as the cover proportions of the pixels. Although this approach will produce some basis spectra, there is no guarantee that they will correspond to any real materials. In this case, one should not refer to them as “pure class spectra”, but with the commonly used term **end members**. The end members may be identified with any of the methods we considered in this book. For example, if we wish the end members to be uncorrelated, we may use PCA. If we wish the end members to be independent, we may use ICA.

Example 7.9

Formulate the problem of linear spectral unmixing so that it can be solved using PCA.

Consider that we computed the autocovariance matrix C of the image according to equation (7.6). For an L -band image, this matrix is $L \times L$ and it is expected to have at most L nonzero eigenvalues. However, most likely, it will have fewer, let us say that

its nonzero eigenvalues are $K < L$. This means that PCA defines a coordinate system with K axes only, in terms of which every pixel of the image may be identified. This is shown schematically in figure 7.9. Let us call the mean spectrum \mathbf{s}_0 . This vector is made up from the mean grey values of the different bands and it is represented by vector $O\Omega$ in figure 7.9. Let us also call the eigenvectors we identified with PCA \mathbf{e}_k , where $k = 1, 2, \dots, K$. We may say that the tip of each eigenvector defines the spectrum of one end member of the L -band image:

$$\mathbf{p}_k \equiv \mathbf{e}_k + \mathbf{s}_0 \quad (7.60)$$

Let us say that a pixel (m, n) , represented by point P in figure 7.9, has spectrum $\mathbf{s}(m, n)$ and, when it is projected on the axes identified by PCA, it has components $[\alpha_1(m, n), \alpha_2(m, n), \dots, \alpha_K(m, n)]$. We may write then:

$$\mathbf{s}(m, n) - \mathbf{s}_0 = \alpha_1(m, n)\mathbf{e}_1 + \alpha_2(m, n)\mathbf{e}_2 + \dots + \alpha_K(m, n)\mathbf{e}_K \quad (7.61)$$

Note that these coefficients do not necessarily sum up to 1. So, they cannot be treated as mixing proportions. We can substitute \mathbf{e}_k from (7.60) in terms of the end member spectra, to obtain:

$$\begin{aligned} \mathbf{s}(m, n) - \mathbf{s}_0 &= \alpha_1(m, n)(\mathbf{p}_1 - \mathbf{s}_0) + \alpha_2(m, n)(\mathbf{p}_2 - \mathbf{s}_0) + \dots + \alpha_K(m, n)(\mathbf{p}_K - \mathbf{s}_0) \\ &= \alpha_1(m, n)\mathbf{p}_1 + \alpha_2(m, n)\mathbf{p}_2 + \dots + \alpha_K(m, n)\mathbf{p}_K \\ &\quad - [\alpha_1(m, n) + \alpha_2(m, n) + \dots + \alpha_K(m, n)]\mathbf{s}_0 \end{aligned} \quad (7.62)$$

We may, therefore, write:

$$\begin{aligned} \mathbf{s}(m, n) &= \alpha_1(m, n)\mathbf{p}_1 + \alpha_2(m, n)\mathbf{p}_2 + \dots + \alpha_K(m, n)\mathbf{p}_K \\ &\quad + \mathbf{s}_0 \left[1 - \sum_{k=1}^K \alpha_k(m, n) \right] \end{aligned} \quad (7.63)$$

Accepting as end member spectra $\mathbf{s}_0, \mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_K$, the mixing proportions are: $1 - \sum_{k=1}^K \alpha_k(m, n)$, $\alpha_1(m, n)$, $\alpha_2(m, n), \dots, \alpha_K(m, n)$, respectively.

So, the algorithm is as follows:

Step 0: Compute the mean of each band and create vector \mathbf{s}_0 .

Step 1: Compute the autocovariance matrix of the image using equation (7.6).

Step 2: Compute the eigenvalues and eigenvectors \mathbf{e}_k of this matrix. Say there are K nonzero eigenvalues.

Step 3: Identify as end member spectra \mathbf{s}_0 and $\mathbf{p}_k \equiv \mathbf{s}_0 + \mathbf{e}_k$, for $k = 1, 2, \dots, K$.

Step 4: For each pixel (m, n) , with spectrum $\mathbf{s}(m, n)$, identify its mixing coefficients by taking the product of $[\mathbf{s}(m, n) - \mathbf{s}_0] \cdot \mathbf{e}_k \equiv \alpha_k(m, n)$.

Step 5: The mixing proportions for pixel (m, n) are

$$\left[1 - \sum_{k=1}^K \alpha_k(m, n), \alpha_1(m, n), \alpha_2(m, n), \dots, \alpha_K(m, n) \right],$$

corresponding to the $K + 1$ end member spectra $\mathbf{s}_0, \mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_K$, respectively.

Note that although $\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_K$ are orthogonal to each other, the end member spectra are not.

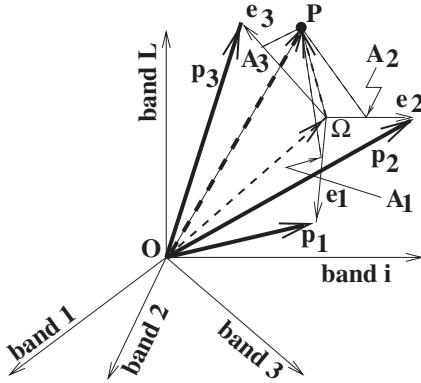


Figure 7.9: Vector $\overline{O\Omega}$ is the mean spectrum s_0 . Vectors $\mathbf{e}_1, \mathbf{e}_2$ and \mathbf{e}_3 are the eigenvectors of the covariance matrix that define the axes of the PCA system. The thick black vectors are the \mathbf{p}_k vectors which represent end member spectra. Lengths $\Omega A_1, \Omega A_2$ and ΩA_3 correspond to coefficients α_1, α_2 and α_3 for pixel P . Vector \overline{OP} represents the spectrum of the pixel and vector $\overline{\Omega P}$ represents the spectrum of the pixel minus the mean spectrum.

Example 7.10

Formulate the problem of linear spectral unmixing so that you can identify the end member spectra using ICA. How can you then use them to work out the mixing proportions?

We may consider that we want to solve the cocktail party problem (see page 234), where every pixel is a microphone that recorded a mixture of the pure signals we are seeking to identify. We are also interested in the mixing proportions with which each pixel recorded the pure signals. We shall solve the problem according to the formulation implied by figure 3.16a and the discussion that starts on page 264. Each image referred to in that figure and that discussion corresponds to a band we have here.

We shall adapt here the ICA algorithm on page 274 to our problem.

Let us assume that we have an image of size $M \times N$, consisting of L bands. Each pixel is a vector of size $L \times 1$. We have MN such vectors. Let us say that index m of a pixel defines in which row of the image the pixel is and index n identifies in which column of the image the pixel is. Then each pixel may be indexed by a unique index $i \equiv (n-1)M + m$. Here we assume that indices m and n take values from 1 to M and from 1 to N , respectively.

Step 1: Write the vectors of the pixels one next to each other, to form a matrix P that will be $L \times MN$ in size.

Step 2: Compute the average of all vectors, say vector \mathbf{m} , and remove it from each vector, thus creating MN vectors $\tilde{\mathbf{p}}_i$, of size $L \times 1$.

Step 3: Compute the autocorrelation matrix of the new vectors. Let us call \tilde{p}_{ki} the k th component of vector $\tilde{\mathbf{p}}_i$. Then the elements of the autocorrelation matrix C are:

$$C_{kj} = \frac{1}{MN} \sum_{i=1}^{MN} \tilde{p}_{ki} \tilde{p}_{ji} \quad \text{for } k, j = 1, 2, \dots, L \quad (7.64)$$

Matrix C is of size $L \times L$ and it may also be computed as:

$$C = \frac{1}{MN} \tilde{P} \tilde{P}^T \quad (7.65)$$

Step 4: Compute the nonzero eigenvalues of C and arrange them in decreasing order. Let us say that they are E . Let us denote by \mathbf{u}_l the eigenvector that corresponds to eigenvalue λ_l . We may write them next to each other to form matrix U .

Step 5: Scale the eigenvectors so that the projected components of vectors $\tilde{\mathbf{p}}_i$ will have the same variance along all eigendirections: $\tilde{\mathbf{u}}_l \equiv \mathbf{u}_l / \sqrt{\lambda_l}$.

You may write the scaled eigenvectors next to each other to form matrix \tilde{U} , of size $L \times E$.

Step 6: Project all vectors $\tilde{\mathbf{p}}_i$ on the scaled eigenvectors to produce vectors $\tilde{\mathbf{q}}_i$, where $\tilde{\mathbf{q}}_i$ is an $E \times 1$ vector with components \tilde{q}_{li} , given by:

$$\tilde{q}_{li} = \tilde{\mathbf{u}}_l^T \tilde{\mathbf{p}}_i \quad (7.66)$$

This step may be performed in a concise way as $\tilde{Q} \equiv \tilde{U}^T \tilde{P}$, with vectors $\tilde{\mathbf{q}}_i$ being the columns of matrix \tilde{Q} .

Step 7: Select randomly an $E \times 1$ vector \mathbf{w}_1 , with the values of its components drawn from a uniform distribution, in the range $[-1, 1]$.

Step 8: Normalise vector \mathbf{w}_1 so that it has unit norm: if w_{i1} is the i th component of vector \mathbf{w}_1 , define vector $\tilde{\mathbf{w}}_1$, with components:

$$\tilde{w}_{i1} \equiv \frac{w_{i1}}{\sqrt{\sum_j w_{j1}^2}} \quad (7.67)$$

Step 9: Project all data vectors $\tilde{\mathbf{q}}_i$ on $\tilde{\mathbf{w}}_1$, to produce the MN different projection components:

$$y_i = \tilde{\mathbf{w}}_1^T \tilde{\mathbf{q}}_i \quad (7.68)$$

These components will be stored in a $1 \times MN$ matrix (row vector), which may be produced in one go as $Y \equiv \tilde{\mathbf{w}}_1^T \tilde{Q}$.

Step 10: Update each component of vector $\tilde{\mathbf{w}}_1$ according to

$$w_{k1}^+ = \tilde{w}_{k1} \frac{1}{MN} \sum_{i=1}^{MN} G''(y_i) - \frac{1}{MN} \sum_{i=1}^{MN} \tilde{q}_{ki} G'(y_i) \quad (7.69)$$

Note that for $G'(y) = \tanh y$, $G''(y) \equiv dG'(y)/dy = 1 - (\tanh y)^2$.

Step 11: Normalise vector \mathbf{w}_1^+ , by dividing each of its elements with the square root

of the sum of the squares of its elements, $\sqrt{\sum_j (w_{j1}^+)^2}$, so that it has unit magnitude. Call the normalised version of vector \mathbf{w}_1^+ , vector $\tilde{\mathbf{w}}_1^+$.

Step 12: Check whether vectors $\tilde{\mathbf{w}}_1^+$ and $\tilde{\mathbf{w}}_1$ are sufficiently close. If, say, $|\tilde{\mathbf{w}}_1^{+T} \tilde{\mathbf{w}}_1| > 0.9999$, the two vectors are considered identical and we may adopt the normalised vector $\tilde{\mathbf{w}}_1^+$ as the first axis of the ICA system.

If the two vectors are different, ie if the absolute value of their dot product is less than 0.9999, we set $\tilde{\mathbf{w}}_1 = \tilde{\mathbf{w}}_1^+$ and go to Step 9.

After the first ICA direction has been identified, we proceed to identify the remaining directions. The steps we follow are the same as Steps 7–12, with one extra step inserted: we have to make sure that any new direction we select is orthogonal to the already selected directions. This is achieved by inserting an extra step between Steps 10 and 11, to make sure that we use only the part of vector \mathbf{w}_e^+ (where $e = 2, \dots, E$), which is orthogonal to all previously identified vectors $\tilde{\mathbf{w}}_t^+$, for $t = 1, \dots, e - 1$. This extra step is as follows.

Step 10.5: When trying to work out vector \mathbf{w}_e^+ , create a matrix B that contains as columns all $\tilde{\mathbf{w}}_t^+$, $t = 1, \dots, e - 1$, vectors worked out so far. Then, in Step 11, instead of using vector \mathbf{w}_e^+ , use vector $\mathbf{w}_e^+ - BB^T\mathbf{w}_e^+$.

The ICA basis vectors we identified correspond to (but are not the same as) the end member spectra. Each of the $\tilde{\mathbf{w}}_e^+$ vectors is of size $E \times 1$. The components of each such vector are measured along the scaled eigenaxes of matrix C . They may, therefore, be used to express vector $\tilde{\mathbf{w}}_e^+$ in terms of the original coordinate system, via vectors $\tilde{\mathbf{u}}_1$ and \mathbf{u}_1 . So, if we want to work out the end member spectra, the following step may be added to the algorithm.

Step 15: We denote by \mathbf{v}_e the position vector of the tip of vector $\tilde{\mathbf{w}}_e^+$ in the original coordinate system:

$$\begin{aligned}\mathbf{v}_e &= \tilde{w}_{1e}^+ \tilde{\mathbf{u}}_1 + \tilde{w}_{2e}^+ \tilde{\mathbf{u}}_2 + \cdots + \tilde{w}_{Ee}^+ \tilde{\mathbf{u}}_E + \mathbf{m} \\ &= \tilde{w}_{1e}^+ \frac{\mathbf{u}_1}{\sqrt{\lambda_1}} + \tilde{w}_{2e}^+ \frac{\mathbf{u}_2}{\sqrt{\lambda_2}} + \cdots + \tilde{w}_{Ee}^+ \frac{\mathbf{u}_E}{\sqrt{\lambda_E}} + \mathbf{m}\end{aligned}\quad (7.70)$$

Here \mathbf{m} is the mean vector we removed originally from the cloud of points to move the original coordinate system to the centre of the cloud. All these vectors may be computed simultaneously as follows. First, we create a diagonal matrix Λ , with $1/\sqrt{\lambda_i}$ along the diagonal. Then, vectors \mathbf{v}_e are the columns of matrix V , given by $V = U\Lambda W + \overline{M}$, where matrix \overline{M} is made up from vector \mathbf{m} repeated E times to form its columns.

There are E vectors \mathbf{v}_e , and they are of size $L \times 1$. These are the end member spectra. Once we have identified the end member spectra, we can treat them as the library spectra and use the algorithm on page 692 to work out the mixing proportions for each pixel.

7.2 The physics and psychophysics of colour vision

What is colour?

Colour is the subjective sensation we get when electro-magnetic radiation with wavelengths in the range [400nm, 700nm] reaches our eyes¹. This range of wavelengths is known as the **optical part of the electromagnetic spectrum**.

Therefore, colour really has meaning only if the human visual system is involved. So, one cannot discuss colour independent from the human visual system.

What is the interest in colour from the engineering point of view?

There are two major areas of research in relation to colour images:

- viewing an object that emits its own light;
- viewing an object that is illuminated by some light source.

In the first case, researchers are interested in the way light sources could be combined to create the desirable effect in the brain of the viewer. This line of research is related to the manufacturing of visual display units, like television sets and computer screens.

In the second case, researchers are interested in the role colour plays in human vision and in ways of emulating this aspect of vision by computers, for better artificial vision systems.

Figure 7.10 shows schematically the stages involved in the process of viewing dark objects and light sources. The only difference really is in the presence of an illuminating source in the former case. The role of the illuminating source in the appearance of the viewed object is part of the research in relation to viewing dark objects.

What influences the colour we perceive for a dark object?

Two factors: the spectrum of the source that illuminates the object and the reflectance function of the surface of the object.

This is expressed as

$$B(\lambda) = I(\lambda)R(\lambda) \quad (7.71)$$

where λ is the wavelength of the electromagnetic spectrum, $I(\lambda)$ is the spectrum of the illumination falling on the surface patch, $R(\lambda)$ is the reflectance function of the material the object is made from (or its surface is painted with) and $B(\lambda)$ is the spectrum of the light that reaches the sensor and is associated with the corresponding pixel of the image. (We assume here that each pixel receives the light from a finite patch of the imaged surface, and that there is one-to-one correspondence between pixels and viewed surface patches, so that either we talk about pixel (x, y) or surface patch (x, y) , we talk about the same thing.)

¹The visible range may be slightly broader. The limits stated are only approximately correct, as individuals vary.

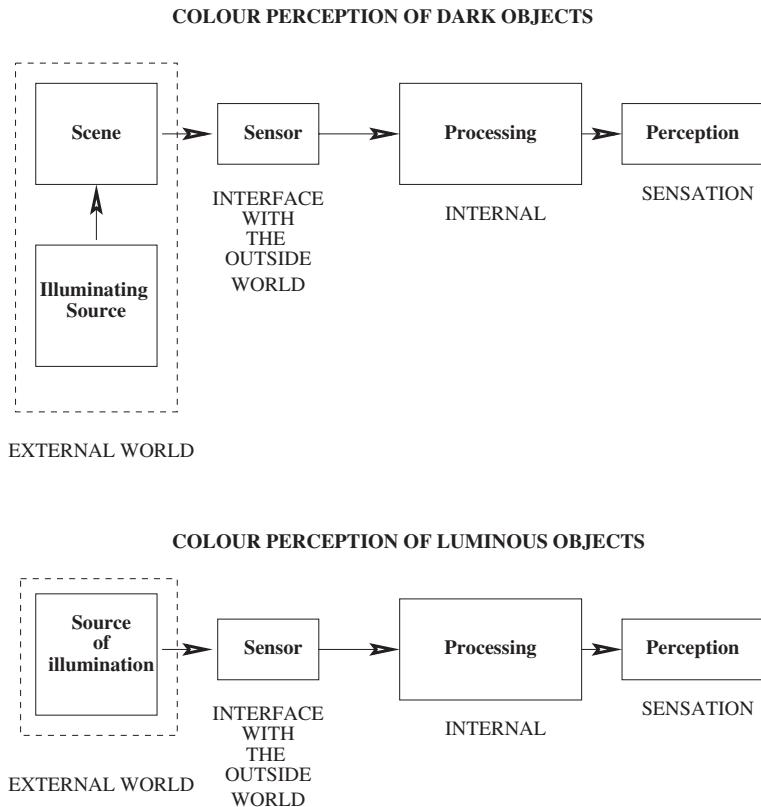


Figure 7.10: The process of vision is the same either we see a dark or a bright object. The difference is in the necessity of an illuminating source in order to view a dark object. The effect of the illuminating source on the way the object appears is very significant.

Note that this is a simplistic model of the effect of illumination. There are materials which do not just reflect the light which falls on them, but they partly absorb it and then re-emit it in a different range of wavelengths than that of the incident light. The object will be visible as long as $B(\lambda)$ is nonzero for at least some values of λ in the range [400nm, 700nm].

In order to avoid having to deal with continuous functions, we often sample the range of wavelengths of the electromagnetic spectrum, and so functions $B(\lambda)$, $I(\lambda)$ and $R(\lambda)$ become continued valued discrete functions. For example, if we discretise the range of λ [400nm, 700nm], considering samples every 10nm, functions $I(\lambda)$, $R(\lambda)$ and $B(\lambda)$ become 31D vectors, the elements of which take continuous values.

The human visual system evolved in the natural world, where the illuminating source was the sun and the materials it was aimed at distinguishing were the natural materials. Its functionality, therefore, is very much adapted to cope with the variations of the daylight and to deal with the recognition of natural materials.

What causes the variations of the daylight?

The quality of the daylight changes with the time of day, the season and the geographical location.

How can we model the variations of the daylight?

Scientists recorded the spectrum of the daylight, between 300nm and 830nm , in a large variety of locations, times of day and seasons. This way, they created an *ensemble* of versions of the spectrum of the daylight. This ensemble of functions, $I(\lambda)$, when considered only in the $[400\text{nm}, 700\text{nm}]$ range and sampled every 10nm , becomes an ensemble of 31D vectors representing the daylight over the range of visible wavelengths. We may consider a 31D space, with 31 coordinate axes, so that we measure one component of these vectors along one of these axes. Then, each 31D vector will be represented by a point and all recorded spectra will constitute a cloud of points in this space. It turns out that this cloud of points does not occupy all 31 dimensions, but only two dimensions. In other words, it turns out that with a careful choice of the coordinate axes, one may fully specify the position of each of these points by using two coordinates only (the remaining 29 being 0), as opposed to having to know the values of all 31 coordinates. We say that the data we have lie on a **2D manifold** in the 31D space. The word manifold means super-surface.

Figure 7.11 explains the idea of points in a 3D space, lying on a 2D manifold. As a consequence, these points may be fully defined by two (instead of three) coordinates, if we select the coordinate axes appropriately. The situation of going from 31D down to 2D is analogous.

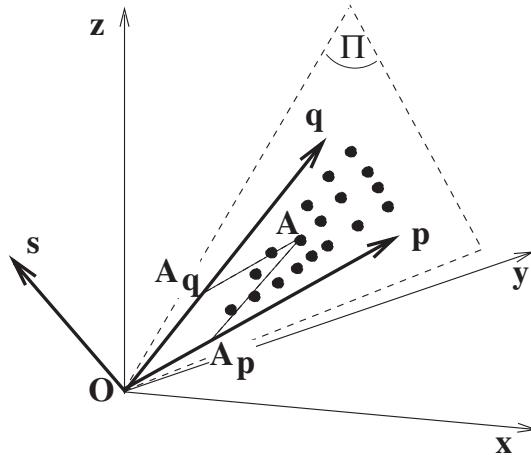


Figure 7.11: The data points in this 3D space happen to fall on a 2D plane Π . We may then choose as our coordinate axes vectors \mathbf{p} and \mathbf{q} , on this plane, and vector \mathbf{s} perpendicular to the plane. In terms of the Opq s coordinate system, a point A may be represented by two numbers only, namely the lengths OA_p and OA_q . Its coordinate with respect to the third axis (\mathbf{s}) is 0. The new basis vectors \mathbf{p} and \mathbf{q} will, of course, be functions of the old coordinates (x, y, z) , so we may write $\overline{OA}(x, y, z) = A_p \mathbf{p}(x, y, z) + A_q \mathbf{q}(x, y, z)$. Here (A_p, A_q) are two numbers fully characterising $A(x, y, z)$, which is the function expressed by point A , as long as we know the basis functions $\mathbf{p}(x, y, z)$ and $\mathbf{q}(x, y, z)$, which are common for all data points.

One, therefore, may write

$$I(\lambda) = i_1(\lambda) + a_2 i_2(\lambda) + a_3 i_3(\lambda) = \sum_{k=1}^3 a_k i_k(\lambda) \quad (7.72)$$

where $i_1(\lambda)$ is the average spectrum of daylight, $a_1 = 1$, (a_2, a_3) are two scalars fully characterising the natural illuminating source, and $i_2(\lambda)$ and $i_3(\lambda)$ are some universal basis functions.

$\lambda(nm)$	$i_1(\lambda)$	$i_2(\lambda)$	$i_3(\lambda)$	$\lambda(nm)$	$i_1(\lambda)$	$i_2(\lambda)$	$i_3(\lambda)$
300	0.04	0.02	0.04	570	96.0	-1.6	0.2
310	6.0	4.5	2.0	580	95.1	-3.5	0.5
320	29.6	22.4	4.0	590	89.1	-3.5	2.1
330	55.3	42.0	8.5	600	90.5	-5.8	3.2
340	57.3	40.6	7.8	610	90.3	-7.2	4.1
350	61.8	41.6	6.7	620	88.4	-8.6	4.7
360	61.5	38.0	5.3	630	84.0	-9.5	5.1
370	68.8	42.4	6.1	640	85.1	-10.9	6.7
380	63.4	38.5	3.0	650	81.9	-10.7	7.3
390	65.8	35.0	1.2	660	82.6	-12.0	8.6
400	94.8	43.4	-1.1	670	84.9	-14.0	9.8
410	104.8	46.3	-0.5	680	81.3	-13.6	10.2
420	105.9	43.9	-0.7	690	71.9	-12.0	8.3
430	96.8	37.1	-1.2	700	74.3	-13.3	9.6
440	113.9	36.7	-2.6	710	76.4	-12.9	8.5
450	125.6	35.9	-2.9	720	63.3	-10.6	7.0
460	125.5	32.6	-2.8	730	71.7	-11.6	7.6
470	121.3	27.9	-2.6	740	77.0	-12.2	8.0
480	121.3	24.3	-2.6	750	65.2	-10.2	6.7
490	113.5	20.1	-1.8	760	47.7	-7.8	5.2
500	113.1	16.2	-1.5	770	68.6	-11.2	7.4
510	110.8	13.2	-1.3	780	65.0	-10.4	6.8
520	106.5	8.6	-1.2	790	66.0	-10.6	7.0
530	108.8	6.1	-1.0	800	61.0	-9.7	6.4
540	105.3	4.2	-0.5	810	53.3	-8.3	5.5
550	104.4	1.9	-0.3	820	58.9	-9.3	6.1
560	100.0	0.0	0.0	830	61.9	-9.8	6.5

Table 7.1: The mean ($i_1(\lambda)$) daylight spectrum and the first two principal components of daylight variations ($i_2(\lambda)$ and $i_3(\lambda)$).

Table 7.1 gives functions $i_k(\lambda)$, for $k = 1, 2, 3$, as functions of λ , sampled every 10nm. Figure 7.12 shows the plots of these three functions. These three functions were computed by applying principal component analysis (PCA) to 622 daylight spectra. Each measured spectrum recorded the spectral radiant power of daylight (measured in watts per square metre) per unit wavelength interval (measured in metres). These absolute measurements were then normalised, so that all spectra had value 100 for wavelength $\lambda = 560nm$. Such spectra are referred to as **relative spectral radiant power distributions**. They were these normalised spectra that were used in the PCA analysis. Note that by talking on “daylight”, as opposed to “solar light”, we imply the total ambient light, not only the light a surface receives directly from the sun, but also whatever light a surface receives reflected from all other surfaces near it. Note also, that the solar light is not only in the visible wavelengths in

the range $[400\text{nm}, 700\text{nm}]$, but it extends well beyond this range. In table 7.1 we include also the daylight spectra in the **ultraviolet** region (300nm to 400nm) and in the **near infrared** region (700nm to 830nm).

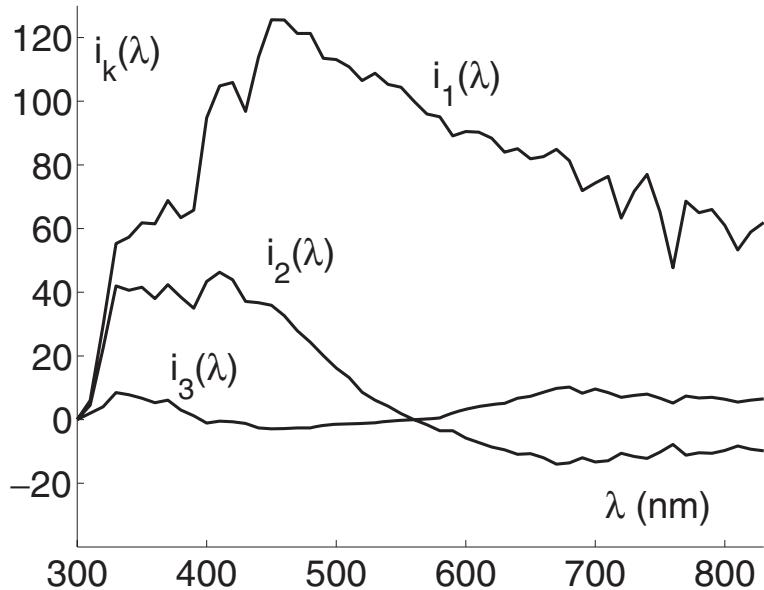


Figure 7.12: The mean ($i_1(\lambda)$) daylight spectrum and the first two principal components ($i_2(\lambda)$ and $i_3(\lambda)$) of daylight variations, as functions of the wavelength λ of the electromagnetic spectrum, measured in nanometres (nm).

Box 7.3. Standard illuminants

In order to facilitate the communication between colour vision scientists, the Commission Internationale de l'Eclairage (CIE) defined certain illumination spectra that are considered as standard. These illumination spectra may be synthesised from the spectra given in table 7.1. Each one of them is fully specified by its so called **correlated colour temperature**. The correlated colour temperature of an illuminant is defined as the temperature of the black body radiator, that produces a colour, which is perceived to be most similar to the colour produced by the illuminant, at the same overall brightness and under the same viewing conditions. The standard daylight illuminants are referred to by the letter D and the first two digits of their correlated colour temperature. Table 7.2 lists them alongside their corresponding correlated colour temperature. The most commonly used illuminant is the D_{65} . The spectra of these illuminants may be computed by using the following formulae, which depend only on the correlated

colour temperature T_c of each illuminant, measured in degrees Kelvin.

Step 0: Compute:

$$s \equiv \frac{10^3}{T_c} \quad (7.73)$$

Step 1: Compute the following.

For correlated colour temperature in the range [4000, 7000]

$$\begin{aligned} x &\equiv -4.6070s^3 + 2.9678s^2 + 0.09911s + 0.244063 \\ & \end{aligned} \quad (7.74)$$

while for correlated colour temperature in the range [7000, 25000]:

$$\begin{aligned} x &\equiv -2.0064s^3 + 1.9018s^2 + 0.24748s + 0.237040 \\ y &\equiv -3.000x^2 + 2.870x - 0.275 \end{aligned} \quad (7.75)$$

Values (x, y) are the so called **chromaticity coordinates** of the illuminant.

Step 2: Compute:

$$\begin{aligned} d_0 &\equiv 0.0241 + 0.2562x - 0.7341y \\ d_1 &\equiv -1.3515 - 1.7703x + 5.9114y \\ d_2 &\equiv 0.0300 - 31.4424x + 30.0717y \end{aligned} \quad (7.76)$$

Step 3: Compute:

$$a_2 \equiv \frac{d_1}{d_0} \quad a_3 \equiv \frac{d_2}{d_0} \quad (7.77)$$

Step 4: Synthesise the relative spectral radiant power of the illuminant, using:

$$I_D(\lambda) = i_1(\lambda) + a_2 i_2(\lambda) + a_3 i_3(\lambda) \quad (7.78)$$

Table 7.2 lists in its last two columns the values of a_2 and a_3 that should be used in conjunction with table 7.1, to produce the spectra of the three standard illuminants.

Illuminant	$T_c(^{\circ}K)$	a_2	a_3
D_{55}	5503	-0.78482	-0.19793
D_{65}	6504	-0.29447	-0.68754
D_{75}	7504	0.14520	-0.75975

Table 7.2: Standard daylight illuminants, their corresponding correlated colour temperature in degrees Kelvin, and the coefficients with which the spectra of table 7.1 must be synthesised to produce their relative spectral radiant power distributions, normalised so that they have value 100 at $\lambda = 560nm$.

Example B7.11

Compute and plot the spectral radiant power distribution of the standard illuminants.

These are shown in figure 7.13.

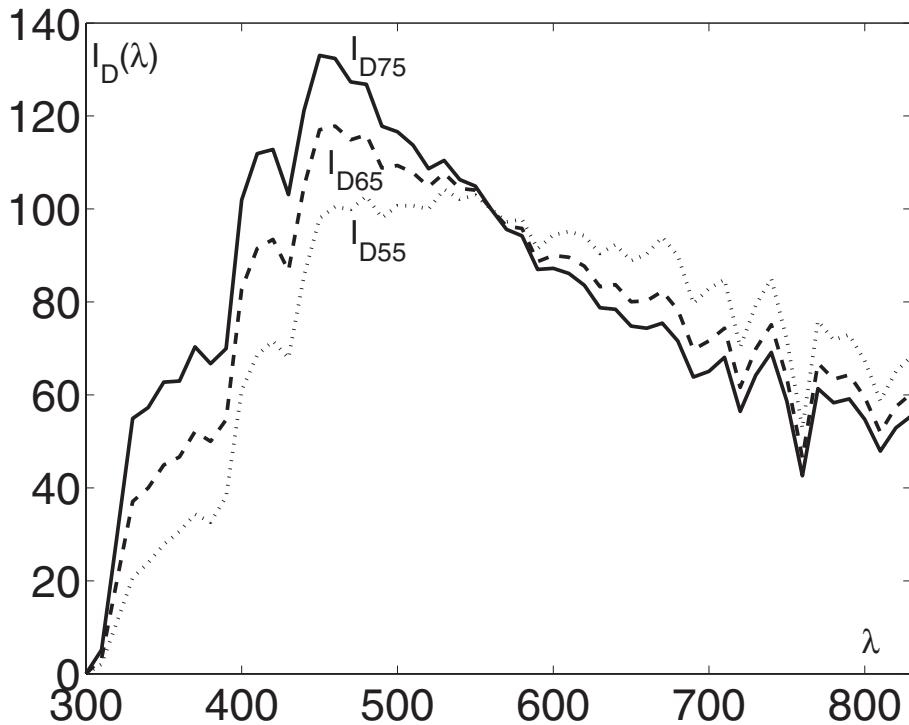


Figure 7.13: The spectral radiant power distribution of illuminants D_{55} (mid-morning daylight), D_{65} (noon daylight) and D_{75} (north sky daylight).

What is the observed variation in the natural materials?

Following the same procedure as for the daylight, scientists recorded the spectra from a large number of natural materials. It turned out, that a large number of the reflectance functions $R(\lambda)$ of the materials can be reproduced by the linear superposition of only six basis functions and in many cases of only three basis functions. In other words, the spectra of the recorded natural objects created an almost 3D manifold in the spectral space, which had a small

thickness along three other directions. If we neglect this small thickness along the three extra directions, we may write

$$R(\lambda) = r_0(\lambda) + u_1 r_1(\lambda) + u_2 r_2(\lambda) + u_3 r_3(\lambda) = \sum_{j=0}^3 u_j r_j(\lambda) \quad (7.79)$$

where $u_0 = 1$, $r_0(\lambda)$ is the mean spectrum, (u_1, u_2, u_3) is the triplet of scalars that characterise almost fully the material of the observed surface, and $r_1(\lambda)$, $r_2(\lambda)$ and $r_3(\lambda)$ are some universal basis functions.

$\lambda(nm)$	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
380	0.055	0.121	0.141	0.051	0.158	0.145	0.053	0.132	0.098	0.095	0.060	0.062
390	0.058	0.148	0.184	0.054	0.209	0.185	0.053	0.171	0.116	0.119	0.061	0.062
400	0.061	0.180	0.254	0.055	0.300	0.250	0.053	0.233	0.131	0.148	0.062	0.063
410	0.062	0.197	0.307	0.056	0.380	0.299	0.053	0.290	0.136	0.172	0.063	0.064
420	0.062	0.201	0.325	0.057	0.412	0.323	0.053	0.329	0.134	0.182	0.064	0.064
430	0.062	0.204	0.331	0.059	0.425	0.340	0.054	0.362	0.132	0.176	0.066	0.065
440	0.061	0.208	0.334	0.060	0.429	0.358	0.054	0.387	0.131	0.160	0.070	0.066
450	0.061	0.216	0.333	0.062	0.429	0.383	0.054	0.399	0.129	0.139	0.075	0.066
460	0.061	0.229	0.327	0.062	0.422	0.420	0.055	0.390	0.126	0.118	0.086	0.067
470	0.061	0.250	0.314	0.063	0.405	0.466	0.056	0.359	0.121	0.100	0.104	0.069
480	0.061	0.277	0.300	0.065	0.381	0.509	0.058	0.313	0.116	0.085	0.137	0.072
490	0.062	0.304	0.287	0.068	0.348	0.545	0.060	0.257	0.111	0.074	0.190	0.077
500	0.065	0.325	0.271	0.076	0.313	0.567	0.067	0.207	0.106	0.065	0.269	0.090
510	0.070	0.330	0.251	0.103	0.282	0.575	0.088	0.167	0.101	0.059	0.377	0.129
520	0.076	0.314	0.231	0.146	0.254	0.571	0.123	0.137	0.096	0.056	0.478	0.208
530	0.079	0.289	0.215	0.177	0.229	0.553	0.151	0.117	0.093	0.053	0.533	0.303
540	0.080	0.277	0.200	0.183	0.214	0.524	0.172	0.105	0.093	0.051	0.551	0.378
550	0.084	0.279	0.184	0.170	0.208	0.488	0.200	0.097	0.094	0.051	0.547	0.426
560	0.090	0.280	0.167	0.149	0.202	0.443	0.250	0.090	0.097	0.052	0.529	0.468
570	0.102	0.294	0.156	0.132	0.195	0.398	0.338	0.086	0.109	0.052	0.505	0.521
580	0.119	0.344	0.150	0.121	0.194	0.349	0.445	0.084	0.157	0.051	0.471	0.575
590	0.134	0.424	0.147	0.114	0.202	0.299	0.536	0.084	0.268	0.053	0.427	0.613
600	0.143	0.489	0.144	0.109	0.216	0.253	0.583	0.084	0.401	0.059	0.381	0.631
610	0.147	0.523	0.141	0.104	0.231	0.223	0.590	0.084	0.501	0.073	0.346	0.637
620	0.151	0.542	0.140	0.103	0.241	0.206	0.586	0.083	0.557	0.095	0.327	0.637
630	0.158	0.558	0.140	0.104	0.253	0.198	0.582	0.084	0.579	0.117	0.317	0.638
640	0.168	0.576	0.141	0.107	0.276	0.192	0.579	0.088	0.587	0.139	0.312	0.641
650	0.179	0.594	0.145	0.109	0.310	0.190	0.579	0.095	0.589	0.162	0.309	0.644
660	0.188	0.611	0.150	0.111	0.345	0.192	0.584	0.105	0.592	0.189	0.314	0.646
670	0.190	0.623	0.151	0.111	0.365	0.200	0.595	0.117	0.595	0.221	0.327	0.648
680	0.188	0.634	0.147	0.111	0.367	0.212	0.611	0.133	0.601	0.256	0.345	0.653
690	0.185	0.651	0.141	0.112	0.363	0.223	0.628	0.155	0.607	0.295	0.362	0.661
700	0.186	0.672	0.134	0.117	0.362	0.231	0.644	0.186	0.614	0.336	0.376	0.671
710	0.192	0.693	0.131	0.123	0.368	0.233	0.653	0.218	0.617	0.370	0.380	0.679
720	0.200	0.710	0.133	0.129	0.377	0.229	0.654	0.255	0.617	0.404	0.378	0.684
730	0.214	0.728	0.144	0.135	0.394	0.229	0.659	0.296	0.618	0.445	0.380	0.689

Table 7.3: The first 12 spectra of the Macbeth colour checker.

Plate Ia shows a chart of 24 basic spectra, which have been selected to represent the variety of spectra observed in natural materials. This chart, called the **Macbeth colour chart**, is used by scientists to calibrate their imaging systems. The spectra of these 24 square patches are given in tables 7.3 and 7.4. We may perform PCA on these 24 spectra, as follows.

$\lambda(nm)$	m13	m14	m15	m16	m17	m18	m19	m20	m21	m22	m23	m24
380	0.064	0.051	0.049	0.056	0.155	0.114	0.199	0.182	0.152	0.109	0.068	0.031
390	0.074	0.053	0.048	0.053	0.202	0.145	0.259	0.240	0.197	0.133	0.076	0.032
400	0.094	0.054	0.047	0.052	0.284	0.192	0.421	0.367	0.272	0.163	0.083	0.032
410	0.138	0.055	0.047	0.052	0.346	0.235	0.660	0.506	0.330	0.181	0.086	0.032
420	0.192	0.057	0.047	0.052	0.362	0.259	0.811	0.566	0.349	0.187	0.088	0.033
430	0.239	0.059	0.047	0.054	0.355	0.284	0.863	0.581	0.356	0.191	0.09	0.033
440	0.280	0.062	0.047	0.056	0.334	0.316	0.877	0.586	0.360	0.194	0.091	0.033
450	0.311	0.066	0.046	0.059	0.305	0.352	0.884	0.587	0.361	0.195	0.091	0.032
460	0.312	0.074	0.045	0.066	0.275	0.390	0.890	0.588	0.361	0.194	0.090	0.032
470	0.281	0.092	0.045	0.081	0.246	0.426	0.894	0.587	0.359	0.193	0.090	0.032
480	0.231	0.123	0.044	0.108	0.217	0.446	0.897	0.585	0.357	0.192	0.089	0.032
490	0.175	0.176	0.044	0.154	0.189	0.444	0.901	0.585	0.356	0.192	0.089	0.032
500	0.126	0.244	0.044	0.228	0.167	0.423	0.905	0.586	0.356	0.192	0.089	0.032
510	0.090	0.306	0.045	0.339	0.148	0.384	0.906	0.586	0.357	0.192	0.089	0.032
520	0.066	0.338	0.045	0.464	0.126	0.335	0.908	0.587	0.358	0.192	0.089	0.032
530	0.052	0.334	0.046	0.557	0.107	0.280	0.907	0.586	0.358	0.192	0.089	0.032
540	0.045	0.316	0.047	0.613	0.099	0.229	0.907	0.586	0.358	0.192	0.089	0.032
550	0.041	0.293	0.049	0.647	0.101	0.183	0.910	0.586	0.358	0.192	0.089	0.032
560	0.039	0.261	0.052	0.670	0.103	0.144	0.910	0.585	0.358	0.192	0.089	0.032
570	0.038	0.228	0.058	0.692	0.109	0.117	0.912	0.587	0.359	0.192	0.089	0.032
580	0.038	0.196	0.071	0.709	0.136	0.100	0.912	0.587	0.359	0.192	0.089	0.031
590	0.038	0.164	0.103	0.722	0.199	0.089	0.912	0.586	0.359	0.191	0.088	0.031
600	0.038	0.134	0.177	0.731	0.290	0.081	0.910	0.584	0.357	0.190	0.088	0.031
610	0.039	0.114	0.313	0.739	0.400	0.076	0.912	0.582	0.356	0.189	0.087	0.032
620	0.040	0.102	0.471	0.747	0.514	0.074	0.915	0.580	0.354	0.188	0.087	0.032
630	0.040	0.096	0.586	0.753	0.611	0.073	0.916	0.577	0.351	0.186	0.086	0.032
640	0.040	0.092	0.651	0.759	0.682	0.073	0.918	0.575	0.348	0.184	0.085	0.032
650	0.041	0.090	0.682	0.764	0.726	0.073	0.920	0.573	0.346	0.182	0.084	0.032
660	0.042	0.090	0.698	0.769	0.754	0.075	0.921	0.572	0.344	0.181	0.084	0.032
670	0.043	0.093	0.707	0.772	0.769	0.076	0.920	0.570	0.341	0.179	0.083	0.032
680	0.043	0.099	0.715	0.777	0.779	0.076	0.921	0.569	0.339	0.178	0.083	0.032
690	0.043	0.104	0.725	0.784	0.789	0.075	0.923	0.568	0.337	0.177	0.082	0.032
700	0.044	0.109	0.734	0.792	0.800	0.072	0.926	0.567	0.334	0.175	0.081	0.032
710	0.047	0.111	0.740	0.798	0.807	0.072	0.928	0.567	0.333	0.174	0.081	0.032
720	0.050	0.110	0.744	0.801	0.813	0.073	0.929	0.565	0.331	0.173	0.081	0.032
730	0.055	0.110	0.748	0.805	0.821	0.079	0.932	0.565	0.330	0.172	0.080	0.032

Table 7.4: The last 12 spectra of the Macbeth colour checker.

Let us call one such spectrum $y_i(j)$, where i identifies the spectrum and j represents the wavelength. So, i takes values from 1 to 24 and j takes values from 1 to 36, as the spectra

are given for 36 sample values of λ . To perform PCA with these 24 spectra, we apply the following algorithm.

- Step 1:** Average the spectra to derive the average spectrum: $m(j)$.
- Step 2:** Remove the average spectrum from each spectrum to produce $z_i(j) \equiv y_i(j) - m(j)$.
- Step 3:** Write these spectra one next to the other, like columns, to form a 36×24 matrix Z .
- Step 4:** Compute ZZ^T and divide it with 24 to produce the covariance matrix C of the data.
- Step 5:** Compute the eigenvalues of C and arrange them in decreasing order.
- Step 6:** Compute the eigenvectors of the first E eigenvalues.
- Step 7:** Write the eigenvectors one under the other as rows to form matrix A of size $E \times 36$.
- Step 8:** Multiply matrix A with matrix Z . The result will be matrix B , of size $E \times 24$.
- Step 9:** Consider the i th column of B : these are the coefficients with which the corresponding eigenvectors have to be multiplied to create an approximation of the i th spectrum. So, to obtain all the reconstructed spectra, multiply matrix A^T with matrix B . The result will be a 36×24 matrix S .
- Step 10:** Add to each column of matrix S the mean spectrum you constructed in Step 1. This will give you matrix T . Each column of this matrix will be an approximated spectrum.

Figure 7.14 is the plot of the obtained first 10 eigenvalues at Step 5. We can see that the values become negligible after the 6th eigenvalue. In fact, the first 12 eigenvalues are:

$$\begin{array}{ccccccc} 1.25518 & 0.43560 & 0.14310 & 0.01703 & 0.00818 & 0.00403 \\ 0.00296 & 0.00066 & 0.00052 & 0.00021 & 0.00010 & 0.00008 \end{array}$$

Table 7.5 lists the eigenvectors constructed at Step 6 that correspond to the first seven eigenvalues. The first six of them and the mean spectrum are the basis spectra appropriate for most natural materials. The first three of them are functions $r_1(\lambda)$, $r_2(\lambda)$ and $r_3(\lambda)$ of equation (7.79). They are plotted in figure 7.15.

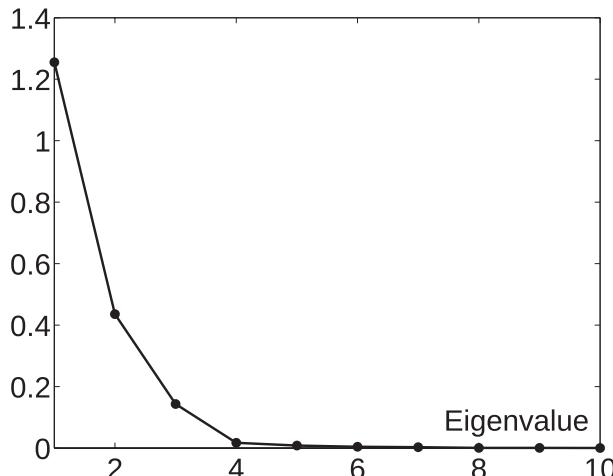


Figure 7.14: The most significant eigenvalues of the reflectance functions given in tables 7.3 and 7.4. It is obvious that three basis functions must be enough to express all reflectance functions adequately.

$\lambda(nm)$	$r_1(\lambda)$	$r_2(\lambda)$	$r_3(\lambda)$	$r_4(\lambda)$	$r_5(\lambda)$	$r_6(\lambda)$	$r_7(\lambda)$
380	0.020506	-0.045631	0.062610	0.006043	-0.048060	0.050640	0.266012
390	0.027144	-0.066618	0.093315	0.009897	-0.074313	0.077337	0.377349
400	0.047397	-0.112613	0.146710	0.031212	-0.125815	0.162142	0.461304
410	0.075593	-0.168199	0.196682	0.077020	-0.169757	0.223400	0.326176
420	0.090415	-0.201078	0.219932	0.113649	-0.184428	0.200833	0.105131
430	0.092957	-0.216809	0.226791	0.135498	-0.168369	0.141992	-0.052165
440	0.091228	-0.227789	0.224879	0.147404	-0.122310	0.070923	-0.170243
450	0.089344	-0.237964	0.216419	0.145758	-0.053601	-0.011351	-0.260007
460	0.089652	-0.246586	0.198617	0.114778	0.030063	-0.090446	-0.272769
470	0.092113	-0.251957	0.169411	0.048899	0.121394	-0.162200	-0.204037
480	0.096595	-0.253135	0.128053	-0.035329	0.197271	-0.208657	-0.080951
490	0.102764	-0.250006	0.072187	-0.133246	0.245160	-0.210257	0.056241
500	0.110416	-0.241373	0.002939	-0.228315	0.244492	-0.172907	0.155331
510	0.119776	-0.224381	-0.082501	-0.295450	0.177094	-0.095415	0.173485
520	0.130614	-0.201407	-0.170474	-0.298266	0.067715	0.000175	0.106118
530	0.140259	-0.177817	-0.232556	-0.243246	-0.037258	0.096363	0.005653
540	0.148489	-0.157773	-0.263867	-0.172285	-0.114689	0.157946	-0.065931
550	0.156040	-0.139805	-0.274118	-0.098272	-0.167139	0.174888	-0.101521
560	0.162992	-0.119845	-0.275821	-0.000402	-0.199050	0.150914	-0.128401
570	0.171146	-0.095495	-0.275670	0.123683	-0.199459	0.061773	-0.126913
580	0.180477	-0.063528	-0.261364	0.250903	-0.144595	-0.082575	-0.046310
590	0.191169	-0.024333	-0.221068	0.343416	-0.027525	-0.212329	0.113841
600	0.200775	0.016684	-0.160472	0.365896	0.103183	-0.226100	0.221752
610	0.209076	0.055401	-0.091462	0.294318	0.198683	-0.111366	0.175590
620	0.215824	0.090016	-0.027793	0.174284	0.248265	0.043674	0.048815
630	0.220383	0.114568	0.019745	0.072717	0.252560	0.161250	-0.045361
640	0.223360	0.129491	0.052857	0.009023	0.226191	0.226345	-0.084185
650	0.224806	0.137030	0.075422	-0.024809	0.183108	0.243736	-0.082530
660	0.225317	0.141354	0.090500	-0.048227	0.131459	0.226676	-0.059165
670	0.225224	0.144604	0.097500	-0.070445	0.074263	0.174581	-0.036580
680	0.225582	0.148451	0.098673	-0.092785	0.017715	0.099392	-0.019465
690	0.226353	0.153486	0.100108	-0.114874	-0.045528	0.004157	-0.007038
700	0.226948	0.158965	0.103640	-0.131776	-0.119655	-0.101870	0.000460
710	0.226636	0.162309	0.110174	-0.137542	-0.184263	-0.193003	0.004018
720	0.225187	0.164183	0.120560	-0.138495	-0.256005	-0.271950	0.000404
730	0.223405	0.164828	0.132616	-0.140173	-0.337889	-0.360059	0.006705

Table 7.5: The basis reflectance functions that correspond to the first 7 eigenvalues of the 24 spectra of tables 7.3 and 7.4.

To calculate the error of the approximation of each spectrum, we compute the difference matrix $P - T$. Then we compute the sum of the squares of each column of this matrix. These are the square errors with which each spectrum is calculated. Tables 7.6 and 7.7 list the errors with which each of the 24 reflectance functions may be approximated, for each number of the retained eigenvalues.

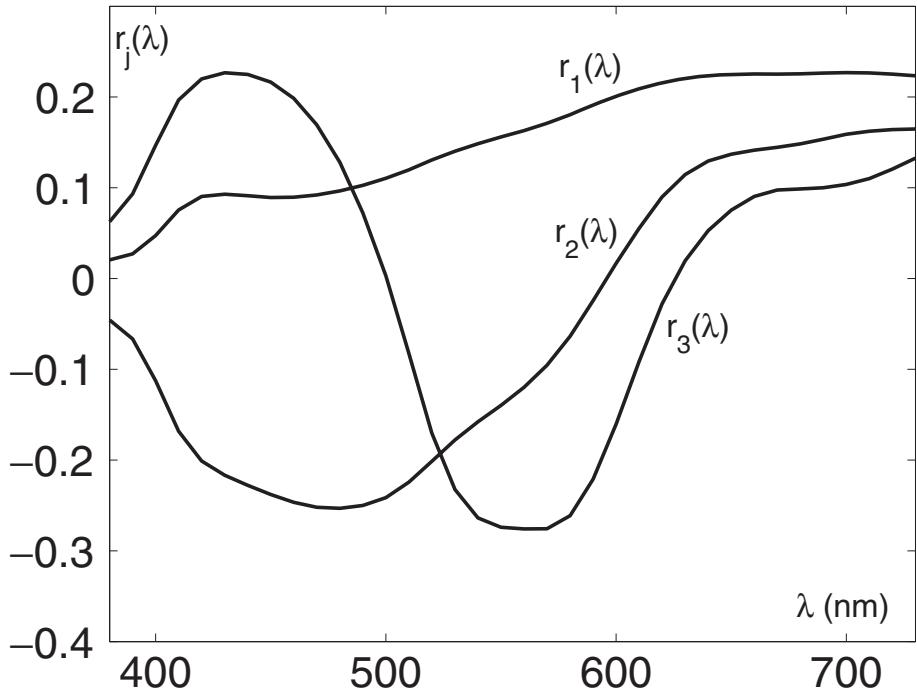


Figure 7.15: The three most significant eigenvectors of the reflectance functions given in tables 7.3 and 7.4. The reflectance functions of most natural materials may be expressed, with a high degree of accuracy, as linear combinations of these three basis functions plus the mean reflectance function.

Finally, tables 7.8–7.11 give the coefficients with which the first three eigenfunctions $r_j(\lambda)$ have to be multiplied to reproduce the reflectance functions of the Macbeth colour checker. The total square errors of these reproductions are given by the rows labelled with $E = 3$ in tables 7.6 and 7.7.

What happens to the light once it reaches the sensors?

Each sensor responds differently to different wavelengths. This is expressed by the sensor sensitivity function $S(\lambda)$. The sensor will record the following value for patch (x, y) :

$$Q = \int_0^{+\infty} S(\lambda) I(\lambda) R(\lambda) d\lambda \quad (7.80)$$

If we substitute $I(\lambda)$ and $R(\lambda)$ from equations (7.72) and (7.79), respectively, we obtain:

$$Q = \int_0^{+\infty} S(\lambda) \sum_{k=1}^3 a_k i_k(\lambda) \sum_{j=0}^3 u_j r_j(\lambda) d\lambda \quad (7.81)$$

E	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
1	0.099	0.212	0.329	0.082	0.380	1.206	0.996	0.416	0.856	0.261	0.666	0.910
2	0.012	0.058	0.041	0.074	0.209	0.119	0.100	0.220	0.137	0.166	0.665	0.239
3	0.004	0.041	0.016	0.005	0.011	0.058	0.075	0.042	0.040	0.086	0.044	0.047
4	0.002	0.033	0.010	0.004	0.010	0.007	0.012	0.036	0.037	0.077	0.012	0.007
5	0.001	0.027	0.010	0.004	0.004	0.006	0.018	0.012	0.010	0.010	0.001	0.004
6	0.001	0.007	0.003	0.002	0.004	0.003	0.003	0.004	0.010	0.002	0.001	0.004
7	0.001	0.001	0.000	0.001	0.003	0.002	0.002	0.003	0.007	0.001	0.001	0.003

Table 7.6: The square errors with which the first 12 spectra of the Macbeth colour checker may be approximated, when the number of retained eigenvalues is E .

E	m13	m14	m15	m16	m17	m18	m19	m20	m21	m22	m23	m24
1	0.194	0.330	1.881	1.084	1.250	0.739	1.690	0.757	0.239	0.044	0.024	0.055
2	0.094	0.288	0.306	0.572	0.746	0.064	0.030	0.018	0.019	0.018	0.022	0.029
3	0.051	0.036	0.098	0.006	0.029	0.052	0.011	0.013	0.019	0.012	0.008	0.009
4	0.026	0.004	0.021	0.005	0.016	0.038	0.010	0.005	0.010	0.006	0.003	0.005
5	0.024	0.003	0.015	0.003	0.012	0.008	0.010	0.005	0.010	0.005	0.002	0.003
6	0.024	0.003	0.013	0.002	0.002	0.001	0.010	0.002	0.005	0.002	0.000	0.002
7	0.001	0.003	0.001	0.001	0.001	0.002	0.000	0.001	0.000	0.000	0.000	0.002

Table 7.7: The square errors with which the last 12 spectra of the Macbeth colour checker may be approximated, when the number of retained eigenvalues is E .

u_j	m1	m2	m3	m4	m5	m6
u_1	-0.958251	0.890052	-0.700982	-1.091990	-0.021951	0.011939
u_2	0.294399	0.393275	-0.536469	0.092743	-0.413129	-1.042620
u_3	-0.088996	0.130014	0.158925	-0.263010	0.445110	-0.247418

Table 7.8: The coefficients with which the first three $r_j(\lambda)$ functions have to be multiplied, in order to reproduce the reflectance functions of samples m1–m6 of the Macbeth colour checker, with square error given in table 7.6, for $E = 3$.

u_j	m7	m8	m9	m10	m11	m12
u_1	0.624948	-0.797700	0.391938	-0.797558	0.170537	0.986600
u_2	0.946811	-0.441647	0.848104	0.307806	-0.029733	0.819337
u_3	-0.157086	0.421878	0.311466	0.283262	-0.788307	-0.437923

Table 7.9: The coefficients with which the first three $r_j(\lambda)$ functions have to be multiplied, in order to reproduce the reflectance functions of samples m7–m12 of the Macbeth colour checker, with square error given in table 7.6, for $E = 3$.

u_j	m13	m14	m15	m16	m17	m18
u_1	-1.287590	-0.914657	0.363834	1.641580	0.879216	-0.852870
u_2	-0.316123	-0.203325	1.255130	0.715840	0.710057	-0.821927
u_3	0.208002	-0.502611	0.456081	-0.752165	0.846522	0.110028

Table 7.10: The coefficients with which the first three $r_j(\lambda)$ functions have to be multiplied, in order to reproduce the reflectance functions of samples m13–m18 of the Macbeth colour checker, with square error given in table 7.7, for $E = 3$.

u_j	m19	m20	m21	m22	m23	m24
u_1	3.255370	1.452050	0.208123	-0.688102	-1.233390	-1.531160
u_2	-1.288540	-0.859597	-0.468760	-0.161850	0.040420	0.159808
u_3	0.137171	0.073706	-0.008776	-0.075409	-0.117581	-0.142883

Table 7.11: The coefficients with which the first three $r_j(\lambda)$ functions have to be multiplied, in order to reproduce the reflectance functions of samples m19–m24 of the Macbeth colour checker, with square error given in table 7.7, for $E = 3$.

We may then exchange the order of summation and integration and also apply the integral only to the factors that depend on λ :

$$Q = \sum_{k=1}^3 \sum_{j=0}^3 a_k u_j \underbrace{\int_0^{+\infty} S(\lambda) i_k(\lambda) r_j(\lambda) d\lambda}_{\equiv S_{kj}} = \sum_{k=1}^3 \sum_{j=0}^3 S_{kj} a_k u_j \quad (7.82)$$

We may assume familiarity with the illumination, ie innate knowledge of a_k , as well as innate knowledge of functions S_{kj} . This implies that we have only three unknowns in (7.82), namely parameters u_1 , u_2 and u_3 , which characterise the viewed surface. Then, if we have three sensors with different sensitivity functions $S(\lambda)$, we have three equations for the three unknowns, and we must be able to solve for them to help recognise the surface. This innate knowledge of functions a_k and S_{kj} , which helps us recognise colours during dawn or dusk, and in summer or in winter equally well, is one of the properties of the human visual system and it is called **colour constancy**.

Is it possible for different materials to produce the same recording by a sensor?

Yes, because a sensor, as shown by equation (7.80), integrates the light it receives. Different combinations of photon energies may result in the same recorded values. Under the same illumination, materials characterised by different reflectance functions may produce identical recordings by a sensor. The different reflectance functions, that produce identical sensor recordings, are known as **metamers** of the sensor.

Example 7.12

Using the approximated spectra of the colour chart of Plate I, construct images which show how these spectra will appear under average daylight, viewed by a camera that has three types of sensor, with spectral sensitivity curves given by

$$S(\lambda) = e^{-\frac{(\lambda-m)^2}{2s^2}} \quad (7.83)$$

with $m = 650$ and $s = 25$, for the R sensor type, $m = 550$ and $s = 25$, for the G sensor type and $m = 450$ and $s = 25$, for the B sensor type.

First, we discretise the spectral sensitivity curves of the three types of sensor of the camera, as follows. For each set of values m and s , we allow λ to take values from

380 to 730 in steps of 10. We call the 36×1 vectors we create this way S_R , S_G and S_B . We show them plotted in figure 7.16.

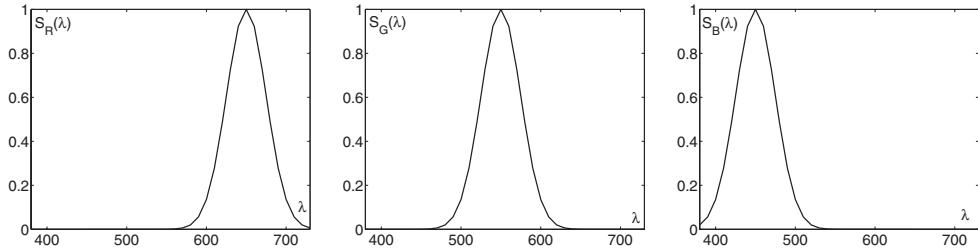


Figure 7.16: The sensitivity curves of the three sensors.

Next, we consider function $i_1(\lambda)$, of table 7.1. This is the average spectrum of daylight. Note that $i_1(\lambda)$ is defined for a broader range of wavelengths than we consider here. We truncate it and use only the values for λ from 380 to 730. We call this vector $I(\lambda)$. Now, let us consider the spectra of the materials. We multiply each spectrum point by point with vector $S_R(\lambda)$ and with vector $I(\lambda)$, and add the 36 results. This is value Q_R for this material. Also, we multiply each spectrum point by point with vector $S_G(\lambda)$ and with vector $I(\lambda)$, and add the 36 results to produce value Q_G . Finally, we multiply each spectrum point by point with vector $S_B(\lambda)$, and with vector $I(\lambda)$, and add the 36 results to produce value Q_B .

For each spectrum, we shall have a triplet (Q_R, Q_G, Q_B) , which represents the (red, green, blue) values of the spectrum. We normalise them to be integers from 0 to 255 and for each spectrum we create a 32×32 image of three bands, the red, green and blue band. All pixels in the red band will have normalised value Q_R , call it R , all pixels in the green band will have normalised value Q_G , call it G , and all pixels in the blue band will have normalised value Q_B , call it B .

In Plate II we show the images we produced by approximating the original spectra with 1, 2,...,6 eigenvalues.

How does the human visual system achieve colour constancy?

We have grown to know scalars a_k and we may speculate that evolution made sure that scalars S_{kj} of equation (7.82) are hard-wired in our brains. After all, functions $i_k(\lambda)$ and $r_j(\lambda)$ are universal and function $S(\lambda)$ is intrinsic to each person.

In addition, our eyes do have three types of sensor² with distinct sensitivities. They are:

- the **L cones** with sensitivity in the range of long optical wavelengths (responsible for the sensation we call “red”);

²We also have a fourth type of sensor, called **rods**, but these sensors do not participate in the perception of colour.

- the **M cones** with sensitivity in the range of middle optical wavelengths (responsible for the sensation we call “green”) and
- the **S cones** with sensitivity in the range of short optical wavelengths (responsible for the sensation we call “blue”).

This is the physiological basis for the **trichromatic theory of colour vision**.

What does the trichromatic theory of colour vision say?

According to this theory, any colour sensation can be created by the blending of three primary spectra. In other words, we may construct three different light sources, each one with a specific spectrum. We may then project the three beams of light on the same spot. By changing the relative intensities of the three lights, we may make a person see any colour we want, ie create in their brain the sensation of any colour we want. The spectra of such three light sources constitute a **colour system**. Because displaying devices have to be controlled and used to create in the brain of the viewers prespecified colour sensations, research in this area was done in relation to the development of television monitors. Different colour systems were proposed by different scientists and for different purposes.

What defines a colour system?

A colour system is defined in terms of the spectral densities of three primary lights: $P_1(\lambda)$, $P_2(\lambda)$ and $P_3(\lambda)$.

In addition, the proposer of the colour system has to supply a graph that shows the so called **tristimulus values**, ie the intensities with which these lights have to be projected on a white screen, so the place where the three beams overlap radiates equal energy at all wavelengths (**ideal white**, also known as **equal energy spectrum on the wavelength basis**). Figure 7.17 shows schematically the graphs that fully define a colour system, as well as the spectrum of the ideal white, which has energy 1 at all wavelengths.

How are the tristimulus values specified?

They are specified by conducting **colour matching experiments**: A white screen is divided into two halves. On the one half a monochromatic (=single wavelength) light of unit radiant power is projected. Let us refer to it as the monochromatic reference stimulus. On the other half, the beams from the three primary lights are projected simultaneously. A human observer adjusts the intensities of the three lights until what she sees on the right half of the screen is not distinguishable from what she sees on the left half of the screen. The three intensities, required to perform the matching, are the three tristimulus values for the particular wavelength of the monochromatic reference stimulus.

Can all monochromatic reference stimuli be matched by simply adjusting the intensities of the primary lights?

No. In some cases, one of the lights has to be projected with the monochromatic reference stimulus, rather than with the other primary lights, for the observer to be able to match what she sees in the two halves of the screen. The light that has to be projected with the reference stimulus is said to have a *negative* tristimulus value at that particular wavelength.

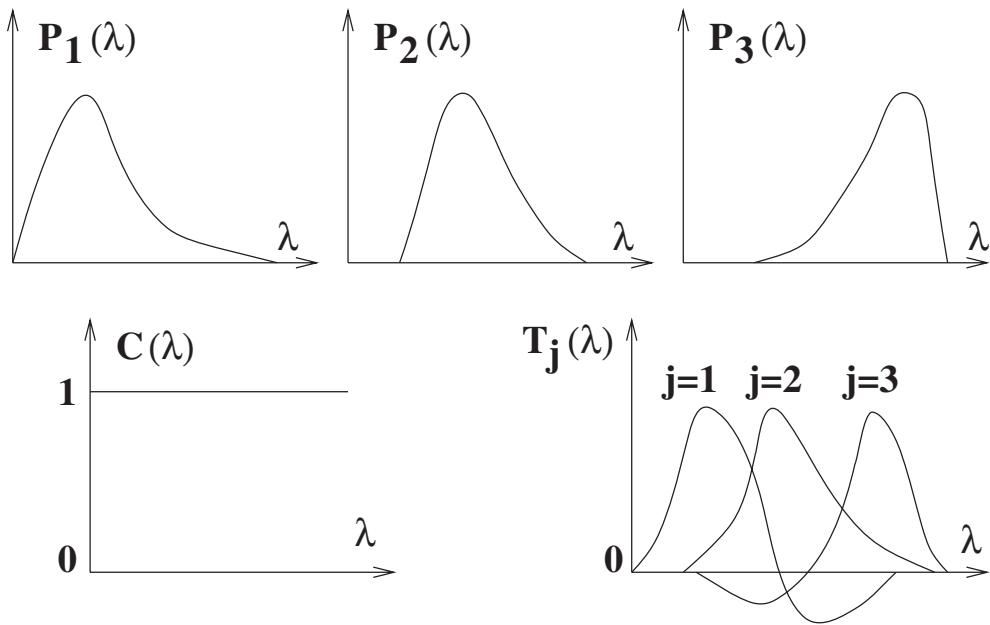


Figure 7.17: Graphs that fully define a colour system. The graph at the bottom left is the spectrum of the ideal white.

Example 7.13

Use the information supplied by the graphs in figure 7.17 to decide what the tristimulus values are, for a spectral distribution with intensity 1 at wavelength ψ and 0 everywhere else.

In figure 7.18 we show on the left the spectral distribution we wish to create and on the right we reproduce the graph at the bottom right of figure 7.17. In this graph we draw a vertical line at wavelength ψ and we read from the vertical axis the coordinates of the points where it cuts the three curves: $T_1(\psi)$, $T_2(\psi)$ and $T_3(\psi)$. In theory, these are the intensities with which the three primary lights should be blended in order to form a spectrum that a human may find identical with the spectrum that has 0 energy at all wavelengths, except at wavelength $\lambda = \psi$, at which it has energy 1. However, since $T_3(\psi)$ is negative, such a colour matching cannot happen in practice. It is obvious, that when the tristimulus graph was being created, the third light had to be projected with intensity $T_3(\psi)$ on the same spot as the monochromatic spectrum $C(\psi)$, in order to create the same colour sensation as the one created by the first and second lights projected together, with intensities $T_1(\psi)$ and $T_2(\psi)$, respectively.

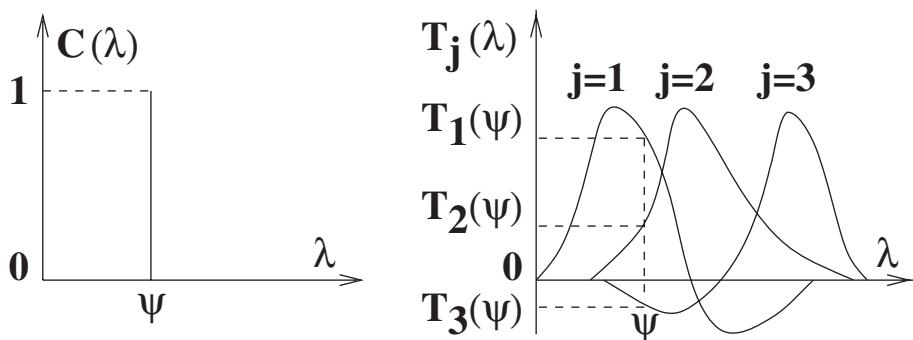


Figure 7.18: To create the same sensation as that of a monochromatic stimulus at wavelength ψ , we have to project together the three primary lights with intensities $T_1(\psi)$, $T_2(\psi)$ and $T_3(\psi)$, read from the plot on the right.

Do all people require the same intensities of the primary lights to match the same monochromatic reference stimulus?

No. Usually, colour matching experiments are repeated several times with different people with normal colour vision. Then average values are reported.

Who are the people with normal colour vision?

They are people who have in their retinas all three types of cone sensor. They are called **trichromats**. People who lack one or two types of cone are called **dichromats** or **monochromats**, respectively. If we also exclude trichromats who deviate significantly in colour perception from the majority of trichromats, we are left with about 95% of the population that may be considered as having normal colour vision and, thus, as being representative enough to be used in colour matching experiments.

What are the most commonly used colour systems?

They are the *CIE RGB*, the *XYZ* and the recently proposed *sRGB*, which has become the standard for digital image processing.

What is the *CIE RGB* colour system?

The primary lights of the *CIE RGB* colour system are monochromatic with wavelengths 700.0nm, 546.1nm and 435.8nm. However, in order to define them fully, we have to define also the unit intensity for each one, so that when we say the viewer needed so much intensity from each to match a particular spectrum, we actually mean so many units from each. To

emulate viewing the ideal white, all wavelengths have to be used simultaneously. As we want the ideal white to have constant energy at all wavelengths, we normalise the tristimulus curves so that the values of each one of them sum up to 1. For this to be so, the intensities (formally called **radiant power**) of the three primary lights of the *CIE RGB* colour system have to be in ratios 72.0962 : 1.3791 : 1.

Figure 7.19 shows the colour matching functions of the *CIE RGB* system, and table 7.12 lists their values. Note the presence of negative tristimulus values, which imply that the particular light has to be added to the reference monochromatic stimulus to match the blending of the other two lights. These tristimulus values have been worked out by averaging the colour matching functions of many human subjects with normal colour vision. So, they constitute the colour matching functions of the so called **standard observer**.

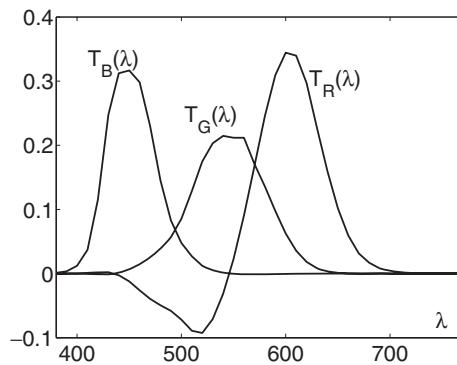


Figure 7.19: The colour matching functions (tristimulus values) of the *CIE RGB* colour system.

What is the *XYZ* colour system?

The *XYZ* colour system is a transformed version of the *CIE RGB* colour system, so the tristimulus values in this system are non-negative numbers. To understand how it is defined, we have to consider first the representation of colours in 3D and in 2D spaces.

How do we represent colours in 3D?

As every colour can be represented by its tristimulus values, we may represent the colours in a 3D space. Along each axis, we measure the intensity with which the corresponding primary light has to be projected to create the colour in question. Colours further away from the origin of the axes (in the positive octant) appear brighter.

How do we represent colours in 2D?

Since brightness increases along the main diagonal of the colour space, we may then separate the brightness component of the light from the pure colour component, ie represent a colour with intensity-invariant tristimulus values. This means that two values only must be enough to represent a colour. The representation of a colour by two normalised tristimulus values can be plotted using two axes. Such a plot is the so called **chromaticity diagram**.

$\lambda(nm)$	$T_R(\lambda)$	$T_G(\lambda)$	$T_B(\lambda)$	$\lambda(nm)$	$T_R(\lambda)$	$T_G(\lambda)$	$T_B(\lambda)$
380	0.00003	-0.00001	0.00117	580	0.24526	0.13610	-0.00108
390	0.00010	-0.00004	0.00359	590	0.30928	0.09754	-0.00079
400	0.00030	-0.00014	0.01214	600	0.34429	0.06246	-0.00049
410	0.00084	-0.00041	0.03707	610	0.33971	0.03557	-0.00030
420	0.00211	-0.00110	0.11541	620	0.29708	0.01828	-0.00015
430	0.00218	-0.00119	0.24769	630	0.22677	0.00833	-0.00008
440	-0.00261	0.00149	0.31228	640	0.15968	0.00334	-0.00003
450	-0.01213	0.00678	0.31670	650	0.10167	0.00116	-0.00001
460	-0.02608	0.01485	0.29821	660	0.05932	0.00037	0.00000
470	-0.03933	0.02538	0.22991	670	0.03149	0.00011	0.00000
480	-0.04939	0.03914	0.14494	680	0.01687	0.00003	0.00000
490	-0.05814	0.05689	0.08257	690	0.00819	0.00000	0.00000
500	-0.07173	0.08536	0.04776	700	0.00410	0.00000	0.00000
510	-0.08901	0.12860	0.02698	710	0.00210	0.00000	0.00000
520	-0.09264	0.17468	0.01221	720	0.00105	0.00000	0.00000
530	-0.07101	0.20317	0.00549	730	0.00052	0.00000	0.00000
540	-0.03152	0.21466	0.00146	740	0.00025	0.00000	0.00000
550	0.02279	0.21178	-0.00058	750	0.00012	0.00000	0.00000
560	0.09060	0.21178	-0.00058	760	0.00006	0.00000	0.00000
570	0.16768	0.17087	-0.00135	770	0.00003	0.00000	0.00000

Table 7.12: The colour matching functions (tristimulus values) of the *CIE RGB* colour system.

What is the chromaticity diagram?

It is a 2D space where we represent colours.

Figure 7.20 shows the colour space that corresponds to some colour system. Tristimulus values (T_1, T_2, T_3), that characterise a particular colour in this colour system, represent a single point in this 3D space. Let us define the normalised tristimulus values as:

$$t_j \equiv \frac{T_j}{T_1 + T_2 + T_3} \quad \text{for } j = 1, 2, 3 \quad (7.84)$$

The normalised tristimulus values (t_1, t_2, t_3), obviously, satisfy the equation:

$$t_1 + t_2 + t_3 = 1 \quad (7.85)$$

This equation represents a plane in the 3D space that cuts the three axes at point 1 along each axis, as shown in figure 7.20. So, the normalisation we perform in (7.84) effectively collapses all points in the 3D space onto that plane. Figure 7.21 shows the part of the plane that is in the octant of the 3D space defined by the three positive semi-axes. It is an equilateral triangle, called the **Maxwell colour triangle**. Any point on this plane can be defined by two numbers. All we have to do is to choose a coordinate system on it. Figure 7.21 shows one such possible coordinate system.

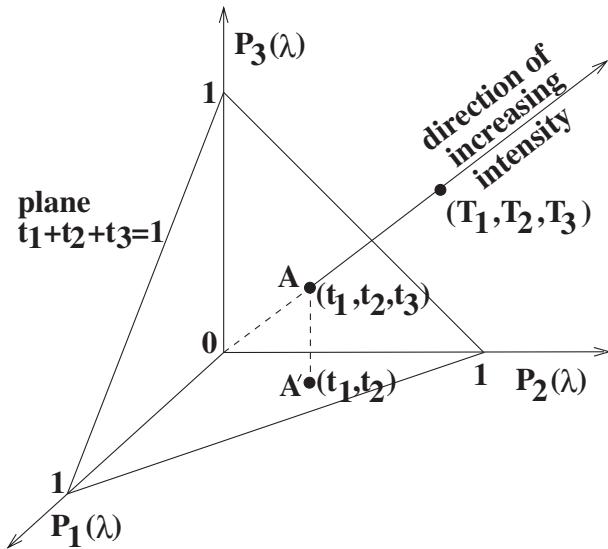


Figure 7.20: A colour space. Increasing the overall intensity ($T_1 + T_2 + T_3$) simply slides the points along lines emanating from the origin of the axes. So, each such line represents a single colour experienced with a variety of intensities. The coordinates of the intersection of the ray with the oblique plane shown here are used to represent the colour.

Because of the oblique position of this plane with respect to the original axes, it is not trivial to define the 2D coordinates of a point on it, in terms of the coordinates of the point in the original 3D space (see example 7.14). On the other hand, we observe that any point A on the Maxwell triangle, in figure 7.20, corresponds to a unique point A' in the right-angle triangle formed by axes OP_1 and OP_2 . So, it is not necessary to use the points on the Maxwell triangle to represent colours. We might as well use the points of the right-angle triangle at the bottom. This plane has a natural coordinate system, namely the two axes OP_1 and OP_2 . Any point with coordinates (T_1, T_2, T_3) in the 3D space corresponds uniquely to a point with coordinates (t_1, t_2, t_3) , which in turn corresponds uniquely to a point in the bottom triangle with coordinates (t_1, t_2) . It is this bottom right-angle triangle with the two axes OP_1 and OP_2 , along which we measure t_1 and t_2 , respectively, that is called **chromaticity diagram**.

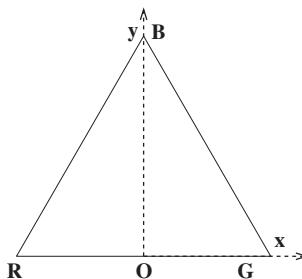


Figure 7.21: All points that represent colours in the 3D colour space of figure 7.20 may be projected radially on this triangle, that passes through the unit points of the three axes. One may define a 2D coordinate system on this plane, to define uniquely all colours (points). This is the **Maxwell colour triangle**.

Box 7.4. Some useful theorems from 3D geometry

In order to reason in a 3D colour space, we may find useful some theorems from 3D geometry.

Theorem 1: Three points in space define a plane (see figure 7.22a).

Theorem 2: Two intersecting lines in space define a plane (see figure 7.22b).

Theorem 3: If a point P belongs to plane Π and a line l_1 lies on Π , then a line l_2 passing through P and parallel to l_1 also lies in plane Π (see figure 7.22c).

Theorem 4: If a line l is orthogonal to two non-parallel lines α and β , that lie in a plane Π , then l is perpendicular to plane Π (see figure 7.23a).

Theorem 5: If a line l is perpendicular to a plane Π , it is orthogonal to all lines that lie in the plane (see figure 7.23b).

Theorem 6: Consider a line l that belongs to a plane Π and a point P that does not belong to plane Π . Consider the line that passes through P and is perpendicular to plane Π . Consider point P' that is the intersection of this line with the plane. Consider also the perpendicular from P to line l , meeting line l at point P'' . Then line $P'P''$ is perpendicular to line l . This is known as the **theorem of the three perpendiculars** (see figure 7.23c).

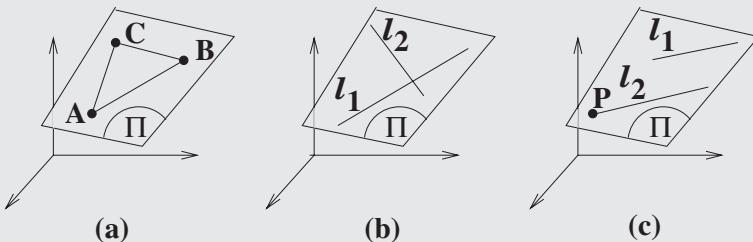


Figure 7.22: (a) Three points A , B and C define a plane Π . (b) Two intersecting lines l_1 and l_2 define a plane Π . (c) If a point P and a line l_1 lie in a plane Π , line l_2 , that passes through P and is parallel to l_1 , also lies in plane Π .

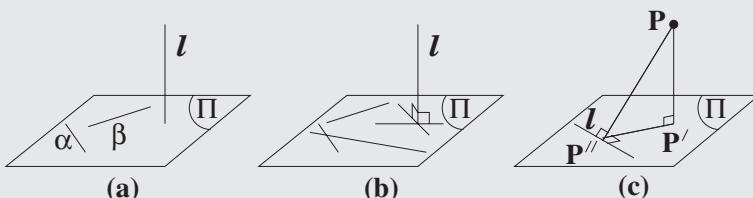


Figure 7.23: (a) If a line l is orthogonal to two non-parallel lines α and β that lie on a plane Π , then line l is perpendicular to plane Π . (b) If a line l is perpendicular to plane Π , then it is orthogonal to all lines that lie in the plane. (c) P' is the projection of point P on plane Π and line PP'' is perpendicular to line l that lies in the plane. Then line $P'P''$ is perpendicular to line l (theorem of the three perpendiculars).

Example B7.14

If a colour is represented by a point with coordinates (R, G, B) in the 3D colour space, work out its coordinates (x, y) in the coordinate system defined in figure 7.21 of the Maxwell colour triangle.

For convenience, we draw first the 3D colour space and the Maxwell colour triangle that corresponds to it. A point with coordinates (R, G, B) in this space will be projected to a point P on triangle AMC , with coordinates (r, g, b) , where

$$r \equiv \frac{R}{R + G + B} \quad g \equiv \frac{G}{R + G + B} \quad b \equiv \frac{B}{R + G + B} \quad (7.86)$$

as shown in figure 7.24.

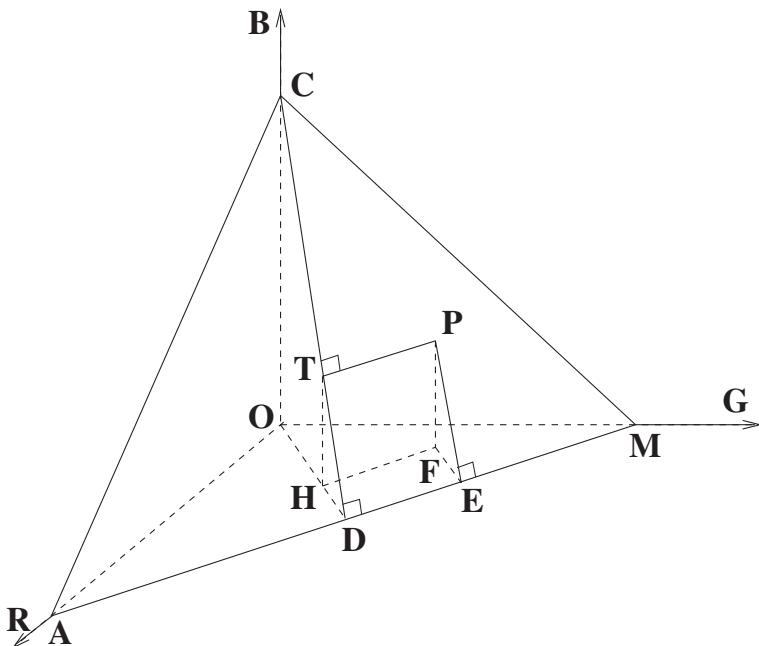


Figure 7.24: A colour space and its corresponding Maxwell colour triangle.

In this figure, point T is the projection of point P on the height CD of the equilateral triangle AMC . Point E is the projection of point P on line AM . To define the coordinates of point P in the coordinate system defined in figure 7.21, we must express lengths PT and PE in terms of coordinates (r, g, b) .

We observe that $PT = ED$ and $PE = TD$, by construction. We aim to compute, therefore, lengths ED and TD . To do that, we project points T and P to points H

and F on plane OAM , respectively. We need to prove three things:

- (i) Point H lies on line OD .
- (ii) $TPFH$ is a parallelogram, so we can infer that $TH = PF$.
- (iii) $FHDE$ is a parallelogram, so we can infer that $DE = HF$.

Proof of (i): First, we observe that, because of Theorem 6 of Box 7.4, $OD \perp AM$. Line TH is parallel to line OC and, therefore, lies inside plane ODC , according to Theorem 3 of Box 7.4. This means that point H lies on line OD .

Proof of (ii): PF is parallel to TH , and by Theorem 3 of Box 7.4, F lies in plane HTP . For $TPFH$ to be a parallelogram, we must also show that TP is parallel to HF . TP is parallel to DE by construction, and as $DE \perp OD$, $TP \perp OD$ too. Also, $TP \perp CD$ by construction. Then, according to Theorem 4 of Box 7.4, TP is perpendicular to plane OCD . We must show next that FH is also perpendicular to plane OCD , in order to show that FH is parallel to TP and complete the proof. TP is orthogonal to any line of plane OCD , which means that $OD \perp TP$. Also, TH was drawn to be orthogonal to plane OAM , which means it is orthogonal to all lines of this plane (Theorem 5 of Box 7.4) and so, $OD \perp TH$. So, OD is orthogonal to two lines of plane $TPFH$, therefore orthogonal to the plane, and all the lines that lie on this plane. Therefore, $FH \perp OD$. At the same time, as OC is perpendicular to plane OAM , it is orthogonal to all lines of this plane, and so $HF \perp OC$. This means that HF is orthogonal to two intersecting lines of plane ODC , therefore perpendicular to this plane, and, thus, parallel to TP . So, $TPFH$ has its sides parallel pairwise, therefore it is a parallelogram.

Proof of (iii): DE is perpendicular to plane ODC , and, thus, parallel to HF . HD and FE , according to Theorem 6 of Box 7.4, are perpendicular to line AM and, thus, parallel to each other. Therefore, $FHDE$ is a parallelogram.

As TH is parallel to CO , triangles DCO and DTH are similar. We may, therefore, write:

$$\frac{DC}{DT} = \frac{DO}{DH} = \frac{CO}{TH} \quad (7.87)$$

In the above expression we know most of the lengths involved.

- From right angle triangle DMC : we know that $MC = \sqrt{2}$ and $DM = \sqrt{2}/2$. Then $CD = \sqrt{MC^2 - DM^2} = \sqrt{3}/2$.
- From right angle isosceles triangle AOM , which has $OA = OM = 1$, we deduce that its height $DO = \sqrt{2}/2$.
- By definition, $CO = 1$
- Since $TH = PF$, $TH = b$, the normalised coordinate of point P along the **B** axis.

From (7.87), we can then work out:

$$DT = \frac{TH}{CO} DC = \sqrt{\frac{3}{2}}b \Rightarrow PE = \sqrt{\frac{3}{2}}b \quad (7.88)$$

Also from (7.87), we deduce that:

$$DH = \frac{TH}{CO} DO = \frac{\sqrt{2}}{2}b \quad (7.89)$$

Then:

$$OH = OD - HD = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}b = \frac{\sqrt{2}}{2}(1 - b) \quad (7.90)$$

By construction, we know that point F in plane OAM has coordinates (r, g) . Then, $OF = \sqrt{r^2 + g^2}$. From the right angle triangle OHF then, we work out the other coordinate we need for point P :

$$HF = \sqrt{OF^2 - OH^2} = \sqrt{r^2 + g^2 - \frac{1}{2}(1 - b)^2} \quad (7.91)$$

Remember, however, that $r + g + b = 1$. So, we may write $1 - b = r + g$. So:

$$\begin{aligned} HF &= \sqrt{\frac{2r^2 + 2g^2 - (r + g)^2}{2}} \\ &= \sqrt{\frac{2r^2 + 2g^2 - r^2 - g^2 - 2rg}{2}} \\ &= \sqrt{\frac{r^2 + g^2 - 2rg}{2}} = \frac{|r - g|}{\sqrt{2}} \end{aligned} \quad (7.92)$$

In summary, the coordinates of point P on the Maxwell colour triangle, in terms of its (r, g, b) coordinates, are

$$\left(\frac{g - r}{\sqrt{2}}, \sqrt{\frac{3}{2}}b \right) \quad (7.93)$$

where we have allowed positive and negative values along axis DM (axis x in figure 7.21).

What is the chromaticity diagram of the CIE RGB colour system?

We may first identify in the chromaticity diagram the points that represent monochromatic colours (ie colours of a single wavelength). To do that, we consider all triplets of tristimulus values (T_R, T_G, T_B) of table 7.12, work out from them the corresponding (t_R, t_G) values, and plot them in the chromaticity diagram. The result is shown in figure 7.25.

We can make the following observations from this figure. The line that is parametrised by the wavelength forms an arc in this chromaticity space. This arc is called **spectrum locus**.

The straight line that connects the points that correspond to the two extreme wavelengths (400nm and 700nm) is called the **purple line**. At the resolution of figure 7.25, the purple line appears to coincide with the t_R axis, but this is not the case. The arc, made up from the tristimulus values of the monochromatic stimuli, is outside the right-angle triangle defined by the acceptable range (ie $0 \leq t_B, t_G \leq 1$) of tristimulus values. This is due to the negative tristimulus values present in table 7.12 and it implies that the selected primary lights are not good enough to allow us to create any colour sensation by simply blending them. That is the reason the XYZ colour system was introduced: it was in response to the desire to have only positive tristimulus values, ie to have all points of the spectrum locus inside the shaded right-angle triangle shown in figure 7.25. In addition, we wish one of the axes of this new colour system to reflect the perceived brightness of the colour.

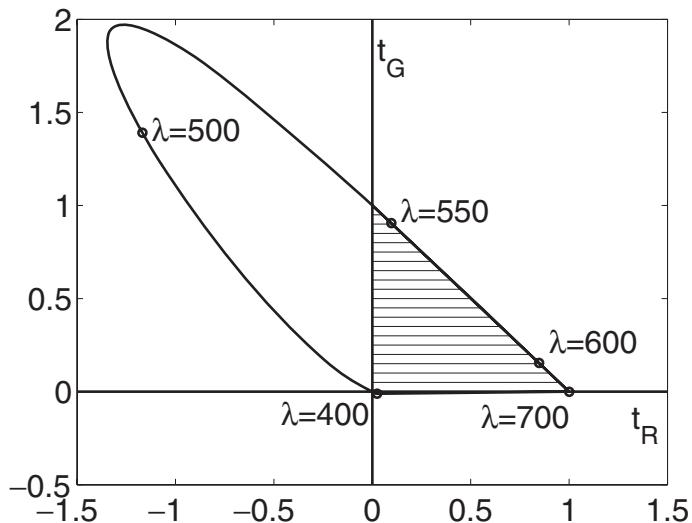


Figure 7.25: The chromaticity diagram that corresponds to the *CIE RGB* colour system. Only colours with (t_R, t_G) values inside the shaded triangle are physically realisable. The curve plotted is parameterised by the wavelength λ . For each value of λ we consider the tristimulus values triplet (T_R, T_G, T_B) , from which we deduce the coordinates (t_R, t_G) , of the point that is plotted here to form the curve.

How does the human brain perceive colour brightness?

We mentioned earlier that perceived brightness increases as we move away from the origin of the *CIE RGB* axes in the 3D colour space, and along the first octant of the space (the positive branches of the colour axes). However, it is wrong to consider that overall brightness is measured along the main diagonal of the colour space. We require different intensities of different colours in order to sense equal brightness. As a result, the axis, along which the perceived overall brightness changes, is different from the main diagonal of the *CIE RGB* space. So, the overall perceived brightness is not measured along the diagonal passing through the ideal white point (the point with coordinates $(1/3, 1/3, 1/3)$). The direction, along which

the perceived brightness is measured, is orthogonal to a plane passing through the origin of the axes in the 3D colour space, called the **alychne**. So, if we want a system that removes the component of brightness from the tristimulus values, and thus represents colours by two numbers only, instead of collapsing all triplets of values on the Maxwell triangle of figure 7.20, we must define another plane, orthogonal to the direction of increased perceived brightness, the alychne plane.

How is the alychne defined in the CIE RGB colour system?

It is the plane with equation:

$$T_R + 4.5907T_G + 0.0601T_B = 0 \quad (7.94)$$

How is the XYZ colour system defined?

It is defined as a linear transformation of the CIE RGB colour system, so that:

- (i) the T_X , T_Y and T_Z tristimulus values are all positive;
- (ii) the equal energy ideal white point, with coordinates $(1/3, 1/3, 1/3)$ in the chromaticity diagram, remains unchanged;
- (iii) the Y component corresponds to the axis of the overall perceived brightness;
- (iv) the points enclosed by the spectrum locus and the purple line fill as much as possible the right-angle triangle, defined by points $(0, 0)$, $(1, 0)$ and $(0, 1)$ in the chromaticity plane;
- (v) the chromaticity coordinate $t_B(\lambda)$, for λ between 650nm and 700nm , is treated as 0. This implies that the spectrum locus for these wavelengths is assumed to coincide with the line that passes through points $(1, 0)$ and $(0, 1)$, ie the line with equation $t_R + t_G = 1$.

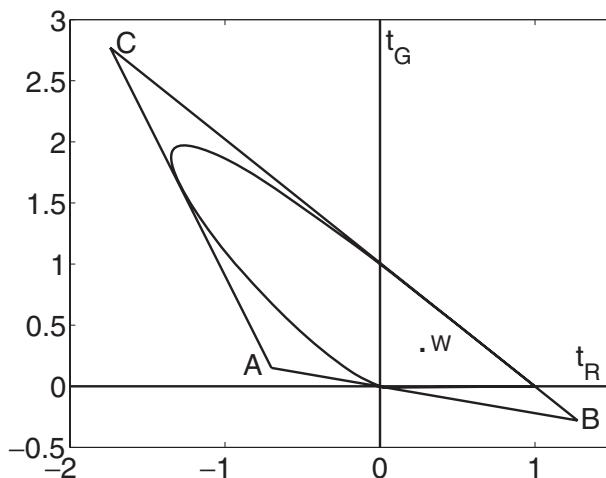


Figure 7.26: The chromaticity diagram of the CIE RGB colour space. Line AB is the intersection of the alychne plane with plane $T_B = 0$. Line AC is drawn tangent to the spectrum locus, near the green (medium) wavelengths. Line BC has been drawn tangent to the spectrum locus, near the red (long) wavelengths. W is the ideal white.

Figure 7.26 shows the chromaticity diagram of the *CIE RGB* space. The line joining points A and B is the line where the alychne plane intersects the (T_R, T_G) plane. The line joining points B and C was drawn to fulfil condition (v), ie it is tangent to the spectrum locus at the point that corresponds to $\lambda = 700\text{nm}$. Point B , being the intersection of these two lines, has coordinates $(t_R, t_G) = (1.275, -0.278)$, and, by inference, it represents colour $(T_R, T_G, T_B) = (1.275, -0.278, 0.003)$. The line passing through points A and C was drawn so that condition (iv) was fulfilled, ie it was chosen to be tangent to the locus spectrum at about the medium wavelengths. Point C turns out to have coordinates $(t_R, t_G) = (-1.740, 2.768)$, and by inference it represents colour $(T_R, T_G, T_B) = (-1.740, 2.768, -0.028)$. Finally, point W represents colour $(T_R, T_G, T_B) = (1/3, 1/3, 1/3)$. The transformation between the two colour systems is assumed linear, ie it has the form:

$$\begin{pmatrix} T_X \\ T_Y \\ T_Z \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} T_R \\ T_G \\ T_B \end{pmatrix} \quad (7.95)$$

To fulfil condition (i) the elements of the transformation matrix should be chosen so that point B is mapped to point $(1, 0, 0)$ and point C to point $(0, 1, 0)$. Further, to fulfil condition (ii), point W should be mapped to point $(1/3, 1/3, 1/3)$. This gives us a system of 9 equations for the 9 unknowns a_{ij} . This way, one may specify the transformation between the two colour systems.

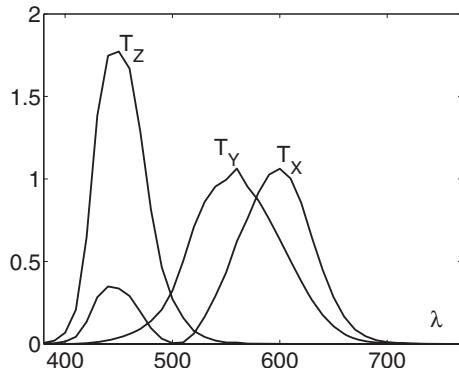


Figure 7.27: The tristimulus values of the *XYZ* colour system.

If a colour is represented in the *CIE RGB* colour space by (T_R, T_G, T_B) , its representation in the *XYZ* space is given by values (T_X, T_Y, T_Z) , which can be calculated from (T_R, T_G, T_B) using the following formulae:

$$\begin{aligned} T_X &= 2.769T_R + 1.752T_G + 1.130T_B \\ T_Y &= T_R + 4.591T_G + 0.060T_B \\ T_Z &= 0.000T_R + 0.057T_G + 5.594T_B \end{aligned} \quad (7.96)$$

Figure 7.27 shows the plots of $T_X(\lambda)$, $T_Y(\lambda)$ and $T_Z(\lambda)$, computed by using equations (7.96) in conjunction with the entries of table 7.12 for the (T_R, T_G, T_B) values³. Note that none of the values is now negative.

³In practical applications, we use a simpler notation: we forget the term “tristimulus values” and we simply refer to the values that define a colour as its (R, G, B) or its (X, Y, Z) values.

Example B7.15

Work out the transformation between the chromaticity coordinates of the XYZ system in terms of the chromaticity coordinates of the *CIE RGB* system.

Using (7.96), we may work out t_X as:

$$\begin{aligned} t_X &= \frac{2.769T_R + 1.752T_G + 1.13T_B}{2.769T_R + 1.752T_G + 1.13T_B + T_R + 4.591T_G + 0.06T_B + 0.057T_G + 5.594T_B} \\ &= \frac{2.769T_R + 1.752T_G + 1.130T_B}{3.769T_R + 6.400T_G + 6.784T_B} \end{aligned} \quad (7.97)$$

If we divide the numerator and denominator with $T_R + T_G + T_B$, we shall have on the right-hand side the chromaticity values of the *CIE RGB* colour system:

$$t_X = \frac{2.769t_R + 1.752t_G + 1.130t_B}{3.769t_R + 6.400t_G + 6.784t_B} \quad (7.98)$$

In a similar way, we may compute t_Y and t_Z . Calculations with more significant figures yield the more accurate results given below:

$$\begin{aligned} t_X &= \frac{0.49000t_R + 0.31000t_G + 0.20000t_B}{0.66697t_R + 1.13240t_G + 1.20063t_B} \\ t_Y &= \frac{0.17697t_R + 0.81240t_G + 0.01063t_B}{0.66697t_R + 1.13240t_G + 1.20063t_B} \\ t_Z &= \frac{0.00000t_R + 0.01000t_G + 0.99000t_B}{0.66697t_R + 1.13240t_G + 1.20063t_B} \end{aligned} \quad (7.99)$$

Note that equation (7.98) is identical with the first of equations (7.99), if we divide its numerator and denominator with 5.651, which is the sum of the coefficients of T_Y in (7.96).

What is the chromaticity diagram of the XYZ colour system?

We use the tristimulus values of the XYZ colour system, shown in figure 7.27, to work out the (t_X, t_Y) values in order to plot the spectrum locus and the purple line of this colour system. The result is shown in Plate Ib. Note that the spectrum locus and the purple line now are entirely inside the right-angle triangle of permissible values. The interesting thing here is that the primary lights, ie the vertices of the right-angle triangle of permissible values, do not correspond to any real colours. These primaries are said to be **imaginary**.

How is it possible to create a colour system with imaginary primaries, in practice?

It is not possible. The imaginary primaries are simply mathematical creations. The tristimulus values of such a colour system are not worked out by performing physical experiments with human subjects, but by simply transforming the tristimulus values of colour systems that have non-imaginary primary lights.

Example B7.16

Show that vector $\mathbf{V} \equiv (1, 4.5907, 0.0601)$ is perpendicular to the alychne plane.

Consider a point on the alychne plane with position vector $\mathbf{r} = (r_1, r_2, r_3)$. Since the alychne plan passes through the origin of the axes, the position vector of every point on the alychne plane lies on the alychne plane. For vector \mathbf{V} to be orthogonal to the alychne plane, it must be orthogonal to any line that lies on the plane. So, for vector \mathbf{V} to be orthogonal to the alychne plane, its dot product with the position vector of any point on the alychne plane must be 0: $r_1 + 4.5907r_2 + 0.0601r_3 = 0$. This is true since (r_1, r_2, r_3) lies on the alychne and thus its coordinates satisfy equation (7.94).

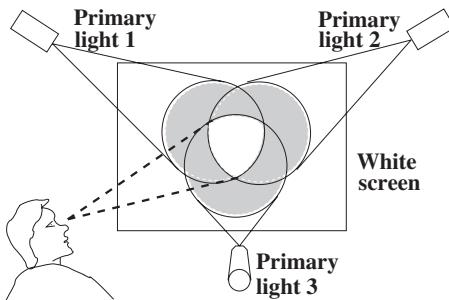


Figure 7.28: The three primary lights are projected onto a white screen. At the place where the three lights overlap, a human observer sees white. The intensities with which the three beams have to be projected depends on the observer. Different people require different intensities to say that they see white.

What if we wish to model the way a particular individual sees colours?

In all experiments done to identify the colour matching function, the viewer was asked to match perceived colours, *not* to name the colour she sees. It is possible, for example, that what a certain viewer considers to be pure white, another viewer may identify as “creamy” or “magnolia white”, etc. This indicates that different individuals see colours differently.

Let us say that in order to create the sensation of white in the brain of a particular viewer, the viewer has to see simultaneously:

spectrum $P_1(\lambda)$ with intensity $A_1(W)$;
 spectrum $P_2(\lambda)$ with intensity $A_2(W)$;
 spectrum $P_3(\lambda)$ with intensity $A_3(W)$.

We may then write:

$$W(\lambda) \tilde{=} A_1(W)P_1(\lambda) + A_2(W)P_2(\lambda) + A_3(W)P_3(\lambda) \quad (7.100)$$

Here sign $\tilde{=}$ means “creates the sensation of” (see figure 7.28).

If different viewers require different intensities of the primary lights to see white, how do we calibrate colours between different viewers?

In order to avoid subjectivity, we usually refer all colours to the so called **reference white**, ie the white of the particular viewer we have in mind. In other words, we calibrate the colours so that they are all measured in terms of the corresponding white of the intended viewer. In order to compare colours then, in an objective way, we say that we correct them for the corresponding reference white.

How do we make use of the reference white?

Let us say that in order to create the sensation of an arbitrary colour $C(\lambda)$ in the brain of the person, for whom equation (7.100) holds, we need to project at the same spot simultaneously:

spectrum $P_1(\lambda)$ with intensity $A_1(C)$;
 spectrum $P_2(\lambda)$ with intensity $A_2(C)$;
 spectrum $P_3(\lambda)$ with intensity $A_3(C)$.

We may then write:

$$C(\lambda) \tilde{=} A_1(C)P_1(\lambda) + A_2(C)P_2(\lambda) + A_3(C)P_3(\lambda) \quad (7.101)$$

We *postulate* here that the normalised quantities $A_j(C)/A_j(W)$ are viewer-independent and we call them **tristimulus values** of colour $C(\lambda)$:

$$T_j(C) \equiv \frac{A_j(C)}{A_j(W)} \quad \text{for } j = 1, 2, 3 \quad (7.102)$$

Note that these tristimulus values are meant to correspond to the objective tristimulus values of the colour system used as shown in figure 7.17. Using equation (7.102) then into equation (7.101), we may write:

$$C(\lambda) \tilde{=} \sum_{j=1}^3 T_j(C)A_j(W)P_j(\lambda) \quad (7.103)$$

In this expression $T_j(C)$ are the *objective* tristimulus values that fully and objectively characterise a colour in relation to a given colour system. Factor $A_j(W)$ in (7.103) is the only

subjective quantity that makes the equation specific for a particular viewer. If we omit it, we shall have an equation that refers to the standard observer, as long as the tristimulus values used are the average colour matching functions of many people with normal vision. The use of the tristimulus values of the standard observer will create different colour sensations for different viewers, according to each viewer's internal hardware, ie their personalised way of seeing colours.

Example 7.17

Use the information supplied by the graphs in figure 7.17 to decide what the tristimulus values are, for a spectral distribution with intensity $X(\psi)$ at wavelength ψ and 0 everywhere else.

At this stage, we are not concerned with the sensor that will view the created spectrum. So, it is assumed that the three spectra are blended linearly, so, the tristimulus values we need now are: $X(\psi)T_{s1}(\psi)$, $X(\psi)T_{s2}(\psi)$ and $X(\psi)T_{s3}(\psi)$.

We may, therefore, write:

$$X(\psi)\delta(\lambda - \psi) = \sum_{j=1}^3 T_{sj}(\psi)X(\psi)P_j(\lambda) \quad (7.104)$$

For a specific viewer, this equation has the form

$$X(\psi)\delta(\lambda - \psi) \tilde{=} \sum_{j=1}^3 X(\psi)T_{sj}(\psi)A_j(W)P_j(\lambda) \quad (7.105)$$

and corresponds to equation (7.103).

Example 7.18

Use the information supplied by the graphs in figure 7.17, on page 716, to decide what the tristimulus values are for a given spectral distribution $X(\lambda)$.

In this case, equation (7.104) has the form:

$$X(\lambda) = \sum_{j=1}^3 T_j(X)P_j(\lambda) \quad (7.106)$$

Let us integrate both sides of equation (7.104) with respect to ψ :

$$\int_0^{+\infty} X(\psi)\delta(\lambda - \psi)d\psi = \sum_{j=1}^3 \int_0^{+\infty} X(\psi)T_{sj}(\psi)d\psi P_j(\lambda) \quad (7.107)$$

By performing the integration, we obtain:

$$X(\lambda) = \sum_{j=1}^3 P_j(\lambda) \int_0^{+\infty} X(\psi) T_{sj}(\psi) d\psi \quad (7.108)$$

By comparing then equations (7.106) and (7.108), we deduce the tristimulus values that are needed to create the particular colour sensation in the brain of the standard observer:

$$T_j(X) = \int_0^{+\infty} X(\psi) T_{sj}(\psi) d\psi \quad (7.109)$$

For a particular observer, equation (7.106) will take the form:

$$X(\lambda) = \sum_{j=1}^3 T_j(X) A_j(W) P_j(\lambda) \quad (7.110)$$

How is the *sRGB* colour system defined?

This colour system was introduced to comply with the way electronic monitors reproduce colour. It is defined from the *XYZ* colour system, so that:

- (i) the (x, y, z) coordinates of its *R* light are $(0.64, 0.33, 0.03)$;
- (ii) the (x, y, z) coordinates of its *G* light are $(0.30, 0.60, 0.10)$;
- (iii) the (x, y, z) coordinates of its *B* light are $(0.15, 0.06, 0.79)$;
- (iv) the (x, y, z) coordinates of its reference white are those of the standard illuminant D_{65} , namely $(0.3127, 0.3290, 0.3583)$.

Assuming that X , Y and Z vary in the range $[0, 1]$, the transformation matrix from (X, Y, Z) values to the *sRGB* values is:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.2419 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (7.111)$$

Values outside the range $[0, 1]$ are clipped. These (R, G, B) values, however, are not the values used to display a digital image. In order to take into consideration the nonlinearities of the displaying monitors, these values are further transformed as follows. If $R, G, B < 0.00304$, define:

$$R' = 12.92R \quad G' = 12.92G \quad B' = 12.92B \quad (7.112)$$

If $R, G, B \geq 0.00304$, define:

$$R' = 1.055R^{\frac{1}{2.4}} - 0.055 \quad G' = 1.055G^{\frac{1}{2.4}} - 0.055 \quad B' = 1.055B^{\frac{1}{2.4}} - 0.055 \quad (7.113)$$

The (R', G', B') multiplied with 255 and rounded to the nearest integer are the values used to display the image on the monitor.

Does a colour change if we double all its tristimulus values?

The sensation of colour depends on the *relative* brightness of the primary lights. So, the perceived colour will not change if all its tristimulus values are multiplied with the same scalar. However, the overall brightness of the seen light will increase. Of course, this is right up to a point. When the brightness becomes too high, the sensors become saturated and the perception of colour is affected. At the other extreme, if the overall brightness becomes too low, our sensors are not triggered and we see only tones of grey, rather than colours. This is called **scotopic vision** and it is performed with the help of different sensors from those responsible for the so called **photopic vision**. The sensors responsible for the photopic vision are the cones we discussed earlier, while the sensors responsible for the scotopic vision are called **rods**.

How does the description of a colour, in terms of a colour system, relate to the way we describe colours in everyday language?

In everyday language we describe colours by using terms like “shade” and “depth”, eg we say “shades of blue”, “deep red”, etc. Formally, these terms are known as **hue** and **saturation**. For example, all shades of red, from light pink to deep red, are recognised as red of different degrees of saturation: pink has low saturation, while deep red has high saturation. On the other hand, we distinguish the hue of green, from the hue of red, irrespective of their saturations.

How do we compare colours?

If a colour is represented by a point in a 3D space, the difference of two colours can be found by using a metric to measure the distance of the two points in that space.

What is a metric?

A metric is a function that takes as input two points, A and B , and gives as output their distance. A function $f(A, B)$ has to have certain properties in order to be a metric:

1. the output of the function has to be a non-negative number, with $f(A, A) = 0$ and $f(A, B) > 0$ if $A \neq B$;
2. the output of the function should not change if we change the order of the two inputs: $f(A, B) = f(B, A)$;
3. if C is a third point, $f(A, C) + f(C, B) \geq f(A, B)$ (**triangle inequality**).

The most well known metric is the Euclidean metric. According to the Euclidean metric, the distance of two points A and B , in 3D, is given by

$$d_2(A, B) \equiv \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2} \quad (7.114)$$

where (x_A, y_A, z_A) and (x_B, y_B, z_B) are the coordinates of points A and B , respectively. This metric sometimes is also referred as the **L_2 norm**. In image processing, we also use the L_1 norm, which is also known as the **city block metric**, and is defined as:

$$d_1(A, B) \equiv |x_A - x_B| + |y_A - y_B| + |z_A - z_B| \quad (7.115)$$

When matching frequency spectra, we often use the L_∞ metric, otherwise known as **Cheby-shev norm**, defined as:

$$d_\infty \equiv \max\{|x_A - x_B|, |y_A - y_B|, |z_A - z_B|\} \quad (7.116)$$

Can we use the Euclidean metric to measure the difference of two colours?

For some colour systems, the Euclidean metric may be used; for some others it may not. For example, consider the chromaticity diagram shown in Plate Ib. We can easily see that the distance of points A and B is much bigger than the distance of points C and D ; and yet, points A and B represent two colours much more similar to each other than the colours represented by points C and D . This means that, the Euclidean distance of points in this chromaticity diagram does not reflect the perceived difference between the colours represented by the points in this space. We say that this colour space is not **perceptually uniform**. This statement is based on the tacit assumption that the metric we use to measure distances is the Euclidean metric. So, more correctly, one should say: “this colour space is not perceptually uniform with respect to the Euclidean metric”. It is of course possible, to define a metric that measures distances in a way that $d(A, B)$ turns out to be much smaller than $d(C, D)$, in accordance with the perceived difference between the colours these points represent. Such a metric would rely on rather complicated functions of the coordinates of the points the distance of which it measures. Instead, once the perceived differences between colours have been worked out (by psychophysical experiments), the colour space itself may be transformed into a new colour space, which is perceptually uniform with respect to the Euclidean metric. The transformation, obviously, has to be nonlinear and such that, for example, it brings closer points A and B , while it may spread more points C and D . Once the colours are represented in such a space, the Euclidean metric may be used to measure their differences.

Which are the perceptually uniform colour spaces?

There are two perceptually uniform colour spaces, the *Luv* and the *Lab*. They are both defined in terms of the *XYZ* colour system and the coordinates of the reference white, denoted by (X_n, Y_n, Z_n) . The transformation formulae between the *XYZ* values⁴ of a colour and the *Luv* or the *Lab* values are empirical formulae, that have been worked out so that the Euclidean metric may be used in these spaces to measure perceived colour differences.

How is the *Luv* colour space defined?

$$\begin{aligned} L &\equiv \begin{cases} 116 \left(\frac{Y}{Y_n} \right)^{1/3} - 16 & \text{if } \frac{Y}{Y_n} > 0.008856 \\ 903.3 \frac{Y}{Y_n} & \text{if } \frac{Y}{Y_n} \leq 0.008856 \end{cases} \\ u &\equiv 13L(u' - u'_n) \\ v &\equiv 13L(v' - v'_n) \end{aligned} \quad (7.117)$$

⁴Note that for the sake of simplicity we use here (X, Y, Z) instead of (T_X, T_Y, T_Z) .

The auxiliary functions that appear in these equations are defined as:

$$\begin{aligned} u' &\equiv \frac{4X}{X + 15Y + 3Z} & u'_n &\equiv \frac{4X_n}{X_n + 15Y_n + 3Z_n} \\ v' &\equiv \frac{9Y}{X + 15Y + 3Z} & v'_n &\equiv \frac{9Y_n}{X_n + 15Y_n + 3Z_n} \end{aligned} \quad (7.118)$$

How is the *Lab* colour space defined?

$$\begin{aligned} L &\equiv \begin{cases} 116 \left(\frac{Y}{Y_n} \right)^{1/3} - 16 & \text{if } \frac{Y}{Y_n} > 0.008856 \\ 903.3 \frac{Y}{Y_n} & \text{if } \frac{Y}{Y_n} \leq 0.008856 \end{cases} \\ a &\equiv 500 \left[f \left(\frac{X}{X_n} \right) - f \left(\frac{Y}{Y_n} \right) \right] \\ b &\equiv 200 \left[f \left(\frac{Y}{Y_n} \right) - f \left(\frac{Z}{Z_n} \right) \right] \end{aligned} \quad (7.119)$$

Function f that appears in these formulae is defined as

$$f(x) = \begin{cases} x^{1/3} & \text{if } x > 0.008856 \\ 7.787x + \frac{4}{29} & \text{if } x \leq 0.008856 \end{cases} \quad (7.120)$$

How do we choose values for (X_n, Y_n, Z_n) ?

The reference white depends on the conditions under which the image was captured. Often these are unknown. As a rule of thumb, we may try to be consistent with the reference white that was assumed when we transformed from the *RGB* to the *XYZ* values.

So, if the (X, Y, Z) values were produced from *CIE RGB* values, using transformation (7.96), the equal energy white E should be used with $X_n = Y_n = Z_n = 100$. If the (X, Y, Z) values were produced from *sRGB* values, using the inverse of the transformation described on page 732 and the inverse of matrix (7.111), the reference white should be the standard illuminant D_{65} , normalised so that $Y_n = 100$. The chromaticity coordinates (x_n, y_n) of a standard illuminant are given by equations (7.75) of Box 7.3, on page 704. From them we can work out the value of z_n as $z_n = 1 - x_n - y_n$. Since we want to have $Y_n = 100$, and since $y_n = Y_n / (X_n + Y_n + Z_n)$, we work out that we must set $X_n + Y_n + Z_n = 100/y_n$. Then we derive $X_n = x_n(X_n + Y_n + Z_n)$ and $Z_n = z_n(X_n + Y_n + Z_n)$. Table 7.13 lists the values of (X_n, Y_n, Z_n) for each one of the commonly used standard illuminants and for the ideal white.

How can we compute the *RGB* values from the *Luv* values?

We start with formula (7.117), to work out Y from the values of the reference white and L , as follows:

$$Y = Y_n \left(\frac{L + 16}{116} \right)^3 \quad (7.121)$$

Channel	D_{55}	D_{65}	D_{75}	E
X_n	95.6509	95.0155	94.9423	100
Y_n	100.0000	100.0000	100.0000	100
Z_n	92.0815	108.8259	122.5426	100

Table 7.13: Reference white values for (X_n, Y_n, Z_n) , for the three standard illuminants.

After we compute Y , we check whether $Y/Y_n > 0.008856$. If this is correct, we accept the value we computed. If it is not correct, we recompute Y using:

$$Y = \frac{LY_n}{903.3} \quad (7.122)$$

From the other two equations (7.117), we can easily work out u' and v' :

$$u' = \frac{u}{13L} + u'_n \quad v' = \frac{v}{13L} + v'_n \quad (7.123)$$

Then, knowing Y and u' , from the second of equations (7.118), we can work out $X+15Y+3Z$:

$$X + 15Y + 3Z = \frac{9Y}{v'} \quad (7.124)$$

We can use this value into the first of equations (7.118), to work out X :

$$X = \frac{1}{4}(X + 15Y + 3Z)u' \quad (7.125)$$

Next, from the knowledge of X , Y and $X + 15Y + 3Z$, we can work out Z :

$$Z = \frac{1}{3} \left[\frac{9Y}{v'} - X - 15Y \right] = \frac{3Y}{v'} - \frac{X}{3} - 5Y \quad (7.126)$$

The final step is to transform the XYZ values to RGB using the inverse of transformation (7.96), on page 727:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 0.4184 & -0.1587 & -0.0828 \\ -0.0912 & 0.2524 & 0.0157 \\ 0.0009 & -0.0026 & 0.1786 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (7.127)$$

How can we compute the RGB values from the Lab values?

We assume here that we know the Lab values and the values of the reference white (X_n, Y_n, Z_n) used to obtain them. We want to work out the corresponding (X, Y, Z) values and from them the corresponding (R, G, B) values.

From the value of L , we can work out the value of Y/Y_n , in the same way as in the case of the inverse Luv transformation.

Once we know the value of Y/Y_n , we can work out the value of $f(Y/Y_n)$, using equation (7.120). Then, we may use the second of formulae (7.119), to work out the value of $f(X/X_n)$:

$$f\left(\frac{X}{X_n}\right) = \frac{a}{500} + f\left(\frac{Y}{Y_n}\right) \quad (7.128)$$

Assuming that $X/X_n > 0.008856$, we can work out that:

$$\frac{X}{X_n} = \left[f\left(\frac{X}{X_n}\right) \right]^3 \quad (7.129)$$

If after this calculation X/X_n turns out to be indeed larger than 0.008856, we accept this value. If not, we rework it as:

$$\frac{X}{X_n} = \frac{1}{7.787} \left[f\left(\frac{X}{X_n}\right) - \frac{4}{29} \right] \quad (7.130)$$

This way, we derive the value of X .

Then, from the last of equations (7.119), we can work out the value of Z in a similar way:

$$f\left(\frac{Z}{Z_n}\right) = f\left(\frac{Y}{Y_n}\right) - \frac{b}{200} \quad (7.131)$$

Assuming that $Z/Z_n > 0.008856$, we can work out that:

$$\frac{Z}{Z_n} = \left[f\left(\frac{Z}{Z_n}\right) \right]^3 \quad (7.132)$$

If after this calculation Z/Z_n turns out to be indeed larger than 0.008856, we accept this value. If not, we rework it as:

$$\frac{Z}{Z_n} = \frac{1}{7.787} \left[f\left(\frac{Z}{Z_n}\right) - \frac{4}{29} \right] \quad (7.133)$$

This way, we derive the value of Z .

Finally, we have to apply the inverse transform to go from the XYZ values to the RGB values.

How do we measure perceived saturation?

In the Luv colour space saturation is defined as

$$S_{uv} = 13\sqrt{(u' - u'_n)^2 + (v' - v'_n)^2} \quad (7.134)$$

Saturation cannot be defined in the Lab colour space.

How do we measure perceived differences in saturation?

This is straightforward by applying formula (7.134) for the two colours and taking the difference of the two values.

How do we measure perceived hue?

We do not measure perceived hue, but perceived *difference* in hue. This is defined in terms of the perceived **hue angle**.

How is the perceived hue angle defined?

The hue angle, h_{uv} or h_{ab} , is defined using angle ϕ , computed from

$$\phi_{uv} \equiv \tan^{-1} \frac{|v|}{|u|} \quad \text{or} \quad \phi_{ab} \equiv \tan^{-1} \frac{|b|}{|a|} \quad (7.135)$$

for the *Luv* and the *Lab* colour spaces, respectively. Then, hue angle h_{uv} or h_{ab} is worked out as

$$h_{ij} \equiv \begin{cases} \phi_{ij} & \text{if Numerator} > 0 \quad \text{Denominator} > 0 \\ 360^\circ - \phi_{ij} & \text{if Numerator} < 0 \quad \text{Denominator} > 0 \\ 180^\circ - \phi_{ij} & \text{if Numerator} > 0 \quad \text{Denominator} < 0 \\ 180^\circ + \phi_{ij} & \text{if Numerator} < 0 \quad \text{Denominator} < 0 \end{cases} \quad (7.136)$$

where ij stands for uv or ab , *Numerator* stands for v or b , and *Denominator* stands for u or a .

How do we measure perceived differences in hue?

The perceived difference in hue is defined by considering the total perceived difference of two colours and analysing it as:

$$(Total_perceived_difference_of_two_colours)^2 \equiv (Perceived_difference_in_lightness)^2 + (Perceived_difference_in_chroma)^2 + (Perceived_difference_in_hue)^2 \quad (7.137)$$

Therefore:

$$(Perceived_difference_in_hue)^2 \equiv (Total_perceived_difference_of_two_colours)^2 - (Perceived_difference_in_lightness)^2 - (Perceived_difference_in_chroma)^2 \quad (7.138)$$

In this expression, the total perceived difference of two colours is given by the Euclidean metric, applied either to the *Luv* or the *Lab* colour space. **Lightness** is the value of L defined by equations (7.117) or (7.119). **Chroma** is defined as:

$$C_{uv} \equiv \sqrt{u^2 + v^2} \quad \text{or} \quad C_{ab} \equiv \sqrt{a^2 + b^2} \quad (7.139)$$

Example B7.19

Show that in the *Lab* system, the perceived difference in hue, ΔH_{ab} , between two colours (a, b) and $(a + \Delta a, b + \Delta b)$, where $\Delta a \ll a$ and $\Delta b \ll b$, may be computed using:

$$\Delta H_{ab} = \frac{|a\Delta b - b\Delta a|}{\sqrt{a^2 + b^2}} \quad (7.140)$$

Let us rewrite equation (7.137). On the left-hand side we have the Euclidean distance in the Lab colour space:

$$\begin{aligned} (\Delta L)^2 + (\Delta a)^2 + (\Delta b)^2 &\equiv (\Delta L)^2 + (\Delta C_{ab})^2 + (\Delta H_{ab})^2 \\ \Rightarrow (\Delta H_{ab})^2 &= (\Delta a)^2 + (\Delta b)^2 - (\Delta C_{ab})^2 \end{aligned} \quad (7.141)$$

From (7.139) we can work out ΔC_{ab} :

$$\begin{aligned} \Delta C_{ab} &= \frac{\partial C_{ab}}{\partial a} \Delta a + \frac{\partial C_{ab}}{\partial b} \Delta b \\ &= \frac{1}{2}(a^2 + b^2)^{-1/2} 2a \Delta a + \frac{1}{2}(a^2 + b^2)^{-1/2} 2b \Delta b \\ &= \frac{a \Delta a + b \Delta b}{\sqrt{a^2 + b^2}} \end{aligned} \quad (7.142)$$

We may then substitute in (7.141):

$$\begin{aligned} (\Delta H_{ab})^2 &= (\Delta a)^2 + (\Delta b)^2 - \frac{(a \Delta a + b \Delta b)^2}{a^2 + b^2} \\ &= \frac{a^2(\Delta a)^2 + b^2(\Delta a)^2 + a^2(\Delta b)^2 + b^2(\Delta b)^2 - a^2(\Delta a)^2 - b^2(\Delta b)^2 - 2ab \Delta a \Delta b}{a^2 + b^2} \\ &= \frac{(a \Delta b - b \Delta a)^2}{a^2 + b^2} \end{aligned} \quad (7.143)$$

Equation (7.140) then follows.

Example B7.20

Show that in the Lab system, the perceived difference in hue, ΔH_{ab} , between two colours (a, b) and $(a + \Delta a, b + \Delta b)$, where $\Delta a \ll a$ and $\Delta b \ll b$, is given by:

$$\Delta H_{ab} = \frac{C_{ab} \Delta h_{ab} \pi}{180} \quad (7.144)$$

From (7.136), it is obvious that $\Delta h_{ab} = |\Delta \phi_{ab}|$. From (7.135), we have:

$$\Delta \phi_{ab} = \frac{\partial}{\partial a} \left(\tan^{-1} \frac{b}{a} \right) \Delta a + \frac{\partial}{\partial b} \left(\tan^{-1} \frac{b}{a} \right) \Delta b \quad (7.145)$$

$$\begin{aligned}
 &= \frac{1}{1 + (\frac{b}{a})^2} \frac{\partial}{\partial a} \left(\frac{b}{a} \right) \Delta a + \frac{1}{1 + (\frac{b}{a})^2} \frac{\partial}{\partial b} \left(\frac{b}{a} \right) \Delta b \\
 &= \frac{a^2}{a^2 + b^2} \left(-\frac{b}{a^2} \Delta a + \frac{1}{a} \Delta b \right) \\
 &= \frac{a \Delta b - b \Delta a}{a^2 + b^2}
 \end{aligned} \tag{7.146}$$

We note that angle ϕ_{ab} and hue angle h_{ab} are measured in degrees. To be able to use their differentials defined above, we must convert them into rads by multiplying them with $\pi/180$. If we substitute then from (7.139) and (7.146) into (7.144), we obtain (7.140), which proves the validity of (7.144).

What affects the way we perceive colour?

Apart from the external factors of illumination and reflectance function, the perception of colour is also affected by the temporal and spatial context of the viewed colour surface.

What is meant by temporal context of colour?

If colour lights are flashed to a person with certain frequency, the colours the person sees do not only depend on the actual spectra of the flashed lights, but also on the frequency with which these spectra change. For example, if they change faster than, roughly, 30 times per second ($30Hz$), no matter what colour the individual lights are when seen in static conditions and individually, the person sees only tones of grey. If they are flashed with temporal frequency between roughly $6Hz$ and $30Hz$, the person sees only tones of green and red, but not tones of blue. The lights have to alternate more slowly than 6 times per second for a person to be able to perceive all colours flashed as if they were shown statically and individually.

What is meant by spatial context of colour?

Spatial context of colour refers to the colours that are next to it in space. The way we perceive colour is very much influenced by the surroundings of the colour patch we are concentrating on. This is more so when the colour of interest is not very spatially extensive. Plate IV demonstrates this very well. Seeing these two panels, we perceive on the left a more greyish square in the middle than the corresponding square on the right. And yet, both central squares were constructed to have exactly the same shade of greyish-yellow.

If the colour context changes with high spatial frequency, we may not even see the real colours but rather different colours depending on the spatial frequency with which the real colours alternate. This is demonstrated in Plate III, where depending on the distance from which the pattern is seen, instead of seeing alterations of yellow and blue the right end of the pattern may appear green.

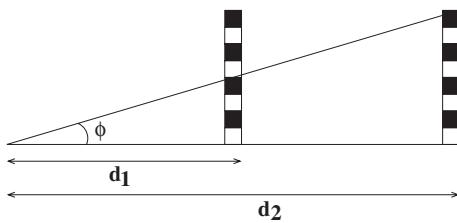


Figure 7.29: When an object is at a distance d_2 , there are more cycles of colour variation within the viewing angle ϕ , than when the same object is at a distance $d_1 < d_2$. If there are more than 16 cycles of colour variation per degree of viewing angle, then we see only grey. If there are between 4 and 16 cycles of colour variation per degree of viewing angle, we do not see blue.

Why distance matters when we talk about spatial frequency?

A viewer has a field of view, ie a cone of directions from which her sensors can receive input. Whether the viewer will see the real colours or not depends on how many alterations of colour happen inside that field of view. That is why spatial frequencies are measured in cycles per degree of viewing angle. The word “cycles” refers to alterations of colours. In the example of Plate III, a cycle is a pair of blue and yellow stripes. So, if we are further away from the pattern we see, we can fit inside one degree of viewing angle more such cycles. This is demonstrated in figure 7.29. There comes a point when the brain cannot cope with the large number of cycles, and stops seeing blue and yellow and starts seeing green.

How do we explain the spatial dependence of colour perception?

The spatial dependence of colour perception is attributed to the paths which the signals from the sensors in the retina have to share in order to reach the brain. The spectral sensitivities of the cones in the retina are highly correlated and, therefore, the signals they produce are correlated too. According to psychophysicists, these signals are not transmitted to the brain separately, but through three distinct pathways, that somehow decorrelate them and encode them for maximal efficiency. One of these pathways transmits an excitatory signal when the L cones are activated and an inhibitory signal when the M cones are activated. We say then that this path transmits the **opponent colour** $R - G$. Another pathway is identified with the transmission of light-dark variation and it has the maximal spatial resolution. The third pathway is responsible for the transmission of the blue-yellow signal and it has the least spatial resolution. That is why light-dark variations are perceived with maximal resolution, at spatial frequencies that are too high for colour identification, while the sensation of blue is lost when the spatial frequency of colour alteration is moderate. It has been worked out that the transformation from the XYZ colour space to the opponent colour space $O_1O_2O_3$ of the decorrelated signals, that eventually reach the brain, is linear:

$$\begin{pmatrix} T_{O1} \\ T_{O2} \\ T_{O3} \end{pmatrix} = \begin{pmatrix} 0.279 & 0.720 & -0.107 \\ -0.449 & 0.290 & -0.077 \\ 0.086 & -0.590 & 0.501 \end{pmatrix} \begin{pmatrix} T_X \\ T_Y \\ T_Z \end{pmatrix} \quad (7.147)$$

7.3 Colour image processing in practice

How does the study of the human colour vision affect the way we do image processing?

The way humans perceive colour is significant in image processing only in special cases:

- (i) when we try to develop an industrial inspection system that is aimed at replacing the human inspector;
- (ii) when we intent to modify images, eg compress or enhance them, with the purpose of being seen by a human.

In all other problems, we may deal with the colour images as if they were simply multispectral images. We still have to solve important problems, but we do not need to worry about the way the image would look to the human eye. For example, if we are talking about colour constancy, the term is only correct if we wish to identify colours the same way humans do. If we are simply talking about a computer recognising the same object under different illumination conditions, then the correct term is spectral constancy rather than colour constancy and the algorithm discussed on page 687 may be used for a colour image, the same way it may be used for any multispectral image.

Often, inadequate understanding of the psychophysics of colour vision lead to bad outcomes, like for example the use of the perceptually uniform colour spaces (which often turn out to be not so perceptually uniform in practice). In this section we shall see how the understanding we gained on the human visual system in the previous section may be adapted and used in practice in order to process colour images.

How perceptually uniform are the perceptually uniform colour spaces in practice?

Both *Lab* and *Luv* spaces are only approximately perceptually uniform. However, the major problem is not the approximation used to go from the standard *RGB* space to these spaces, but the approximations we make when we use these spaces in practice. For a start, formulae (7.117) and (7.119) were derived using the *RGB* colour system as defined by the CIE standard. When we apply these formulae, we use the *RGB* values we recorded with whatever sensor we are using. These values do not necessarily correspond to the *RGB* values that are meant to be used in these formulae. Next, we very seldomly know the exact values of the reference white (which is user dependent) and we use the standard illuminant instead, which, however, may not apply in our experimental set up. As a result, very often the use of *Lab* and *Luv* colour spaces in practice turns out to produce disappointing results. This is largely due to the sloppy use of these formulae, which are often applied without careful consideration of their meaning.

How should we convert the image *RGB* values to the *Luv* or the *Lab* colour spaces?

Given that most images we are dealing with have been captured by digital media, it is best to assume that the appropriate colour space for them is the *sRGB* space. We must then follow a process that is the inverse of the process presented on page 732. In the following process, the *RGB* values are assumed to be in the range [0, 255].

Step 1: Divide the RGB values of your image by 255, to convert them to the range $[0, 1]$. Call these scaled values R' , G' and B' .

Step 2: Apply to the (R', G', B') values the following nonlinear transformation, to produce the (R, G, B) values.

If $R', G', B' > 0.04045$, use:

$$\begin{aligned} R &= \left(\frac{R' + 0.055}{1.055} \right)^{2.4} \\ G &= \left(\frac{G' + 0.055}{1.055} \right)^{2.4} \\ B &= \left(\frac{B' + 0.055}{1.055} \right)^{2.4} \end{aligned} \quad (7.148)$$

If $R', G', B' \leq 0.04045$, use:

$$R = \frac{R'}{12.92} \quad G = \frac{G'}{12.92} \quad B = \frac{B'}{12.92} \quad (7.149)$$

Step 3: From these (R, G, B) values, work out the (X, Y, Z) values:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (7.150)$$

Step 4: Use either formulae (7.117) or (7.119), on page 734, with reference white the D_{65} standard illuminant, to obtain the values in the perceptually uniform colour spaces. Because of the way the XYZ values have been derived, the values of the reference white that must be used for the transformation to the perceptually uniform space have to be divided by 100. That is, the (X_n, Y_n, Z_n) values for D_{65} that must be used are: $(0.950155, 1.0000, 1.088259)$.

Example B7.21

Assume that instead of the correct reference white (X_n, Y_n, Z_n) , you use $(X_n + \Delta X_n, Y_n, Z_n)$, in the transformation formulae (7.117) and (7.118). Work out the error when you measure the distance of two colours in the Luv colour space, assuming that the two colours have the same lightness L . Use only first order perturbations.

Using first order perturbation theory, we may write:

$$\begin{aligned} u_{\text{measured}} &= u_{\text{true}} \pm \left| \frac{\partial u}{\partial X_n} \right| \Delta X_n \equiv u_{\text{true}} \pm \epsilon_u \\ v_{\text{measured}} &= v_{\text{true}} \pm \left| \frac{\partial v}{\partial X_n} \right| \Delta X_n \equiv v_{\text{true}} \pm \epsilon_v \end{aligned} \quad (7.151)$$

From (7.118) we can easily work out that:

$$\begin{aligned}\frac{\partial u}{\partial X_n} &= -13L \frac{60Y_n + 12Z_n}{(X_n + 15Y_n + 3Z_n)^2} \\ \frac{\partial v}{\partial X_n} &= 13L \frac{9Y_n}{(X_n + 15Y_n + 3Z_n)^2}\end{aligned}\quad (7.152)$$

The difference between two colours ΔE is given by the Euclidean metric in the Luv colour space. Since the two colours have the same lightness,

$$\begin{aligned}\Delta E &= \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2} \\ &= \sqrt{(u_{1t} \pm \epsilon_u - u_{2t} \pm \epsilon_u)^2 + (v_{1t} \pm \epsilon_v - v_{2t} \pm \epsilon_v)^2}\end{aligned}\quad (7.153)$$

where (u_{1t}, v_{1t}) are the true values of one colour and (u_{2t}, v_{2t}) are the true values of the other colour. Since both colours have the same value for L , both colours have the same error ϵ_u and ϵ_v in the calculation of u and v , respectively. Also, when we add or subtract values, we must always assume that the errors add (worst case scenario). Then, we may write:

$$\begin{aligned}\Delta E &= \sqrt{(u_{1t} - u_{2t})^2 + 4\epsilon_u^2 + 4\epsilon_u(u_{1t} - u_{2t}) + (v_{1t} - v_{2t})^2 + 4\epsilon_v^2 + 4\epsilon_v(v_{1t} - v_{2t})} \\ &= \underbrace{\sqrt{(u_{1t} - u_{2t})^2 + (v_{1t} - v_{2t})^2}}_{\Delta E_t} \left[1 + \frac{4\epsilon_u^2 + 4\epsilon_u(u_{1t} - u_{2t}) + 4\epsilon_v^2 + 4\epsilon_v(v_{1t} - v_{2t})}{(u_{1t} - u_{2t})^2 + (v_{1t} - v_{2t})^2} \right]^{1/2} \\ &\simeq \Delta E_t \left[1 + \frac{1}{2} \frac{|4\epsilon_u^2 + 4\epsilon_u(u_{1t} - u_{2t}) + 4\epsilon_v^2 + 4\epsilon_v(v_{1t} - v_{2t})|}{(u_{1t} - u_{2t})^2 + (v_{1t} - v_{2t})^2} \right] \\ &\simeq \Delta E_t \left[1 + \frac{|2\epsilon_u(u_{1t} - u_{2t}) + 2\epsilon_v(v_{1t} - v_{2t})|}{(u_{1t} - u_{2t})^2 + (v_{1t} - v_{2t})^2} \right]\end{aligned}\quad (7.154)$$

The final expression was derived by using $(1+x)^n \simeq 1+n$ when $0 < x \ll 1$ and by omitting terms of second order in the error values. Then, we can work out the relative error in the computed distance between the two colours, as

$$\begin{aligned}\text{Relative_Error} &\equiv \frac{|\Delta E - \Delta E_t|}{\Delta E_t} \\ &= \frac{|2\epsilon_u(u_{1t} - u_{2t}) + 2\epsilon_v(v_{1t} - v_{2t})|}{(u_{1t} - u_{2t})^2 + (v_{1t} - v_{2t})^2}\end{aligned}\quad (7.155)$$

where:

$$\begin{aligned}\epsilon_u &\equiv \left| 13L \frac{60Y_n + 12Z_n}{(X_n + 15Y_n + 3Z_n)^2} \Delta X_n \right| \\ \epsilon_v &\equiv \left| 13L \frac{9Y_n}{(X_n + 15Y_n + 3Z_n)^2} \Delta X_n \right|\end{aligned}\quad (7.156)$$

Note that the relative error is a function of the true values of the colours. This means that it is different at different parts of the colour space. This is characteristic of systems

that depend on the perturbed parameter in a nonlinear way. This nonuniformity of the expected error by itself damages the perceptual uniformity of the colour space.

Figure 7.30 shows the plot of the relative error in the computed distance between two colours, for different combinations of the true colour differences $\Delta u \equiv |u_{1t} - u_{2t}|$ and $\Delta v \equiv |v_{1t} - v_{2t}|$, for $(X_n, Y_n, Z_n) = (95.0155, 100, 108.8259)$, $L = 1$ and $\Delta X_n = 1$. Note that as L increases, these errors grow linearly.

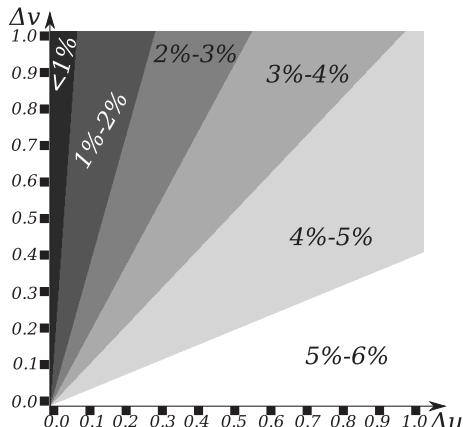


Figure 7.30: Relative error in the estimation of the distance of two colours, when their true distance is $\sqrt{(\Delta u)^2 + (\Delta v)^2}$, for $L = 1$ and $\Delta X_n = 1$.

Example B7.22

Assume that instead of the correct reference white (X_n, Y_n, Z_n) , you use $(X_n + \Delta X_n, Y_n, Z_n)$, in the transformation formulae (7.119) and (7.120). Work out the error when you measure the distance of two colours in the Lab colour space, assuming that the two colours have the same lightness L . Use only first order perturbations.

From (7.119), we note that b does not depend on X_n , and so an error in X_n will not affect its value. Using first order perturbation theory, we may then write:

$$a_{\text{measured}} = a_{\text{true}} \pm \left| \frac{\partial a}{\partial X_n} \right| \Delta X_n \equiv a_{\text{true}} \pm \epsilon_a \quad (7.157)$$

From (7.120), we can easily work out that:

$$\begin{aligned}\frac{\partial a}{\partial X_n} &= 500 \frac{df}{dX_n} \\ &= \begin{cases} 500 \left(-\frac{1}{3}\right) \frac{X^{1/3}}{X_n^{4/3}} & \text{if } \frac{X}{X_n} > 0.008856 \\ 500 \left(-\frac{7.787X}{X_n^2}\right) & \text{if } \frac{X}{X_n} \leq 0.008856 \\ -\frac{500}{3} \left(\frac{X}{X_n^4}\right)^{1/3} & \text{if } \frac{X}{X_n} > 0.008856 \\ -\frac{3893.5X}{X_n^2} & \text{if } \frac{X}{X_n} \leq 0.008856 \end{cases} \quad (7.158)\end{aligned}$$

The difference between two colours ΔE is given by the Euclidean metric in the Lab colour space. Since the two colours have the same lightness,

$$\begin{aligned}\Delta E &= \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2} \\ &= \sqrt{(a_{1t} \pm \epsilon_{a1} - a_{2t} \pm \epsilon_{a2})^2 + (b_{1t} - b_{2t})^2} \quad (7.159)\end{aligned}$$

where (a_{1t}, b_{1t}) are the true values of one colour and (a_{2t}, b_{2t}) are the true values of the other colour. The errors depend on the value of X , that characterises each colour, so the two colours have different errors in a : ϵ_{a1} and ϵ_{a2} , respectively. Then, we may write:

$$\begin{aligned}\Delta E &= \sqrt{(a_{1t} - a_{2t})^2 + \epsilon_{a1}^2 + \epsilon_{a2}^2 + 2(\epsilon_{a1} + \epsilon_{a2})(a_{1t} - a_{2t}) + 2\epsilon_{a1}\epsilon_{a2} + (b_{1t} - b_{2t})^2} \\ &= \underbrace{\sqrt{(a_{1t} - a_{2t})^2 + (b_{1t} - b_{2t})^2}}_{\Delta E_t} \left[1 + \frac{\epsilon_{a1}^2 + \epsilon_{a2}^2 + 2(\epsilon_{a1} + \epsilon_{a2})(a_{1t} - a_{2t}) + 2\epsilon_{a1}\epsilon_{a2}}{(a_{1t} - a_{2t})^2 + (b_{1t} - b_{2t})^2} \right]^{1/2} \\ &\simeq \Delta E_t \left[1 + \frac{1}{2} \frac{|\epsilon_{a1}^2 + \epsilon_{a2}^2 + 2(\epsilon_{a1} + \epsilon_{a2})(a_{1t} - a_{2t}) + 2\epsilon_{a1}\epsilon_{a2}|}{(a_{1t} - a_{2t})^2 + (b_{1t} - b_{2t})^2} \right] \\ &\simeq \Delta E_t \left[1 + \frac{|(\epsilon_{a1} + \epsilon_{a2})(a_{1t} - a_{2t})|}{(a_{1t} - a_{2t})^2 + (b_{1t} - b_{2t})^2} \right] \quad (7.160)\end{aligned}$$

Then, we can work out the relative error in the computed distance between the two colours, as

$$\begin{aligned}\text{Relative_Error} &\equiv \frac{|\Delta E - \Delta E_t|}{\Delta E_t} \\ &= \frac{|(\epsilon_{a1} + \epsilon_{a2})(a_{1t} - a_{2t})|}{(a_{1t} - a_{2t})^2 + (b_{1t} - b_{2t})^2} \quad (7.161)\end{aligned}$$

where the errors are computed with the help of equations (7.157) and (7.158).

How do we measure hue and saturation in image processing applications?

There do not seem to be any standard formulae for this. One may quantify these two concepts on the plane, shown in figure 7.21. We use as reference point the reference white. We may then define saturation as the distance from that point. The further away we go from the reference white, the more saturated a colour becomes. We may also choose an orientation as the reference orientation. Let us say that our reference orientation is the line that connects the origin with the vertex marked R . We may measure the hue of a colour by the angle it forms with the reference direction. If this angle is 0, the hue may be red, and, as it increases, it passes through the shades of green and blue, before it becomes red again.

If (r, g, b) are the normalised colour coordinates of a point, and (r_w, g_w, b_w) are the coordinates of the reference white, then hue and saturation for that colour are given by (see example 7.27):

$$\text{Saturation} \equiv \frac{1}{\sqrt{2}} \sqrt{(g - r - g_w + r_w)^2 + 3(b - b_w)^2} \quad (7.162)$$

$$\phi = \tan^{-1} \left\{ \frac{\sqrt{3}|b_w(1+g-r) - b(1+g_w-r_w)|}{|(1+g_w-r_w)(g_w-r_w-g+r) + 3b_w(b_w-b)|} \right\} \equiv \tan^{-1} \left\{ \frac{\sqrt{3}|\text{Numerator}|}{|\text{Denominator}|} \right\} \quad (7.163)$$

Then:

$$\text{Hue} \equiv \begin{cases} \phi & \text{if Numerator} > 0 \quad \text{Denominator} > 0 \\ 360^\circ - \phi & \text{if Numerator} < 0 \quad \text{Denominator} > 0 \\ 180^\circ - \phi & \text{if Numerator} > 0 \quad \text{Denominator} < 0 \\ 180^\circ + \phi & \text{if Numerator} < 0 \quad \text{Denominator} < 0 \end{cases} \quad (7.164)$$

If we use as reference white the ideal white with values $r_w = g_w = b_w = 1/3$ and scaling so that the maximum value of saturation is 1, the above formulae take the form (see example 7.23):

$$\text{Saturation}_{\text{ideal_white}} \equiv \frac{\sqrt{3}}{2} \sqrt{(g - r)^2 + 3 \left(b - \frac{1}{3} \right)^2} \quad (7.165)$$

$$\phi_{\text{ideal_white}} = \tan^{-1} \left\{ \frac{\sqrt{3} \left| \frac{1}{3}(1+g-r) - b \right|}{\left| r - g - b + \frac{1}{3} \right|} \right\} \quad (7.166)$$

Example 7.23

Work out the values of the hue and saturation for the red, green and blue colours of the Maxwell triangle, when as reference white we use the ideal white.

The (r, g, b) coordinates of the red, green and blue colours are $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$, respectively. For the ideal white, $r_w = g_w = b_w = 1/3$. Then using formula (7.162), the saturation of all three colours is worked out to be: $\sqrt{2/3}$. One may use

this value to normalise the definition of the saturation, so that it takes values between 0 and 1, and thus derive formula (7.165).

For angle ϕ , we use (7.166):

$$\begin{aligned}\phi_R &= 0 \\ \phi_G &= \tan^{-1} \left\{ \frac{\sqrt{3} \left| \frac{1}{3} \times 2 \right|}{\left| -1 + \frac{1}{3} \right|} \right\} = \tan^{-1} \sqrt{3} = 60^\circ \\ \phi_B &= \tan^{-1} \left\{ \frac{\sqrt{3} \left| \frac{1}{3} - 1 \right|}{\left| -1 + \frac{1}{3} \right|} \right\} = \tan^{-1} \sqrt{3} = 60^\circ\end{aligned}\quad (7.167)$$

Applying then the rules of (7.164), we work out that:

$$\begin{aligned}Hue_{R,ideal:white} &= 0^\circ \\ Hue_{G,ideal:white} &= 180^\circ - \phi_G = 120^\circ \\ Hue_{B,ideal:white} &= 180^\circ + \phi_B = 240^\circ\end{aligned}\quad (7.168)$$

Example B7.24

The Maxwell colour triangle of a colour system is shown in figure 7.31. Point W is the reference white. The reference direction is defined to be line WR . Hue is the angle formed by the direction of a point P , with coordinates (x, y) , with respect to the reference direction, measured counterclockwise. Use vector calculus to work out the hue of point P .

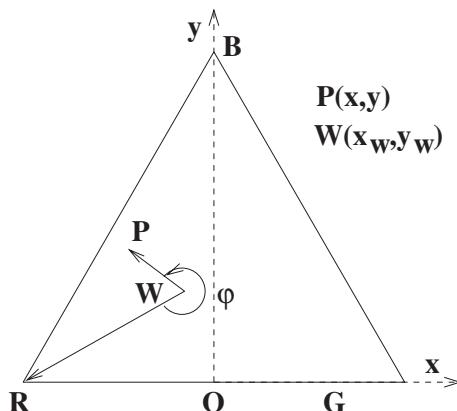


Figure 7.31: The Maxwell colour triangle. Angle ϕ is the hue of point P .

Point R has coordinates $(-\sqrt{2}/2, 0)$. Therefore, vectors \overline{WR} and \overline{WP} have coordinates:

$$\overline{WR} = \left(-\frac{\sqrt{2}}{2} - x_w, -y_w \right) \quad \overline{WP} = (x - x_w, y - y_w) \quad (7.169)$$

We consider the dot product of these two vectors:

$$\begin{aligned} \overline{WR} \cdot \overline{WP} &= |\overline{WR}| |\overline{WP}| \cos \phi \\ \Rightarrow \cos \phi &= \frac{\overline{WR} \cdot \overline{WP}}{|\overline{WR}| |\overline{WP}|} \end{aligned} \quad (7.170)$$

Therefore:

$$\cos \phi = \frac{\left(\frac{\sqrt{2}}{2} + x_w\right)(x_w - x) + y_w(y_w - y)}{\sqrt{\left(\frac{\sqrt{2}}{2} + x_w\right)^2 + y_w^2} \times \sqrt{(x - x_w)^2 + (y - y_w)^2}} \quad (7.171)$$

As ϕ is measured from 0° to 360° , to specify it fully we need to know its sine as well as its cosine (or at least the sign of its sine so we can work out in which quadrant it belongs). To compute the sine, we have to take the cross product of vectors \overline{WR} and \overline{WP} . Let us associate unit vectors \mathbf{i} and \mathbf{j} with axes Ox and Oy , respectively. For a right-handed coordinate system, the third axis will have unit vector, say, \mathbf{k} that sticks out of the page. The cross product of vectors \overline{WR} and \overline{WP} then will be:

$$\begin{aligned} \overline{WR} \times \overline{WP} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -\frac{\sqrt{2}}{2} - x_w & -y_w & 0 \\ x - x_w & y - y_w & 0 \end{vmatrix} \\ &= \mathbf{k} \left[\left(-\frac{\sqrt{2}}{2} - x_w \right) (y - y_w) - (-y_w)(x - x_w) \right] \\ &= \mathbf{k} \left[\left(\frac{\sqrt{2}}{2} + x_w \right) (y_w - y) + y_w(x - x_w) \right] \end{aligned} \quad (7.172)$$

We know that the component of the cross product along unit vector \mathbf{k} is given by $|\overline{WR}| |\overline{WP}| \sin \phi$. So, we deduce that:

$$\sin \phi = \frac{\left(\frac{\sqrt{2}}{2} + x_w \right) (y_w - y) + y_w(x - x_w)}{\sqrt{\left(\frac{\sqrt{2}}{2} + x_w \right)^2 + y_w^2} \sqrt{(x - x_w)^2 + (y - y_w)^2}} \quad (7.173)$$

Using (7.171) and (7.173) allows us to work out the value of ϕ in the range $[0^\circ, 360^\circ]$.

Example B7.25

Use trigonometry to compute the sine of angle ϕ in the Maxwell colour triangle of figure 7.31.

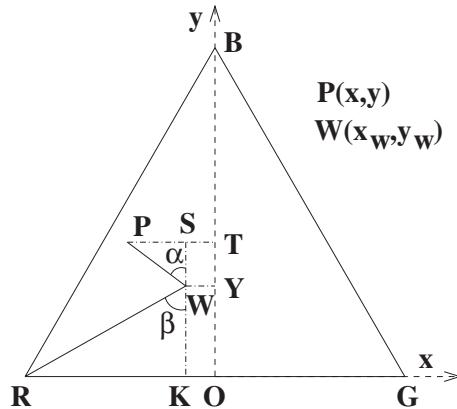


Figure 7.32: The Maxwell colour triangle of the RGB colour space.

Figure 7.32 repeats figure 7.31 and is conveniently labelled for this example. We note that $\phi = \beta + 180^\circ + \alpha$. So:

$$\sin \phi = -\sin(\alpha + \beta) = -\sin \alpha \cos \beta - \sin \beta \cos \alpha \quad (7.174)$$

We project points P and W on the OB axis to points T and Y , respectively, and point W on the RG axis to point K . From the right angle triangles PSW and WKR , we have:

$$\sin \alpha = \frac{PS}{PW} \quad \cos \alpha = \frac{WS}{PW} \quad \sin \beta = \frac{RK}{RW} \quad \cos \beta = \frac{WK}{RW} \quad (7.175)$$

Let us say that the coordinates of the reference white are (x_w, y_w) . The coordinates of R obviously are $(-\sqrt{2}/2, 0)$. All quantities that appear in (7.175) are lengths of line segments, and, therefore, positive numbers. So, when we express them in terms of the coordinates of the projected points, we must be careful when these coordinates are negative. For example, we cannot say that $PT = x$, because x is obviously negative from the geometry shown in figure 7.32. With that in mind, we may write:

$$PS = -x + x_w \quad WK = y_w \quad RK = \frac{\sqrt{2}}{2} + x_w \quad WS = y - y_w \quad (7.176)$$

Lengths PW and RW are obviously given by

$$PW = \sqrt{(x - x_w)^2 + (y - y_w)^2} \quad RW = \sqrt{\left(\frac{\sqrt{2}}{2} + x_w\right)^2 + y_w^2} \quad (7.177)$$

Substituting from (7.176) and (7.177) into (7.175) and from there into (7.174), we can easily work out the expression for $\sin \phi$, which turns out to be the same as the one given by equation (7.173).

Example B7.26

Use trigonometry to compute the sine of the hue angle in the Maxwell colour triangle of figure 7.33.

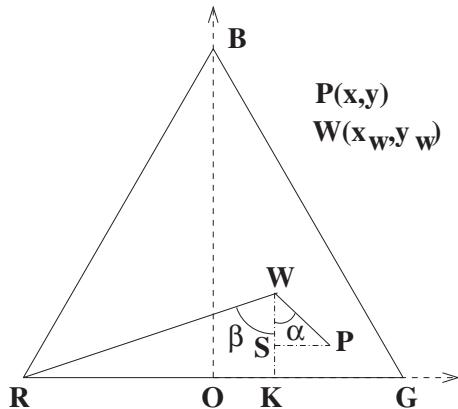


Figure 7.33: A colour space and its corresponding Maxwell triangle.

We note that here the hue angle $\phi = 360^\circ - \alpha - \beta$. Therefore:

$$\sin \phi = -\sin(\alpha + \beta) = -\sin \alpha \cos \beta - \sin \beta \cos \alpha \quad (7.178)$$

The trigonometric functions that appear here can be expressed again by equations (7.175). In terms of the coordinates of the points involved, the lengths that appear in (7.175) are:

$$PS = x - x_w \quad WS = y_w - y \quad RK = \frac{\sqrt{2}}{2} + x_w \quad WK = y_w \quad (7.179)$$

Lengths PW and RW are again given by equations (7.177). Using these equations into (7.178), we work out that $\sin \phi$ is given by equation (7.173) in this case too.

Example B7.27

Work out the values of the tangent of the hue angle and of the saturation in terms of the *RGB* values of a colour.

For the tangent of the hue angle all we have to do is to take the ratio of equations (7.171) and (7.173), and substitute (x, y) and (x_w, y_w) in terms of the normalised (r, g, b) values of the corresponding colours, given by equation (7.93), on page 724:

$$\begin{aligned}\tan \phi &= \frac{\left(\frac{\sqrt{2}}{2} + x_w\right)(y_w - y) + y_w(x - x_w)}{\left(\frac{\sqrt{2}}{2} + x_w\right)(x_w - x) + y_w(y_w - y)} \\ &= \frac{\frac{1+g_w-r_w}{\sqrt{2}} \times \frac{\sqrt{3}(b_w-b)}{\sqrt{2}} + \frac{\sqrt{3}b_w}{\sqrt{2}} \times \frac{g-r-g_w+r_w}{\sqrt{2}}}{\frac{1+g_w-r_w}{\sqrt{2}} \times \frac{g_w-r_w-g+r}{\sqrt{2}} + \frac{\sqrt{3}b_w}{\sqrt{2}} \times \frac{\sqrt{3}(b_w-b)}{\sqrt{2}}} \\ &= \frac{\sqrt{3}[b_w(1+g-r) - b(1+g_w-r_w)]}{(1+g_w-r_w)(g_w-r_w-g+r) + 3b_w(b_w-b)}\end{aligned}\quad (7.180)$$

The value of the saturation is given by the length of vector \overline{WP} of figure 7.31:

$$\begin{aligned}S &= \sqrt{(x - x_w)^2 + (y - y_w)^2} \\ &= \sqrt{\frac{(g - r - g_w + r_w)^2 + 3(b - b_w)^2}{2}}\end{aligned}\quad (7.181)$$

How can we emulate the spatial dependence of colour perception in image processing?

First, the *RGB* bands of an image are converted into the *XYZ* bands using transformation (7.96):

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 2.769 & 1.752 & 1.130 \\ 1.000 & 4.591 & 0.060 \\ 0.000 & 0.057 & 5.594 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}\quad (7.182)$$

Then the opponent colour bands $O_1O_2O_3$ are computed using transformation (7.147), which is repeated here for convenience:

$$\begin{pmatrix} O_1 \\ O_2 \\ O_3 \end{pmatrix} = \begin{pmatrix} 0.279 & 0.720 & -0.107 \\ -0.449 & 0.290 & -0.077 \\ 0.086 & -0.590 & 0.501 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}\quad (7.183)$$

The opponent colour bands may be blurred with kernels that are designed to imitate the way the human eye blurs colours. Three blurring kernels are required, one for each opponent

Channel	w_{ki}	h_{ki}
O_1 ($N_1 = 3$)	$w_{11} = 1.003270$	$h_{11} = 0.0500$
	$w_{12} = 0.114416$	$h_{12} = 0.2250$
	$w_{13} = -0.117686$	$h_{13} = 7.0000$
O_2 ($N_2 = 2$)	$w_{21} = 0.616725$	$h_{21} = 0.0685$
	$w_{22} = 0.383275$	$h_{22} = 0.8260$
O_3 ($N_3 = 2$)	$w_{31} = 0.567885$	$h_{31} = 0.0920$
	$w_{32} = 0.432115$	$h_{32} = 0.6451$

Table 7.14: Parameters for blurring the O_1 , O_2 and O_3 channels in order to imitate the human way of seeing colours from a distance.

colour band. Each of these blurring kernels is constructed as the sum of some Gaussian functions, as follows:

$$g_k(x, y) = \sum_{i=1}^{N_k} \frac{w_{ki}}{T_i} e^{-\alpha_{ki}^2(x^2+y^2)} \quad (7.184)$$

Here index k identifies the opponent colour band, (x, y) are spatial coordinates of the 2D blurring mask, N_k is the number of components that are used to create the mask for channel k , T_i is a normalising constant that makes the sum of the elements of the i th component equal to 1, w_{ki} is the weight with which the i th component contributes to the mask for channel k and α_{ki} is the spreading parameter for the i th component of channel k , computed from the parameter h_{ki} , using:

$$\alpha_{ki} = \frac{2\sqrt{\ln 2}}{Fh_{ki} - 1} \quad \text{for } Fh_{ki} > 1 \quad (7.185)$$

This formula comes about because h_{ki} is the **half width half maximum** of the Gaussian function, ie it measures the distance from the centre where the value of the Gaussian drops to half its central value. This quantity is measured in degrees of visual angle. Factor F is used to convert the degrees of visual angle in pixels. Usually, product Fh_{ki} is quite high, so removing 1 from it does not change it much. However, if this product is 1 or lower, the implication is that the spreading of the Gaussian is in subpixel values, and so this Gaussian may be omitted. (Note that α_{ki} is inversely proportional to the standard deviation, and if $\alpha_{ki} \rightarrow +\infty$, the standard deviation goes to 0.) So, setting $\alpha_{ki} = 0$ for $Fh_{ki} \leq 1$, safeguards against creating filters with sub-pixel spread. Table 7.14 lists the values of parameters w_{ki} and h_{ki} for each channel.

Note that spatial coordinates (x, y) have to be in pixels. To emulate the human visual system, we must relate the resolution of the sensor measured in mm per pixel, to the distance from which the particular object is assumed to be seen.

After the $O_1O_2O_3$ channels have been blurred, they are transformed back to the XYZ space, using the inverse of transformation (7.147), on page 741, and from there back to the RGB space using the inverse of transformation (7.182), for displaying or further processing.

This sequence of transformations is called **S-CIELAB** colour system.

Example B7.28

Consider the image of Plate VIa. It was scanned so that a 13cm picture was represented by 512 pixels. Blur this image so that it looks the way it would appear when seen from distances 2, 4 and 6 metres. To avoid having to introduce a background, use the full image as the scene, but concentrate your processing only to the central part.

The scale of the image is $r = 512/130 \simeq 4$ pixels per mm. Next, we shall work out the size of an object at a distance of 2m, 4m and 6m. Figure 7.34 shows how the distance D of the seen object is related to the size S of the object and the visual angle.

$$\tan \theta = \frac{S}{D} \Rightarrow S = D \tan^{-1} \theta \quad (7.186)$$

By setting $\theta = 1^\circ$, and measuring D in mm, we can work out how many mm 1° of visual angle corresponds to, for the selected distance. This way, we work out that $S_2 = 34.91$, $S_4 = 69.82$ and $S_6 = 104.73$ mm per degree of visual angle, for distances 2, 4 and 6 metres, respectively. If we then multiply these values with r , we shall convert them into pixels per visual angle and thus have the factor F we need in order to work out the smoothing filters. It turns out that $F_2 = 139.64$, $F_4 = 279.28$ and $F_6 = 418.92$. These values of F are used in conjunction with formulae (7.184) and (7.185), and table 7.14 to work out the three smoothing masks for each distance. For a 4m distance, the cross-sections of the constructed filters are shown in figure 7.35. We note that the filter for smoothing the O_1 component, that corresponds to image brightness, is the narrowest, while the filter for smoothing the O_3 component, that corresponds to the blue-yellow colours, is the broadest. In fact, the three filters for the three distances, with truncation error 0.1, turned out to be of sizes 11×11 , 15×15 and 21×21 , for 2m, 23×23 , 33×33 and 45×45 for 4m, and 37×37 , 51×51 and 69×69 , for 6m. Figure 7.36 shows the original opponent colour channels of the input image and the blurred ones, with the filters used for the 4m distance. The final colour images constructed are shown in Plates VIb, VIc and VId.

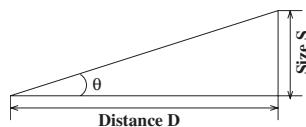


Figure 7.34: The relationship between the distance and size of an object, and the visual angle.

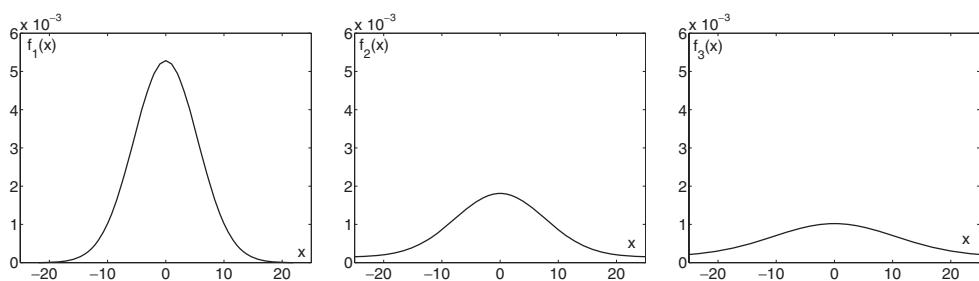


Figure 7.35: The smoothing filters for distance 4m, for the O_1 , O_2 and O_3 channels, from left to right, respectively.

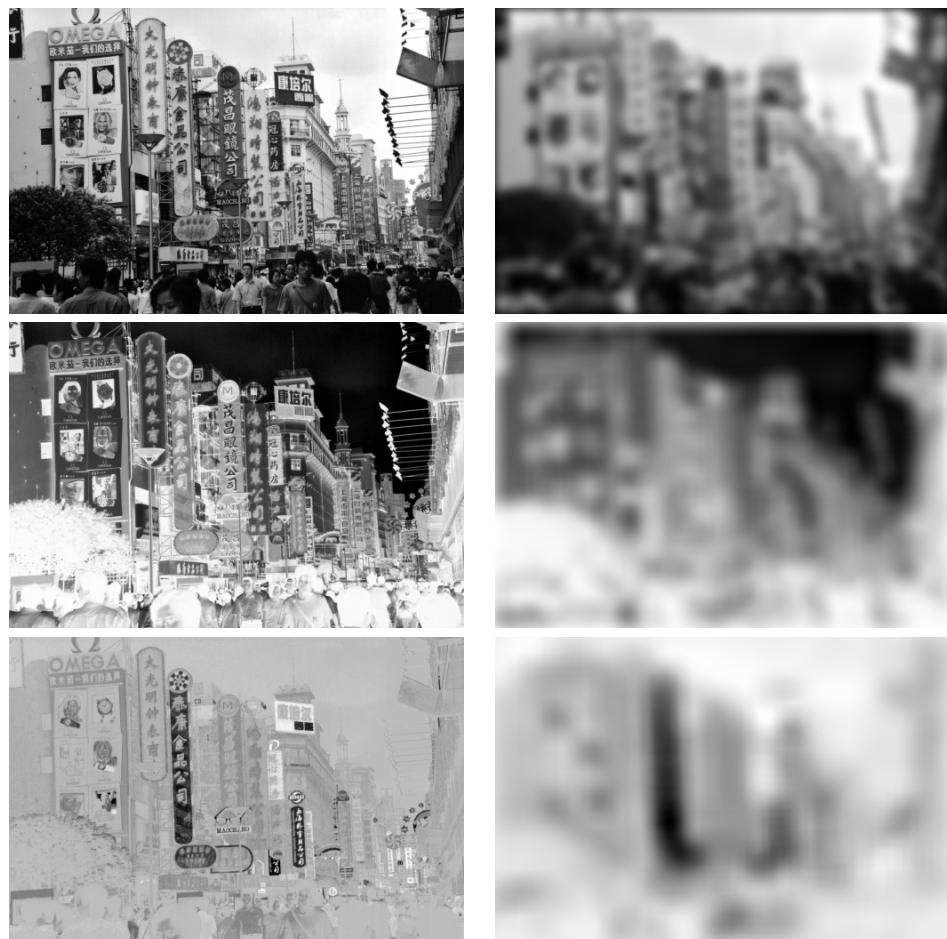


Figure 7.36: The original opponent channels (on the left) and after they have been blurred with the filters for a distance of 4m (on the right).

What is the relevance of the phenomenon of metamerism to image processing?

Obviously, the metamers of the human eye are different from the metamers of the electronic sensors, because of the different spectral responses of the two types of sensor. The phenomenon of metamerism then is of paramount importance when one wishes to use computer vision to grade colours of objects that are aimed for the consumer market. For example, if one wishes to grade ceramic tiles in batches of the same colour shade, tiles placed in the same batch by the computer should also be placed in the same batch by a human, otherwise the grading performed will not be acceptable to the consumer: people may receive tiles that are of different colour shade, while the computer thought the tiles packed together belonged to the same colour shade.

How do we cope with the problem of metamerism in an industrial inspection application?

The problem we wish to solve here may be expressed as follows. “If the values recorded by a set of three colour sensors for a particular pixel are (Q_1, Q_2, Q_3) , what would the recorded values be if the same surface patch was seen by another set of three colour sensors with different sensitivities, but under the same illumination conditions?” In practice, the first set of recorded values (Q_1, Q_2, Q_3) would be the (R, G, B) values of a pixel in the image captured by an ordinary colour camera and the second set of values would be the values “recorded” by the human eye.

Assuming that we work with sampled values of the wavelength λ , equation (7.80), on page 711, may be used in digital form to express the values that will be recorded by the camera sensors, with sensitivities $S_1(\lambda)$, $S_2(\lambda)$ and $S_3(\lambda)$, under illumination $I(\lambda)$, for an object with reflectance function $R(\lambda)$:

$$\begin{aligned} Q_1 &= \sum_{i=1}^N S_1(\lambda_i) I(\lambda_i) R(\lambda_i) \\ Q_2 &= \sum_{i=1}^N S_2(\lambda_i) I(\lambda_i) R(\lambda_i) \\ Q_3 &= \sum_{i=1}^N S_3(\lambda_i) I(\lambda_i) R(\lambda_i) \end{aligned} \quad (7.187)$$

Here N is the number of points we use to sample the range of wavelengths λ . Typically, $N = 31$, as λ is in the range $[400\text{nm}, 700\text{nm}]$ and we sample with a step of 10nm .

If we want to know what values another set of sensors would record, under the same illumination and for the same object, we have to solve these equations for $R(\lambda_i)$ and use these values in another set of similar equations, where the sensitivity functions are replaced with the sensitivity functions of the new set of sensors. This, clearly, is not possible, as we have $N \gg 3$ unknowns, and only 3 equations. So, we have to work in an indirect way to recover the transformation from the 3D space of the colour camera values to the 3D space of the human eye values.

As the equations involved are linear, we may expect that the transformation we are seeking between the two spaces of the recorded values are also linear, expressed by a matrix A . This is expressed schematically in figure 7.37.

We may then work as follows. Here for simplicity we assume that $N = 31$, so that a reflectance function is a 31D vector.

- Choose with a Monte-Carlo method M points in the 31D reflectance function space, that map through equations (7.187) into a small “ball” of radius δE centred at point (E_1^0, E_2^0, E_3^0) in the 3D human sensor space. Call the values of these points in the 3D human sensor space (E_1^i, E_2^i, E_3^i) , where index i is used to identify the selected point.
- Find the values the camera sensors would record for the chosen 31-tuples, which represent the selected reflectance functions and which are mapped within the ball of radius δE in the human sensor space. Let us call them (Q_1^i, Q_2^i, Q_3^i) .
- Solve in the least square error sense the system of equations

$$\begin{pmatrix} E_1^i \\ E_2^i \\ E_3^i \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} Q_1^i \\ Q_2^i \\ Q_3^i \end{pmatrix} \quad \text{for } i = 1, 2, \dots, M \quad (7.188)$$

for the unknown values a_{ij} of matrix A .

- Repeat that for many balls of radius δE , centred at several locations (E_1^0, E_2^0, E_3^0) of the 3D human sensor space.

Once the transformation from the camera space to the human visual system space has been worked out, the (R, G, B) values of a colour seen by the camera may be converted to the corresponding values in the human visual system space. Subsequent clustering of colours can then proceed in this human visual system space.

In a practical application of grading ceramic tiles, it has been reported that all matrices A , constructed for different locations (E_1^0, E_2^0, E_3^0) , turned out to be identical, within some tolerance. Thus, the transformation from the camera recordings to the human “recordings” had been identified, and used to work out, from the camera values, for each ceramic tile, the values that would have been seen by a human observer. By clustering the transformed values, batches of tiles of indistinguishable colour shade were then created, the same way they would have been created by human inspectors.

What is a Monte-Carlo method?

A Monte-Carlo method is a rather “brute force” method of solving a problem by using random numbers. In the particular case of the metamers of a reflectance function with respect to a particular sensor, we have to choose other reflectance functions that are mapped to the same values by the sensor. There is no analytic way to do that. All we can do is to choose at random reflectance functions, test each one of them to see whether it produces eye response values near enough to the values of the original reflectance function, so that the human brain will confuse them as identical, and if so, keep it, if not disregard it. Another application of the Monte-Carlo approach is the estimation of the volume of an object defined by a surface that cannot be easily described analytically (see example 7.29).

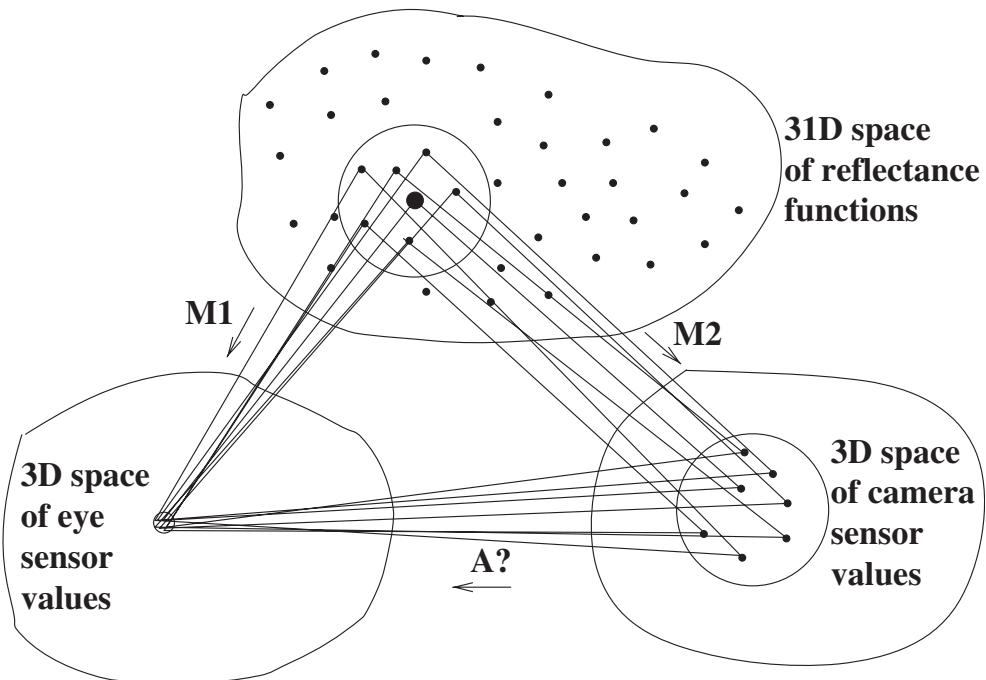


Figure 7.37: Assume that you are given a uniformly coloured object with reflectance function represented by the point marked with the largest filled circle in the 31D space of discretised reflectance functions. You are required to find metamers of this reflectance function. By randomly trying many 31-tuples of values, you deduce that the points marked inside the indicated circle in the 31D space of reflectance functions are mapped to points inside a small sphere of radius δE in the 3D space of human sensor values, such that the human eye cannot distinguish them, although they are mathematically distinct. Mapping M_1 , from the reflectance function space to the human sensor space, is done through equations (7.187), with the knowledge of the sensitivity curves of the sensors of the human eye. All reflectance functions that are mapped via M_1 to the particular sphere of radius δE , in the 3D space of human sensor, constitute the metamers of the original reflectance function we were given. The same points are also mapped via mapping M_2 to the 3D space of the values of the camera we are using for the inspection task. Mapping M_2 is done with the same equations as mapping M_1 , only now the sensitivity curves we use are those of the camera. As mappings M_1 and M_2 are different, the points in the 3D space of the values of the camera are not expected to be particularly close to each other. That is, they are not expected to constitute metamers of the original reflectance function with respect to the camera sensors. We can use the pairs of corresponding points between the two sensor spaces, to work out the elements of a 3×3 matrix A , that will allow the mapping from camera sensor values to eye sensor values, and which will be valid locally in these two spaces. It has been reported that matrix A worked out this way may also be valid globally, as it does not depend on the exact location of the sphere of metamers in the 3D space of human sensor values.

Example B7.29

Estimate the area inside the closed curve in figure 7.38a, by using the Monte-Carlo approach. Assume that you have means to know when a point is inside the curve and when it is outside.

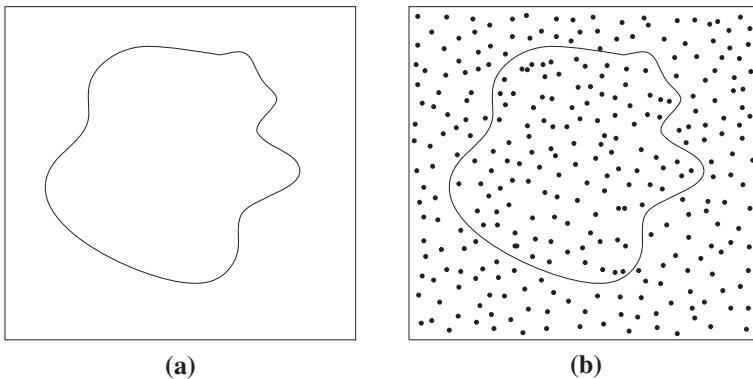


Figure 7.38: (a) We would like to estimate the area enclosed by the curve. (b) Randomly selected points uniformly distributed inside the rectangle.

The shape of the curve cannot be described by an analytic function, which we could integrate to obtain the area enclosed by the curve. We can, however, draw uniformly distributed numbers in the area $(x_{\max} - x_{\min}) \times (y_{\max} - y_{\min})$, which defines the rectangle inside which the curve lies, and test whether the drawn points are inside the curve or outside. In this example, we drew 300 points randomly placed inside the rectangle. We counted 112 points inside the curve. We deduce, therefore, that the area of the curve is $112/300 = 0.37$ of the total area of the rectangle. The area of the rectangle was measured to be equal to 380 unit tiles, so the area of the curve is $380 \times 0.37 = 140.6$ unit tiles. Obviously, the more points we draw, the more accurate our estimate is.

How do we remove noise from multispectral images?

Gaussian noise is usually removed by low pass filtering each band separately. The problem is the removal of impulse noise. As we saw in Chapter 4, impulse noise requires the use of a rank order filter, like the median filter. When, however, the values of the pixels are vectors, it is not straightforward to define a rank of the spectra of the pixels inside a window. Nevertheless, scientists have done that and defined the so called **median vector filter**.

How do we rank vectors?

Consider vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$. Let us assign to each vector \mathbf{x}_i a number d_i , which is the sum of the distances this vector has from all other vectors. The median vector is the one with the minimum sum distance from all other vectors. The distance between any two vectors may be measured by adopting an appropriate norm. For example, one may use the L_1 norm (see page 733). Alternatively, one may use the Euclidean metric. This is a good idea, if one deals with colour images expressed in the *Lab* or *Luv* colour space, as the Euclidean metric expresses the perceived difference between colours in these spaces. This makes the use of this metric, in conjunction with the *Lab* or *Luv* colour space, most appropriate for colour image denoising.

Example 7.30

Work out the vector median of the following set of vectors:

$$\begin{array}{lll} \mathbf{x}_1 = (1, 2, 3) & \mathbf{x}_2 = (0, 1, 3) & \mathbf{x}_3 = (2, 2, 1) \\ \mathbf{x}_4 = (3, 1, 2) & \mathbf{x}_5 = (2, 3, 3) & \mathbf{x}_6 = (1, 1, 0) \\ \mathbf{x}_7 = (3, 3, 1) & \mathbf{x}_8 = (1, 0, 0) & \mathbf{x}_9 = (2, 2, 2) \end{array} \quad (7.189)$$

The sum of distance of vector \mathbf{x}_1 from all other vectors is:

$$\begin{aligned} d_1 &\equiv (\mathbf{x}_1 - \mathbf{x}_2)^2 + (\mathbf{x}_1 - \mathbf{x}_3)^2 + (\mathbf{x}_1 - \mathbf{x}_4)^2 + (\mathbf{x}_1 - \mathbf{x}_5)^2 + (\mathbf{x}_1 - \mathbf{x}_6)^2 \\ &\quad + (\mathbf{x}_1 - \mathbf{x}_7)^2 + (\mathbf{x}_1 - \mathbf{x}_8)^2 + (\mathbf{x}_1 - \mathbf{x}_9)^2 \\ &= (1^2 + 1^2) + (1^2 + 2^2) + (2^2 + 1^2 + 1^2) + (1^2 + 1^2) + (1^2 + 3^2) \\ &\quad + (2^2 + 1^2 + 2^2) + (2^2 + 3^2) + (1^2 + 1^2) \\ &= 49 \end{aligned} \quad (7.190)$$

In a similar way, we compute the d_i values associated with all other vectors:

$$\begin{array}{lll} d_1 = 49 & d_2 = 73 & d_3 = 34 \\ d_4 = 49 & d_5 = 61 & d_6 = 61 \\ d_7 = 64 & d_8 = 82 & d_9 = 31 \end{array} \quad (7.191)$$

According to the d_i values, the median is vector \mathbf{x}_9 .

How do we deal with mixed noise in multispectral images?

If an image is affected by Gaussian as well as impulse noise, then we may use the **α -trimmed vector median filter**: after we rank the vectors inside the smoothing window, we may keep only the $N(1 - \alpha)$ vectors with the smallest distance from the others. We may then compute only from them the mean spectrum that we shall assign to the central pixel of the window.

Example 7.31

For the vectors of example 7.30 calculate the α -trimmed mean vector, for $\alpha = 0.2$.

Since we have $N = 9$ vectors, from the ranked sequence of vectors we must use only the first $9 \times (1 - 0.2) = 7.2 \simeq 7$ vectors. This means that we must ignore vectors \mathbf{x}_2 and \mathbf{x}_8 with the two largest distances. The mean value of the remaining vectors is: $\bar{\mathbf{x}} = (2.0000, 2.0000, 1.7143)$.

Example 7.32

Plate Va shows a 3-band 512×512 image affected by impulse noise while Plate Vb shows the same image affected by impulse and Gaussian noise. Use vector median filtering and α -trimmed vector median filtering, with $\alpha = 0.2$, respectively, to clean these images.

As we are going to use Euclidean distances to compute the median, we first convert the image into Luv space, using the process described on page 742. Then, we adopt a 3×3 window and identify the pixel with the median colour in the Luv space, from the 9 pixels inside the window. The RGB values of that pixel are assigned to the central pixel of the window. The result is shown in Plate Vc.

For the mixed noise, we decide to use a window of size 5×5 , so that after we have kept only the 80% of the pixels with the smallest distances from all others inside the window, we shall have a reasonable number of values to average to reduce the Gaussian noise. For 25 pixels, 80% means that we keep the 20 pixels with the smallest distance from all other pixels. We then average their RGB values and assign the result to the central pixel. We average the RGB values, as opposed to averaging the Luv values, because the noise in the RGB space is uncorrelated and additive Gaussian to a good approximation, while in the Luv space, most likely, it is not, due to the nonlinear transformations used to obtain the Luv values. So, it makes more sense to average values we have good reasons to believe they suffer from zero-mean, additive and uncorrelated noise, rather than to average values we know they suffer from much more complicated noise. The result of using this filter is shown in Plate Vd.

How do we enhance a colour image?

We may convert the *RGB* values into hue and saturation. Then set the saturation of each pixel to a fraction of the maximum possible and work back new *RGB* values, keeping the same hue. The algorithm is as follows.

Step 0: Work out the (r, g, b) values of each pixel from its (R, G, B) values by using (7.86), on page 722. Select a value of γ in the range $[0, 1/\sqrt{6}]$.

Step 1: Work out the so called **value** V_p for each pixel, defined as the maximum of its R, G and B values.

Step 2: Work out the (x_p, y_p) coordinates of the pixel on the Maxwell triangle as given by (7.93) on page 724.

Step 3: Compute the saturation of the pixel using formula (7.165) on page 747.

Step 4: If the saturation of the pixel is below a threshold, leave its values unchanged. This will allow the retention of the white pixels of the image.

Step 5: If the saturation of the pixel is above the threshold, compute the hue of the pixel using formula (7.166) in conjunction with formulae (7.164). Note that *Numerator* means the quantity that is inside the absolute value in the numerator of (7.166) and *Denominator* means the quantity that is inside the absolute value in the denominator of (7.166).

Step 6: Set $\beta = (4\sqrt{3}\gamma + \sqrt{2})/6$ and calculate:

$$\begin{aligned} x_0 &= \begin{cases} \frac{\sqrt{6}\gamma x_p}{1-\sqrt{6}y_p} & \text{if } 0^\circ \leq hue < 120^\circ \\ \frac{(3\sqrt{2}\beta-1)x_p}{3\sqrt{2}x_p+\sqrt{6}y_p-1} & \text{if } 120^\circ \leq hue < 240^\circ \\ \frac{(1-3\sqrt{2}\beta)x_p}{3\sqrt{2}x_p-\sqrt{6}y_p+1} & \text{if } 240^\circ \leq hue < 360^\circ \end{cases} \\ y_0 &= \begin{cases} \frac{1}{\sqrt{6}} - \gamma & \text{if } 0^\circ \leq hue < 120^\circ \\ \sqrt{3} \frac{x_p+\beta(\sqrt{6}y_p-1)}{3\sqrt{2}x_p+\sqrt{6}y_p-1} & \text{if } 120^\circ \leq hue < 240^\circ \\ \sqrt{3} \frac{x_p-\beta(\sqrt{6}y_p-1)}{3\sqrt{2}x_p-\sqrt{6}y_p+1} & \text{if } 240^\circ \leq hue < 360^\circ \end{cases} \end{aligned} \quad (7.192)$$

Some of these formulae are derived in examples 7.34 and 7.35. The others can be derived in a similar way.

Step 7: Calculate new (r, g, b) values for the pixel as follows:

$$\begin{aligned} b_{new} &= \sqrt{\frac{2}{3}}y_0 \\ r_{new} &= \frac{1 - b_{new} - \sqrt{2}x_0}{2} \\ g_{new} &= \frac{1 - b_{new} + \sqrt{2}x_0}{2} \end{aligned} \quad (7.193)$$

For the derivation of these formulae, see example 7.36.

Step 8: Calculate new (R, G, B) values for the pixel as follows:

$$R_{new} = \frac{V_p r_{new}}{c_{max}} \quad G_{new} = \frac{V_p g_{new}}{c_{max}} \quad B_{new} = \frac{V_p b_{new}}{c_{max}} \quad (7.194)$$

where $c_{max} \equiv \max\{r_{new}, g_{new}, b_{new}\}$

Plates VII and VIII show two images and their enhanced versions produced by this algorithm.

Example 7.33

Work out the maximum saturation you can assign to a pixel with hue in the range $[240^\circ, 360^\circ]$ without altering its hue.

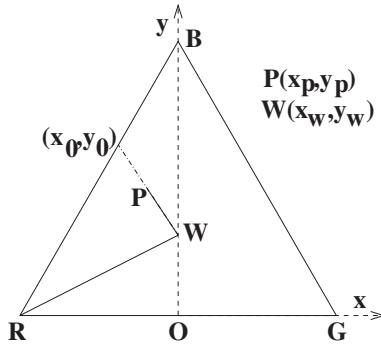


Figure 7.39: The maximum saturation a pixel P can have for fixed hue is given by the colour of point (x_0, y_0) .

Consider a pixel P with coordinates (x_p, y_p) on the Maxwell triangle, as shown in figure 7.39. Since the hue of this pixel is in the range $[240^\circ, 360^\circ]$, the pixel is nearest to the Red-Blue side of the triangle (see example 7.23). We assume that the reference white W is the ideal white, with coordinates $r_w = g_w = b_w = 1/3$, which translate into $(0, 1/\sqrt{6})$ in the coordinate system of the Maxwell triangle, if we use transformation formulae (7.93). Since hue is measured by the angle vector \overline{WP} forms with vector \overline{WR} , and since saturation is measured along line WP , in order to maximise saturation without altering the hue, we must slide point P along line WP , until it reaches the side of the Maxwell triangle, where the saturation is maximal. So, we must work out the intersection point of line WP with line RB . The coordinates of point R are $(-\sqrt{2}/2, 0)$, while those of point B are $(0, \sqrt{3}/2)$. Therefore the equation of line RB is:

$$\begin{aligned} \frac{x - 0}{y - \sqrt{3}/2} &= \frac{-\sqrt{2}/2 - 0}{0 - \sqrt{3}/2} \\ \Rightarrow x &= (y - \sqrt{3}/2) \frac{1}{\sqrt{3}} \\ \Rightarrow x &= \frac{y}{\sqrt{3}} - \frac{1}{\sqrt{2}} \end{aligned} \quad (7.195)$$

The equation of the WP line is:

$$\begin{aligned} \frac{x - 0}{y - 1/\sqrt{6}} &= \frac{x_p - 0}{y_p - 1/\sqrt{6}} \\ \Rightarrow x &= \left(y - \frac{1}{\sqrt{6}} \right) \frac{x_p}{y_p - 1/\sqrt{6}} \end{aligned} \quad (7.196)$$

We can combine these two equations to solve for (x_0, y_0) which is the point where the two lines intersect:

$$\begin{aligned}
 \frac{y_0}{\sqrt{3}} - \frac{1}{\sqrt{2}} &= \left(y_0 - \frac{1}{\sqrt{6}}\right) \frac{x_p}{y_p - 1/\sqrt{6}} \\
 \Rightarrow y_0 \left(\frac{1}{\sqrt{3}} - \frac{x_p}{y_p - 1/\sqrt{6}}\right) &= \frac{1}{\sqrt{2}} - \frac{x_p}{\sqrt{6}y_p - 1} \\
 \Rightarrow y_0 &= \frac{\frac{1}{\sqrt{2}} - \frac{x_p}{\sqrt{6}y_p - 1}}{\frac{1}{\sqrt{3}} - \frac{\sqrt{6}x_p}{\sqrt{6}y_p - 1}} \\
 \Rightarrow y_0 &= \frac{\sqrt{3}}{2} \frac{\sqrt{6}y_p - 1 - \sqrt{2}x_p}{\sqrt{6}y_p - 1 - 3\sqrt{2}x_p} \tag{7.197}
 \end{aligned}$$

We may then set $y = y_0$ in (7.195) to work out the value of x_0 :

$$x_0 = \frac{-2x_p}{3\sqrt{2}x_p - \sqrt{6}y_p + 1} \tag{7.198}$$

The new saturation of pixel P will be:

$$\text{New_Saturation} = \sqrt{x_0^2 + \left(y_0 - \frac{1}{\sqrt{6}}\right)^2} \tag{7.199}$$

Example 7.34

Work out a value for increased saturation you may assign to the pixel represented by point P in figure 7.39, without altering its hue.

This problem is similar to the problem of example 7.33, but now we are seeking the intersection of line WP with a line parallel to line RB and closer to P (see figure 7.40). The equation of line RB is (7.195). A line parallel to it has equation

$$x = \frac{y}{\sqrt{3}} - \beta \tag{7.200}$$

where β is some positive constant. Note that if $\beta = \frac{1}{\sqrt{2}}$, this line is the RB line. Obviously, the limiting case is this line to pass through point P . In that case, parameter

β takes the value:

$$\begin{aligned} x_p &= \frac{y_p}{\sqrt{3}} - \beta \\ \Rightarrow \beta &= \frac{y_p}{\sqrt{3}} - x_p \end{aligned} \quad (7.201)$$

So, the range of allowable values of beta is $\frac{y_p}{\sqrt{3}} - x_p < \beta \leq \frac{1}{\sqrt{2}}$.

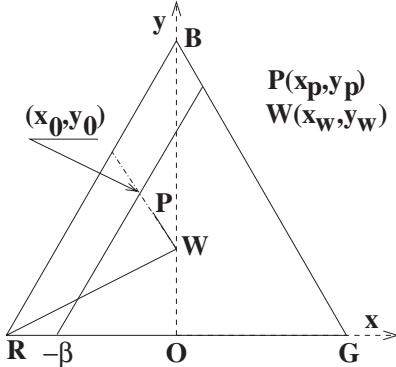


Figure 7.40: The saturation of pixel P can be increased without altering its hue, if pixel P is given the colour of point (x_0, y_0) , which is the intersection of line WP and a line parallel to RB .

We can now combine equations (7.200) and (7.196) to work out their intersection point:

$$\begin{aligned} \frac{y_0}{\sqrt{3}} - \beta &= \left(y_0 - \frac{1}{\sqrt{6}} \right) \frac{x_p}{y_p - 1/\sqrt{6}} \\ \Rightarrow y_0 \left(\frac{1}{\sqrt{3}} - \frac{x_p}{y_p - 1/\sqrt{6}} \right) &= \beta - \frac{x_p}{\sqrt{6}y_p - 1} \\ \Rightarrow y_0 &= \frac{\beta - \frac{x_p}{\sqrt{6}y_p - 1}}{\frac{1}{\sqrt{3}} - \frac{\sqrt{6}x_p}{\sqrt{6}y_p - 1}} \\ y_0 &= \sqrt{3} \frac{\beta(\sqrt{6}y_p - 1) - x_p}{\sqrt{6}y_p - 1 - 3\sqrt{2}x_p} \end{aligned} \quad (7.202)$$

We may then set $y = y_0$ in (7.195) to work out the value of x_0 :

$$x_0 = \frac{(1 - 3\sqrt{2}\beta)x_p}{3\sqrt{2}x_p - \sqrt{6}y_p + 1} \quad (7.203)$$

The new saturation of pixel P can be computed using (7.199).

Example 7.35

We decide to saturate all pixels of an image on the perimeter of the grey triangle in figure 7.41, characterised by the value of parameter γ as defined in the same figure. Work out the relationship between β of example 7.34 and γ .

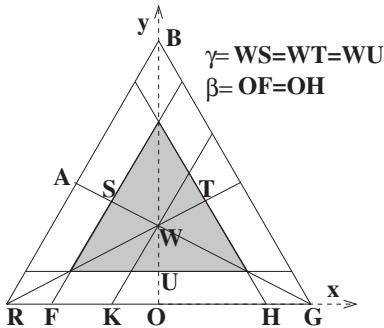


Figure 7.41: When we do not want extreme saturation for the enhanced image pixels, we may assign to each pixel the saturation defined by the perimeter of the grey triangle, characterised by parameter γ .

Remembering that $RG = \sqrt{2}$ and using the property of equilateral triangles that $GK/GR = GW/GA = 2/3$, we work out that when the pixel has hue in the range $[240^\circ, 360^\circ]$:

$$\beta = \frac{4\sqrt{3}\gamma + \sqrt{2}}{6} \quad (7.204)$$

Example 7.36

Compute the enhanced RGB values for the pixel of example 7.33.

We remember that the (x, y) coordinates of a pixel in the Maxwell triangle are expressed in terms of the (r, g, b) coordinates of the pixel as $((g - r)/\sqrt{2}, \sqrt{3}b/\sqrt{2})$. We can see immediately that

$$b = \sqrt{\frac{2}{3}}y_0 \quad (7.205)$$

We also remember that $r + g + b = 1 \Rightarrow r + g = 1 - b$. This, in combination with $g - r = \sqrt{2}x_0$, yields:

$$g = \frac{1 - b + \sqrt{2}x_0}{2} \quad r = \frac{1 - b - \sqrt{2}x_0}{2} \quad (7.206)$$

How do we restore multispectral images?

In general, blurring, particularly motion blurring, is expected to affect all channels the same. So, the restoration filter we deduce for one channel is applicable to all channels and it should be used to restore each channel separately, unless we have reasons to believe that the channels have different levels of noise, in which case the component of the filter (Wiener filter (page 429), or constraint matrix inversion filter (page 456)) that is designed to deal with noise will have to be different for the different channels.

How do we compress colour images?

The human visual system is much more sensitive to variations of brightness than to variations in colour. So, image compression schemes use more bits to encode the luminance component of the colour system and fewer bits to represent the chroma component (eg (u, v) or (a, b)). This, of course, applies to colour systems that separate the luminance component from the two chroma components. For multispectral images and for *RGB* colour images, where all components contain a mixture of luminance and chroma information, all channels are treated the same.

How do we segment multispectral images?

The most commonly used method for segmenting multispectral images is to treat the values of each pixel in the different bands as the features of the pixel. In other words, for an L -band image we consider an L -dimensional space and we measure the value of a pixel in each band along one of the axes in this space. This is the **spectral histogram** of the image. For a 3-band image this is known as the **colour histogram** of the image.

Pixels with similar spectra are expected to cluster together. If we know a priori the number of different spectra we expect to have in the image, ie if we know a priori the number of clusters in the L -dimensional spectral space, then we can identify the clusters by using simple k -means clustering.

Alternatively, if we wish to take into consideration the locality of the pixels too, we may use the generalised mean shift algorithm we saw in Chapter 6, on page 574. The generalisation of this algorithm for multispectral images is trivial: each pixel instead of being characterised by three numbers (its grey value and its two spatial coordinates) is characterised by $L + 2$ numbers, namely its values in the L bands and its two spatial coordinates. The values of these $L + 2$ parameters are updated according to the mean shift algorithm.

How do we apply k -means clustering in practice?

Step 0: Decide upon the number k of clusters you will create.

Step 1: Select at random k vectors that will be the centres of the clusters you will create.

Step 2: For each vector (ie each pixel) identify the nearest cluster centre and assign the pixel to that cluster.

Step 3: After all pixels have been assigned to clusters, recalculate the cluster centres as the average vectors of the pixels that belong to each cluster and go to Step 2.

The algorithm stops when at some iteration step no pixel changes cluster. At the end, all pixels that belong to the same cluster may be assigned the spectrum of the centre of the cluster. As the centre of the cluster may not coincide with a pixel, the colour of the pixel that is nearest to the centre is selected. This algorithm has the problem that it often converges to bad clusters, as it strongly depends on the initial choice of cluster centres. A good strategy is to run the algorithm several times, with different initial guesses of the cluster centres each time, and at the end to select the cluster assignment that minimises the sum of the distances of all pixels from their corresponding clusters.

If the input image is a 3-band image and the outcome of the segmentation is to produce an image that is more pleasing to the human eye, or the segmentation is aimed at reducing the number of colours in the image to achieve image compression, then it is a good idea for this algorithm to be applied in the *Lab* or the *Luv* colour spaces, where the Euclidean metric reflects human colour perception. If, however, we are not interested in correct colour assignment or if we are dealing with multispectral images, the algorithm may be applied directly to the raw spectral values.

Example 7.37

Segment the image of Plate IXa using k -means clustering and mean shift clustering. At the end assign to each pixel the mean colour of the cluster to which it belongs.

As we use the Euclidean distance to compute colour differences, before we perform any clustering, we have to convert the image into either the *Lab* or the *Luv* space. We decided to work in the *Luv* space. Assuming that the RGB values of this image are sRGB, we must use the procedure described on page 742 for this transformation.

The k -means clustering was run with 10 different random initialisations of the cluster centres and at the end the clustering with the minimum total energy was selected. (The energy was computed as the sum of the square distances of all pixels from their corresponding cluster centres). The number of clusters was selected to be $k = 10$. The result is shown in Plate IXb. The average RGB values of the pixels that belong to a cluster were assigned to all the pixels of that cluster.

For the mean shift algorithm, we must select values for the scaling parameters. We selected $h_x = h_y = 15$, because in previous experiments presented in this book, and for images of this size, these values gave good results. A quick check of the range of values, of the *Luv* representation of the image, revealed that the ranges of values for the three components were: $L \in [2, 100]$, $u \in [-42, 117]$ and $v \in [-77, 72]$. These ranges are of roughly the same order of magnitude, and we selected to use $h_L = h_u = h_v = 15$. The result of the mean shift algorithm run in the static feature space is shown in Plate IXc. The trajectory of each pixel in the static feature space was assumed to have converged,

when the square difference between two successive points was less than 0.0001. For comparison, the mean shift algorithm was run assuming CIE RGB values for the original image. In this case, $L \in [71, 266]$, $u \in [-191, 424]$ and $v \in [-182, 225]$, so we selected $h_L = h_u = h_v = 25$ and again $h_x = h_y = 15$. The result is shown in Plate IXd. The trajectory convergence threshold was again 0.0001.

How do we extract the edges of multispectral images?

Often, edge detection is performed to the average band of a multispectral image. This is a quick and easy way to proceed, but it is not the best.

Edge detection is best performed in the first principal component of a multispectral image, as that has the maximum contrast over all original image bands.

If, however, the calculation of the first principal component is to be avoided, one may apply gradient magnitude estimation to each band separately and then, for each pixel, select the maximum gradient value it has over all bands. This way, a composite gradient map is created, which contains the maximum contrast a pixel has with its neighbours over all bands. The subsequent processing of this gradient map may then proceed as for the ordinary gradient map of a grey image.

Example 7.38

Extract the edges of a 64×64 subimage of image IXa, using its first principal component, its average grey band and the composite gradient map.

The red band of the image, its average grey band (obtained by averaging the grey values of the three bands) and its first principal component are shown in figure 7.42. The Sobel edge detector was applied directly to the images in 7.42b and 7.42c. The results, superimposed on the original colour image, are shown in Plates Xa and Xb. To work out the composite gradient map, we computed the gradient magnitude in each band separately, using the Sobel filters, and accepted as gradient magnitude the maximum value of the three. The output of this edge detection is shown in Plate Xc. In all three cases, non-maxima suppression was performed using only the horizontal and vertical directions. In all three cases, the gradient magnitude was thresholded at 0.3 times the maximum value, to remove weak edges.

From Plate X, we can see that the use of the first principal component detected correctly some edges that were missed by using the average band, while the use of all three bands allowed the detection of some edges missed by the previous method, since they had very similar brightness in the grey versions of the image.

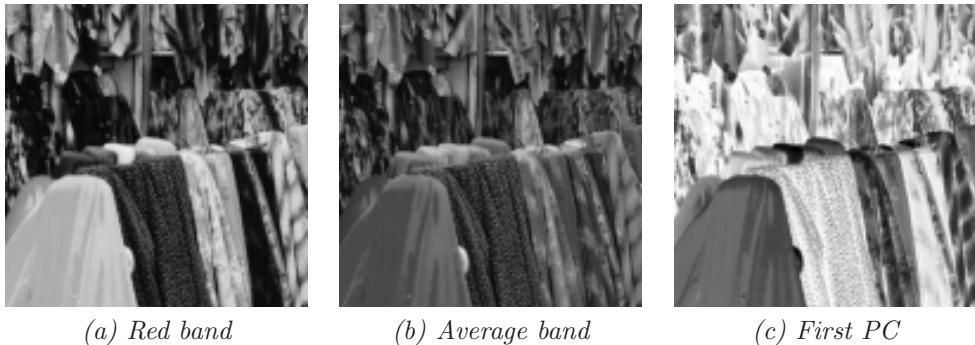


Figure 7.42: The red band of a subimage of Plate IXa, its average band and its first principal component.

What is the “take home” message of this chapter?

This chapter first of all stressed the difference between colour image processing and multispectral image processing. If the outcome of image processing is to be an image to be viewed by humans, or if the outcome has to be identical with the outcome a human would produce by taking into consideration the colour of the image, then one has to apply image processing taking into consideration the way humans see colours. Examples of such applications are image compression and visual industrial inspection of certain products, like cosmetics, ceramic tiles, etc.

In all other cases a colour image may be treated like any other multispectral image. For some processes, a grey version of the image may be adequate, and in such a case the first principal component may be used, as it contains most of the image information. In cases where spectral information is important, one may use the generalisations for multispectral images of several of the methods that were discussed in the previous chapters. The area where spectral information is most useful is that of image segmentation, where the spectral signature of a pixel may be used for segmentation and recognition, ie classification. Indeed, the spectral signature of a pixel may be used to work out what type of object this pixel comes from. This is less so in robotic vision, where colour is not really a strong indicator of object class (eg there are green and red and brown doors), but more so in remote sensing, where the spectral signature of a pixel may be used to identify its class (eg the spectral signature of grass is different from the spectral signature of concrete). In particular, hyperspectral images, made up from hundreds of bands, may be used to identify different minerals present in the soil. In the case of robotic vision, colour is hardly useful for generic class identification, but it may be very useful in tracking a particular object. For example, one may not use colour to identify cars, but a particular car may be tracked in a series of images with the aid of its specific colour. In those cases, a form of spectral constancy is necessary, ie methodology that allows the computer to compensate for the variation of the different illuminants under which the same object is seen.

Bibliographical notes

This book is not aimed at covering the most recently proposed algorithms. Conference proceedings and the latest journal issues are more appropriate for that. This book is about methods that have already been established and earned their worth in the recent years. So, we do not include here a comprehensive list of papers on image processing—that would have been impossible anyway, due to the shear volume of the output of the research community. We give, however, the key references that helped in shaping this book. This said, the authors have benefitted enormously from hundreds of publications over the years. The distilled information is very difficult to be attributed to specific books and papers.

The references will be commented chapter by chapter, where this is relevant, but there are some key books that helped us throughout. References [1], [24] and [50] proved to be invaluable sources of functions and formulae. Papoulis' book [47] is irreplaceable for anybody who wants to understand stochastic processes. For Fourier and Hilbert transforms, Bracewell's book is a classic [9] and excellent worked out examples can be found in [29]. Other classical books were very helpful too [61; 65; 62; 30; 23]. In certain places, reference to Book II is made. This is reference [55], where detailed information on many topics, which were simply touched upon here, like, for example, wavelets and mathematical morphology, may be found.

Chapter 2: For Walsh functions we relied heavily on [2]. The book contains some minor mistakes, but it is the only comprehensive book on Walsh functions and it is generally well written.

Chapter 3: For z -transforms and the Butterworth filter [20] is a very readable book. For ICA we relied on [31], [32] and [35] and on many useful discussions with Nikos Mitianoudis, where we sorted out several inconsistencies and misunderstandings in the published literature.

Chapter 4: Examples 4.10 and 4.12, showing that noise can be white but not independent, were devised by Mike Brookes. For weighted median and mode filtering our source is [26]. For the retinex algorithm useful references were [34; 40; 41; 63]. Unsharp masking was first developed in Germany in the 1930s, but the first paper in the digital image processing community was [68]. The idea of pairwise image enhancement came from [33] and [36]. Toboggan enhancement was proposed in [17]. Details about the algorithm of anisotropic diffusion were found in [13; 49; 42].

Chapter 5: Example 5.66 on lens distortion is from [10], while [3] was found to be a very useful tutorial on radial lens distortion models. The simulated annealing algorithm is based on the classical paper by Geman and Geman [21]. We also found enormously helpful [43] and [48]. The Renormalisation Group transform was introduced to image processing by Gidas [22]. A tutorial can be found in [58] while an application to signal processing can be found in [66]. Details on the super-coupling transform can be found in [4].

Chapter 6: The theory of Otsu's thresholding was proposed in [45]. Modifications and

refinements can be found in [38] and [59]. The ideas for nonlinear edge operators come from [25] and [60] and more recently from [54]. The theory of linear step edge operators and hysteresis thresholding come from the classical work by Canny [11; 12] and from [70]. The extension to ramp edges was proposed in [56] and [53], while the adaptation of the theory for lines (page 609) was proposed in [51] and [57]. These works make use of the results in [64], which is a must for anybody who wishes to learn about filtering of noise. How to select optimally the two thresholds for hysteresis thresholding was studied in [27] and it can be found in tutorial [52] in a distilled form. The working out of the Sobel filter weights, in Boxes 6.5 and 6.6, was published first in [37]. The mean shift algorithm is from [14] and [15]. The normalised graph cuts algorithm was proposed in [69]. Good sources on phase congruency are [39] and [46], while the monogenic signal was introduced in [18].

Chapter 7: The algorithm on spectral constancy is from [19]. The *sRGB* system was proposed in [71]. A useful book for colour transformations is [67]. Reference [72] is a delightful book to read, if one wants to understand the psychophysics of colour vision, while the “bible” of the physics of colour is undoubtedly [73]. It is from [73] that the entries of tables 7.1, 7.3, 7.4 and 7.12 were taken. For linear spectral unmixing with robust or second order statistics one should consult [5; 6]. For linear spectral unmixing with negative and superunity mixing proportions, when the reference spectra are mixtures themselves, the reader is referred to [16]. It was possible to reproduce the colours of the Macbeth colour checker thanks to the information supplied in [28]. The work reported on how to deal with metamers in industrial inspection came from [7; 8]. The information on opponent colour space and the sCIELAB transformation is from [74]. This transformation was used for the segmentation of colour images in [44].

References

- [1] M Abramowitz and I A Stegun (eds), 1970. *Handbook of Mathematical Functions*, Dover Publications, New York, ISBN 486 61272 4, Library of Congress Catalogue 65–12253.
- [2] K G Beauchamp, 1975. *Walsh Functions and their Applications*, Academic Press, ISBN 0-12-084050-2.
- [3] A Bisognianni, “Measurements and correction of geometric distortion”, Stanford University, http://scien.stanford.edu/class/psych221/projects/07/geometric_distortion/project.htm
- [4] M Bober, M Petrou and J Kittler, 1998. “Non-linear motion estimation using the super-coupling approach”. PAMI-20:550–555.
- [5] P Bosdogianni, M Petrou and J Kittler, 1997. “Mixed Pixel Classification with Robust Statistics”. TGRS-35:551–559.
- [6] P Bosdogianni, M Petrou and J Kittler, 1997. “Mixture models with higher order moments”. TGRS-35:341–353.
- [7] C Boukouvalas and M Petrou, 1998. “Perceptual Correction for Colour Grading using Sensor Transformations and Metameric Data”. Machine Vision and Applications, 11:96–104.
- [8] C Boukouvalas and M Petrou, 2000. “Perceptual Correction for Colour Grading of Random Textures”. Machine Vision and Applications, 12:129–136.
- [9] R N Bracewell, 1978. *The Fourier Transform and its Applications*, McGraw Hill, ISBN 0-07-007013-X.
- [10] F M Candocia, “A scale-preserving lens distortion model and its application to image registration”, Florida Conference on Recent Advances in Robotics, FCRAR, Miami, May 25–26, 2006.
- [11] J Canny, 1983. *Finding edges and lines in images*. MIT AI Lab Technical Report 720.
- [12] J Canny, 1986. “A computational approach to edge detection”. PAMI-8:679–698.
- [13] F Catte, P-L Lions, J-M Morel and T Coll, 1992. “Image selective smoothing and edge detection by non-linear diffusion”. SIAM Journal of Numerical Analysis, 29:182–193.
- [14] Y Cheng, 1995. “Mean shift, mode seeking and clustering”. PAMI-17:21–46.
- [15] D Comaniciu and P Meer, 2002. “Mean shift: a robust approach toward feature space analysis”. PAMI-24:603–619.
- [16] O Duran and M Petrou, 2009. “Spectral unmixing with negative and superunity abundances for subpixel anomaly detection”. TGRS-6:152–156.

- [17] J Fairfield, 1992. "Toboggan contrast enhancement". Proceedings of the SPIE Conference Applications of Artificial Intelligence X: Machine Vision and Robotics, K W Bowyer (ed), 22–24 April, Orlando, 1708:282–292.
- [18] M Felsberg and G Sommer, 2001. "The monogenic signal". TIP-49:3136–3144.
- [19] G D Finlayson, B Schiele and J L Crowley, 1998. "Comprehensive colour image normalisation", ECCV, I:475–490.
- [20] R A Gabel and R A Roberts, 1987. *Signals and Linear Systems*, J Wiley, ISBN 0-471-83821-7.
- [21] S Geman and D Geman, 1984. "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images". PAMI-6:721–741.
- [22] B Gidas, 1989. "A Renormalisation Group Approach to Image Processing Problems". PAMI-11:164–180.
- [23] R C Gonzalez and R E Woods, 1992. *Digital Image Processing*, Addison Wesley, ISBN 0-201-50803-6.
- [24] I S Gradshteyn and I M Ryzhik, 1980. *Table of Integrals, Series and Products*, Academic Press, ISBN 0-12-294760-6.
- [25] J Graham and C J Taylor, 1988. "Boundary cue operators for model-based image processing". Proceedings of the fourth Alvey Vision Conference, AVC88, Manchester, 31 August–2 September, 59–64.
- [26] L D Griffin, 2000. "Mean, median and mode filtering of images". Proceedings of the Royal Society of London, 456:2995–3004.
- [27] E R Hancock and J Kittler, 1991. "Adaptive Estimation of Hysteresis Thresholds", CVPR, 196–201.
- [28] D Holloway, 2004. <http://www.mambo.net/cgi-bin/TempProcessor/view/113>
- [29] H P Hsu, 1970. *Fourier Analysis*, Simon and Schuster, New York.
- [30] T S Huang (ed), 1979. *Picture Processing and Digital Filtering*, Topics in Applied Physics, Vol 6, Springer-Verlag, ISBN 0-387-09339-7.
- [31] A Hyvärinen, 1998. "New approximations of differential entropy for independent component analysis and projection pursuit". Advances in Neural Information Processing Systems, 10:272–279.
- [32] A Hyvärinen and E Oja, 2000. "Independent Component Analysis: Algorithms and Applications". Neural Networks, 13:411–430.
- [33] T Jen, B Hsieh and S Wang, 2005. "Image contrast enhancement based on intensity pair distribution". International Conference on Image Processing, 1:913-916.
- [34] D J Jobson, Z Rahman and G A Woodell, 1997. "Properties and performance of a center/surround retinex". TIP-6:451–462.
- [35] M Jones and R Sibson, 1987. "What is projection pursuit?". Journal of the Royal Statistical Society, Series A, 150:1–36.
- [36] M H Kabir, M Abdullah-Al-Wadud and O Chae, 2006. "Image contrast enhancement based on block-wise intensity pair distribution with two expansion forces". 11th Iberoamerican Congress on Pattern Recognition, CIARP2006, November 14-17, Cancun, Mexico, pp 247–256.

- [37] J Kittler, 1983. "On the accuracy of the Sobel edge detector". *Image and Vision Computing*, 1:37–42.
- [38] J Kittler and J Illingworth, 1985. "On threshold selection using clustering criteria". *SMC-15*:652–655.
- [39] P Kovesi, 1997. "Invariant Measures of Image Features From Phase Information", PhD thesis, University of Western Australia,
<http://www.cs.uwa.edu.au/pub/robvis/theses/PeterKovesi/>.
- [40] E Land, 1986. "An alternative technique for the computation of the designator in the retinex theory of colour vision". *Proceedings of the National Academy of Science*, 83:3078–3080.
- [41] Y Li, R He, C Xu, C Hou, Y Sun, L Guo, L Rao and W Yan, 2008. "Retinex enhancement of infrared images", 30th Annual International IEEE EMBS Conference, Vancouver, British Columbia, Canada, August 20–24, 2189–2192.
- [42] B Mackiewich, 1995. *Intracranial boundary detection and radio frequency correction in MRI*, PhD thesis, Simon Fraser University, Canada,
<http://www.cs.sfu.ca/~stella/papers/blairthesis/main/main.htm>
- [43] K V Mardia and G K Kanji, editors, 1993. *Advances in Applied Statistics*. Carfax Publishing Company, ISBN 0-902879-25-1.
- [44] M Mirmehdi and M Petrou, 2000. "Segmentation of colour textures". *PAMI-22*:142–159.
- [45] N Otsu, 1979. "A threshold selection method from gray level histograms". *SMC-9*:62–66.
- [46] R Owens, 1997. "Feature detection via phase congruency".
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT2/node3.html
- [47] A Papoulis, 1965. *Probability, Random Variables and Stochastic Processes*, McGraw-Hill, Library of Congress Catalogue 64-22956.
- [48] K S Pedersen, 2004. "From Bayes to pdes, Part II: Mumford-Shah, Geman-Geman and MRFs", <http://www.itu.dk/courses/MBAG/E2004/>
- [49] P Perona and J Malik, 1990. "Scale-space and edge detection using anisotropic diffusion". *PAMI-12*:629–639.
- [50] S Persidis, 2007. *Mathematical Handbook*. ISBN 978-960-7610-13-3, ESPI, Athens.
- [51] M Petrou, 1993. "Optimal convolution filters and an algorithm for the detection of wide linear features." *IEE Proceedings I, Vision, Signal and Image Processing*, 140:331–339.
- [52] M Petrou, 1994. "The Differentiating Filter Approach to Edge Detection". *Advances in Electronics and Electron Physics*, 88:297–345.
- [53] M Petrou, 1995. "Separable 2D filters for the detection of ramp edges". *IEE Proceedings Vision, Image and Signal Processing*, 142:228–231.
- [54] M Petrou, V A Kovalev and J R Reichenbach, 2006. "Three dimensional nonlinear invisible boundary detection". *TIP-15*:3020–3032.
- [55] M Petrou and P Garcia Sevilla, 2006. *Image Processing, dealing with texture*. John Wiley & Sons, Ltd, ISBN-13-978-0-470-02628-1.
- [56] M Petrou and J Kittler, 1991. "Optimal edge detectors for ramp edges". *PAMI-13*:483–491.

- [57] M Petrou and A Kolomvass, 1992. "The recursive implementation of the optimal filter for the detection of roof edges and thin lines". *Signal Processing VI, Theory and Applications*, 1489–1492.
- [58] M Petrou, 1995. "Accelerated optimisation in Image Processing via the Renormalisation Group Transformation". *Complex Stochastic Systems and Engineering*, Ed: D M Titterington, Clarendon Press, ISBN 0 19 853485 X, 105–120.
- [59] M Petrou and A Matrucceli, 1998. "On the stability of thresholding SAR images". *Pattern Recognition*, 31:1791–1796.
- [60] I Pitas and A N Venetsanopoulos, 1986. "Non-linear order statistic filters for image filtering and edge detection". *Signal Processing*, 10:395–413.
- [61] W K Pratt, 1978. *Digital Image Processing*, John Wiley & Sons, Inc, ISBN 0-471-01888-0.
- [62] R M Pringle and A A Rayner, 1971. *Generalised inverse matrices with applications to Statistics*, Being number twenty eight of Griffin's Statistical monographs and courses, edited by A Stuart, ISBN 0-85264-181-8.
- [63] E Provenzi, M Fierro, A Rizzi, L De Carli, D Gadia and D Marini, 2007. "Random spray retinex: a new retinex implementation to investigate the local properties of the model". *TIP-16*:162–171.
- [64] S O Rice, 1945. "Mathematical analysis of random noise". *Bell Systems Tech. J.*, 24:46–156.
- [65] A Rosenfeld and A C Kak, 1982. *Digital Picture Processing*, Academic Press, ISBN 0-12-597301-2.
- [66] M Samonas and M Petrou, 1999. "Multiresolution Restoration of Medical Signals using the Renormalisation Group and the Super-coupling Transforms". *Computers in Biology and Medicine*, 29:191–206.
- [67] S J Sangwine and R E N Horne, (eds), 1998. *The Colour Image Processing Handbook*. Chapman and Hall, ISBN 0-412-80620-7.
- [68] W F Schreiber, 1970. "Wirephoto Quality Improvement by Unsharp Masking". *Pattern Recognition*, 2:117–121.
- [69] J Shi and J Malik, 2000. "Normalised cuts and image segmentation". *PAMI*-22:888–905.
- [70] L A Spacek, 1986. "Edge detection and motion detection". *Image and Vision Computing*, 4:43–53.
- [71] M Stokes, M Anderson, S Chandrasekhar and R Motta, 1996. "A standard default colour space for the Internet-sRGB",
<http://www.w3.org/Graphics/Color/sRGB.html>
- [72] B A Wandell, 1995. *Foundations of Vision*. Sinauer Associates, Inc. Publishers, Sunderland, Massachusetts, ISBN 0-87893-853-2.
- [73] G Wyszecki and W S Stiles, 1982. *Color Science*. ISBN 0-471-02106-7, John Wiley & Sons, Inc, New York.
- [74] X Zhang and B A Wandell, 1996. "A spatial extension of CIELAB for digital color image reproduction". SID journal.
<http://white.stanford.edu/~brian/scielab/introduction.html>

Index

- D_{55} , 705, 706
 D_{65} , 705, 706
 for *sRGB*, 732
 D_{75} , 705, 706
 α -trimmed filter, 760
 χ^2 -test, 237, 239
k-means clustering, 767
z-transform, 294, 301
3D geometry, 721
- algorithm
 k-means clustering, 767
 anisotropic diffusion, 349
 averaging angles, 629
 colour constancy, 687
 colour enhancement, 761
 constrained matrix inversion, 464
 DFT, 102
 edge detection, 602
 edge detection as zero crossings, 623
 endmember spectra by ICA, 697
 endmember spectra by PCA, 696
 expectation maximisation, 537
 Gauss-Seidel method, 485
 gradient descent, 487
 Haar transform, 89
 histogram equalisation, 370
 histogram equalisation with random additions, 372
 histogram hyperbolisation, 373
 histogram hyperbolisation with random additions, 374
 Hough, 520
 hysteresis edge linking, 621
ICA, 274
inverse filtering, 410
Jacobi's method, 482
- K-L transform, 214, 215
knee, 549
linear spectral unmixing, 692, 696, 697
local energy, 651
local variance, 339
metamer mapping, 757
monogenic signal, 660
normalised graph cuts, 589
pairwise image enhancement, 378
PCA, 674
PCA for material spectra, 709
power method, 682
reconstruction with unknown degradation matrix, 489
retinex, 360
Riesz transform, 660
simulated annealing, 539
simulated annealing with Gibbs sampler, 507
simulated annealing with Metropolis sampler, 506
spectral constancy, 687
sRGB to XYZ, 743
standard illuminant spectra, 705
successive doubling, 124
SVD, 69
thresholding a unimodal histogram, 549
toboggan enhancement, 387, 389
unsharp masking, 357
unsharp masking, adaptive, 358
Walsh transform, 91
watershed, 566
whirl transform, 469, 472
whirl transform, scrambling matrix for, 473
Wiener filtering, 431
alychné, 725

- ambient light, 351
- analytic signal, 659
- anisotropic diffusion, 337, 342, 348
 - algorithm for, 349
- approximation of image
 - by DFT, 103, 176
 - by EDCT, 148, 176
 - by EDST, 165, 166, 176
 - by Haar transform, 79, 89, 176
 - by Hadamard transform, 91, 176
 - by K-L, 220
 - by ODCT, 156, 176
 - by ODST, 175, 176
 - by SVD, 63, 176
 - by Walsh transform, 91, 176
 - error of
 - by DFT, 103, 176
 - by EDCT, 148, 176
 - by EDST, 165, 166, 176
 - by Haar transform, 79, 89, 176
 - by Hadamard transform, 91, 176
 - by K-L, 220
 - by ODCT, 156, 176
 - by ODST, 175, 176
 - by SVD, 63, 176
 - by Walsh transform, 91, 176
- astronomical image, 402
- attribute, 551
- autocorrelation function, 211, 216, 313, 424
 - ensemble, 190
 - spatial, 196, 325
- autocorrelation matrix, 211, 216
 - spatial, 215
- autocovariance, 190
- autocovariance matrix, 201
- average angle, 629
- axis of symmetry, 202
- band, 2
- barrel distortion, 513
- basis images, 75
 - DFT, 101
 - EDCT, 146
 - EDST, 163
 - Haar, 80, 93
 - Hadamard, 88
 - ICA, 281
 - independent, 292
 - K-L, 221
 - ODCT, 154
 - ODST, 173
 - orthogonal, 292
 - uncorrelated, 292
 - Walsh, 88
- basis signals in ICA, 283
- Bessel function, 298, 300
 - approximation of, 300
- binary order, 86
- blind source separation, 289
- block circulant matrix, 437
 - diagonalisation of, 445
 - inversion of, 438
- camera
 - colour, sensor arrangement of, 3
 - distortion due to lens, 513
 - multispectral, 2
 - point spread function of, 12
- Canny criteria, 606
- Canny filter, 608
- central limit theorem, 235, 264
- central moments, 239
- Chebyshev norm, 733
- checkerboard effect, 7
- chroma, 738
- chromaticity diagram, 718
 - definition of, 720
- CIE, 704, 717
- circulant matrix, 438
- city block metric, 733
- class of a pixel, 528
- cocktail party problem, 234, 264, 283
- colour
 - calibration between different viewers, 730
 - comparison of, 733
 - definition of, 700
 - emulation of spatial dependence of, 752
 - enhancement, 761
 - factors influencing its perception, 740
 - spatial context of, 740
 - temporal context of, 740
- colour camera, 3
- colour constancy, 713, 742
- colour histogram, 767

- colour matching experiments, 715
colour space
 Lab, 734, 742
 Lab, errors in, 745
 Lab, inverse transform, 736
 Luv, 734, 742
 Luv, definition of, 734
 Luv, errors in, 743
 Luv, inverse transform, 735
 2D, 718
 3D, 718
 opponent, 741
 perceptually uniform, 734, 742
colour system
 CIE RGB, 717
 CIE RGB, chromaticity diagram for, 724
 CIE RGB, colour matching functions of, 718
 CIE RGB, definition of, 717
 CIE RGB, definition of alychne for, 726
 CIE RGB, tristimulus values of, 718
 definition of, 715
 S-CIELAB, 753
 sRGB, 717, 742
 sRGB, definition of, 732
 transformation CIE RGB to XYZ, 727
 transformation from *Lab* to RGB, 736
 transformation from *Luv* to RGB, 735
 transformation from sRGB to *Lab*, 742
 transformation from sRGB to *Luv*, 742
 tristimulus values of, 715
XYZ, 717, 718
XYZ, chromaticity diagram of, 728
XYZ, definition of, 726
XYZ, imaginary primaries of, 728
XYZ, tristimulus values of, 727
complete set of functions, 72
cones, 715, 741
confidence of χ^2 -test, 239
conjugate transpose, 50
constrained matrix inversion, 436
 algorithm for, 464
 comparison with Wiener filtering, 462
 filter derivation for, 459
 filter for, 456
 inhomogeneous degradation, 477
contrast, inhomogeneous, 375
convolution, 632
convolution theorem, 105, 108
convolution theorem, for *z*-transform, 302
cooling schedule, 506
correlated colour temperature, 705
cosine transform
 even symmetric, 137, 138
 basis images of, 146
 inverse of 1D, 143
 inverse of 2D, 145
 odd symmetric, 137, 149
 basis images of, 154
 inverse of 1D, 152
 inverse of 2D, 154
cost function, 477, 481
 for image restoration, 499
minimisation of, 503
minimisation of quadratic, 487
 minimum of, 491
covariance, 184
cover proportions, 689
cross correlation, 193, 423
cross covariance, 193
data whitening, 265, 266
daylight, 703
daylight, variations of, 701, 703
degree, of a graph node, 577, 586
degrees of freedom of χ^2 -test, 239
delta function, 95, 142, 144, 152, 405
 Fourier transform of, 325
 shifting property of, 15
DFT, 94, 176
 algorithm for, 102
 convolution theorem for, 105
dc component of, 118
display of, 112
for rotated image, 113
for scaled image, 119
for shifted image, 114
imaginary, 130
inverse of, 95
magnitude and phase of, 411
matrix for, 99
 real valued, 126
dichromats, 717
differentiation with respect to a vector, 267

- direct component, 118
discrete Fourier transform, *see* DFT
distribution function, 178, 180
divergence, 345
dual grid, 569, 593
- Earth observation, 688
EDCT, 138, 176
 - basis images of, 146
 - inverse
 - 1D, 143
 - 2D, 145
edge detection, 527, 591
 - algorithm, 602
 - and noise, 593, 605, 606
 - by linear filtering, 592, 602
 - by nonlinear filtering, 591
 - Canny's criteria for, 606
 - for multispectral images, 769
 - in the first principal component, 769
 - statistical, 591
 - using phase congruency, 626
 - via local energy, 652
 - with Laplacian of Gaussian, 621, 623
edge preserving smoothness constraint, 502
edge, of a graph, 576
edgel, 593
EDST, 157, 176
 - basis images of, 163
 - inverse
 - 1D, 160
 - 2D, 162
eigenface, 292
eigenimage, 60, 222, 292
eigenvector, 222
elementary images
 - DFT, 101
 - EDCT, 146
 - EDST, 163
 - Haar, 80, 93
 - Hadamard, 88
 - ICA, 281
 - independent, 292
 - K-L, 221
 - ODCT, 154
 - ODST, 173
 - orthogonal, 292
uncorrelated, 292
Walsh, 88
EM algorithm, 537
end members, 695, 697
energy, local, 660
ensemble autocorrelation function, 190
ensemble autocorrelation matrix, 210
ensemble of images, 190–192
ensemble statistics, 195, 292
entropy, 239, 243
 - of an image, 673
entropy of a Gaussian pdf, 243, 246
entropy of a uniform pdf, 244
equal energy spectrum, 715
ergodicity, 195, 197, 199, 200, 215
error
 - due to wrong reference white, 743, 745
 - in image approximation
 - by DFT, 103, 176
 - by EDCT, 148, 176
 - by EDST, 165, 166, 176
 - by Haar transform, 79, 89, 176
 - by Hadamard transform, 91, 176
 - by K-L, 220
 - by ODCT, 156, 176
 - by ODST, 175, 176
 - by SVD, 63, 176
 - by Walsh transform, 91, 176
 - least mean square, 226
 - least mean square in image approx., 292
 - least square in image approx., 292
error function, 237, 251
estimation
 - least square error, 419
 - MAP, 490
 - maximum a posteriori, 490
Euclidean metric
 - definition of, 733
 - use in measuring colour difference, 734
even antisymmetric sine transform, *see* EDST
even symmetric cosine transform, *see* EDCT
event, 178
 - probability of, 179
expectation maximisation algorithm, 537
expectation value, 181
false contouring, 7

- false maxima, 619
fast Fourier transform, 124
feature, 551
 symmetry of, 661
feature detection
 antisymmetric, 650, 652
 symmetric, 648, 650, 652
feature extraction, 44
feature space, 551
Fiedler vector, 586
filter
 α -trimmed, 337, 760
 z -transform of, 294
 Butterworth, 303
 definition of, 294
 edge adaptive, 574
 edge preserving, 574
 first derivative of Gaussian, 352
 flat, 327, 328
 frequency response function of, 294
 Gaussian, 309, 332
 high pass, 351
 ideal 1D low pass, 296, 300
 ideal 2D band pass, 299
 ideal 2D high pass, 299
 ideal 2D low pass, 296, 300
 ideal low pass 1D-2D comparison, 301
 IIR, 296
 impulse response of, 294
 infinite impulse response, 296
 Laplace transform of, 294
 Laplacian of Gaussian, 621
 median, 326–328
 mode, 326, 328, 574
 nonlinear, 357
 nonrecursive, 301
 rank order, 326
 real, frequency response of, 294
 real, unit sample response of, 294
 recursive, 301, 303
 Robinson, 406
 second derivative of Gaussian, 352
 separable, 605
 Sobel, 605
 stability of, 294
 statistical, 326
 system transfer function of, 294
 unit sample response of, 294
 vector median, 760
 weighted median, 333
 weighted mode, 333
filtering
 averaging, 327, 328
 lowpass, 328
 mean shift, 574
 median, 326, 327
 mode, 328
first principal component
 for edge detection, 769
 for maximum contrast, 672
fixed temperature annealing, 505
flatfielding, 364, 366, 407, 548
floor operator, 216
Fourier series, 625
Fourier slice theorem, 404
Fourier transform, 72, 94, 177, 293, 637
 definition of, 637
 duality of, 637, 638
 fast, 124
 inverse of, 637
 of a constant, 637
 of a delta function, 637
 of an integral, 637
 of derivative function, 637
 Parseval's theorem for, 637
 properties of, 637
 scaling property of, 637
 shifting property of, 637
 slice theorem of, 404
 Wiener-Khinchine theorem of, 325
fourth order cumulant, 240
frequency convolution theorem, 108
frequency response function, 294, 396
Fresnel integrals, 400
 asymptotic behaviour of, 401
function
 Γ , 183
 ζ , 183
 delta, 405
 Gabor, 126
 gradient vector of, 345
 Haar, 73, 74
 harmonic expansion of, 641
 Laplacian of, 345

- periodic, 141
- Rademacher, 74, 86
- Riemann, 183
- step, 405
- Walsh, 73, 74, 86
- functions
 - complete set of, 72
 - orthogonal, 72
 - orthonormal, 72
- fuzzy logic, 200
- Gabor functions, 126
- Gamma function, 183
- Gauss-Seidel method, 482, 485
- Gaussian filter, 309
 - weights of, 329
- Gaussian filtering, 342
- Gaussian function
 - Fourier transform of, 309
- Gaussian mixture model, 537
- Gaussian probability density function
 - parameter estimation for, 537, 539
- geometric image degradation
 - global, 513
 - inhomogeneous, 515
- geometric progression, 95
- Gibbs sampler, 503
- GMM, 537
- good locality measure, 617
- gradient, 345
 - magnitude of, 557
- gradient descent algorithm, 487
- gradient vector, 345
- Gram matrix, 228
- graph, 576
 - Laplacian matrix of, 577, 586
 - relational, 576
 - undirected, 576
 - weights, 576
- Gray code, 86
- grey level, 1
- grid, dual, 569, 593
- Haar functions, 73, 74
 - discrete version of, 76
 - image basis from, 75
- Haar transform, 74, 177
- advantages of, 92
- algorithm for, 76, 89
- basis images of, 80
- disadvantages of, 92
- elementary images of, 80
- Haar wavelet, 93
- Hadamard matrices, 85
- Hadamard transform, 74, 88
 - basis images of, 88
- half width half maximum, 753
- heat equation, 342, 348
- Hermitian transpose, 50
- Hilbert pair, 631
- Hilbert transform, 631, 639
 - generalisation of, 660
- hill in mathematical morphology, 568
- histogram, 528
 - equalisation, 370
 - equalisation with random additions, 372
 - hyperbolisation, 373, 374
 - hyperbolisation with random additions, 374
 - manipulation, 367, 368
 - normalised, 236
 - stretching, 367
 - with upsampling, 553
- homomorphic filter, 364
- Hough transform, 520
- hue, 733
 - in practice, 747
 - perceived differences in, 738
 - perceived, measurement of, 737
- hue angle, 738
- human vision, 700, 701
 - colour constancy of, 714
 - ganglion cells, 292
 - perception of brightness, 725
 - photopic, 733
 - physics of, 700
 - psychophysics of, 700
 - rods, 714, 733
 - scotopic, 733
- hysteresis edge linking, 621
- hysteresis thresholding, 528
- ICA, 234
 - algorithm for, 274

- blind source separation, 289
characteristics of, 289
differences in image and signal processing, 290
for linear spectral unmixing, 697
for signal processing, 283
in image processing, 264
independent components, 290
medical images, 260
sparse representation, 290
ideal white, 715, 716, 718
IIR, 296
image
 approximation of
 by DFT, 103, 176
 by EDCT, 148, 176
 by EDST, 165, 166, 176
 by Haar transform, 79, 89, 176
 by Hadamard transform, 91, 176
 by K-L, 220, 226
 by ODCT, 156, 176
 by ODST, 175, 176
 by SVD, 62, 63, 176
 by Walsh transform, 91, 176
 average of, 118
 bit size of, 6
 classification of, 528
 coding for optimisation algorithms, 505
 colouring for optimisation algorithms, 505
 compression, 44
 contrast of, 10
 dc component of, 118, 176
 definition of, 1
 degradation, model of, 396
 DFT of scaled, 119
 digital, 1
 digital, formation of, 3
 dilation, 554
 edge detection of, 527
 eigenimages of, 60
 enhancement, 43, 293, 358, 395
 local, 375
 pairwise, 377, 378
 toboggan, 383
 entropy, 673
 erosion, 555
 filtering, 293, 574
 first difference of, 449
 gradient magnitude of, 557
 Haar transform of, 74, 76
 Hadamard transform of, 74
 histogram of, 528
 information of, 673
 inhomogeneous degradation of, 468
 labelling of, 528
 Laplacian of, 342, 449
 mean value of, 220
 morphological reconstruction of, 554
 multispectral, 1, 669
 multispectral, compression of, 767
 multispectral, edge detection, 769
 multispectral, restoration of, 767
 multispectral, segmentation of, 767
 of an ideal line, 403
 panchromatic, 1
 quality of, 7
 registration, 395
 representation by a graph, 576
 resolution of, 7
 restoration, 44, 395
 as MAP estimation, 490
 constrained matrix inversion, 436
 degradation matrix unknown, 489
 geometric, 395, 513
 grey value, 395
 inverse filtering, 396
 nonlinear, 490
 Wiener filtering, 419
 rotation of, 519
 SAR, 326
 scaling, 112
 second difference of, 449
 segmentation of, 527, 528
 sharpening, 351
 singular value decomposition of, 50, 51,
 60
 synthetic aperture radar, 326
 thresholding, 528
 transform, cosine even symmetric, 137,
 138
 transform, cosine odd symmetric, 137, 149
 transform, DFT, 94, 95
 transform, EDCT, 138
 transform, EDST, 157

- transform, Fourier, 94, 95
 transform, Hadamard, 88
 transform, ODCT, 149
 transform, ODST, 167
 transform, sine even antisymmetric, 137, 157
 transform, sine odd antisymmetric, 137, 167
 transform, unitary, 49
 transform, Walsh, 88
 transforms, comparison of, 176
 Walsh transform of, 74, 76
 whirl transform of, 468
 image basis from
 DFT, 101
 EDCT, 146
 EDST, 163
 Haar functions, 75
 Haar transform, 80
 Hadamard transform, 88
 ICA, 281
 K-L, 221
 ODCT, 154
 ODST, 173
 SVD, 60
 Walsh functions, 75
 image noise, 311
 image smoothing, 328
 image thresholding and variable illumination, 545
 images ensemble, 190
 imaginary primaries, 728, 729
 impulse response, 294
 independent component analysis,
 see ICA
 independent components, 277
 industrial inspection, 364, 366, 742
 and metamers, 756
 infinite impulse response, 296
 information content, 673
 information of a symbol, 673
 interference, low frequency, 351
 inverse filtering, 396
 algorithm for, 410
 comparison with Wiener filtering, 430
 iterative conditional modes, 503
 Jacobi's method, 482
 Jacobian matrix, 270
 joint distribution function, 184
 joint probability density function, 184
 JPEG, 176
 K-L transform, 200, 201, 214
 as a first step of ICA, 265
 basis images of, 221
 comparison with SVD, 292
 error of, 220
 error of approximation with, 226
 Karhunen-Loeve, *see K-L*
 kernel-based method, 553
 knee algorithm, 549
 Kronecker order, 86
 Kronecker product, 38
 kurtosis, 240
 excess, 240
 Pearson, 240
 proper, 240
 L cones, 715
 label of a pixel, 528, 529
 labelling, 528, 529
 Lagrange multiplier, 269
 Lagrange multipliers, 460
 method of, 268
 used in ICA, 269
 Lambertian surface, 685
 Laplace transform, 294
 Laplacian, 345
 as a smoothness constraint, 455
 matrix operator, 450
 matrix operator, eigenvalues of, 454
 of an image, 342, 449
 Laplacian matrix of a graph, 577, 586
 Laplacian of Gaussian filter, 621
 least square error estimation, 419
 least square error solution, 692
 Leibniz rule, 348, 531, 620
 lens distortion
 algorithm for, 522
 modelling of, 514
 leptokurtic probability density function, 240

- level set methods, 621
lexicographic order, 86
library spectra, 689
lightness for colour description, 738
line detection, 609
 using phase congruency, 626
 via local energy, 652
linear degradation process
 inversion of its matrix, 446
 matrix of, 444
linear filter, 293
linear filtering, 293
 high pass, 351
linear operator, 12
 point spread function of, 12
linear spectral unmixing
 with ICA, 697
 with library spectra, 692
 with PCA, 695
 without library spectra, 695
linear system of equations
 solution by Gauss-Seidel method, 482, 485
 solution by Jacobi's method, 482
local energy, 626, 647, 660
 algorithm for, 651
 filters for, 648
 in 2D, 659
 measurement of, 630
local image enhancement, 375

M cones, 715
Macbeth colour checker, 707, 708
magnetoencephalography, 288
manifold, 702
MAP estimation, 490
marginal probability density function, 188
Markov chain, 503
matrix
 block circulant, 22, 437, 600
 circulant, 22, 438
 conjugate transpose of, 50
 diagonalisation of, 50
 eigenvalues of, 587
 Gram, 228
 Hadamard, 85
 Hermitian transpose of, 50
 Jacobian, 270
 Kronecker product of, 38, 445
 norm of, 63, 64, 226
 orthogonal, 50
 partition of, 19
 positive semidefinite, 228, 586
 symmetric, 586
 Toeplitz, 213
 trace of, 64, 226
 unitary, 49, 99
maximum a posteriori estimation, 490
Maxwell triangle, 720, 747
 coordinates in, 722
MCMC minimisation, 503
mean distance of zero crossings, 619
mean shift, 339
 for colour image segmentation, 767
 for segmentation, 574
 for smoothing, 337, 339
mean square error, 416
mean value, 181, 185
median, 326
medical images and ICA, 260
membrane model, 501
 edge preserving, 502
metamerism, 756
metamers, 713, 756
 in industrial inspection, 756
metric
 L_1 norm, 733
 L_2 norm, 733
 L_∞ norm, 733
 Chebyshev norm, 733
 city block, 733
 definition of, 733
 Euclidean, 733
Metropolis sampler, 503
minimum error threshold, 530, 534, 535
 drawbacks of, 541
mode, 328
mode filter, 574
 edge adaptive, 574
 edge preserving, 574
mode filtering, 326
modulus, 100
monochromats, 717
monogenic signal, 659, 660
Monte Carlo Markov chain, 503

- Monte-Carlo method, 757
morphological image reconstruction, 557
morphological reconstruction, 554, 558, 560
motion blurring, 397
 - point spread function of, 398
MSE, 416
multiple illumination sources, 687
multiresolution, 511
 - via renormalisation group transform, 512
 - via super-coupling transform, 512
multiresolution optimisation, 512
multispectral camera, 2
multispectral image, 1, 669
- natural materials
 - spectra of, 708
 - variation of, 706, 708
natural order, 86
near infrared, 704
negentropy, 239
 - approximation of, 246, 252, 254, 257
 - definition of, 243
 - maximisation of, 269
noise
 - additive, 311, 337
 - additive Gaussian, 492
 - autocorrelation function of, 313
 - biased, 312, 314
 - blue, 313
 - coloured, 313, 314
 - dependent, 316, 318
 - filtered, 615
 - fixed pattern, 312
 - Gaussian, 311, 317, 326–328, 332, 337
 - homogeneous, 311
 - iid, 311, 313–315, 317, 492
 - impulse, 311, 328, 337
 - in multispectral images, 759
 - independent, 311, 312
 - independent identically distributed, 313
 - mixed, 337, 760
 - multiplicative, 311, 325
 - probability density function of, 235
 - salt and pepper, 311
 - shot, 311, 326
 - spec, 311
 - unbiased, 311, 312
uncorrelated, 311–313
uniform, 315
white, 311, 313–316, 318
zero-mean, 311–313, 318
- non-Gaussianity
 - measure of, 239, 240
non-maxima suppression, 601
norm
 - L_1 , 733
 - L_2 , 733
 - L_∞ , 733
 - Chebyshev, 733
 - of a matrix, 64, 226
normal colour vision, 717
normal distribution, 534
normal order, 86
normalised graph cuts algorithm, 576, 589
 - as an eigenvalue problem, 576
normalised histogram, 236
- ODCT, 149, 176
 - basis images of, 154
 - inverse
 - 1D, 152
 - 2D, 154
odd antisymmetric sine transform, *see* ODST
odd symmetric cosine transform, *see* ODCT
ODST, 167, 176
 - basis images of, 173
 - inverse
 - 1D, 171
 - 2D, 172
opponent colour space, 741, 752
opponent colours, 741
optimisation, 503–505, 512
orthogonal matrix, 50
orthogonal set of functions, 72
orthonormal set of functions, 72
orthonormal vectors, 50
Otsu method, 541, 542
 - drawbacks of, 545
p-tile method, 530
pairwise image enhancement, 377, 378
Paley order, 86
panchromatic image, 1
Parseval's theorem, 496, 637, 647

- path
 non-descending, 568
 non-ascending, 568
- PCA, 672
 advantages of, 675
 algorithm for, 674
 disadvantages of, 675
 for linear spectral unmixing, 695, 696
 for material spectra, 708
 of daylight spectra, 703
- pel, 1
- perceptually uniform colour space, 742
- period of a function, 141
- phase congruency, 625
 in 2D, 659
 in practice, 630
 measure of, 627
- photometric stereo, 364, 366
- photopic vision, 733
- picture element, 1
- pin-cushion distortion, 513
- pixel, 1
- platykurtic probability density function, 240
- point source, 12, 14
- point spread function, 12, 13, 396
 from astronomical image, 402
 deduction of, 27
 from an ideal edge, 404
 of a camera, 405
 separable, 13
 shift invariant, 13
- Poisson distribution, 311
- power method
 algorithm for, 682
 for eigenvector estimation, 682
- power spectrum, 313
- primary lights, 715
- principal component analysis, *see* PCA
- probability
 a posteriori, 491
 a priori, 491
 definition of, 179
 of an event, 178
 posterior, 491
 prior, 491
- probability density function, 181, 528, 530
 fourth order moment of, 239
- Gaussian, 235, 237, 240
Gaussian, entropy of, 243, 246
Gaussian, fourth moment of, 242
Gaussian, kurtosis of, 242
Gaussian, skewness of, 241
leptokurtic, 240
non-Gaussianity of, 235
normal, 238
of a function of a random variable, 320
of sum of two random variables, 545, 546
platykurtic, 240
second order moment of, 239
skewness of, 239
sub-Gaussian, 240
super-Gaussian, 240
third moment of, 239
third order moment of, 241
uniform, definition of, 244
uniform, entropy of, 244
uniform, fourth moment of, 247
uniform, negentropy of, 244, 247, 251
uniform, variance of, 244
variance of, 239
- probability theory, 200
- pseudo-convolution, 632
- pseudorandom, 178
- purple line, 725
- quad tree, 554
- Rademacher function, 74, 86
- radiant power, 718
- radiometric correction, 689
- ramp detection, 609
- random experiment, 178
- random field, 177, 178, 189, 190
 autocorrelation function of, 424
 autocorrelation of, 190
 autocovariance, 190
 ergodic, 195, 197, 428
 homogeneous, 195, 424
 spatial autocorrelation function of, 325
 spatial statistics of, 196
 stationary, 195
- random fields
 cross correlation of, 193, 423
 cross covariance of, 193

- uncorrelated, 193
- random number generator, 178
 - according to given probability density function, 510
- random variable, 177, 178
 - central moments of, 239
 - distribution function of, 180
 - entropy of, 243
 - expectation value of, 181, 190
 - function of, 320
 - marginal probability density function of, 188
 - mean value of, 181, 185
 - moments of, 239
 - probability density function of, 181
 - standard deviation of, 181
 - variance of, 181
 - zero-mean, 234
- random variables, 190
 - covariance of, 184
 - expectation value of, 187
 - independent, 184, 234
 - joint distribution function of, 184
 - joint probability density function of, 184
 - orthogonal, 184, 234
 - uncorrelated, 184, 234
- rank order filtering, 326
- ranking vectors, 760
- Rayleigh quotient, 578, 682
 - minimisation of, 585
- recursive filter, 303
- reference white, 730, 742
 - and transformations to *Luv* and *Lab*, 735
 - error due to, 743, 745
- reflectance function, 545, 684, 700
- region growing, 553
- regularisation term, 501
- relational graph, 576
- remote sensing, 671, 675, 688, 689, 770
- renormalisation group transform, 512
- retinex algorithm, 357, 360
- ridges in the watershed algorithm, 568
- Riemann's ζ function, 183
- Riesz transform, 659, 660
- Robinson operators, 406
- rods, 733
- rotation transformation, 519
- S cones, 715
- SAD, 693
- saturation, 733
 - in practice, 747
 - perceived, measurement of, 737
- scale space, 342
- scaling function, 93
- scotopic vision, 733
- seed of a random number generator, 178
- seed pixel, 554, 557
- segmentation, 527, 528
 - by filtering, 574
 - by normalised graph cuts, 576
 - via edge detection, 620
 - with mean shift, 574
 - with region growing, 553
 - with split and merge, 554
 - with watershed, 553
- sensitivity function of multispectral camera, 2
- sensor sensitivity function, 711
- sequency order, 86
- sharpening, 351
- signal
 - analytic, 659
 - local energy of, 626, 647
 - monogenic, 659, 660
 - projection of, 632
- signal to noise ratio, 616
- simulated annealing, 503, 539
 - accelerated, 511
 - cooling schedule for, 506
 - fixed temperature, 505
 - properties of, 511
 - termination criterion for, 505
 - with Gibbs sampler, algorithm for, 507
 - with Metropolis sampler, algorithm for, 506
- sine transform
 - even antisymmetric, 137, 157
 - basis images of, 163
 - inverse of 1D, 160
 - inverse of 2D, 162
 - odd antisymmetric, 137, 167
 - basis images of, 173

- inverse of 1D, 171
- inverse of 2D, 172
- singular value decomposition, *see* SVD
- skewness, 239
- smoothing, edge adaptive, 337
- smoothness constraint, 455, 493, 501
- snakes, 621
- Sobel filter, 596, 605
- source signal, 290
- sparse representation, 290
- spatial autocorrelation function, 196
- spatial autocorrelation matrix, 215, 222
- spatial mean, 196
- spatial statistics, 195, 196
- spectral angular distance, 693
- spectral band, 2
- spectral constancy, 669, 684, 742
 - algorithm for, 687
 - assumptions for, 687
- spectral density, 427
- spectral histogram, 767
- spectral signature of a pixel, 684
- spectral unmixing, 671, 688
 - linear, 688, 689
- spectrum locus, 724
- split and merge algorithm, 554
- stacking operator, 29
- standard deviation, 181
- standard illuminant, 704
 - as reference white, 735
- standard illuminants
 - chromaticity coordinates of, 705
 - correlated colour temperature of, 705
 - spectral radiant power of, 705, 706
- standard observer, 718
- statistics
 - ensemble, 190, 195
 - spatial, 195, 196
- step function, 405
- structuring element, 554, 555, 560
- successive doubling, 124
- super-coupling transform, 512
- SVD, 50, 51, 60, 176
 - algorithm for, 69
 - comparison with K-L, 292
 - intuitive explanation of, 62
- symmetry axis, 202
- system transfer function, 294
- temperature parameter, 501
- theorem of the three perpendiculars, 721
- thin plate model, 501
- threshold
 - and variable illumination, 545, 548
 - minimum error, 530, 534, 535, 541
 - Otsu, 541, 542, 545
- thresholding, 528
 - a unimodal histogram, 549
 - drawbacks of, 550
 - hysteresis, 528
 - minimum error, 530, 534
 - Otsu method for, 541, 542, 545
 - p-tile method, 530
 - under variable illumination, 548
- time convolution theorem, 108
- toboggan enhancement, 383
- Toeplitz matrix, 213
- trace of a matrix, 64, 226
- transform
 - DFT, 176
 - EDCT, 176
 - EDST, 176
 - Fourier, 177
 - Haar, 176, 177
 - Hilbert, 639, 660
 - K-L, 200, 201, 214
 - ODCT, 176
 - ODST, 176
 - Riesz, 660
 - Walsh, 176, 177
- triangle inequality, 733
- trichromatic theory of colour vision, 715
- trichromats, 717
- tristimulus values, 715
 - negative, 715
- ultraviolet, 704
- unit sample response, 294
- unitary matrix, 49
- unitary transform, 49
- unsharp masking, 357
 - local, 357
 - locally adaptive, 358
- value in colour representation, 761

-
- variable illumination, 351, 364, 407, 545, 548
 - as low frequency interference, 351
 - variance, 181
 - efficient computation of, 339
 - interclass, 541, 544
 - intraclass, 541, 544
 - vector α -trimmed median filter, 760
 - vector median filter, 760
 - vector outer product, 47
 - vector ranking, 760
 - vectors
 - orthonormal, 50
 - outer product of, 47
 - vertex of a graph, 576
 - Walsh functions, 73, 74, 86
 - Paley order of, 86
 - binary order of, 86
 - discrete version of, 76
 - dyadic order of, 86
 - from Hadamard matrices, 85
 - from Rademacher functions, 74
 - image basis from, 75
 - Kronecker order of, 86
 - lexicographic order of, 86
 - natural order of, 86
 - normal order of, 86
 - ordering of, 86
 - sequency order of, 86
 - Walsh order of, 86
 - Walsh-Kaczmarz order of, 86
 - Walsh order, 86
 - Walsh transform, 74, 88, 177
 - advantages of, 92
 - algorithm for, 76, 81, 91
 - basis images of, 88
 - disadvantages of, 92
 - Walsh-Kaczmarz order, 86
 - waterhole in mathematical morphology, 568
 - waterlines for the watershed algorithm, 568
 - watershed algorithm, 566
 - drawbacks of, 568
 - watershed segmentation, 553
 - wavelets, 93
 - Weber-Fechner law, 360
 - whirl transform, 468
 - algorithm for, 469, 472, 473
 - Wiener filter, 417, 420, 429
 - comparison with constrained matrix inversion filter, 462
 - derivation of, 428
 - in practice, 430
 - relationship with inverse filter, 430
 - Wiener filtering, 419
 - algorithm for, 431
 - Wiener-Khinchine theorem, 313, 325, 427