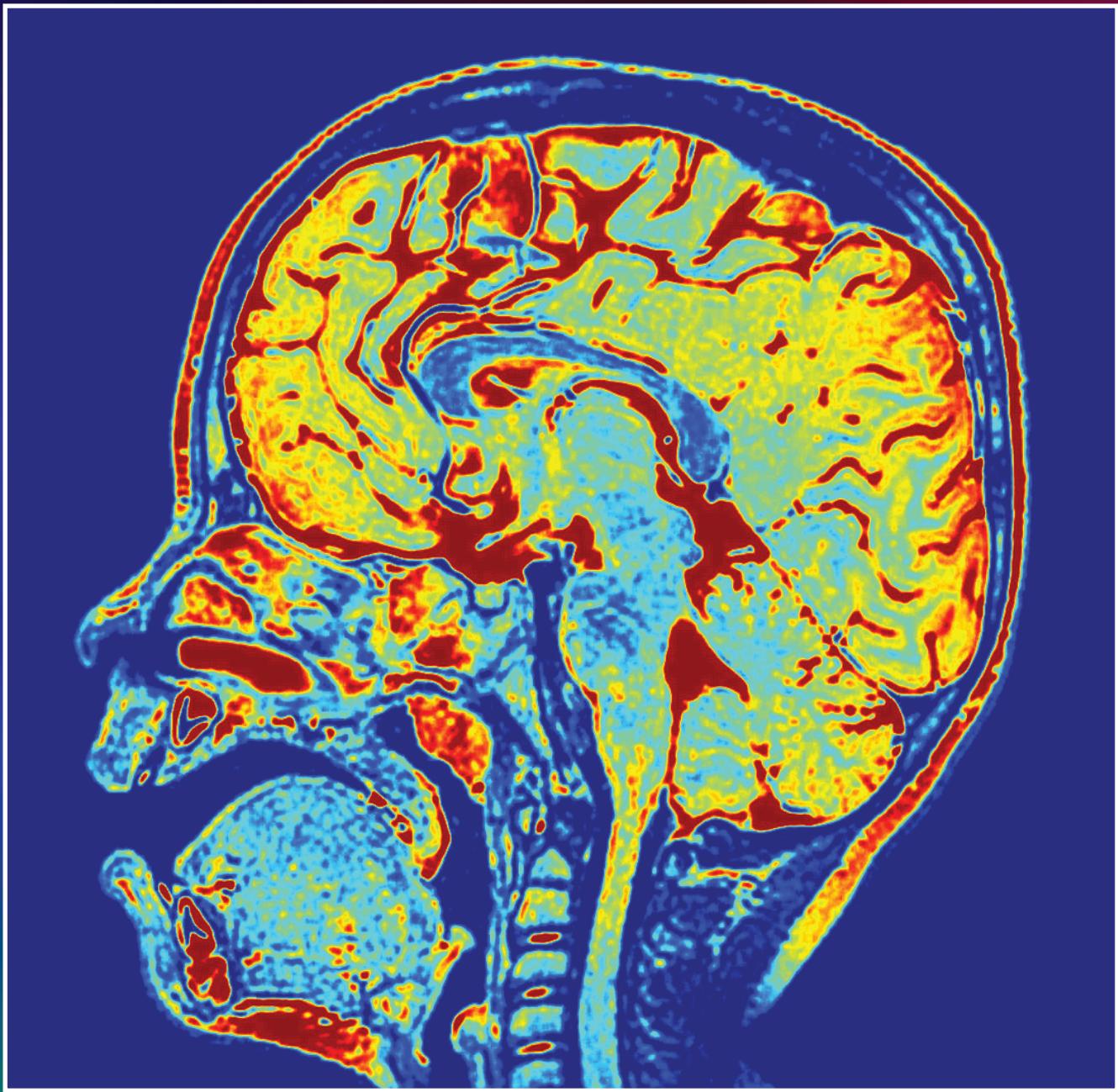


# IMAGE PROCESSING FOR ENGINEERS



*by Andrew E. Yagle and Fawwaz T. Ulaby*

**Book companion website:**

IP : [ip.eecs.umich.edu](http://ip.eecs.umich.edu)





# IMAGE PROCESSING FOR ENGINEERS

**Andrew E. Yagle**  
*The University of Michigan*

**Fawwaz T. Ulaby**  
*The University of Michigan*

Copyright © 2018 Andrew E. Yagle and Fawwaz T. Ulaby

This book is published by Michigan Publishing under an agreement with the authors.  
It is made available free of charge in electronic form to any student or instructor  
interested in the subject matter.

Published in the United States of America by  
Michigan Publishing  
Manufactured in the United States of America

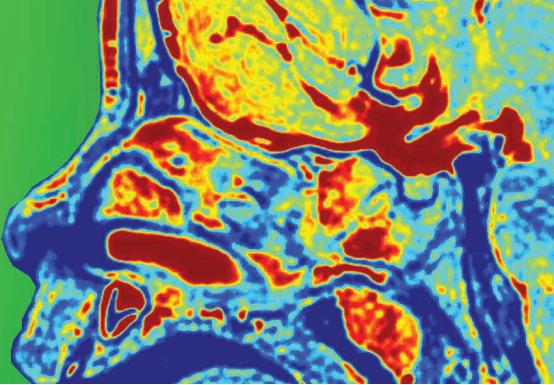
ISBN 978-1-60785-488-3 (hardcover)  
ISBN 978-1-60785-489-0 (electronic)

The Free ECE Textbook Initiative is sponsored by the ECE  
Department at the University of Michigan.



*This book is dedicated to the memories of*  
**Professor Raymond A. Yagle and Mrs. Anne Yagle**

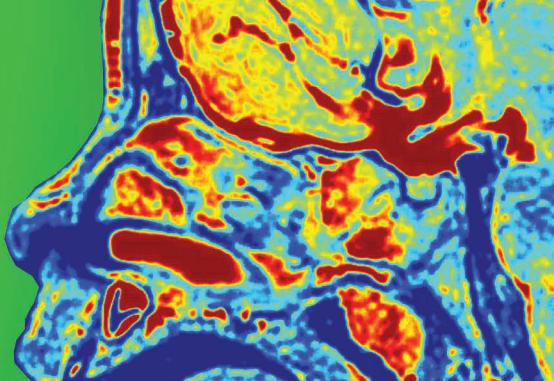
# CONTENTS



<b>Preface</b>			
<b>Chapter 1 Imaging Sensors</b>			
1-1 Optical Imagers	1	4-1 Interpolation Using Sinc Functions	129
1-2 Radar Imagers	13	4-2 Upsampling and Downsampling Modalities	130
1-3 X-Ray Computed Tomography (CT)	18	4-3 Upsampling and Interpolation	133
1-4 Magnetic Resonance Imaging	19	4-4 Implementation of Upsampling Using 2-D DFT in MATLAB	137
1-5 Ultrasound Imager	23	4-5 Downsampling	140
1-6 Coming Attractions	27	4-6 Antialias Lowpass Filtering	141
<b>Chapter 2 Review of 1-D Signals and Systems</b>	38	4-7 B-Splines Interpolation	143
2-1 Review of 1-D Continuous-Time Signals	41	4-8 2-D Spline Interpolation	149
2-2 Review of 1-D Continuous-Time Systems	43	4-9 Comparison of 2-D Interpolation Methods	150
2-3 1-D Fourier Transforms	47	4-10 Examples of Image Interpolation Applications	152
2-4 The Sampling Theorem	53		
2-5 Review of 1-D Discrete-Time Signals and Systems	59	<b>Chapter 5 Image Enhancement</b>	159
2-6 Discrete-Time Fourier Transform (DTFT)	66	5-1 Pixel-Value Transformation	160
2-7 Discrete Fourier Transform (DFT)	70	5-2 Unsharp Masking	163
2-8 Fast Fourier Transform (FFT)	76	5-3 Histogram Equalization	167
2-9 Deconvolution Using the DFT	80	5-4 Edge Detection	171
2-10 Computation of Continuous-Time Fourier Transform (CTFT) Using the DFT	82	5-5 Summary of Image Enhancement Techniques	176
<b>Chapter 3 2-D Images and Systems</b>	89		
3-1 Displaying Images	90	<b>Chapter 6 Deterministic Approach to Image         Restoration</b>	180
3-2 2-D Continuous-Space Images	91	6-1 Direct and Inverse Problems	181
3-3 Continuous-Space Systems	93	6-2 Denoising by Lowpass Filtering	183
3-4 2-D Continuous-Space Fourier Transform (CSFT)	94	6-3 Notch Filtering	188
3-5 2-D Sampling Theorem	107	6-4 Image Deconvolution	191
3-6 2-D Discrete Space	113	6-5 Median Filtering	194
3-7 2-D Discrete-Space Fourier Transform (DSFT)	118	6-6 Motion-Blur Deconvolution	195
3-8 2-D Discrete Fourier Transform (2-D DFT)	119		
3-9 Computation of the 2-D DFT Using MATLAB	121	<b>Chapter 7 Wavelets and Compressed Sensing</b>	202
		7-1 Tree-Structured Filter Banks	203
		7-2 Expansion of Signals in Orthogonal Basis Functions	206
		7-3 Cyclic Convolution	209
		7-4 Haar Wavelet Transform	213

<b>7-5</b>	Discrete-Time Wavelet Transforms	218	<b>Chapter 10 Color Image Processing</b>	<b>334</b>
<b>7-6</b>	Sparsification Using Wavelets of Piecewise-Polynomial Signals	223	<b>10-1</b> Color Systems	335
<b>7-7</b>	2-D Wavelet Transform	228	<b>10-2</b> Histogram Equalization and Edge Detection	340
<b>7-8</b>	Denoising by Thresholding and Shrinking	232	<b>10-3</b> Color-Image Deblurring	343
<b>7-9</b>	Compressed Sensing	236	<b>10-4</b> Denoising Color Images	346
<b>7-10</b>	Computing Solutions to Underdetermined Equations	238	<b>Chapter 11 Image Recognition</b>	<b>353</b>
<b>7-11</b>	Landweber Algorithm	241	<b>11-1</b> Image Classification by Correlation	354
<b>7-12</b>	Compressed Sensing Examples	242	<b>11-2</b> Classification by MLE	357
<b>Chapter 8 Random Variables, Processes, and Fields</b>	<b>254</b>		<b>11-3</b> Classification by MAP	358
<b>8-1</b>	Introduction to Probability	255	<b>11-4</b> Classification of Spatially Shifted Images	360
<b>8-2</b>	Conditional Probability	259	<b>11-5</b> Classification of Spatially Scaled Images	361
<b>8-3</b>	Random Variables	261	<b>11-6</b> Classification of Rotated Images	366
<b>8-4</b>	Effects of Shifts on Pdfs and Pmfs	263	<b>11-7</b> Color Image Classification	367
<b>8-5</b>	Joint Pdfs and Pmfs	265	<b>11-8</b> Unsupervised Learning and Classification	373
<b>8-6</b>	Functions of Random Variables	269	<b>11-9</b> Unsupervised Learning Examples	377
<b>8-7</b>	Random Vectors	272	<b>11-10</b> K-Means Clustering Algorithm	380
<b>8-8</b>	Gaussian Random Vectors	275	<b>Chapter 12 Supervised Learning and Classification</b>	<b>389</b>
<b>8-9</b>	Random Processes	278	<b>12-1</b> Overview of Neural Networks	390
<b>8-10</b>	LTI Filtering of Random Processes	282	<b>12-2</b> Training Neural Networks	396
<b>8-11</b>	Random Fields	285	<b>12-3</b> Derivation of Backpropagation	403
<b>Chapter 9 Stochastic Denoising and Deconvolution</b>	<b>291</b>		<b>12-4</b> Neural Network Training Examples	404
<b>9-1</b>	Estimation Methods	292	<b>Appendix A Review of Complex Numbers</b>	<b>411</b>
<b>9-2</b>	Coin-Flip Experiment	298	<b>Appendix B MATLAB® and MathScript</b>	<b>415</b>
<b>9-3</b>	1-D Estimation Examples	300	<b>Index</b>	<b>421</b>
<b>9-4</b>	Least-Squares Estimation	303		
<b>9-5</b>	Deterministic versus Stochastic Wiener Filtering	307		
<b>9-6</b>	2-D Estimation	309		
<b>9-7</b>	Spectral Estimation	313		
<b>9-8</b>	1-D Fractals	314		
<b>9-9</b>	2-D Fractals	320		
<b>9-10</b>	Markov Random Fields	322		
<b>9-11</b>	Application of MRF to Image Segmentation	327		

# PREFACE



*“A picture is worth a thousand words.”*

This is an image processing textbook with a difference. Instead of just a picture gallery of before-and-after images, we provide (on the accompanying website) MATLAB programs (.m files) and images (.mat files) for each of the examples. These allow the reader to experiment with various parameters, such as noise strength, and see their effect on the image processing procedure. We also provide general MATLAB programs, and Javascript versions of them, for many of the image processing procedures presented in this book. We believe studying image processing without actually performing it is like studying cooking without turning on an oven.

Designed for a course on image processing (IP) aimed at both graduate students as well as undergraduates in their senior year, in any field of engineering, this book starts with an overview in Chapter 1 of how imaging sensors—from cameras to radars to MRIs and CAT—form images, and then proceeds to cover a wide array of image processing topics. The IP topics include: image interpolation, magnification, thumbnails, and sharpening, edge detection, noise filtering, de-blurring of blurred images, supervised and unsupervised learning, and image segmentation, among many others. As a prelude to the chapters focused on image processing (Chapters 3–12), the book offers in Chapter 2 a review of 1-D signals and systems, borrowed from our 2018 book *Signals and Systems: Theory and Applications*, by Ulaby and Yagle.

## Book highlights:

- A section in Chapter 1 called “Coming Attractions,” offering a sampling of the image processing applications covered in the book.
- MATLAB programs and images (.m and .mat files) on the book’s website for all examples and problems. All of these also run on NI LabVIEW Mathscript.
- Coverage of standard image processing techniques, including upsampling and downsampling, rotation and scaling, histogram equalization, lowpass filtering, classification, edge detection, and an introduction to color image processing.

- An introduction to discrete wavelets, and application of wavelet-based denoising algorithms using thresholding and shrinkage, including examples and problems.
- An introduction to compressed sensing, including examples and problems.
- An introduction to Markov random fields and the ICM algorithm.
- An introduction to supervised and unsupervised learning and neural networks.
- Coverage of both deterministic (least-squares) and stochastic (a priori power spectral density) image deconvolution, and how the latter gives better results.
- Interpolation using B-splines.
- A review of probability, random processes, and MLE, MAP, and LS estimation.

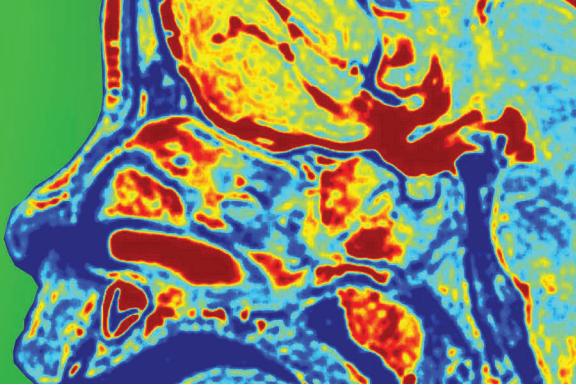
## Book Companion Website: [ip.eecs.umich.edu](http://ip.eecs.umich.edu)

The book website is a rich resource developed to extend the educational experience of the student beyond the material covered in the textbook. It contains MATLAB programs, standard images to which the reader can apply the image processing tools outlined in the book, and Javascript image processing modules with selectable parameters. It also contains solutions to “concept questions” and “exercises,” and, for instructors, solutions to homework problems.

**Acknowledgments:** Mr. Richard Carnes—our friend, our piano performance teacher, and our L<sup>A</sup>T<sub>E</sub>X super compositor—deserves singular thanks and praise for the execution of this book. We are truly indebted to him for his meticulous care and attention. We also thank Ms. Rose Anderson for the elegant design of the cover and for creating the printable Adobe InDesign version of the book.

ANDREW YAGLE AND FAWWAZ ULABY, 2018

# CHAPTER 1



## 1

# Imaging Sensors

### Contents

- Overview, 2
- 1-1** Optical Imagers, 3
- 1-2** Radar Imagers, 13
- 1-3** X-Ray Computed Tomography (CT), 18
- 1-4** Magnetic Resonance Imaging, 19
- 1-5** Ultrasound Imager, 12
- 1-6** Coming Attractions, 27
- Problems, 36

### Objectives

Learn about:

- How a digital camera forms an image, and what determines the angular resolution of the camera.
- How a thermal infrared imager records the distribution of energy emitted by the scene.
- How a radar can create images with very high resolution from satellite altitudes.
- How an X-ray system uses computed tomography (CT) to generate 3-D images.
- How magnetic resonance is used to generate 3-D MRI images.
- How an ultrasound instrument generates an image of acoustic reflectivity, much like an imaging radar.
- The history of image processing.
- The types of image-processing operations examined in detail in follow-up chapters.

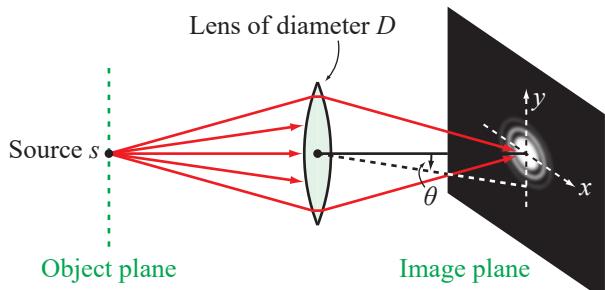


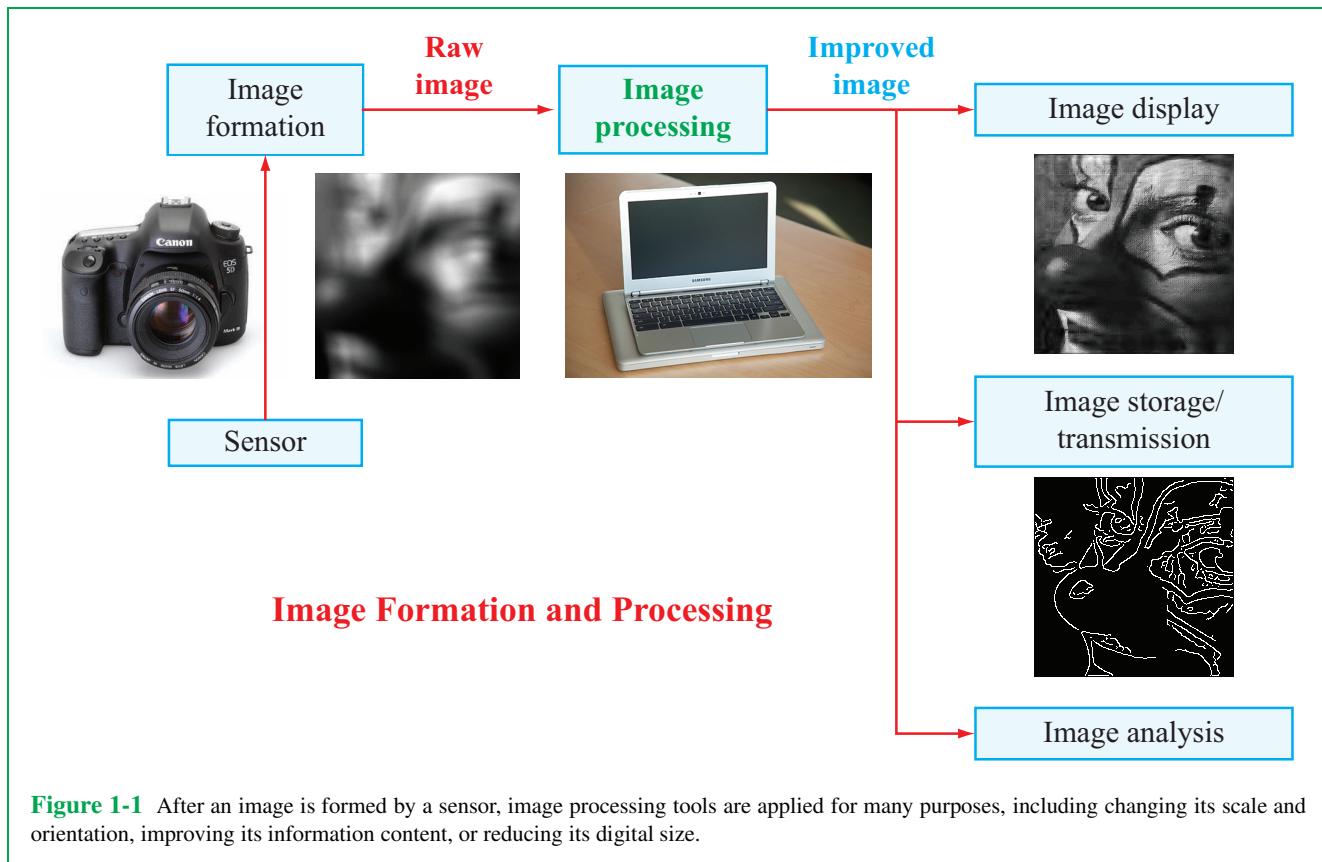
Image processing has applications in medicine, robotics, human-computer interface, and manufacturing, among many others. This book is about the mathematical methods and computational algorithms used in processing an image from its **raw form**—whether generated by a digital camera, an ultrasound monitor, a high-resolution radar, or any other 2-D imaging system—into an **improved form** suitable for the intended application. As a prelude, this chapter provides overviews of the image formation processes associated with several sensors.

## Overview

In today's world we use ***two-dimensional (2-D)*** images generated by a variety of different sensors, from optical cameras and ultrasound monitors to high-resolution radars and others. A camera uses light rays and lenses to form an image of the brightness distribution across the scene observed by the lens, ultrasound imagers use sound waves and transducers to measure the reflectivity of the scene or medium exposed to the sound waves, and radar uses antennas to illuminate a scene with microwaves and then detect the fraction of energy scattered back toward the radar. The three image formation processes are markedly different, yet their output product is similar: a 2-D analog or digital image. An X-ray ***computed tomography (CT)*** scanner measures the attenuation of X-rays along many directions through a 3-D object, such as a human head, and then processes the data to generate one or more 2-D cross-

sectional images (called ***slices***) of the attenuation for specific areas of interest. A rather different process occurs in ***magnetic resonance imaging (MRI)***.

For these and many other sensing processes, the formation of the 2-D image is only the first step. As depicted in **Fig. 1-1**, we call such an image the ***raw image***, because often we subject the raw image to a sequence of image processing steps designed to transform the image into a product more suitable for the intended application (**Table 1-1**). These steps may serve to filter out (most of) the noise that may have accompanied the (desired) signal in the image detection process, rotate or interpolate the image if called for by the intended application, enhance certain image features to accentuate recognition of objects of interest, or compress the number of pixels representing the image so as to reduce data storage (number of bits), as well as other related actions.



**Figure 1-1** After an image is formed by a sensor, image processing tools are applied for many purposes, including changing its scale and orientation, improving its information content, or reducing its digital size.

**Table 1-1 Examples of image-processing applications.**

- Medicine (radiological diagnoses, microscopy)
- Defense (radar, sonar, infrared, satellites, etc.)
- Robotics / machine vision (e.g., “intelligent” vehicles)
- Human-computer interfaces (face/fingerprint “recognition” for security, character recognition)
- Compression for storage, transmission from space probes, etc.
- Entertainment industry
- Manufacturing (e.g., part inspection)

- This book covers the mathematical bases and computational techniques used to realize these image-processing transformations. ◀

To set the stage for the material covered in future chapters, this introductory chapter introduces the reader to overviews of the image formation processes associated with several different types of sensors. Each of Sections 1-1 through 1-5 sketches the fundamental physical principles and the terminology commonly used in connection with that particular imaging sensor. The chapter concludes with Section 1-6, which provides visual demonstrations of the various image operations that the image

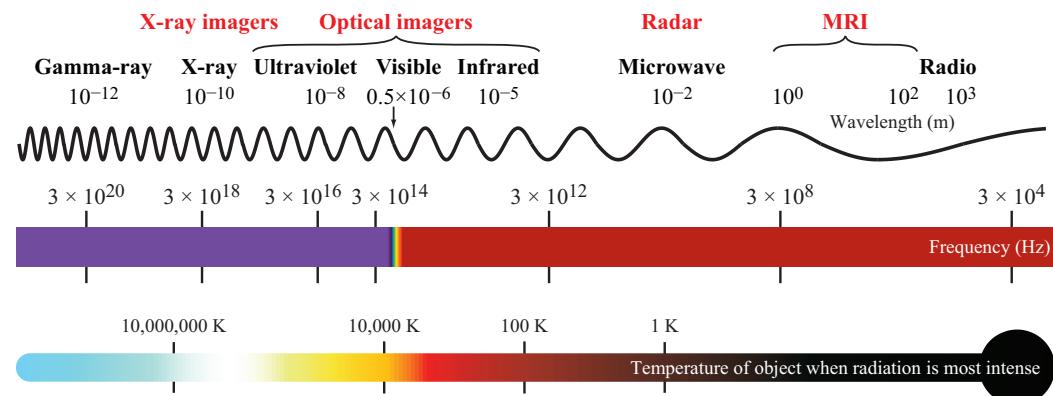
processing techniques covered in future chapters can accomplish.

## 1-1 Optical Imagers

Even though the prime objective of this book is to examine the various image processing techniques commonly applied to a raw image (Fig. 1-1) to transform it into an improved image of specific utility, it will prove helpful to the reader to have a fundamental understanding of the image formation process that led to the raw image in the first place. We consider five types of imaging sensors in this introductory chapter, of which four are **electromagnetic (EM)**, and the fifth is acoustic. **Figure 1-2** depicts the EM spectrum, extending from the gamma-ray region to the radio region. Optical imagers encompass imaging systems that operate in the visible, ultraviolet, and infrared segments of the EM spectrum. In the present section we feature digital cameras, which record reflected energy in the visible part of the spectrum, and infrared imagers, which sense thermal radiation self-emitted by the observed scene.

### 1-1.1 Digital Cameras

In June of 2000, Samsung introduced the first mobile phone with a built-in digital camera. Since then, cameras have become integral to most mobile phones, computer tablets, and laptop computers. And even though cameras may vary widely in terms of their capabilities, they all share the same imaging process. As

**Figure 1-2** Electromagnetic spectrum.

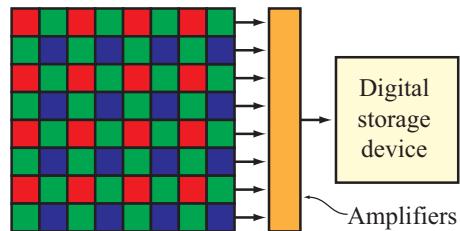
an optical imager, the camera records the spatial distribution of visible light reflected by a scene due to illumination by the sun or an artificial light source. In the simplified diagram shown in **Fig. 1-3**, the converging lens of the camera serves to focus the light reflected by the apple to form a sharp image in the image plane of the camera. To “focus” the image, it is necessary to adjust the location of the lens so as to satisfy the **lens law**

$$\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f} \quad (\text{lens law}), \quad (1.1)$$

where  $d_o$  and  $d_i$  are the distances between the lens and the object and image planes, respectively, and  $f$  is the focal length of the lens.

In a traditional analog camera, the image is captured by a film containing light-sensitive silver halide crystals. The crystals undergo a chemical change—and an associated darkening—in proportion to the amount of light absorbed by the crystals.

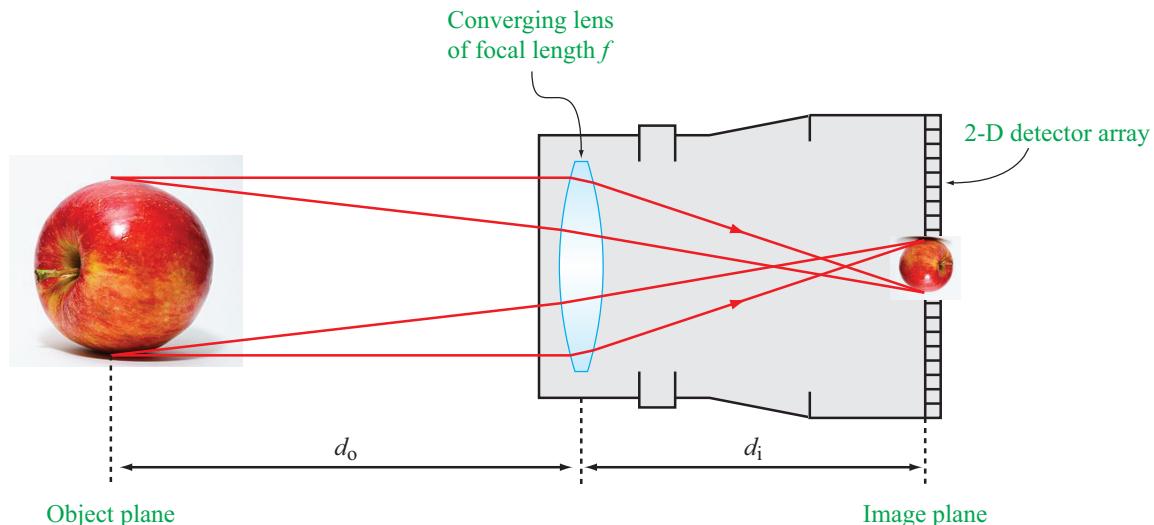
Modern cameras use arrays of **charge-coupled devices (CCDs)** or **active pixel sensors (APSs)**, placed in the image plane, to capture the image and then transfer the intensity readings to a data storage device (**Fig. 1-4**). The CCD relies on charge transfer in response to incident photons, whereas an APS uses a photodetector and an amplifier. CCD arrays were the sensors of choice in the 1970–2000 era, but they have been



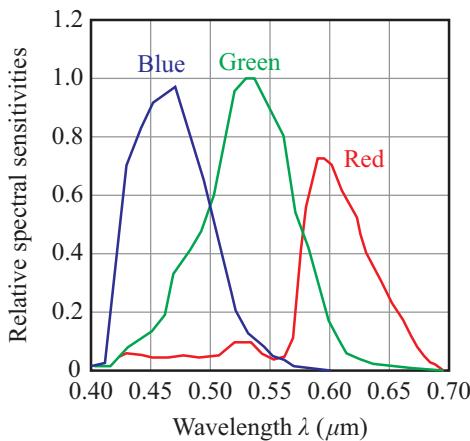
**Figure 1-4** An active pixel sensor uses a 2-D array of photodetectors, usually made of CMOS, to detect incident light in the red, green, and blue bands.

replaced with APS arrays over the past 20 years (Moynihan, 2015) because APS arrays consume less power to operate and are less expensive to fabricate (but they are more susceptible to noise than CCDs).

A photodetector uses **CMOS** (complementary metal-oxide semiconductor) technology to convert incident photons into an output voltage. Because both CCD and CMOS are sensitive to the entire visible spectrum, from about  $0.4 \mu\text{m}$  to  $0.7 \mu\text{m}$ , as well as part of the near-infrared (NIR) spectrum from  $0.7 \mu\text{m}$  to



**Figure 1-3** Camera imaging system.



**Figure 1-5** Spectral sensitivity plots for photodetectors.  
(Courtesy Nikon Corporation.)

1  $\mu\text{m}$ , it is necessary to use a filter to block the IR spectrum and to place red (R), green (G), or blue (B) filters over each pixel so as to separate the visible spectrum of the incident light into the three primary colors. Thus, the array elements depicted in **Fig. 1-4** in red respond to red light, and a similar correspondence applies to those depicted in green and blue. Typical examples of color sensitivity spectra are shown in **Fig. 1-5** for a Nikon camera.

Regardless of the specific detection mechanism (CCD or APS), the array output is transferred to a digital storage device with specific markers denoting the location of each element of the array and its color code (R, G, or B). Each array consists of three subarrays, one for red, another for green, and a third for blue. This information is then used to synchronize the output of the 2-D detector array with the 2-D pixel arrangement on an **LCD** (*liquid crystal display*) or other electronic displays.

## A. Continuous and Discrete Images

By the time an image appears on an LCD screen, it will have undergone a minimum of three transformations, involving a minimum of three additional images. With  $\lambda$  denoting the light wavelength and using **Fig. 1-6** as a guide, we define the following images:

$I_o(x',y';\lambda)$ : continuous intensity brightness in the **object plane**, with  $(x',y')$  denoting the coordinates of the object plane.

$I_i(x,y;\lambda)$ : continuous intensity image in the **image plane** of the camera, with  $(x,y)$  denoting the coordinates of the image plane.

$V[n_1,m_1] = \{V_{\text{red}}[n_1,m_1], V_{\text{green}}[n_1,m_1], V_{\text{blue}}[n_1,m_1]\}$  distribution: discrete 2-D array of the **voltage outputs** of the CCD or photo detector array.

$B[n_2,m_2] = \{B_{\text{red}}[n_2,m_2], B_{\text{green}}[n_2,m_2], B_{\text{blue}}[n_2,m_2]\}$  distribution: discrete 2-D array of the **brightness** across the LCD array.

► Our notation uses parentheses ( ) with continuous-space signals and images, as in  $I_o(x',y')$ , and square brackets [ ] with discrete-space images, as in  $V[n,m]$ . ◀

The three associated transformations are:

**(1) Optical Transformation:** from  $I_o(x',y';\lambda)$  to  $I_i(x,y;\lambda)$ .

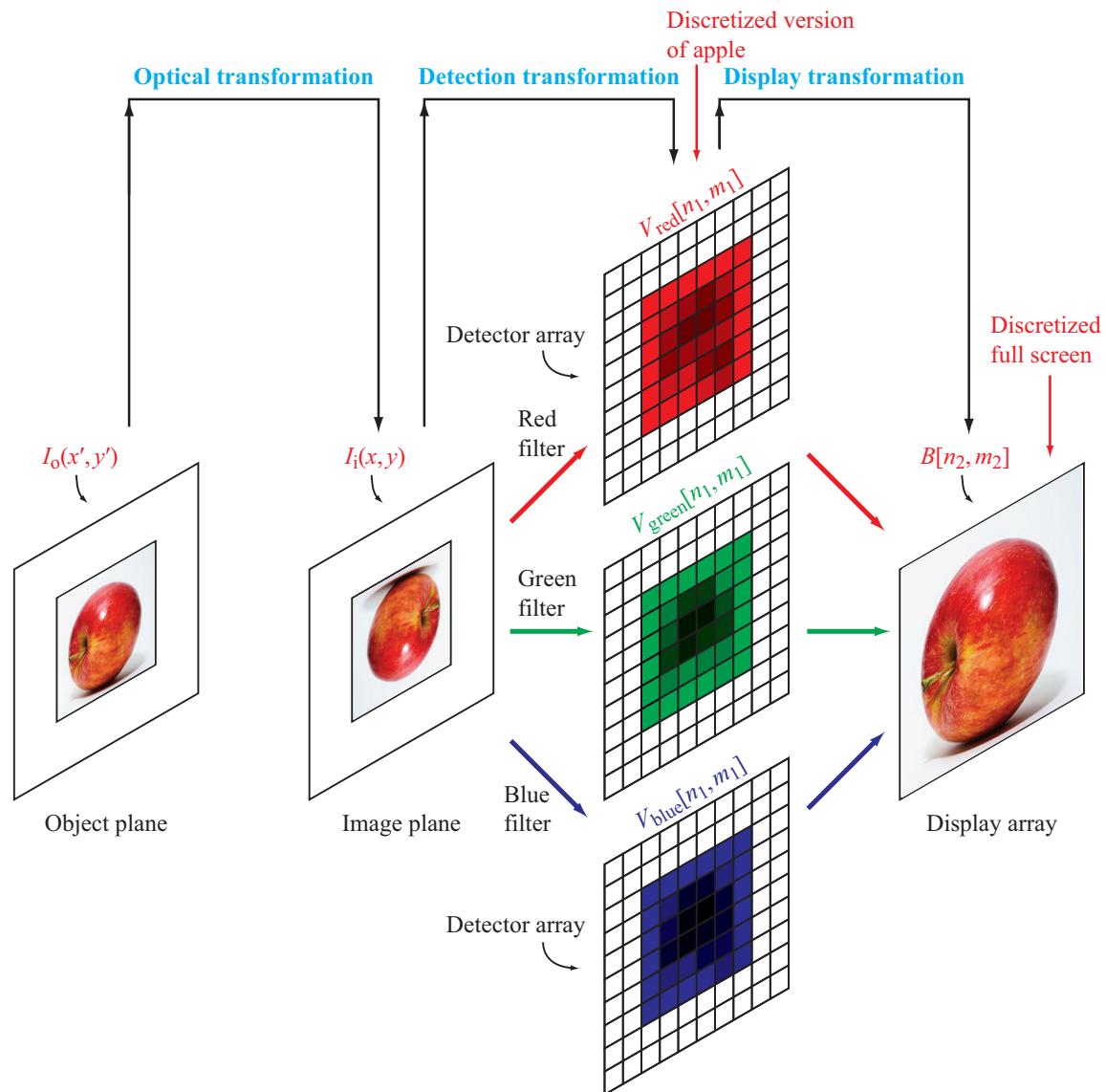
**(2) Detection Transformation:** from  $I_i(x,y;\lambda)$  to  $V[n_1,m_1]$ .

**(3) Display Transformation:** from  $V[n_1,m_1]$  to  $B[n_2,m_2]$ .

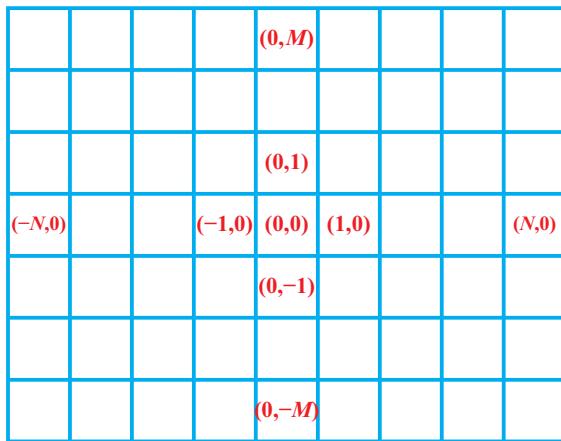
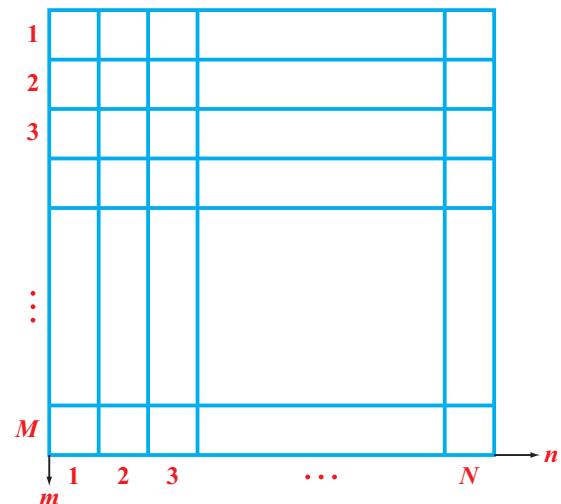
Indices  $[n_1,m_1]$  and  $[n_2,m_2]$  vary over certain ranges of discrete values, depending on the chosen notation. For a discrete image, the two common formats are:

**(1) Centered Coordinate System:** The central pixel of  $V[n,m]$  is at  $(n = 0, m = 0)$ , as shown in **Fig. 1-7(a)**, and the image extends to  $\pm N$  for  $n$  and to  $\pm M$  for  $m$ . The total image size is  $(2M + 1) \times (2N + 1)$  pixels. Note that index  $n$  varies horizontally.

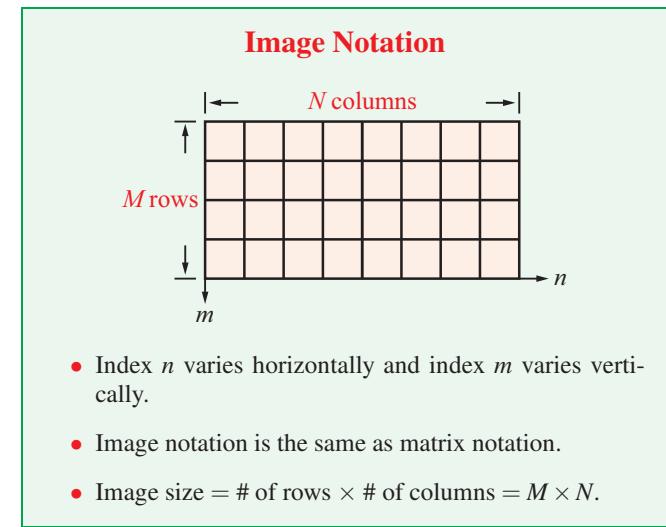
**(2) Corner Coordinate System:** In **Fig. 1-7(b)**, indices  $n$  and  $m$  of  $V[n,m]$  start at 1 (rather than zero). Image size is  $M \times N$ .



**Figure 1-6**  $I_o(x', y'; \lambda)$  and  $I_i(x, y; \lambda)$  are continuous scene brightness and image intensities, whereas  $V[n_1, m_1]$  and  $B[n_2, m_2]$  are discrete images of the detected voltage and displayed brightness, respectively.

(a) Centered coordinates with  $(2M+1) \times (2N+1)$  pixels(b) Corner coordinates with  $(M \times N)$  pixels

**Figure 1-7** (a) In the centered coordinate system, index  $m$  extends between  $-M$  and  $+M$  and index  $n$  varies between  $-N$  and  $+N$ , whereas (b) in the corner coordinate system, indices  $n$  and  $m$  start at 1 and conclude at  $N$  and  $M$ , respectively.



The detection and display images may or may not be of the same size. For example, if image compression is used to generate a “thumbnail,” then far fewer pixels are used to represent the imaged object in the display image than in the detected image. Conversely, if the detected image is to be enlarged through interpolation, then more pixels are used to display the object than in the detected image.

## B. Point Spread Function

Consider the scenario depicted in **Fig. 1-8(a)**. An infinitesimally small source of monochromatic (single wavelength) light, denoted  $s$ , is located in the center of the object plane, and the lens location is adjusted to satisfy Eq. (1.1), thereby producing in the image plane the best-possible image of the source. We assume that the lens has no aberrations due to shape or material imperfections. We observe that even though the source is infinitesimal in spatial extent—essentially like a spatial impulse, its image is definitely not impulse-like. The image exhibits a circularly symmetric **diffraction pattern** caused by the phase interference of the various rays of light that had emanated from the source and traveled to the image plane through the lens. The pattern is called an **Airy disc**.

**Figure 1-8(b)** shows a 1-D plot of the image pattern in terms of the intensity  $I_i(\theta)$  as a function of  $\theta$ , where  $\theta$  is the angular deviation from the central horizontal axis (**Fig. 1-8(a)**). The expression for  $I_i(\theta)$  is

$$I_i(\theta) = I_o \left[ \frac{2J_1(\gamma)}{\gamma} \right]^2, \quad (1.2)$$

where  $J_1(\gamma)$  is the first-order **Bessel function** of the first kind, and

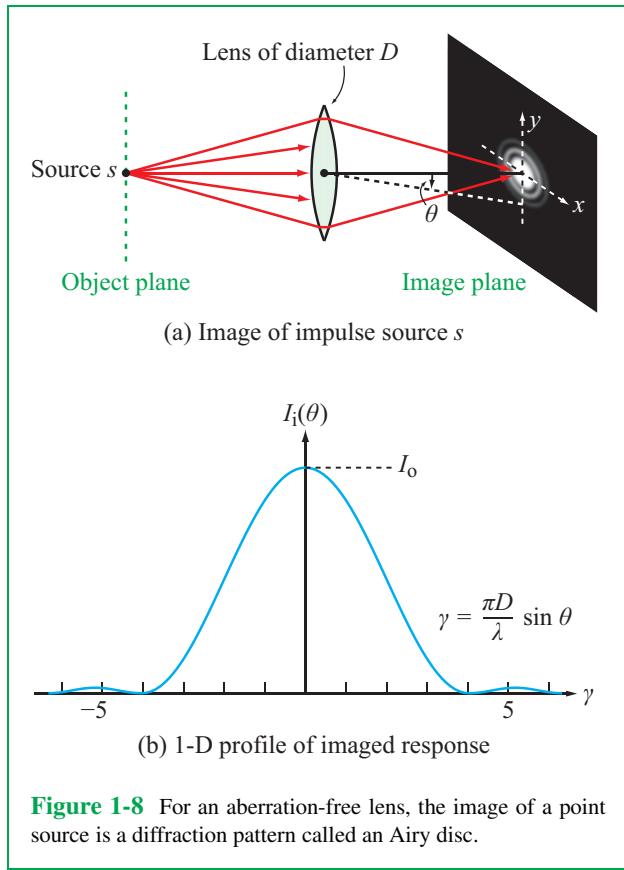
$$\gamma = \frac{\pi D}{\lambda} \sin \theta. \quad (1.3)$$

Here,  $\lambda$  is the wavelength of the light (assumed to be monochromatic for simplicity) and  $D$  is the diameter of the converging lens. The normalized form of Eq. (1.2) represents the **impulse response**  $h(\theta)$  of the imaging system,

$$h(\theta) = \frac{I_i(\theta)}{I_o} = \left[ \frac{2J_1(\gamma)}{\gamma} \right]^2. \quad (1.4)$$

For a 2-D image, the impulse response is called the **point spread function (PSF)**.

Detector arrays are arranged in rectangular grids. For a pixel at  $(x, y)$  in the image plane (**Fig. 1-9**),



**Figure 1-8** For an aberration-free lens, the image of a point source is a diffraction pattern called an Airy disc.

$$\sin \theta = \sqrt{\frac{x^2 + y^2}{x^2 + y^2 + d_i^2}}, \quad (1.5)$$

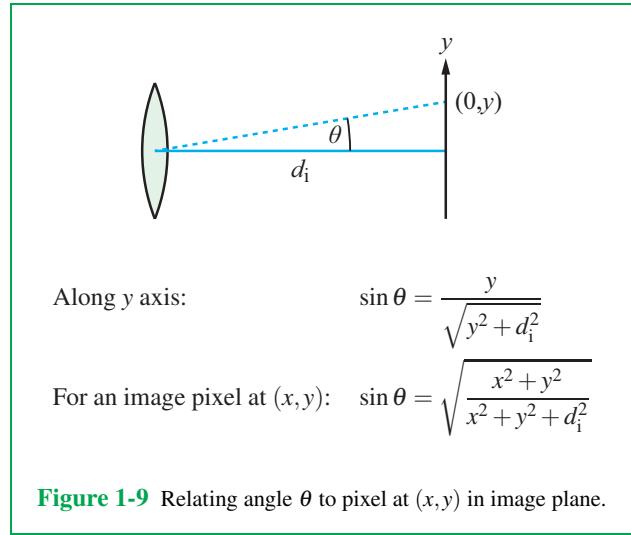
and Eq. (1.4) can be rewritten as

$$h(x, y) = \frac{I_i(x, y)}{I_o} = \left[ \frac{2J_1(\gamma)}{\gamma} \right]^2, \quad (1.6)$$

with

$$\gamma = \frac{\pi D}{\lambda} \sqrt{\frac{x^2 + y^2}{x^2 + y^2 + d_i^2}}. \quad (1.7)$$

The expressions given by Eqs. (1.2) through (1.7) pertain to coherent monochromatic light. Unless the light source is a laser, the light source usually is panchromatic, in which case the diffraction pattern that would be detected by each of the three-color detector arrays becomes averaged over the wavelength range of that array. The resultant diffraction pattern maintains the general shape of the pattern in **Fig. 1-8(b)**, but it exhibits a gentler variation with  $\theta$  (with no distinct minima). Here,  $h(x, y)$  denotes the PSF in rectangular coordinates relative to the center of the image plane.



**Figure 1-9** Relating angle  $\theta$  to pixel at  $(x, y)$  in image plane.

► The implication of this PSF is that when the optical system is used to image a scene, the image it forms in the image plane is the result of a 2-D convolution (as defined later in Section 3-3) of the brightness distribution of the scene in the object plane,  $I_o(x, y)$ , with the PSF given by Eq. (1.7):

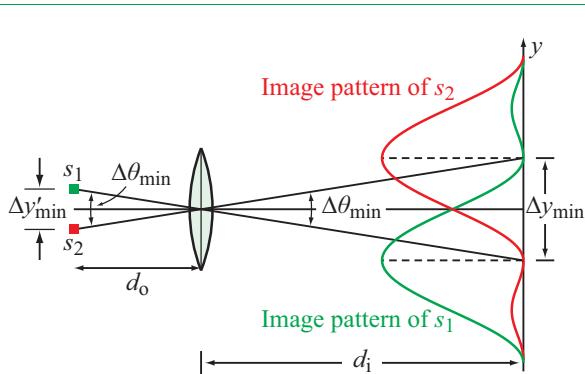
$$I_i(x, y) = I_o(x, y) \ast \ast h(x, y). \quad (1.8)$$

The convolution effect is then embedded in the discrete 2-D detected image as well as in all subsequent manifestations. ◀

### C. Spatial Resolution

Each of the two coherent, monochromatic sources shown in Fig. 1-10 produces a diffraction pattern. If the two sources are sufficiently far apart so that their patterns are essentially distinct, then we should be able to distinguish them from one another. But as we bring them closer together, their diffraction patterns in the image plane start to overlap, making it more difficult to discern their images as those of two distinct sources.

One definition of the ***spatial resolution*** capability of the imaging system along the  $y'$  direction is the separation  $\Delta y'_{\min}$  between the two point sources (Fig. 1-10) such that the peak of the diffraction pattern of one of them occurs at the location of the first null of the diffraction pattern of the other one, and vice versa. Along the  $y$  direction in the image plane, the first null occurs when  $[2J_1(\gamma)/\gamma]^2 = 0$  or, equivalently,  $\gamma = 3.832$ . Use of



**Figure 1-10** The separation between  $s_1$  and  $s_2$  is such that the peak of the diffraction pattern due to  $s_1$  is coincident with the first null of the diffraction pattern of  $s_2$ , and vice versa.

the geometry in Fig. 1-10 with  $\sin \theta \approx \theta$  leads to

$$\Delta\theta_{\min} \approx \frac{\Delta y_{\min}}{d_i} \approx 1.22 \frac{\lambda}{D} \quad (1.9a)$$

**(angular resolution).**

The angular width  $\Delta\theta_{\min}$  is the ***angular resolution*** of the imaging system and  $\Delta y_{\min}$  is the ***image spatial resolution***. On the object side of the lens, the ***scene spatial resolution*** is

$$\Delta y'_{\min} = 1.22 d_o \Delta\theta_{\min} = 1.22 d_o \frac{\lambda}{D}. \quad (1.9b)$$

**(scene spatial resolution)**

This is known as the ***Rayleigh resolution criterion***. Because the lens diameter  $D$  is in the denominator, using a larger lens improves spatial resolution. Thus telescopes are made with very large lenses and/or mirrors.

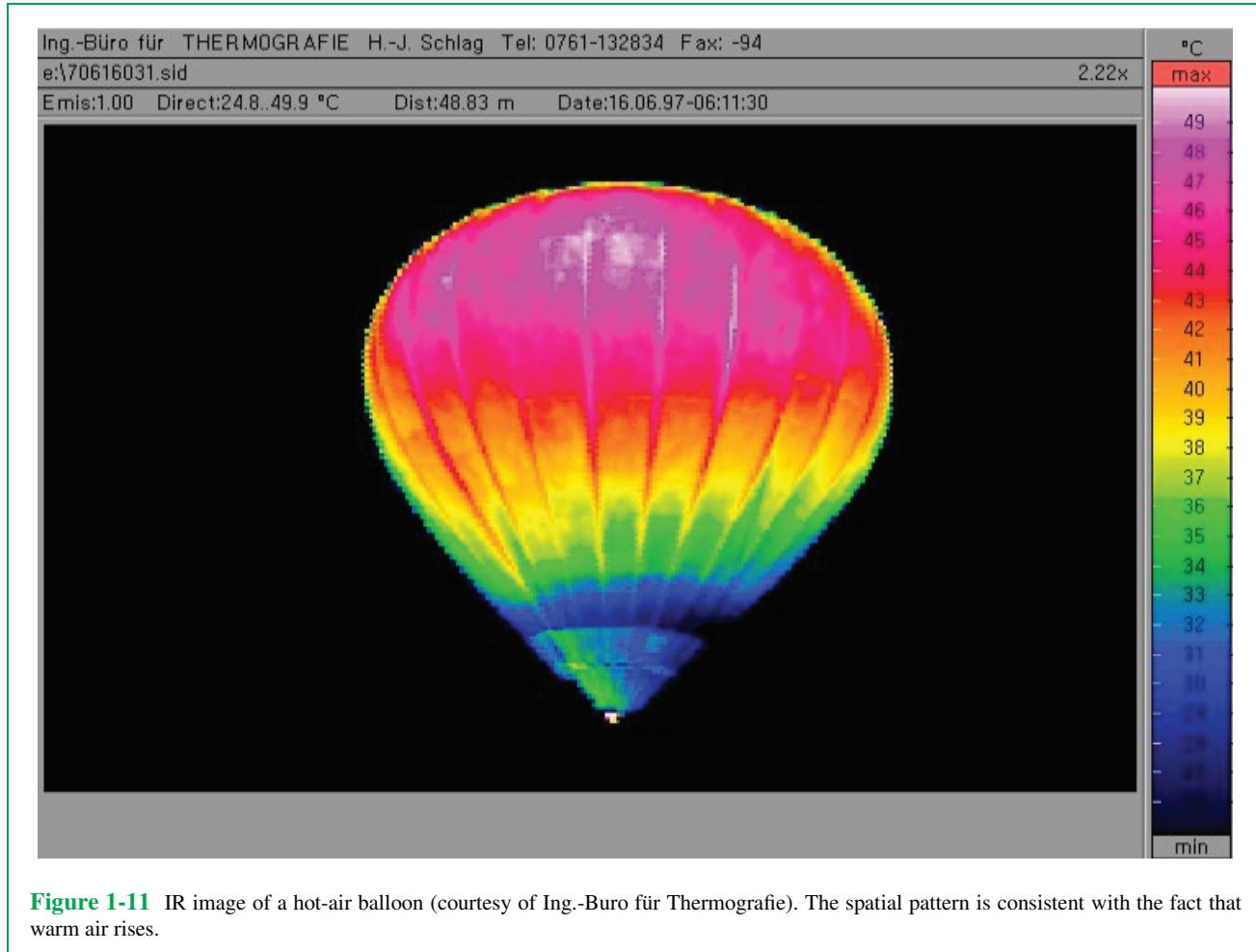
These expressions apply to the  $y$  and  $y'$  directions at wavelength  $\lambda$ . Since the three-color detector arrays operate over different wavelength ranges, the associated angular and spatial resolutions are the smallest at  $\lambda_{\text{blue}} \approx 0.48 \mu\text{m}$  and the largest at  $\lambda_{\text{red}} \approx 0.6 \mu\text{m}$  (Fig. 1-5). Expressions with identical form apply along the  $x$  and  $x'$  direction (i.e., upon replacing  $y$  and  $y'$  with  $x$  and  $x'$ , respectively).

### D. Detector Resolution

The inherent spatial resolution in the image plane is  $\Delta y_{\min} = d_i \lambda / D$ , but the detector array used to record the image has its own ***detector resolution***  $\Delta p$ , which is the pixel size of the active pixel sensor. For a black and white imaging camera, to fully capture the image details made possible by the imaging system, the pixel size  $\Delta p$  should be, at most, equal to  $\Delta y_{\min}$ . In a color camera, however, the detector pixels of an individual color are not adjacent to one another (see Fig. 1-4), so  $\Delta p$  should be several times smaller than  $\Delta y_{\min}$ .

#### 1-1.2 Thermal IR Imagers

**Density slicing** is a technique used to convert a parameter of interest from amplitude to ***pseudocolor*** so as to enhance the visual display of that parameter. An example is shown in Fig. 1-11, wherein color represents the ***infrared (IR)*** temperature of a hot-air balloon measured by a ***thermal infrared imager***. The vertical scale on the right-hand side provides the color-



**Figure 1-11** IR image of a hot-air balloon (courtesy of Ing.-Büro für Thermografie). The spatial pattern is consistent with the fact that warm air rises.

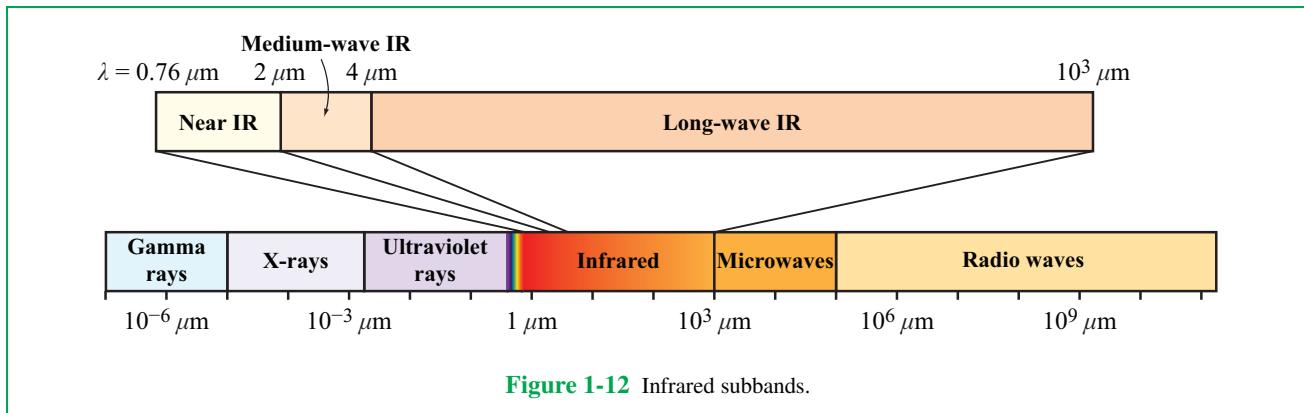


Figure 1-12 Infrared subbands.

temperature conversion. Unlike the traditional camera—which measures the **reflectance** of the observed scene—a thermal IR imager measures the **emission** by the scene, without an external source of illumination. IR imagers are used in many applications including night vision, surveillance, fire detection, and thermal insulation in building construction.

### A. IR Spectrum

The wavelength range of the infrared spectrum extends from the end of the red part of the visible spectrum at about  $0.76 \mu\text{m}$  to the edge of the millimeter-wave band at  $1000 \mu\text{m}$  (or, equivalently,  $\lambda = 1 \text{ mm}$ ). For historical reasons, the IR band has been subdivided into multiple subbands, but these subbands do not have a standard nomenclature, nor standard definitions for their wavelength extents. The prevailing practice assigns the following names and wavelength ranges (Fig. 1-12):

- The **near IR (NIR)** extends from  $\lambda = 0.76 \mu\text{m}$  to  $\lambda = 2 \mu\text{m}$ .
- The **middle-wave IR (MWIR)** extends from  $\lambda = 2 \mu\text{m}$  to  $\lambda = 4 \mu\text{m}$ .
- The **long-wave IR (LWIR)** extends from  $\lambda = 4 \mu\text{m}$  to  $\lambda = 1000 \mu\text{m}$ .

Most sensors operating in the NIR subband are similar to visible light cameras in that they record light reflectance, but only in the  $0.76\text{--}2 \mu\text{m}$  range, whereas IR sensors operating at the longer wavelengths rely on measuring energy self-emitted by the observed object, which depends, in part, on the temperature of the object. Hence, such IR sensors are called **thermal imagers**.

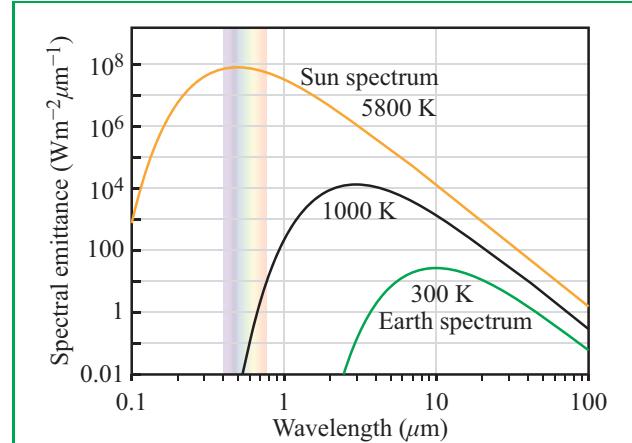
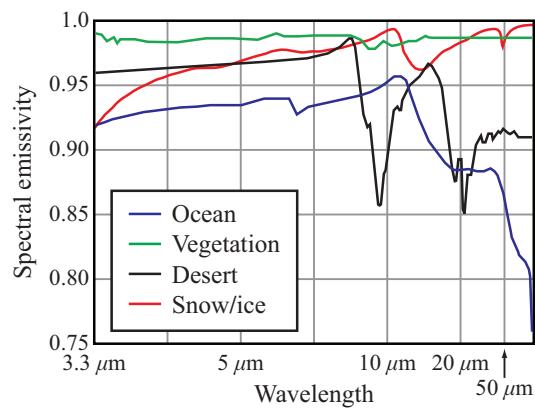


Figure 1-13 The peak of the blackbody radiation spectrum of the sun is in the visible part of the EM spectrum, whereas the peak for a terrestrial object is in the IR (at  $\approx 10 \mu\text{m}$ ).

The basis for the self-emission is the **blackbody radiation law**, which states that all material objects radiate EM energy, and the spectrum of the radiated energy depends on the physical temperature of the object, its material composition, and its surface properties. A **blackbody** is a perfect emitter and perfect absorber, and its radiation spectrum is governed by Planck's law. Figure 1-13 displays plots of spectral emittance for the sun (at an effective radiating temperature of 5800 K) and a terrestrial blackbody at 300 K (27 °C). We observe from Fig. 1-13 that the peak of the terrestrial blackbody is at approximately

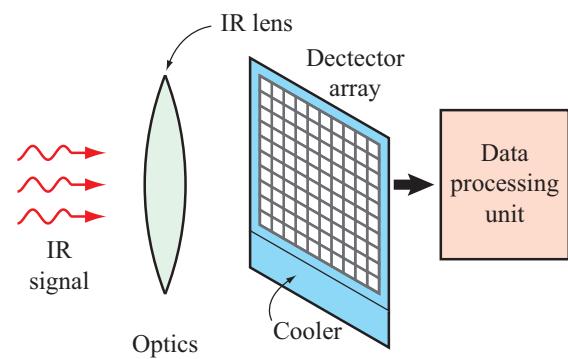


**Figure 1-14** Emissivity spectra for four types of terrain.  
(Courtesy of the National Academy Press.)

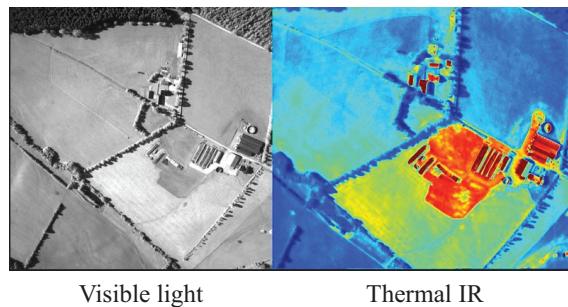
10  $\mu\text{m}$ , which is towards the short wavelength end of the LWIR subband. This means that the wavelength range around 10  $\mu\text{m}$  is particularly well suited for measuring radiation self-emitted by objects at temperatures in the range commonly encountered on Earth. The amount of energy emitted at any specific wavelength depends not only on the temperature of the object, but also on its material properties. The **emissivity** of an object is defined as the ratio of the amount of energy radiated by that object to the amount of energy that would have been radiated by the object had it been an ideal blackbody at the same physical temperature. By way of an example, **Fig. 1-14** displays spectral plots of the emissivity for four types of terrain: an ocean surface, a desert surface, a surface covered with snow or ice, and a vegetation-covered surface.

## B. Imaging System

The basic configuration of a thermal IR imaging system (**Fig. 1-15**) is similar to that of a visible-light camera, but the lenses and detectors are designed to operate over the intended IR wavelength range of the system. Two types of detectors are used, namely **uncooled detectors** and **cooled detectors**. By cooling a semiconductor detector to very low temperatures, typically in the 50–100 K range, its self-generated thermal noise is reduced considerably, thereby improving the signal-to-noise ratio of the detected IR signal emitted by the observed scene. Cooled detectors exhibit superior sensitivity in comparison with uncooled detectors, but the cooling arrangement requires the



**Figure 1-15** Thermal IR imaging systems often use cryogenic cooling to improve detection sensitivity.



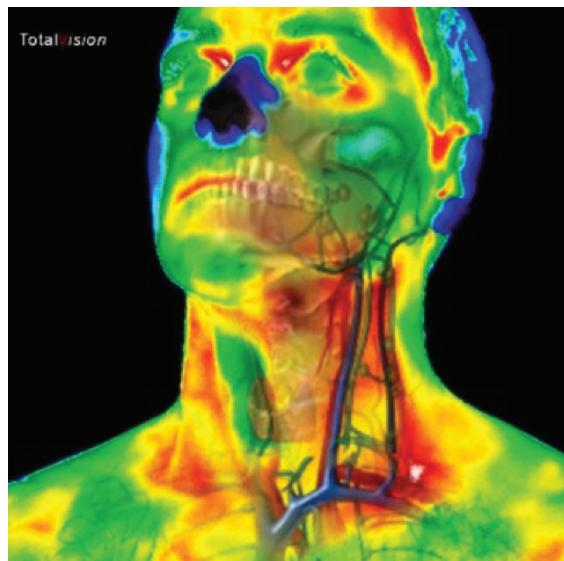
**Figure 1-16** Comparison of black-and-white visible-light photography with an IR thermal image of the same scene.

availability and use of a cryogenic agent, such as liquid nitrogen, as well as placing the detectors in a vacuum-sealed container. Consequently, cooled IR imagers are significantly more expensive to construct and operate than uncooled imagers.

We close this section with two image examples. **Figure 1-16** compares the image of a scene recorded by a visible-light black-and-white camera with a thermal IR image of the same scene. The IR image is in pseudocolor, with red representing high IR emission and blue representing (comparatively) low IR emission. The two images convey different types of information, but they also have significantly different spatial resolutions. Today, digital cameras with 16 megapixel detector arrays are readily available and fairly inexpensive. In contrast, most standard

detector arrays of thermal IR imagers are under 1 megapixel in size. Consequently, IR images appear “blurry” when compared with their photographic counterparts.

Our second image, shown in [Fig. 1-17](#), is an IR thermal image of a person’s head and neck. Such images are finding increased use in medical diagnostics, particularly for organs close to the surface [Ring and Ammer, 2012].



**Figure 1-17** Thermal IR image of a person’s head and neck.

**Concept Question 1-1:** What is a camera’s point spread function? What role does it play in the image formation process?

**Concept Question 1-2:** How are the image and scene spatial resolutions related to one another?

**Concept Question 1-3:** What is the emissivity of an object?

**Concept Question 1-4:** Why is an IR imager called a thermal imager?

**Exercise 1-1:** An imaging lens used in a digital camera has a diameter of 25 mm and a focal length of 50 mm. Considering only the photodetectors responsive to the red band centered at  $\lambda = 0.6 \mu\text{m}$ , what is the camera’s spatial resolution in the image plane, given that the image distance from the lens is  $d_i = 50.25 \text{ mm}$ ? What is the corresponding resolution in the object plane?

**Answer:**  $\Delta y_{\min} = 1.47 \mu\text{m}$ ;  $\Delta y'_{\min} = 0.3 \text{ mm}$ .

**Exercise 1-2:** At  $\lambda = 10 \mu\text{m}$ , what is the ratio of the emissivity of a snow-covered surface relative to that of a sand-covered surface? (See [Fig. 1-14](#).)

**Answer:**  $e_{\text{snow}}/e_{\text{sand}} \approx 0.985/0.9 = 1.09$ .

## 1-2 Radar Imagers

Conceptually, a radar can generate an image of the reflectivity of a scene by scanning its antenna beam across the scene in a raster-like format, as depicted in [Fig. 1-18](#). Even though the imaging process is very different from the process used by a lens in a camera, the radar and the camera share the same fundamental relationship for angular resolution. In Section 1-1.1, we stated in Eq. (1.9a) that the angular resolution of a converging lens is approximately  $\Delta\theta_{\min} = 1.22\lambda/D$ , and the corresponding spatial resolution is

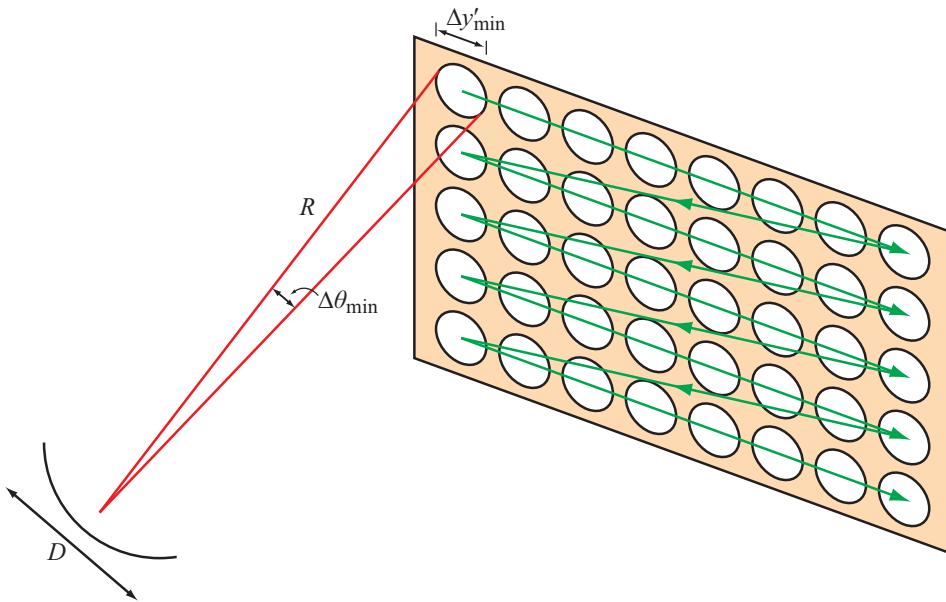
$$\Delta y'_{\min} = d_o \Delta\theta_{\min} = 1.22d_o \frac{\lambda}{D} \quad (\text{camera}). \quad (1.10a)$$

Here,  $\lambda$  is the wavelength of the light and  $D$  is the diameter of the lens.

Equation (1.10a) is approximately applicable to a microwave radar with a dish antenna of diameter  $D$  (in the camera case, the scene illumination is external to the camera, so the lens gets involved in only the receiving process, whereas in the radar case the antenna is involved in both the transmitting and receiving processes). In the radar literature, the symbol usually used to denote the **range** between the radar antenna and the target is the symbol  $R$ . Hence, upon replacing  $d_o$  with  $R$ , we have

$$\Delta y'_{\min} \approx R \frac{\lambda}{D} \quad (\text{radar}). \quad (1.10b)$$

It is important to note that  $\lambda$  of visible light is much shorter than  $\lambda$  in the microwave region. In the middle of the visible spectrum,  $\lambda_{\text{vis}} \approx 0.5 \mu\text{m}$ , whereas at a typical microwave radar



**Figure 1-18** Radar imaging of a scene by raster scanning the antenna beam.

frequency of 6 GHz,  $\lambda_{\text{mic}} \approx 5 \text{ cm}$ . The ratio is

$$\frac{\lambda_{\text{mic}}}{\lambda_{\text{vis}}} = \frac{5 \times 10^{-2}}{0.5 \times 10^{-6}} = 10^5 !$$

This means that the angular resolution capability of an optical system is on the order of 100,000 times better than the angular resolution of a radar, if the lens diameter is the same size as the antenna diameter.

To fully compensate for the large wavelength ratio, a radar antenna would need a diameter on the order of 1 km to produce an image with the same resolution as a camera with a lens 1 cm in diameter. Clearly, that is totally impractical. In practice, most radar antennas are on the order of centimeters to meters in size, but certainly not kilometers. Yet, radar can image the Earth surface from satellite altitudes with spatial resolutions on the order of 1 m—equivalent to antenna sizes several kilometers in extent! How is that possible?

### A. Synthetic-Aperture Radar

As we will see shortly, a **synthetic-aperture radar (SAR)** uses a **synthesized aperture** to achieve good resolution in one dimension

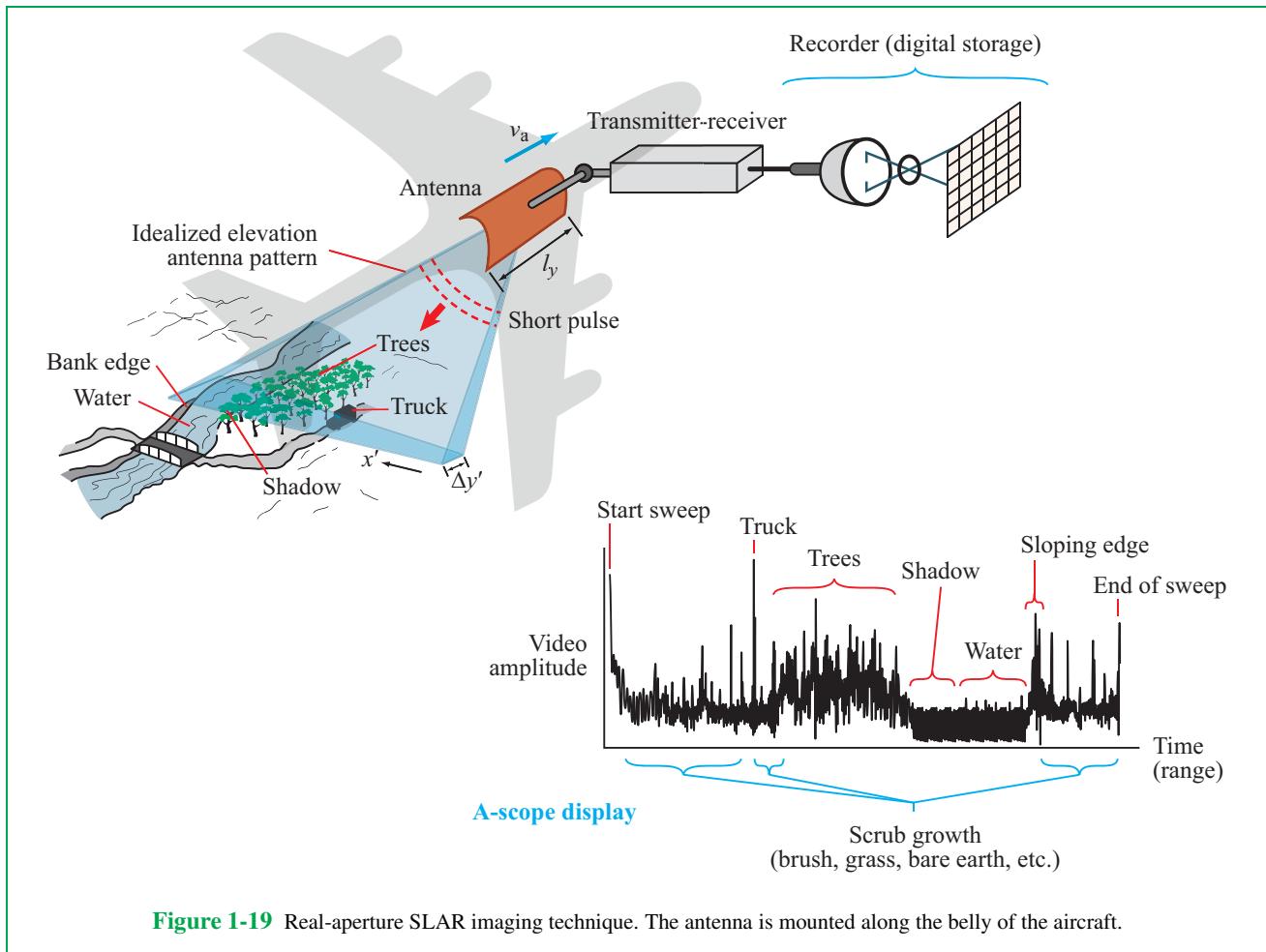
and transmits very short pulses to achieve fine resolution in the orthogonal dimension. The predecessor to SAR is the real-aperture **side-looking airborne radar (SLAR)**. A SLAR uses a rectangular- or cylindrical-shaped antenna that gets mounted along the longitudinal direction of an airplane, and pointed partially to the side (**Fig. 1-19**).

Even though the antenna beam in the elevation direction is very wide, fine discrimination can be realized along the  $x'$  direction in **Fig. 1-19** by transmitting a sequence of very short pulses. At any instant in time, the extent of the pulse along  $x'$  is

$$\Delta x'_{\min} = \frac{c\tau}{2 \sin \theta} \quad (\text{scene range resolution}), \quad (1.11)$$

where  $c$  is the velocity of light,  $\tau$  is the pulse width, and  $\theta$  is the incidence angle relative to nadir-looking. This represents the **scene spatial resolution** capability along the  $x'$  direction. At a typical angle of  $\theta = 45^\circ$ , the spatial resolution attainable when transmitting pulses each 5  $\mu\text{s}$  in width is

$$\Delta x'_{\min} = \frac{3 \times 10^8 \times 5 \times 10^{-9}}{2 \sin 45^\circ} \approx 1.05 \text{ m.}$$



**Figure 1-19** Real-aperture SLAR imaging technique. The antenna is mounted along the belly of the aircraft.

Not only is this an excellent spatial resolution along the  $x'$  direction, but it is also independent of range  $R$  (distance between the radar and the surface), which means it is equally applicable to a satellite-borne radar.

As the aircraft flies along the  $y$  direction, the radar beam sweeps across the terrain, while constantly transmitting pulses, receiving their echoes, and recording them on an appropriate medium. The sequential echoes are then stitched together to form an image.

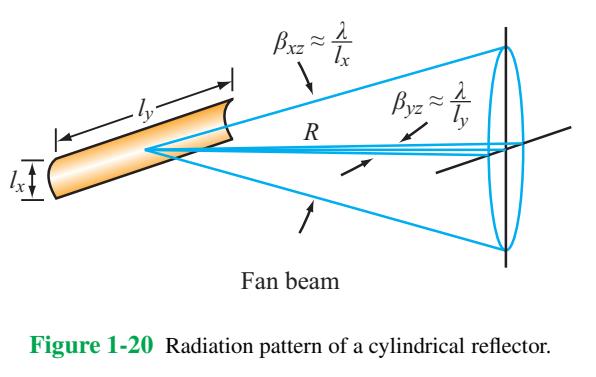
By designing the antenna to be as long as practicable along the airplane velocity direction, the antenna pattern exhibits a relatively narrow beam along that direction ( $y'$  direction in

**Fig. 1-19**). The shape of the beam of the cylindrical antenna is illustrated in **Fig. 1-20**. From range  $R$ , the extent of the beam along the  $y$  direction is

$$\Delta y'_{\min} \approx \frac{\lambda}{l_y} R = \frac{\lambda h}{l_y \cos \theta}, \quad (1.12)$$

**(real-aperture azimuth resolution)**

where  $h$  is the aircraft altitude. This is the spatial resolution capability of the radar along the flight direction. For a 3 m long antenna operating at  $\lambda = 3$  cm from an altitude of 1 km, the



**Figure 1-20** Radiation pattern of a cylindrical reflector.

resolution  $\Delta y'_\text{min}$  at  $\theta = 45^\circ$  is

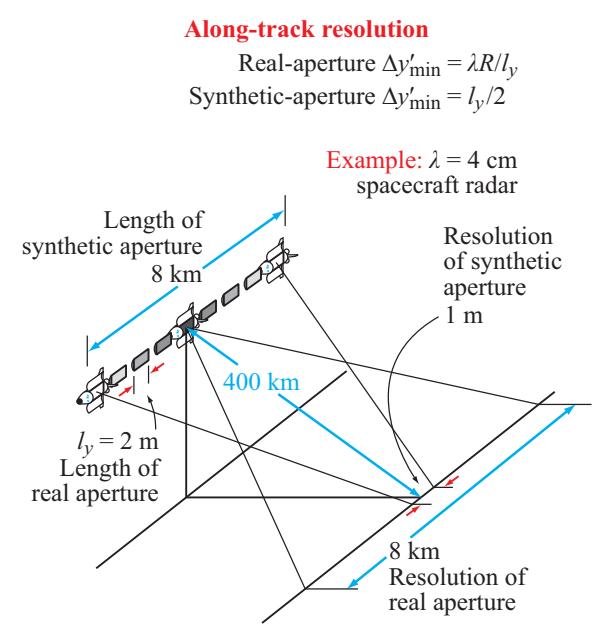
$$\Delta y'_\text{min} \approx \frac{3 \times 10^{-2}}{3 \cos 45^\circ} \times 10^3 \approx 14 \text{ m.}$$

Ideally, an imaging system should have similar resolution capabilities along both directions of the imaged scene. In the present case,  $\Delta x'_\text{min} \approx 1.05 \text{ m}$ , which is highly desirable, but  $\Delta y'_\text{min} \approx 14 \text{ m}$ , which for most imaging applications is not so desirable, particularly if the altitude  $h$  is much higher than 1 km. Furthermore, since  $\Delta y'_\text{min}$  is directly proportional to the altitude  $h$  of the flying vehicle, whereas  $\Delta x'_\text{min}$  is independent of  $h$ , the disparity between  $\Delta x'_\text{min}$  and  $\Delta y'_\text{min}$  will get even greater when we consider radars flown at satellite altitudes.

To improve the resolution  $\Delta y'_\text{min}$  and *simultaneously remove its dependence on the range R*, we artificially create an array of antennas as depicted in **Fig. 1-21**. In the example shown in **Fig. 1-21**, the *real* satellite-borne radar antenna is 2 m long and the synthetic aperture is 8 km long! The latter consists of pulse returns recorded as the real antenna travels over a distance of 8 km, and then processed later as if they had been received by an 8 km long array of antennas, each 2 m long, simultaneously. The net result of the processing is an image with a resolution along the  $y$  direction given by

$$\Delta y'_\text{min} = \frac{l}{2} \quad (\text{SAR azimuth resolution}), \quad (1.13)$$

where  $l$  is the length of the real antenna. For the present example,  $l = 2 \text{ m}$  and  $\Delta y'_\text{min} = 1 \text{ m}$ , which is approximately the same as  $\Delta x'_\text{min}$ . Shortening the antenna length would improve the azimuth resolution, but considerations of signal-to-noise ratio would require the transmission of higher power levels.



**Figure 1-21** An illustration of how synthetic aperture works.

## B. Point Spread Function

The first of our two SAR-image examples displays a large part of Washington, D.C. (**Fig. 1-22**). The location information of a particular pixel in the observed scene is computed, in part, from the round-trip travel time of the transmitted pulse. Consequently, a target that obscures the ground beneath it, such as the Washington Monument in **Fig. 1-22**, ends up generating a **radar shadow** because no signal is received from the obscured area. The radar shadow of the obelisk of the Washington Monument appears as a dark line projected from the top onto the ground surface.

Radar shadow is also apparent in the SAR image of the plane and helicopter of **Fig. 1-23**.

In Section 1-1.1, we stated that the image formed by the lens of an optical camera represents the convolution of the reflectivity of the scene (or the emission distribution in the case of the IR imager) with the point spread function (PSF) of the imaging system. The concept applies equally well to the imaging radar case. For an  $x$ - $y$  SAR image with  $x$  denoting the side-looking direction and  $y$  denoting the flight direction, the **SAR PSF** is



**Figure 1-22** SAR image collected over Washington, D.C. Right of center is the Washington Monument, though only the shadow of the obelisk is readily apparent in the image. [Courtesy of Sandia National Laboratories.]

given by

$$h(x, y) = h_x(x) h_y(y), \quad (1.14)$$

with  $h_x(x)$  describing the shape of the transmitted pulse and  $h_y(y)$  describing the shape of the synthetic antenna-array pattern. Typically, the pulse shape is like a Gaussian:

$$h_x(x) = e^{-2.77(x/\tau)^2}, \quad (1.15a)$$

where  $\tau$  is the effective width of the pulse (width between half-peak points). The synthetic array pattern is sinc-like in shape, but the sidelobes may be suppressed further by assigning different weights to the processed pulses. For the equally weighted case,

$$h_y(y) = \text{sinc}^2\left(\frac{1.8y}{l}\right), \quad (1.15b)$$

where  $l$  is the length of the real antenna, and the sinc function is defined such that  $\text{sinc}(z) = \sin(\pi z)/(\pi z)$  for any variable  $z$ .

**Concept Question 1-5:** Why is a SAR called a “synthetic”-aperture radar?

**Concept Question 1-6:** What system parameters determine the PSF of a SAR?



**Figure 1-23** High-resolution image of an airport runway with a plane and helicopter. [Courtesy of Sandia National Laboratories.]

**Exercise 1-3:** With reference to the diagram in [Fig. 1-21](#), suppose the length of the real aperture were to be increased from 2 m to 8 m. What would happen to (a) the antenna beamwidth, (b) length of the synthetic aperture, and (c) the SAR azimuth resolution?

**Answer:** (a) Beamwidth is reduced by a factor of 4, (b) synthetic aperture length is reduced from 8 km to 2 km, and (c) SAR resolution changes from 1 m to 4 m.

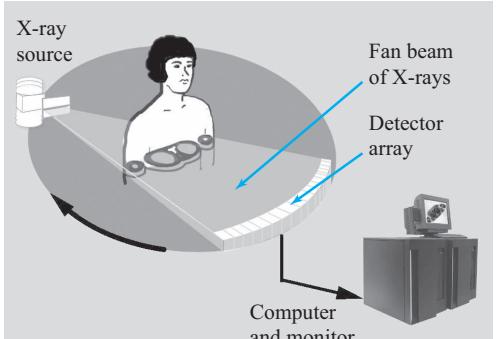
## 1-3 X-Ray Computed Tomography (CT)

**Computed tomography**, also known as **CT scan**, is a technique capable of generating 3-D images of the X-ray attenuation (absorption) properties of an object, such as the human body. The X-ray absorption coefficient of a material is strongly dependent on the density of that material. CT has the sensitivity necessary to image body parts across a wide range of densities, from soft tissue to blood vessels and bones.

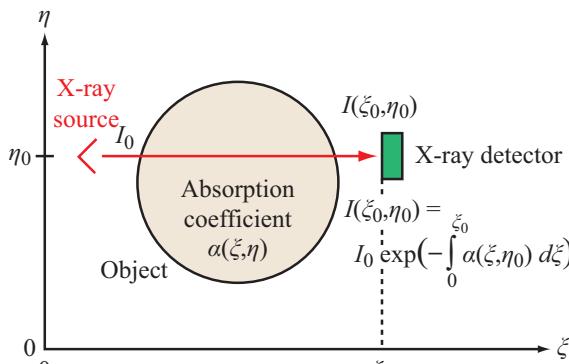
As depicted in [Fig. 1-24\(a\)](#), a CT scanner uses an X-ray source, with a narrow slit to generate a fan-beam, wide enough to encompass the extent of the body, but only about 1 mm thick. The attenuated X-ray beam is captured by an array of ~ 900 detectors. The X-ray source and the detector array are mounted on a circular frame that rotates in steps of a fraction of a degree over a full 360° circle around the object or patient, each time recording an X-ray attenuation profile from a different angular direction. Typically, on the order of 1000 such profiles are recorded, each composed of measurements by 900 detectors. For each horizontal **slice** of the body, the process is completed in less than 1 second. CT uses image reconstruction algorithms to generate a 2-D image of the absorption coefficient of that horizontal slice. To image an entire part of the body, such as the chest or head, the process is repeated over multiple slices (layers).

For each anatomical slice, the CT scanner generates on the order of  $9 \times 10^5$  measurements (1000 angular orientations  $\times$  900 detectors). In terms of the coordinate system shown in [Fig. 1-24\(b\)](#), we define  $\alpha(\xi, \eta)$  as the **absorption coefficient** of the object under test at location  $(\xi, \eta)$ . The X-ray beam is directed along the  $\xi$  direction at  $\eta = \eta_0$ . The X-ray intensity received by the detector located at  $\xi = \xi_0$  and  $\eta = \eta_0$  is given by

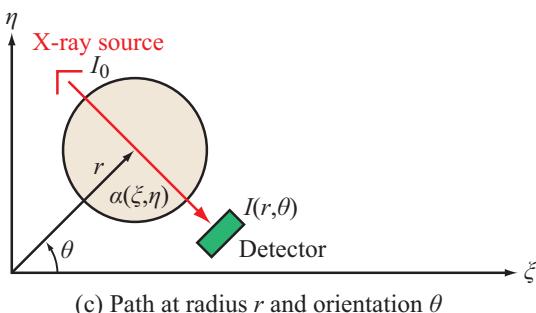
$$I(\xi_0, \eta_0) = I_0 \exp \left[ - \int_0^{\xi_0} \alpha(\xi, \eta_0) d\xi \right], \quad (1.16)$$



(a) CAT scanner



(b) Horizontal path



(c) Path at radius r and orientation theta

**Figure 1-24** (a) CT scanner, (b) X-ray path along  $x$ , and (c) X-ray path along arbitrary direction.

where  $I_0$  is the X-ray intensity radiated by the source. Outside the body,  $\alpha(\xi, \eta) = 0$ . The corresponding logarithmic **path attenuation**  $p(\xi_0, \eta_0)$  is defined as

$$p(\xi_0, \eta_0) = -\log \frac{I(\xi_0, \eta_0)}{I_0} = \int_0^{\xi_0} \alpha(\xi, \eta_0) d\xi. \quad (1.17)$$

The path attenuation  $p(\xi_0, \eta_0)$  is the integrated absorption coefficient across the X-ray path.

In the general case, the path traversed by the X-ray source is at a range  $r$  and angle  $\theta$  in a polar coordinate system, as depicted in **Fig. 1-24(c)**. The direction of the path is orthogonal to the direction of  $r$ . For a path corresponding to a specific set  $(r, \theta)$ , Eq. (1.17) becomes

$$p(r, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \alpha(\xi, \eta) \delta(r - \xi \cos \theta - \eta \sin \theta) d\xi d\eta, \quad (1.18)$$

where the Dirac impulse  $\delta(r - \xi \cos \theta - \eta \sin \theta)$  dictates that only those points in the  $(\xi, \eta)$  plane that fall along the path specified by fixed values of  $(r, \theta)$  are included in the integration.

The relation between  $p(r, \theta)$  and  $\alpha(\xi, \eta)$  is known as the 2-D **Radon transform** of  $\alpha(\xi, \eta)$ . The goal of CT is to reconstruct  $\alpha(\xi, \eta)$  from the measured path attenuations  $p(r, \theta)$ , by inverting the Radon transform given by Eq. (1.18), which is accomplished with the help of the Fourier transform.

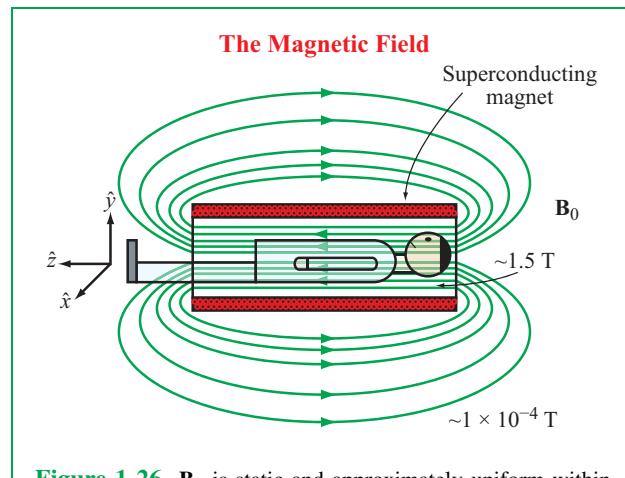
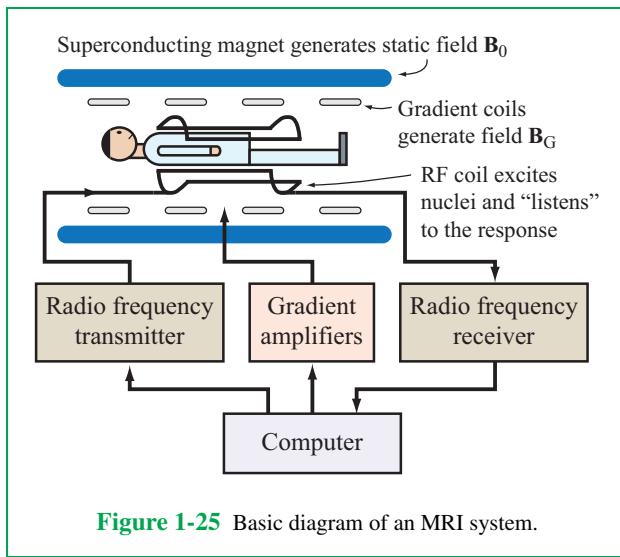
**Concept Question 1-7:** What physical attribute of the imaged body is computed and displayed by a CT scanner?

## 1-4 Magnetic Resonance Imaging

Since its early demonstration in the 1970s, **magnetic resonance imaging (MRI)** has become a highly valuable tool in diagnostic radiology, primarily because it can generate high-resolution anatomical images of the human body, without exposing the patient to ionizing radiation. Like X-ray CT scanners, **magnetic resonance (MR)** imagers can generate 3-D images of the body part of interest, from which **2-D slices** can be extracted along any orientation of interest. The name MRI derives from the fact that the MRI scanner measures **nuclear magnetic resonance (NMR)** signals emitted by the body's tissues and blood vessels in response to excitation by a magnetic field introduced by a radio frequency (RF) system.

### 1-4.1 Basic System Configuration

The MRI system shown in **Fig. 1-25** depicts a human body lying



**Figure 1-26**  $\mathbf{B}_0$  is static and approximately uniform within the cavity. Inside the cavity,  $\mathbf{B}_0 \approx 1.5 \text{ T}$  (teslas), compared with only 0.1 to 0.5 milliteslas outside.

inside a magnetic core. The magnetic field at a given location  $(x, y, z)$  within the core and at a given instant in time  $t$  may consist of up to three magnetic field contributions:

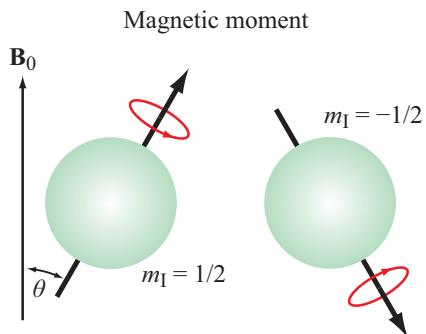
$$\mathbf{B} = \mathbf{B}_0 + \mathbf{B}_G + \mathbf{B}_{RF},$$

where  $\mathbf{B}_0$  is a **static field**,  $\mathbf{B}_G$  is the **field gradient**, and  $\mathbf{B}_{RF}$  is the **radio frequency (RF) excitation** used to solicit a response from the biological material placed inside the core volume. Each of these three components plays a critical role in making MRI possible, so we will discuss them individually.

### A. Static Field $\mathbf{B}_0$

Field  $\mathbf{B}_0$  is a strong, static (non-time varying) magnetic field created by a magnet designed to generate a uniform (constant) distribution throughout the magnetic core (**Fig. 1-26**). Usually, a superconducting magnet is used for this purpose because it can generate magnetic fields with much higher magnitudes than can be realized with resistive and permanent magnets. The direction of  $\mathbf{B}_0$  is longitudinal ( $\hat{z}$  direction in **Fig. 1-26**) and its magnitude is typically on the order of 1.5 teslas (T). The conversion factor between teslas and gauss is  $1 \text{ T} = 10^4 \text{ gauss}$ . Earth's magnetic field is on the order of 0.5 gauss, so  $\mathbf{B}_0$  inside the MRI core is on the order of 30,000 times that of Earth's magnetic field.

Biological tissue is composed of chemical compounds, and each compound is organized around the nuclei (protons) of the atoms comprising that compound. Some, but not all, nuclei be-



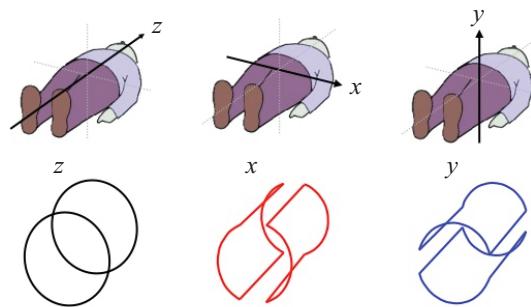
**Figure 1-27** Nuclei with spin magnetic number of  $\pm 1/2$  precessing about  $\mathbf{B}_0$  at the Larmor angular frequency  $\omega_0$ .

come **magnetized** when exposed to a magnetic field. Among the substances found in a biological material, the hydrogen nucleus has a strong susceptibility to magnetization, and hydrogen is highly abundant in biological tissue. For these reasons, a typical MR image is related to the concentration of hydrogen nuclei.

The strong magnetic field  $\mathbf{B}_0$  causes the nuclei of the material inside the core space to temporarily magnetize and to spin (precess) like a top about the direction of  $\mathbf{B}_0$ . The precession orientation angle  $\theta$ , shown in **Fig. 1-27**, is determined by the **spin quantum number**  $I$  of the spinning nucleus and the **magnetic**

**Table 1-2** Gyromagnetic ratio  $\gamma$  for biological nuclei.

Isotope	Spin $I$	% Abundance	$\gamma$ MHz/T
$^1\text{H}$	1/2	99.985	42.575
$^{13}\text{C}$	1/2	1.108	10.71
$^{14}\text{N}$	1	99.63	3.078
$^{17}\text{O}$	5/2	0.037	5.77
$^{19}\text{F}$	1/2	100	40.08
$^{23}\text{Na}$	3/2	100	11.27
$^{31}\text{P}$	1/2	100	17.25

**Figure 1-28** Magnetic coils used to generate magnetic fields along three orthogonal directions. All three gradient fields  $\mathbf{B}_G$  point along  $\hat{\mathbf{z}}$ , but their intensities vary linearly along  $x$ ,  $y$ , and  $z$ .

**quantum number**  $m_I$ . A material, such as hydrogen, with a spin system of  $I = 1/2$  has magnetic quantum numbers  $m_I = \pm 1/2$ . Hence, the nucleus may spin along two possible directions defined by  $\cos \theta = m_I / \sqrt{I(I+1)}$ , which yields  $\theta = \pm 54^\circ 44'$  [Liang and Lauterbur, 2000].

The associated **angular frequency** of the nuclear precession is called the **Larmor frequency** and is given by

$$\omega_0 = \gamma \mathbf{B}_0 \quad (\text{Larmor angular frequency}), \quad (1.19\text{a})$$

with  $\omega_0$  in rad/s,  $\mathbf{B}_0$  in teslas (T), and  $\gamma$ , the **gyromagnetic ratio** of the material, in (rad/s)/T. Alternatively, we can express the precession in terms of the frequency  $f_0 = \omega_0 / 2\pi$ , in which case Eq. (1.19a) assumes the equivalent form

$$f_0 = \gamma \mathbf{B}_0 \quad (\text{Larmor frequency}), \quad (1.19\text{b})$$

where  $\gamma = \gamma / 2\pi$ . This fundamental relationship between  $f_0$  and  $\mathbf{B}_0$  is at the heart of what makes magnetic resonance imaging possible. **Table 1-2** provides a list of nuclei of biological interest that have nonzero spin quantum numbers, along with their corresponding gyromagnetic ratios. For the hydrogen isotope  $^1\text{H}$ ,  $\gamma = 42.575$  MHz/T, so the Larmor frequency for hydrogen at  $\mathbf{B}_0 = 1.5$  T is  $f_0 = 63.8625$  MHz, which places it in the RF part of the EM spectrum. Since the human body is made up primarily of water, the most commonly imaged nucleus is hydrogen.

## B. Gradient Field $\mathbf{B}_G$

The MRI system includes three current-activated gradient coils (**Fig. 1-28**) designed to generate magnetic fields pointed along the  $\hat{\mathbf{z}}$  direction—the same as  $\mathbf{B}_0$ , but whose magnitudes exhibit linear spatial variations along the  $\hat{\mathbf{x}}$ ,  $\hat{\mathbf{y}}$ , and  $\hat{\mathbf{z}}$  directions. That is why they are called **gradient fields**. The three coils can be activated singly or in combination. The primary purpose of the

gradient magnetic field is **localization** (in addition to other information that can be extracted about the tissue material contained in the core volume during the activation and deactivation cycles of the gradient fields). Ideally, the gradient fields assume the following spatial variation inside the core volume:

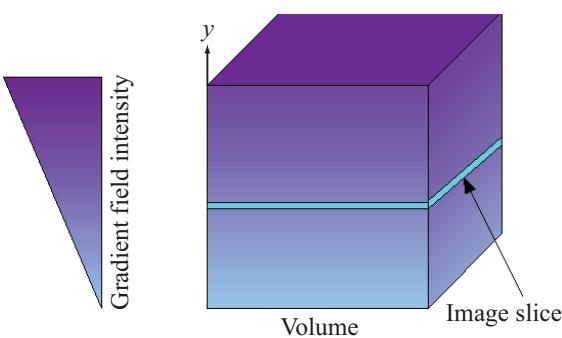
$$\mathbf{B}_G = (G_x \mathbf{x} + G_y \mathbf{y} + G_z \mathbf{z}) \hat{\mathbf{z}},$$

with the center of the  $(x, y, z)$  coordinate system placed at the center of the core volume. The gradient coefficients  $G_x$ ,  $G_y$ , and  $G_z$  are on the order of 10 mT/m, and they are controlled individually by the three gradient coils.

Let us consider an example in which  $G_x = G_z = 0$  and  $G_y = 10$  mT/m, and let us assume that the vertical dimension of the core volume is 1 m. If  $\mathbf{B}_0 = 1.5$  T, the combined field will vary from

$$B = B_0 + G_y y = \begin{cases} 1.495 \text{ T} @ y = -\frac{1}{2} \text{ m}, \text{ to} \\ 1.505 \text{ T} @ y = \frac{1}{2} \text{ m}, \end{cases}$$

as depicted in **Fig. 1-29**. By Eq. (1.19b), the corresponding Larmor frequency for hydrogen will vary from 63.650 MHz for hydrogen nuclei residing in the plane at  $y = -0.5$  m to 64.075 MHz for nuclei residing in the plane at  $y = +0.5$  m. As we will explain shortly, when an RF signal at a particular frequency  $f_{RF}$  is introduced inside the core volume, those nuclei whose Larmor frequency  $f_0$  is the same as  $f_{RF}$  will resonate by absorbing part of the RF energy and then reemitting it at the same frequency (or slightly shifted in the case of certain chemical reactions). The strength of the emitted response is proportional to the density of nuclei. By varying the total magnetic field  $\mathbf{B}$  linearly along



**Figure 1-29** Imposing a gradient field that varies linearly with  $y$  allows stratification into thin slices, each characterized by its own Larmor frequency.

the vertical direction, the total core volume can be discretized into horizontal layers called slices, each corresponding to a different value of  $f_0$  (Fig. 1-29). This way, the RF signal can communicate with each slice separately by selecting the RF frequency to match  $f_0$  of that slice. In practice, instead of sending a sequence of RF signals at different frequencies, the RF transmitter sends out a short pulse whose frequency spectrum covers the frequency range of interest for all the slices in the volume, and then a Fourier transformation is applied to the response from the biological tissue to separate the responses from the individual slices.

The gradient magnetic field along the  $\hat{y}$  direction allows discretization of the volume into  $x$ - $y$  slices. A similar process can be applied to generate  $x$ - $z$  and  $y$ - $z$  slices, and the combination is used to divide the total volume into a three-dimensional matrix of **voxels** (**volume pixels**). The voxel size defines the spatial resolution capability of the MRI system.

### C. RF System

The combination of the strong static field  $\mathbf{B}_0$  and the gradient field  $\mathbf{B}_G$  (whose amplitude is on the order of less than 1% of  $\mathbf{B}_0$ ) defines a specific Larmor frequency for the nuclei of every isotope within each voxel. As we noted earlier through **Table 1-1**, at  $\mathbf{B}_0$  intensities in the 1 T range, the Larmor frequencies of common isotopes are in the MHz range. The RF system consists of a transmitter and a receiver connected to separate coils, or the same coil can be used for both functions. The transmitter generates a burst of narrow RF pulses. In practice, many different pulse configurations are used, depending on the

intended application [Liang and Lauterbur, 2000]. The magnetic field of the transmitted energy causes the exposed biological tissue to resonate at its Larmor frequency. With the transmitter off, the receiver picks up the resonant signals emitted by the biological tissue. The received signals are Fourier transformed so as to establish a one-to-one correspondence to the locations of the voxels responsible for the emission. For each voxel, the strength of the associated emission is related to the density of  $^1\text{H}$  nuclei in that voxel as well as to other parameters that depend on the tissue properties and pulse timing.

### 1-4.2 Point Spread Function

Generating the MR image involves applying the discrete form of the Fourier transform. Accordingly, the point spread function of the MR image is given by a discrete form of the sinc function, namely [Liang and Lauterbur, 2000]:

$$h_x(x) = \Delta k \frac{\sin(\pi N \Delta k x)}{\sin(\pi \Delta k x)}, \quad (1.20)$$

where  $x$  is one of the two MR image coordinates,  $k$  is a spatial frequency,  $\Delta k$  is the sampling interval in  $k$  space, and  $N$  is the total number of Fourier samples. A similar expression applies to  $h_y(y)$ . The spatial resolution of the MR image is equal to the equivalent width of  $h_x(x)$ , which can be computed as follows:

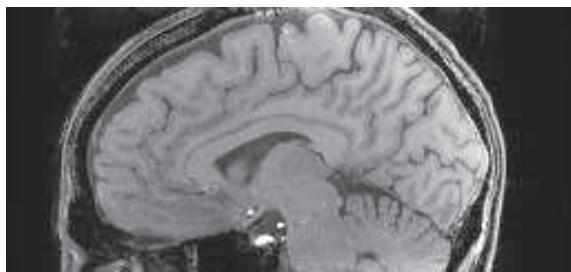
$$\Delta x_{\min} = \frac{1}{h(0)} \int_{-1/(1 \Delta k)}^{1/(1 \Delta k)} h_x(x) dx = \frac{1}{N \Delta k}. \quad (1.21)$$

The integration was performed over one period ( $1/\Delta k$ ) of  $h_x(x)$ . According to Eq. (1.21), the image resolution is inversely proportional to the product  $N \Delta k$ . The choices of values for  $N$  and  $\Delta k$  are associated with signal-to-noise ratio and scan time considerations.

### 1-4.3 MRI-Derived Information

Generally speaking, MRI can provide three types of information about the imaged tissue:

- (a) The magnetic characteristics of tissues, which are related to biological attributes and blood vessel conditions.
- (b) Blood flow, made possible through special time-dependent gradient excitations.
- (c) Chemical properties discerned from measurements of small shifts in the Larmor frequency.



**Figure 1-30** MR image.

An example of an MR image is shown in **Fig. 1-30**.

**Concept Question 1-8:** An MRI system uses three different types of magnetic fields. For what purpose?

**Concept Question 1-9:** What determines the Larmor frequency of a particular biological material?

## 1-5 Ultrasound Imager

Human hearing extends up to 20 kHz. **Ultrasound** is defined as sound at frequencies above that range. Ultrasound imaging systems, which operate in the 2 to 20 MHz range, have numerous industrial and medical applications, and the latter include both diagnosis and therapy. Fundamentally, ultrasound imagers are similar to radar imagers in that both sensors employ **phase shifting** (or, equivalently, **time delaying**) to focus and steer their beams at the desired distances and along the desired directions. Imaging radars use 1-D or 2-D arrays of antennas, and likewise, ultrasound imagers use 1-D or 2-D arrays of **transducers**. However, electromagnetic waves and sound waves have different propagation properties, so the focusing and steering techniques are not quite identical.

### 1-5.1 Ultrasound System Architecture

Ultrasound imagers use both 1-D and 2-D transducer arrays (with some as large as  $2000 \times 8000$  elements and each on the order of  $5 \mu\text{m} \times 5 \mu\text{m}$  in size), but for the sake of simplicity, we show in **Fig. 1-31** only a 1-D array with four elements. The system has a transmitting unit and a receiving unit, with the transducer array connected to the two units through a **transmit/receive switch**.

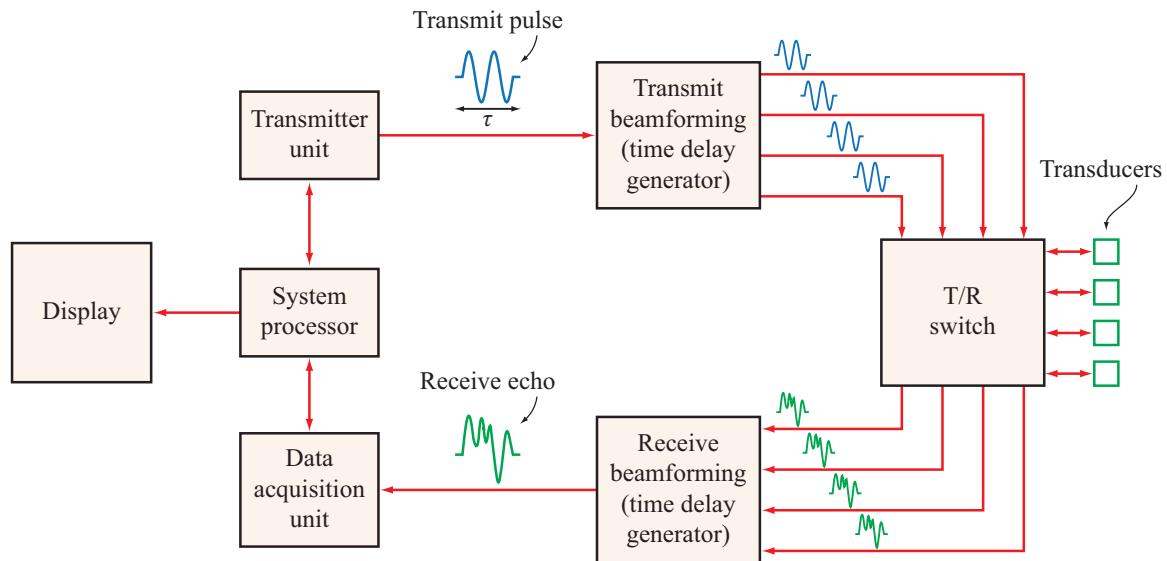
**receive switch.** Thus, the array serves to both launch acoustic waves in response to electrical excitation as well as to receive the consequent acoustic echoes and convert them back into electrical signals. The echoes are reflections from organs and tissue underneath the skin of the body part getting imaged by the ultrasound imager (**Fig. 1-32**).

The transmitter unit in **Fig. 1-31**, often called the **pulsar**, generates a high-voltage short-duration pulse (on the order of a few microseconds in duration) and sends it to the **transmit beamforming unit**, which applies individual time delays to the pulse before passing it on to the transducers through the transmit/receive switch. The choice of time delays determines the range at which the acoustic waves emitted by the four transducers interfere constructively, as well as the direction of that location relative to the axis of the array. The range operation is called **focusing** and the directional operation is called **steering**. Reciprocal operations are performed by the **receive beamforming unit**; it applies the necessary time delays to the individual signals made available by the transducers and then combines them together coherently to generate the receive echo. The focusing and steering operations are the subject of the next subsection.

### 1-5.2 Beam Focusing and Steering

The focusing operation is illustrated by the diagram in **Fig. 1-33** using eight transducers. In response to the electrical stimulations introduced by the beamforming unit, all of the transducers generate outward-going acoustic waves that are identical in every respect except for their phases (time delays). The specific distribution of the time delays shown in **Fig. 1-33(a)** causes the eight acoustic waves to interfere constructively at the point labeled *Focus 1* at range  $R_{f_1}$ . The time-delay distribution is symmetrical relative to the center of the array, so the direction of Focus 1 is broadside to the array axis. Changing the delay shifts between adjacent elements, while keeping the distribution symmetrical, as in **Fig. 1-33(b)**, causes the focal point to move to *Focus 2* at range  $R_{f_2}$ . If no time delay is applied to any of the eight transducer signals, the focal point moves to infinity.

- ▶ The combined beam of the transducer array can be focused as a function of depth by varying the incremental time delay between adjacent elements in a symmetrical time-delay distribution. ◀



**Figure 1-31** Block diagram of an ultrasound system with a 4-transducer array.

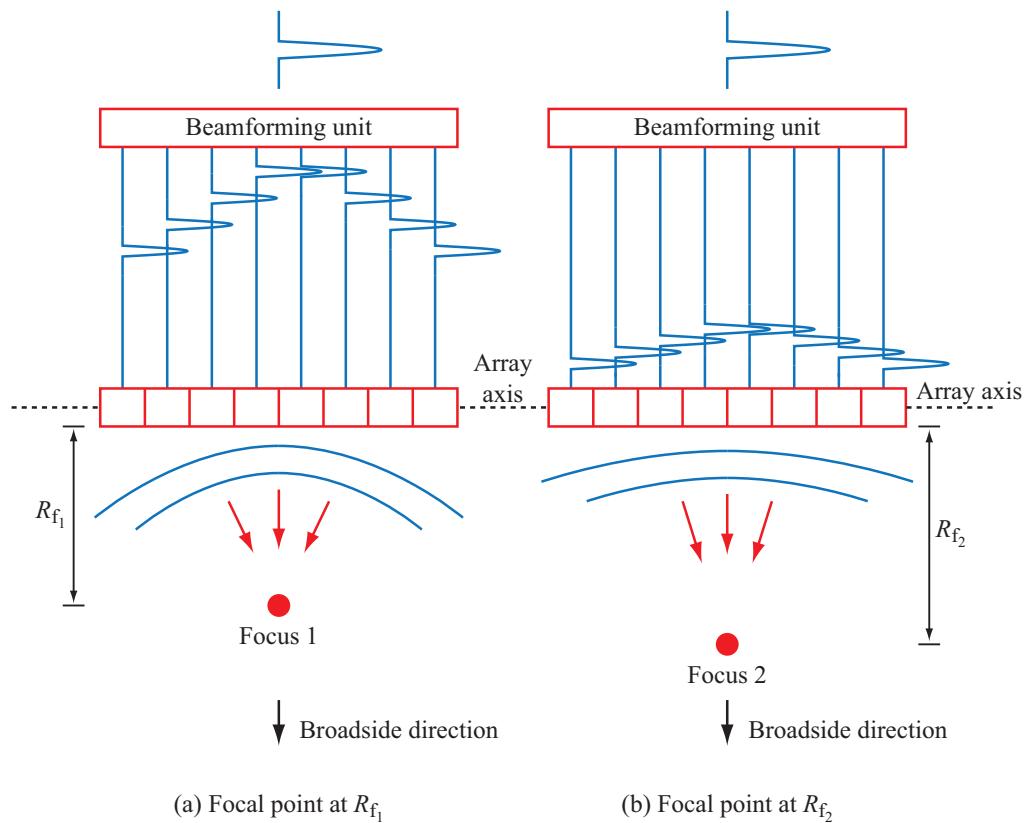


**Figure 1-32** Ultrasound imaging of the thyroid gland.

The image displayed in [Fig. 1-34](#) is a simulation of acoustic energy across two dimensions, the lateral dimension parallel to the array axis and the axial dimension along the range direction. The array consists of 96 elements extending over a length of 20 mm, and the beam is focused at a range  $R = 40$  mm.

The delay-time distribution shown in [Fig. 1-35](#) is symmetrical relative to the broadside direction. By shifting the axis of symmetry to another direction, the focal point moves to a new direction. This is called **steering the beam**, and is illustrated in [Fig. 1-35](#) and simulated in part (b) of [Fig. 1-34](#).

With a 2-D array of transducers, the steering can be realized along two orthogonal directions, so the combination of focusing and steering ends up concentrating the acoustic energy radiated by the transducer array into a small voxel within the body getting imaged by the ultrasound probe. Similar operations are performed by the receive beamforming unit so as to focus and steer the beam of the transducer array to receive the echo from the same voxel.



**Figure 1-33** Changing the inter-element time delay across a symmetrical time-delay distribution shifts the location of the focal point in the range direction.

### 1-5.3 Spatial Resolution

For a 2-D transducer array of size  $(L_x \times L_y)$  and focused at range  $R_f$ , as shown in **Fig. 1-36** (side  $L_y$  is not shown in the figure), the size of the resolution voxel is given by an **axial resolution**  $\Delta R_{\min}$  along the range direction and by **lateral resolutions**  $\Delta x_{\min}$  and  $\Delta y_{\min}$  along the two lateral directions. The axial resolution is given by

$$\Delta R_{\min} = \frac{\lambda N}{2} \quad (\text{axial resolution}), \quad (1.22)$$

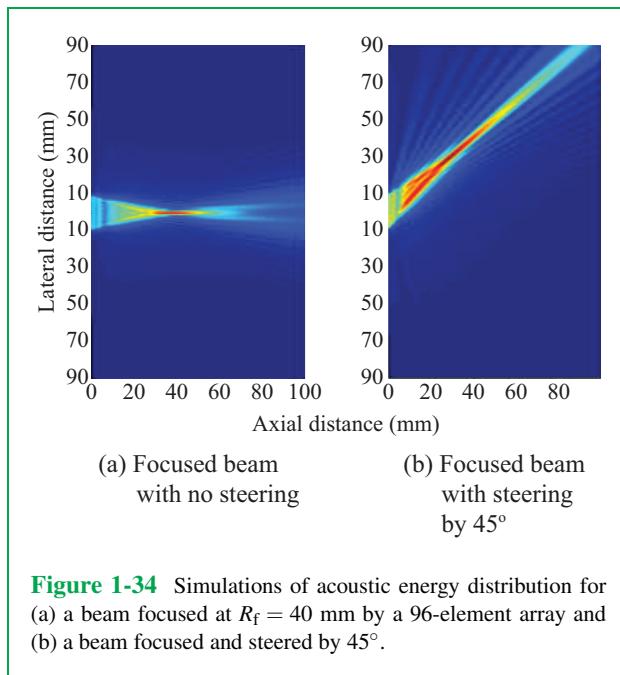
where  $\lambda$  is the **wavelength** of the pulse in the material in which the acoustic waves are propagating and  $N$  is the number of

cycles in the pulse. The wavelength is related to the signal frequency by

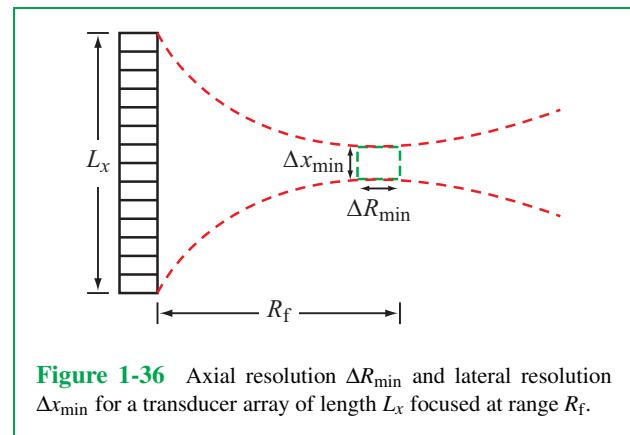
$$\lambda = \frac{v}{f}, \quad (1.23)$$

where  $v$  is the wave velocity and  $f$  is the frequency. In biological tissue,  $v \approx 1540$  m/s. For an ultrasound system operating at  $f = 5$  MHz and generating pulses with  $N = 2$  cycles per pulse,

$$\Delta R_{\min} = \frac{vN}{2f} = \frac{1540 \times 2}{2 \times 5 \times 10^6} \approx 0.3 \text{ mm.}$$



**Figure 1-34** Simulations of acoustic energy distribution for (a) a beam focused at  $R_f = 40$  mm by a 96-element array and (b) a beam focused and steered by 45°.



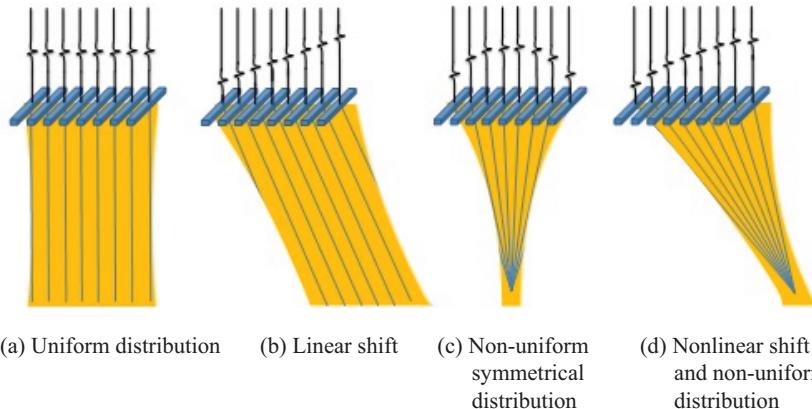
**Figure 1-36** Axial resolution  $\Delta R_{\min}$  and lateral resolution  $\Delta x_{\min}$  for a transducer array of length  $L_x$  focused at range  $R_f$ .

The lateral resolution  $\Delta x_{\min}$  is given by

$$\Delta x_{\min} = R_f \frac{\lambda}{L_x} = \frac{R_f v}{L_x f} \quad (\text{lateral resolution}), \quad (1.24)$$

where  $R_f$  is the focal length (range at which the beam is focused). If the beam is focused at  $R_f = 5$  cm and the array length  $L_x = 4$  cm and  $f = 5$  MHz, then

$$\Delta x_{\min} = \frac{5 \times 10^{-2} \times 1540}{4 \times 10^{-2} \times 5 \times 10^6} \approx 0.4 \text{ mm},$$

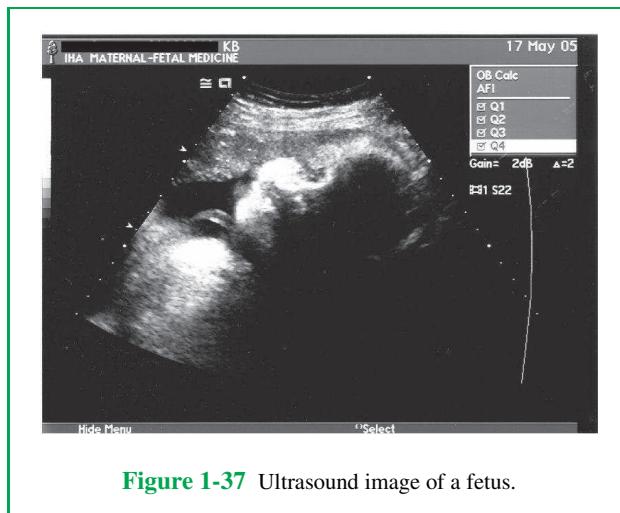


**Figure 1-35** Beam focusing and steering are realized by shaping the time-delay distribution.

which is comparable with the magnitude of the axial resolution  $\Delta R_{\min}$ . The resolution along the orthogonal lateral direction,  $\Delta y_{\min}$ , is given by Eq. (1.24) with  $L_x$  replaced with  $L_y$ . The size of the resolvable voxel is

$$\Delta V = \Delta R_{\min} \times \Delta x_{\min} \times \Delta y_{\min}. \quad (1.25)$$

**Figure 1-37** displays an ultrasound image of a fetus.



**Figure 1-37** Ultrasound image of a fetus.

**Concept Question 1-10:** How does an ultrasound imager focus its beam in the range direction and in the lateral direction? There are two orthogonal lateral directions, so how is that managed?

**Exercise 1-4:** A 6 MHz ultrasound system generates pulses with 2 cycles per pulse using a 5 cm  $\times$  5 cm 2-D transducer array. What are the dimensions of its resolvable voxel when focused at a range of 8 cm in a biological material?

**Answer:**

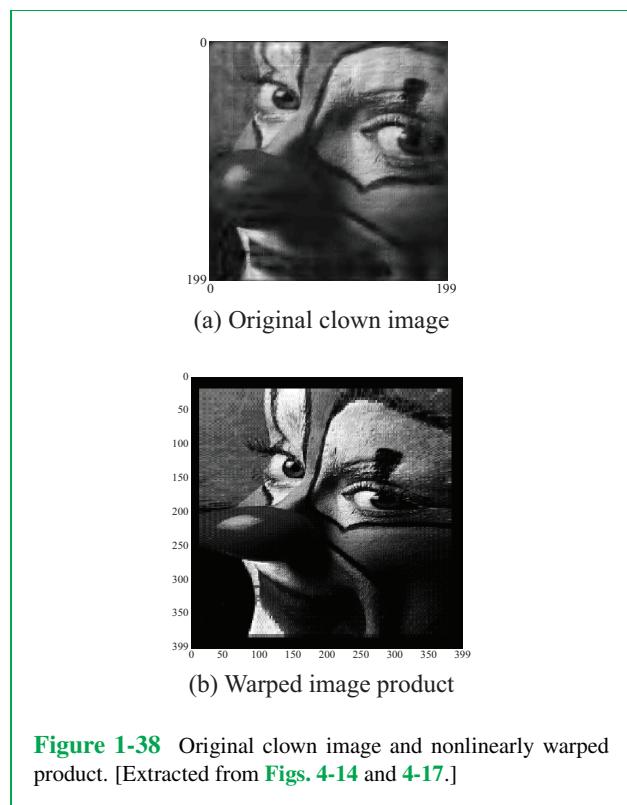
$$\begin{aligned}\Delta V &= \Delta R_{\min} \times \Delta x_{\min} \times \Delta y_{\min} \\ &= 0.26 \text{ mm} \times 0.41 \text{ mm} \times 0.41 \text{ mm}.\end{aligned}$$

## 1-6 Coming Attractions

Through examples of image processing products, this section presents images extracted from various sections in the book. In each case, we present a transformed image, along with a reference to its location within the text.

### 1-6.1 Image Warping by Interpolation

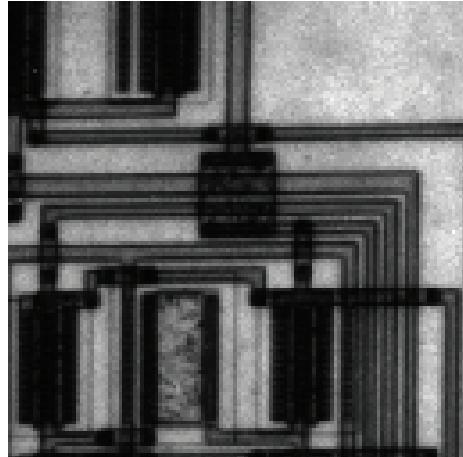
Section 4-10 demonstrates how an image can be warped by nonlinear shifts of its pixel locations and then interpolating the results to generate a smooth image. An example is shown in **Fig. 1-38**.



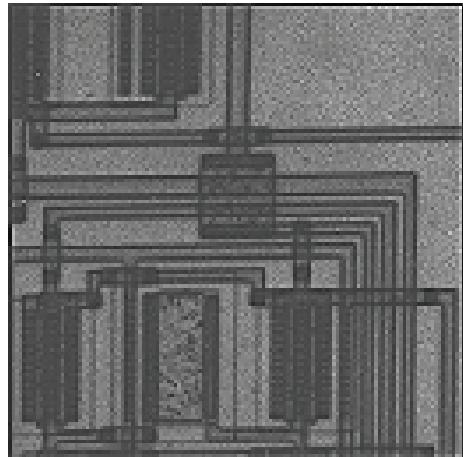
**Figure 1-38** Original clown image and nonlinearly warped product. [Extracted from **Figs. 4-14** and **4-17**.]

### 1-6.2 Image Sharpening by Highpass Filtering

Section 5-2 illustrates how an image can be sharpened by spatial highpass filtering, and how this amplifies noise in the image. The original image of an electronic circuit and its highpass-filtered version are displayed in **Fig. 1-39**.



(a) Original image

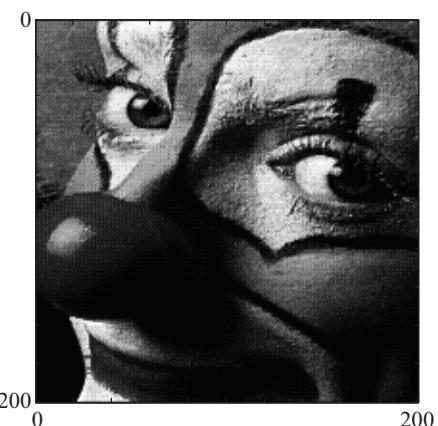


(b) Sharpened image

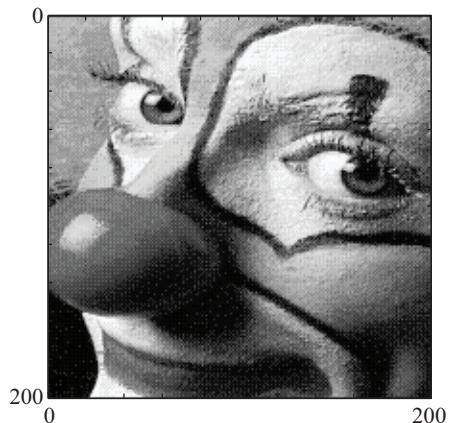
**Figure 1-39** Image of electronic circuit before and after application of a highpass sharpening filter. [Extracted from **Fig. 5-6**.]

### 1-6.3 Brightening by Histogram Equalization

Histogram equalization (nonlinear transformation of pixel values) can be used to brighten an image, as illustrated by the pair of images in **Fig. 1-40**.



(a) Dark clown image

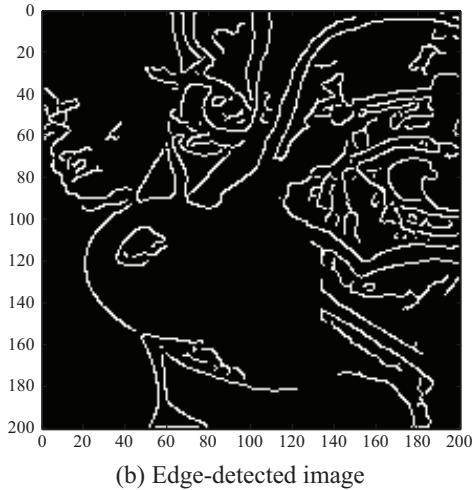
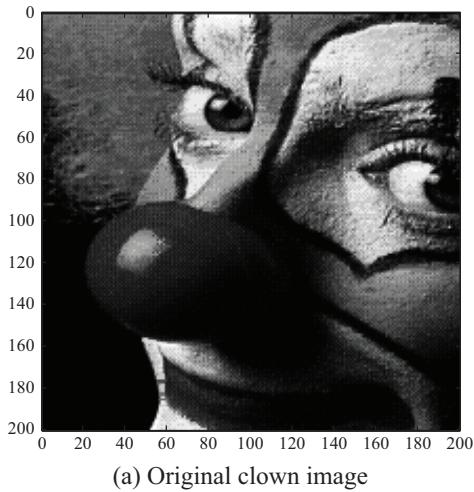


(b) Brightened clown image

**Figure 1-40** Application of histogram equalization to the dark image in (a) leads to the brighter image in (b). [Extracted from **Fig. 5-8**.]

### 1-6.4 Edge Detection

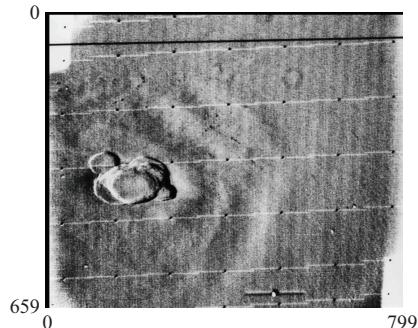
Edges can be enhanced in an image by applying edge detection algorithms, such as the Canny edge detector that was applied to the image in [Fig. 1-41\(a\)](#).



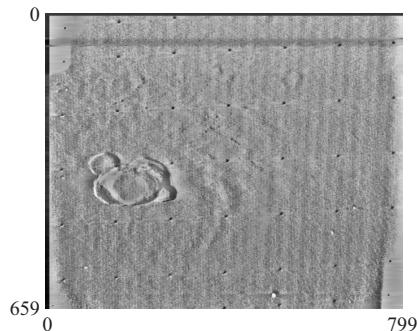
**Figure 1-41** Original clown image and its Canny edge-detected version. [Extracted from [Fig. 5-16](#).]

### 1-6.5 Notch Filtering

Notch filtering can be used to remove sinusoidal interference from an image. The original Mariner space probe image and its notch-filtered version are shown in [Fig. 1-42](#).



(a) Original Mariner image



(b) Notch-filtered Mariner image

**Figure 1-42** The horizontal lines in (a) are due to sinusoidal interference in the recorded image. The lines were removed by applying notch filtering. [Extracted from [Fig. 6-7](#).]

### 1-6.6 Motion-Blur Deconvolution

Section 6-6 shows how to deblur a motion-blurred image. A blurred image and its motion-deblurred version are shown in [Fig. 1-43](#).



(a) Original motion-blurred image caused by taking a photograph in a moving car



(b) Motion deblurred image

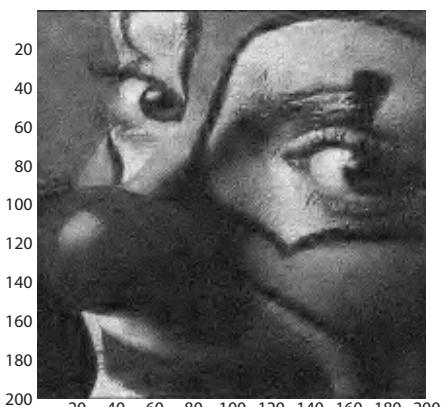
**Figure 1-43** Motion blurring is removed. [Extracted from [Fig. 6-11](#).]

### 1-6.7 Denoising of Images Using Wavelets

One method used to denoise an image is by thresholding and shrinking its wavelet transform. An example is shown in [Fig. 1-44](#).



(a) A noisy clown image

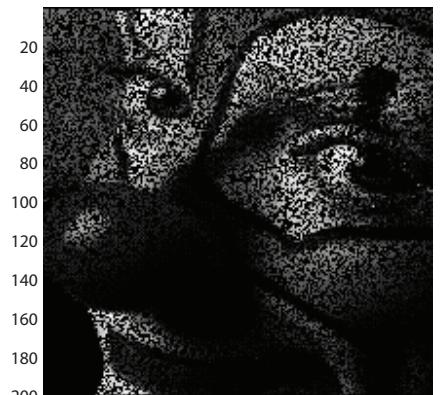


(b) Wavelet-denoised clown image

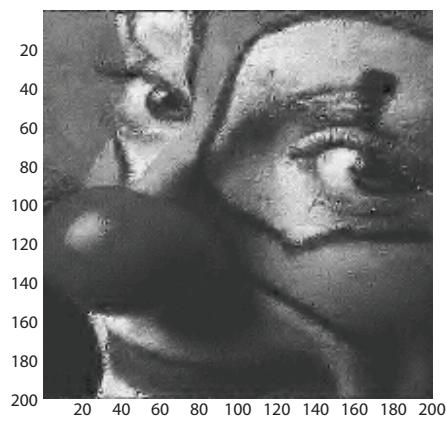
**Figure 1-44** Image denoising. [Extracted from [Fig. 7-21](#).]

### 1-6.8 Image Inpainting

The wavelet transform of an image can be used to “inpaint” (restore missing pixels in an image). An image with deleted pixels and its inpainted version are shown in [Fig. 1-45](#).



(a) Image with missing pixels

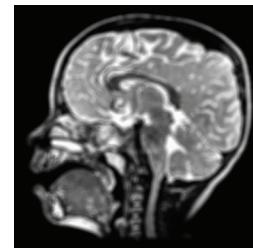


(b) Inpainted image

**Figure 1-45** The image in (b) was created by “filling in” values for the missing pixels in the upper image. [Extracted from [Fig. 7-23](#).]

### 1-6.9 Deconvolution Using Deterministic and Stochastic Image Models

Chapters 8 and 9 provide reviews of probability and estimation. The reason for these reviews is that using a stochastic image model, in which the 2-D power spectral density is modeled as that of a fractal image, can yield much better results in refocusing an image than is possible with deterministic models. An example is shown in [Fig. 1-46](#).



(a) Unfocused MRI image



(b) Deterministically refocused MRI image



(c) Stochastically refocused MRI image

**Figure 1-46** The images in (b) and (c) demonstrate two methods used for refocusing an unfocused image. [Extracted from [Fig. 9-4](#).]

### 1-6.10 Markov Random Fields for Image Segmentation

In a Markov random field (MRF) image model, the value of each pixel is stochastically related to its surrounding values. This is useful in segmenting images, as presented in Section 9-11. **Figure 1-47** illustrates how an MRF image model can improve the segmentation of an X-ray image of a foot into tissue and bone.



(a) Noisy image



(b) Segmented image

**Figure 1-47** Segmenting a noisy image into two distinct classes: bone and tissue. [Extracted from [Fig. 9-14](#).]

### 1-6.11 Motion-Deblurring of a Color Image

Chapters 1–9 consider grayscale (black-and-white) images, since color images consist of three (red, green, blue) images. Motion deblurring of a color image is presented in Section 10-3, an example of which is shown in **Fig. 1-48**.



(a) Motion-blurred Christmas tree

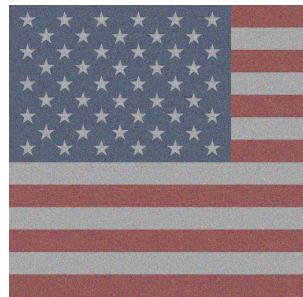


(b) Deblurred Christmas tree

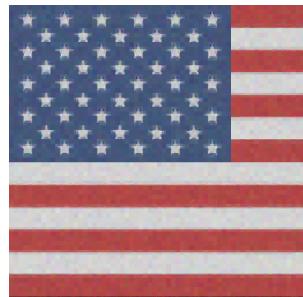
**Figure 1-48** Deblurring a motion-blurred color image. [Extracted from [Fig. 10-11](#).]

### 1-6.12 Wavelet-Based Denoising of a Color Image

Wavelet-based denoising can be used on each color component of a color image, as presented in Section 10-4. An illustration is given in **Fig. 1-49**.



(a) Noisy flag image



(b) Denoised image

**Figure 1-49** Denoising an image of the American flag using wavelet-based denoising. [Extracted from [Fig. 10-12](#).]

### 1-6.13 Histogram Equalization (Brightening) of a Color Image

Color images can be brightened using histogram equalization, as presented in Section 10-5. **Figure 1-50(a)** displays a dark image of a toucan, and part (b) of the same figure displays the result of applying histogram equalization to each color.



(a) Original dark toucan image

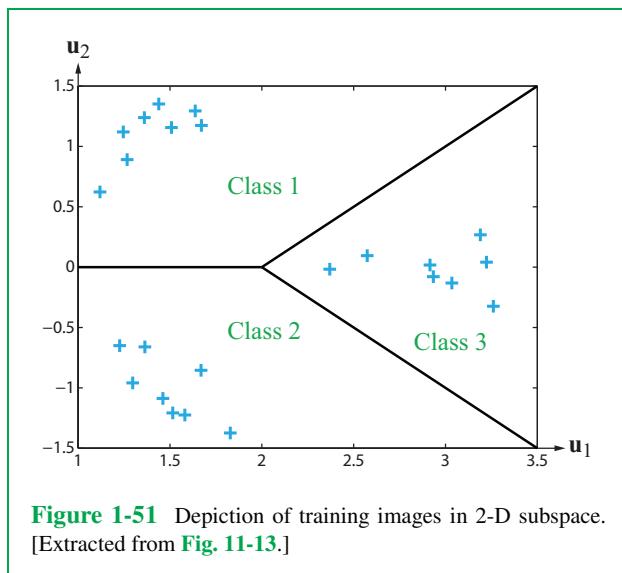


(b) Brightened image

**Figure 1-50** Application of histogram equalization to a color image. [Extracted from [Figs. 10-13](#) and [10-14](#).]

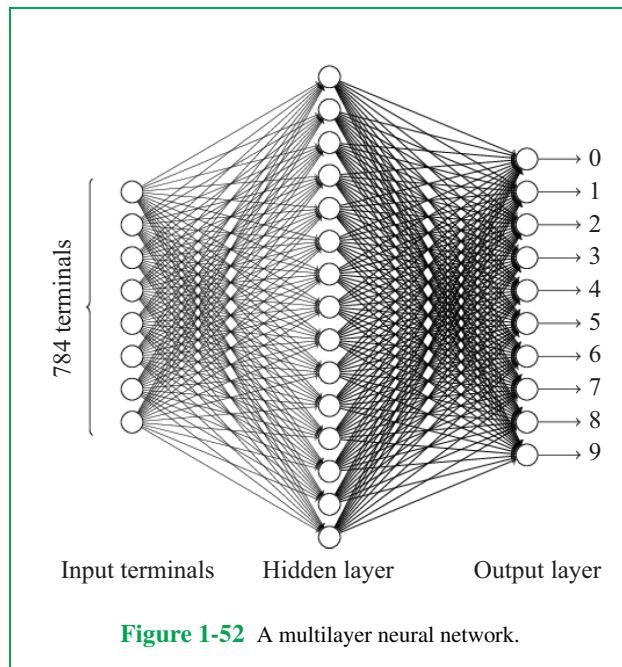
### 1-6.14 Unsupervised Learning

In unsupervised learning, a set of training images is used to determine a set of reference images, which are then used to classify an observed image. The training images are mapped to a subspace spanned by the most significant singular vectors of the singular value decomposition of a training matrix. In **Fig. 1-51**, the training images, depicted by blue “+” symbols, cluster into different image classes.



### 1-6.15 Supervised Learning

Supervised learning by neural networks is presented in Chapter 12. An example of a neural network is shown in **Fig. 1-52**.



## Summary

### Concepts

- Images may be formed using any of these imaging modalities: optical, infrared, radar, x-rays, ultrasound, and magnetic resonance imaging.
- Image processing is needed to process a raw image, formed directly from data, into a final image, which has been deblurred, denoised, interpolated, or enhanced, all of which are subjects of this book.

- Color images are actually triplets of red, green, and blue images, displayed together.
- The effect of an image acquisition system on an image can usually be modelled as 2-D convolution with the point spread function of the system (see below).
- The resolution of an image acquisition system can be computed using various formulae (see below).

### Mathematical Formulae

**Lens law**  $\frac{1}{d_0} + \frac{1}{d_i} = \frac{1}{f}$

**Optical point spread function**

$$h(\theta) = \left[ \frac{2J_1(\gamma)}{\gamma} \right]^2, \quad \gamma = \frac{\pi D}{\lambda} \sin \theta$$

**SAR point spread function**

$$h(x, y) = e^{-2.77(x/\tau)^2} \operatorname{sinc}^2 \left( \frac{1.8y}{l} \right)$$

**MRI point spread function**  $h_x(x) = \Delta k \frac{\sin(\pi N \Delta k x)}{\sin(\pi \Delta k x)}$

**2-D convolution**

$$I_i(x, y) = I_o(x, y) \ast \ast h(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I_o(x - x', y - y') h(x', y') dx' dy'$$

**X-ray tomography path attenuation**

$$p(r, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} a(\xi, \eta) \delta(r - \xi \cos \theta - \eta \sin \theta) d\xi d\eta$$

**Optical resolution**  $\Delta\theta_{\min} \approx 1.22 \frac{\lambda}{D}$

**Radar resolution**  $\Delta y'_{\min} \approx R \frac{\lambda}{D}$

**Ultrasound resolution**  $\Delta R_{\min} = \frac{\lambda N}{2}$

### Important Terms

Provide definitions or explain the meaning of the following terms:

active pixel sensor

liquid crystal display

radar

X-ray computed tomography

beamforming

magnetic resonance imaging (MRI)

resolution

charge-coupled device

optical imaging

synthetic-aperture radar

infrared imaging

point spread function

ultrasound imaging

## PROBLEMS

### Section 1-1: Optical Imagers

**1.1** An imaging lens in a digital camera has a focal length of 6 cm. How far should the lens be from the camera's CCD array to focus on an object

- (a) 12 cm in front of the lens?
- (b) 15 cm in front of the lens?

**1.2** An imaging lens in a digital camera has a focal length of 4 cm. How far should the lens be from the camera's CCD array to focus on an object

- (a) 12 cm in front of the lens?
- (b) 8 cm in front of the lens?

**1.3** The following program loads an image stored in `clown.mat` as  $I_o(x,y)$ , passes it through an imaging system with the PSF given by Eq. (1.6), and displays  $I_o(x,y)$  and  $I_i(x,y)$ . Parameters  $\Delta$ ,  $D$ ,  $d_i$ , and  $\lambda$  (all in mm) are specified in the program's first line.

```
clear;Delta=0.0002;D=0.03;
lambda=0.000005;di=0.003;
T=round(0.01/Delta);
for I=1:T;for J=1:T;
x2y2(I,J)=(I-T/2).* (I-T/2)+(J-T/2) .*
(J-T/2);end;end;
gamma=pi*D/lambda*sqrt(x2y2./(x2y2+di*di/Delta/Delta));
h=2*besselj(1,gamma)./.gamma;
h(T/2,T/2)=(h(T/2+1,T/2)+h(T/2-1,T/2) +
h(T/2,T/2+1)+h(T/2,T/2-1))/4;
h=h.*h;H=h(T/2-5:T/2+5,T/2-5:T/2+5);
load clown.mat;Y=conv2(X,H);
figure,imagesc(X),axis off,colormap(gray),
figure,imagesc(Y),axis off,colormap(gray)
```

Run the program and display  $I_o(x,y)$  (input) and  $I_i(x,y)$  (output).

### Section 1-2: Radar Imagers

**1.4** Compare the azimuth resolution of a real-aperture radar with that of a synthetic-aperture radar, with both pointed at the ground from an aircraft at a range  $R = 5$  km. Both systems operate at  $\lambda = 3$  cm and utilize a 2-m-long antenna.

**1.5** A 2-m-long antenna is used to form a synthetic-aperture radar from a range of 100 km. What is the length of the synthetic aperture?

**1.6** The following program loads an image stored in `sar.mat` as  $I_o(x,y)$ , passes it through an imaging system with the PSF given by Eq. (1.15), and displays  $I_o(x,y)$  and  $I_i(x,y)$ . Parameters  $\Delta$ ,  $\tau$  and  $l$  are specified in the program's first line.

```
clear;Delta=0.1;l=5;tau=1;I=[-15:15];
z=pi*1.8*Delta*I/l;load sar.mat;
hy=sin(pi*z)./(pi*z);hy(16)=1;hy=hy.*hy;
hx=exp(-2.77*Delta*Delta*I.*I/tau/tau);
H=hy'*hx;Y=conv2(X,H);
figure,imagesc(X),axis off,colormap(gray),
figure,imagesc(Y),axis off,colormap(gray)
```

Run the program and display  $I_o(x,y)$  (input) and  $I_i(x,y)$  (output).

### Section 1-3: X-Ray Computed Tomography (CT)

**1.7** (This problem assumes prior knowledge of the 1-D Fourier transform (FT)). The basic CT problem is to reconstruct  $\alpha(\xi, \eta)$  in Eq. (1.18) from  $p(r, \theta)$ . One way to do this is as follows:

- (a) Take the FT of Eq. (1.18), transforming  $r$  to  $f$ . Define  $p(-r, \theta) = p(r, \theta + \pi)$ .
- (b) Define and substitute  $\mu = f \cos \theta$  and  $\nu = f \sin \theta$  in this FT.
- (c) Show that the result defines 2 FTs, transforming  $\xi$  to  $\mu$  and  $\eta$  to  $\nu$ , and that  $\mathbf{A}(\mu, \nu) = \mathbf{P}(f, \theta)$ . Hence,  $\alpha(\xi, \eta)$  is the inverse FT of  $\mathbf{P}(f, \theta)$ .

### Section 1-4: Magnetic Resonance Imaging

**1.8** The following program loads an image stored in `mri.mat` as  $I_o(x,y)$ , passes it through an imaging system with the PSF given by Eq. (1.20), and displays  $I_o(x,y)$  and  $I_i(x,y)$ . Parameters  $\Delta$ ,  $N$ , and  $dk$  are specified in the program's first line.

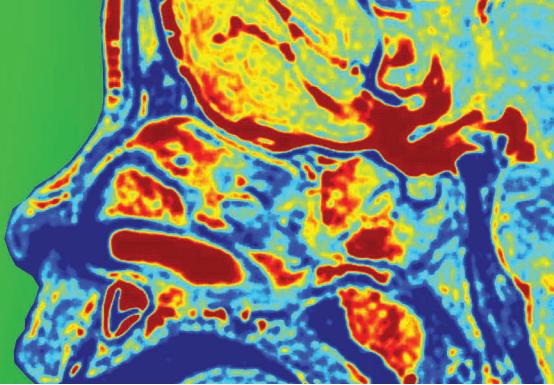
```
clear;N=16;Delta=0.01;dk=1;
I=[-60:60];load mri.mat;
h=dk*sin(pi*N*dk*I*Delta)./.sin(pi*dk*I*Delta);
h(61)=N;H=h'*h;Y=conv2(X,H);
figure,imagesc(X),axis off,colormap(gray),
figure,imagesc(Y),axis off,colormap(gray)
```

Run the program and display  $I_o(x,y)$  (input) and  $I_i(x,y)$  (output).

### Section 1-5: Ultrasound Imager

**1.9** This problem shows how beamforming works on a linear array of transducers, as illustrated in [Fig. 1-35](#), in a medium with a wave speed of 1540 m/s. We are given a linear array of transducers located 1.54 cm apart along the  $x$  axis, with the  $n$ th transducer located at  $x = 1.54n$  cm. Outputs  $\{y_n(t)\}$  from the transducers are delayed and summed to produce the signal  $y(t) = \sum_n y_n(t - 0.05n)$ . In what direction (angle from perpendicular to the array) is the array focused?

# CHAPTER 2



## 2 Review of 1-D Signals and Systems

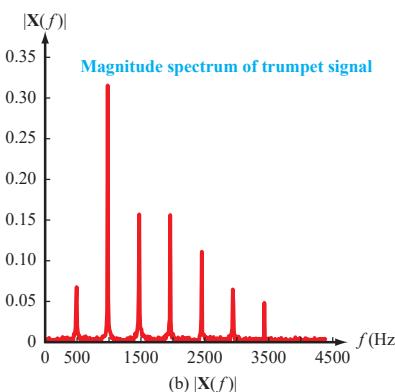
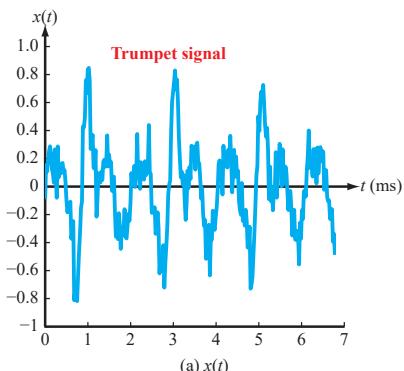
### Contents

- Overview, 39
- 2-1** Review of 1-D Continuous-Time Signals, 41
- 2-2** Review of 1-D Continuous-Time Systems, 43
- 2-3** 1-D Fourier Transforms, 47
- 2-4** The Sampling Theorem, 53
- 2-5** Review of 1-D Discrete-Time Signals and Systems, 59
- 2-6** Discrete-Time Fourier Transform (DTFT), 66
- 2-7** Discrete Fourier Transform (DFT), 70
- 2-8** Fast Fourier Transform (FFT), 76
- 2-9** Deconvolution Using the DFT, 80
- 2-10** Computation of Continuous-Time Fourier Transform (CTFT) Using the DFT, 82
- Problems, 86

### Objectives

Learn to:

- Compute the response of an LTI system to a given input using convolution.
- Compute the frequency response (response to a sinusoidal input) of an LTI system.
- Compute the continuous-time Fourier transform of a signal or impulse response.
- Use the sampling theorem to convert a continuous-time signal to a discrete-time signal.
- Perform the three tasks listed above for continuous-time signals on discrete-time signals.
- Use the discrete Fourier transform (DFT) to denoise, filter, and deconvolve signals.



Many techniques and transforms in image processing are direct generalizations of techniques and transforms in 1-D signal processing. Reviewing these 1-D concepts enhances the understanding of their 2-D counterparts. These include: linear time-invariant (LTI) systems, convolution, frequency response, filtering, Fourier transforms for continuous and discrete-time signals, and the sampling theorem. This chapter reviews these 1-D concepts for generalization to their 2-D counterparts in Chapter 3.

## Overview

Some topics and concepts in 1-D signals and systems generalize directly to 2-D. This chapter provides quick reviews of those topics and concepts in 1-D so as to simplify their repeat presentation in 2-D in future chapters. We assume the reader is already familiar with 1-D signals and systems\*—in both continuous and discrete time, so the presentation in this chapter is more in the form of a *refresher* than an extensive treatment. *Moreover, we limit the coverage to topics that generalize directly from 1-D to 2-D.* These topics include those listed in the box below. Some topics that do *not* generalize readily from 1-D to 2-D include: causality; differential and difference equations; transfer functions; poles and zeros; Laplace and  $\mathbf{z}$ -transforms. Hence, these topics will not be covered in this book.

### 1-D Signals and Systems

- (1) Linear time-invariant (LTI) 1-D systems
- (2) Frequency response of LTI systems
- (3) Impulse response of 1-D systems
- (4) 1-D filtering and convolution
- (5) Sampling theorem in 1-D
- (6) Discrete-time Fourier transform (DTFT)

### 2-D Signals and Systems

- Linear shift-invariant (LSI) 2-D systems
- Spatial frequency response of LSI systems
- Point-spread function of LSI systems
- 2-D filtering and convolution
- Sampling theorem in 2-D
- Discrete-space Fourier transform (DSFT)

---

\*For a review, see *Engineering Signals and Systems in Continuous and Discrete Time*, Ulaby and Yagle, NTS Press, 2016.

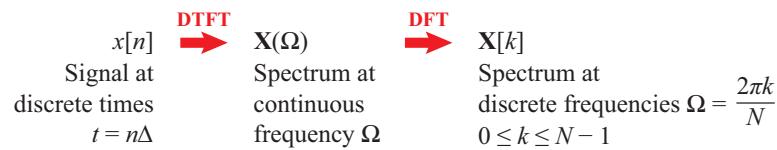
For the sake of clarity, we start this chapter with a synopsis of the terminology and associated symbols used to represent 1-D continuous-time and discrete-time signals and 2-D continuous-space and discrete-space signals, and their associated spectra.

## 1-D Signals

### Continuous Time



### Discrete Time

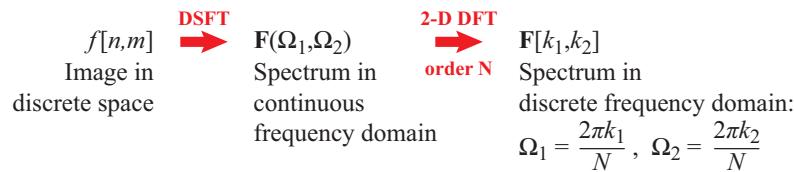


## 2-D Images

### Continuous Space



### Discrete Space



## 2-1 Review of 1-D Continuous-Time Signals

A continuous-time signal is a physical quantity, such as voltage or acoustic pressure, that varies with time  $t$ , where  $t$  is a real number having units of time (usually seconds). Mathematically a continuous-time signal is a function  $x(t)$  of  $t$ . Although  $t$  can also be a spatial variable, we will refer to  $t$  as time, to avoid confusion with the spatial variables used for images.

### 2-1.1 Fundamental 1-D Signals

In [Table 2-1](#), we consider three types of fundamental, continuous-time signals.

#### A. Eternal Sinusoids

An (eternal) sinusoid with amplitude  $A$ , phase angle  $\theta$  (radians), and frequency  $f_0$  (Hz), is described by the function

$$x(t) = A \cos(2\pi f_0 t + \theta), \quad -\infty < t < \infty. \quad (2.1)$$

The period of  $x(t)$  is  $T = 1/f_0$ .

Even though an eternal sinusoid cannot exist physically (since it would extend from before the beginning of the universe until after its end), it is used nevertheless to mathematically describe periodic signals in terms of their Fourier series. Also, another useful aspect of eternal sinusoids is that the response of a [linear time-invariant LTI](#) system (defined in Section 2-2.1) to an eternal sinusoid is another eternal sinusoid at the same frequency as that of the input sinusoid, but with possibly different amplitude and phase. Despite the fact that the sinusoids are eternal (and therefore, unrealistic), they can be used to compute the response of real systems to real signals. Consider, for example, a sinusoid that starts at  $t = 0$  as the input to a stable and causal LTI system. The output consists of a transient response that decays to zero, plus a sinusoid that starts at  $t = 0$ . The output sinusoid has the same amplitude, phase, and frequency as the response that the system would have had to an eternal input sinusoid.

#### B. Pulses

A (rectangular) [pulse](#) of duration  $T$  centered at time  $t_0$  ([Table 2-1](#)) is defined as

$$x(t) = \begin{cases} 1 & \text{for } (t_0 - T/2) < t < (t_0 + T/2), \\ 0 & \text{otherwise.} \end{cases} \quad (2.2)$$

The [rectangle function](#)  $\text{rect}(t)$  is defined as

$$\text{rect}(t) = \begin{cases} 1 & \text{for } -1/2 < t < 1/2, \\ 0 & \text{otherwise.} \end{cases} \quad (2.3a)$$

The pulse  $x(t)$  defined in Eq. (2.2) can be written in terms of  $\text{rect}(t)$  as

$$x(t) = \text{rect}\left(\frac{t - t_0}{T}\right). \quad (2.3b)$$

**Rectangle Pulse**

#### C. Impulses

An [impulse](#)  $\delta(t)$  is defined as a function that has the [sifting](#) property

$$\int_{-\infty}^{\infty} x(t) \delta(t - t_0) dt = x(t_0). \quad (2.4)$$

**Sifting Property**

► Multiplying a function  $x(t)$  that is continuous at  $t = t_0$  by a delayed impulse  $\delta(t - t_0)$  and integrating over  $t$  “sifts out” the value  $x(t_0)$ . ◀

Setting  $x(t) = 1$  in Eq. (2.4) shows that an impulse has an area of unity.

An impulse can be thought of (non-rigorously) as the limiting case of a pulse of width  $T = 2\epsilon$  multiplied by an amplitude  $A = 1/(2\epsilon)$  as  $\epsilon \rightarrow 0$ :

$$\delta(t) = \lim_{\epsilon \rightarrow 0} \frac{1}{2\epsilon} \text{rect}\left(\frac{t}{2\epsilon}\right). \quad (2.5)$$

Because the width of  $\delta(t)$  is the reciprocal of its amplitude ([Fig. 2-1](#)), the area of  $\delta(t)$  remains 1 as  $\epsilon \rightarrow 0$ .

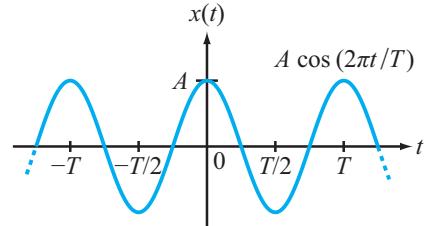
The limit is undefined, but it is useful to think of an impulse as the limiting case of a short duration and high pulse with unit area. Also, the pulse shape need not be rectangular; a Gaussian or sinc function can also be used.

Changing variables from  $t$  to  $t' = at$  yields the [time scaling](#) property ([Table 2-1](#)) of impulses:

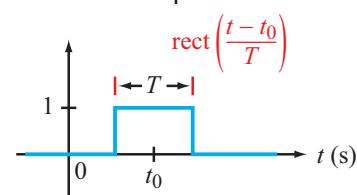
$$\delta(at) = \delta(t)/|a| \quad (2.6)$$

**Table 2-1** Types of signals and signal properties.**Types of Signals**

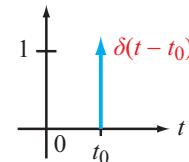
**Eternal sinusoid**  $x(t) = A \cos(2\pi f_0 t + \theta)$ ,  $-\infty < t < \infty$



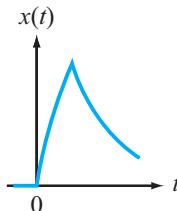
**Pulse (rectangle)**  $x(t) = \text{rect}\left(\frac{t-t_0}{T}\right)$   
 $= \begin{cases} 1 & \text{for } (t_0 - T/2) < t < (t_0 + T/2), \\ 0 & \text{otherwise} \end{cases}$



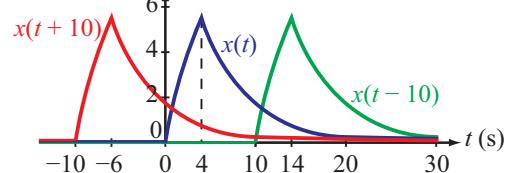
**Impulse  $\delta(t)$**   $\int_{-\infty}^{\infty} x(t) \delta(t-t_0) dt = x(t_0)$

**Properties**

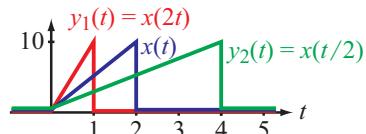
**Causal**  $x(t) = 0$  for  $t < 0$  (starts at or after  $t = 0$ )



**Time delay by  $t_0$**   $x(t) \rightarrow x(t-t_0)$



**Time scaling by  $a$**   $x(t) \rightarrow x(at)$



**Signal energy**  $E = \int_{-\infty}^{\infty} |x(t)|^2 dt$

The scaling property can be interpreted using Eq. (2.5) as follows. For  $a > 1$  the width of the pulse in Eq. (2.5) is compressed by  $|a|$ , reducing its area by a factor of  $|a|$ , but its height is unaltered. Hence the area under the pulse is reduced to  $1/|a|$ .

Impulses are important tools used in defining the impulse responses of 1-D systems and the point-spread functions of 2-D spatial systems (such as a camera or an ultrasound), as well as for deriving the sampling theorem.

## 2-1.2 Properties of 1-D Signals

### A. Time Delay

Delaying signal  $x(t)$  by  $t_0$  generates signal  $x(t - t_0)$ . If  $t_0 > 0$ , the waveform of  $x(t)$  is shifted to the right by  $t_0$ , and if  $t_0 < 0$ , the waveform of  $x(t)$  is shifted to the left by  $|t_0|$ . This is illustrated by the time-delay figure in **Table 2-1**.

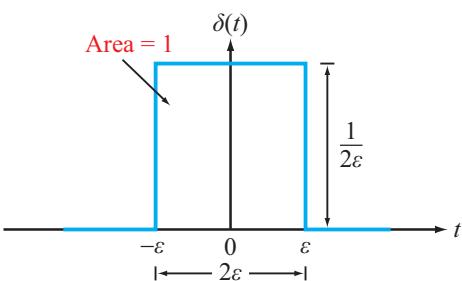
### B. Time Scaling

A signal  $x(t)$  **time-scaled** by  $a$  becomes  $x(at)$ . If  $a > 1$ , the waveform of  $x(t)$  is compressed in time by a factor of  $a$ . If  $0 < a < 1$ , the waveform of  $x(t)$  is expanded in time by a factor of  $1/a$ , as illustrated by the scaling figure in **Table 2-1**. If  $a < 0$ , the waveform of  $x(t)$  is compressed by  $|a|$  or expanded by  $1/|a|$ , and then time-reversed.

### C. Signal Energy

The **energy**  $E$  of a signal  $x(t)$  is

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt. \quad (2.7)$$



**Figure 2-1** Rectangular pulse model for  $\delta(t)$ .

A nonzero signal  $x(t)$  that is zero-valued outside the interval

$$[a, b] = \{t : a \leq t \leq b\},$$

(i.e.,  $x(t) = 0$  for  $t \notin [a, b]$ ), has **support**  $[a, b]$  and **duration**  $b - a$ .

**Concept Question 2-1:** Why does scaling time in an impulse also scale its area?

**Exercise 2-1:** Compute the value of  $\int_{-\infty}^{\infty} \delta(3t - 6) t^2 dt$ .

**Answer:**  $\frac{4}{3}$ . (See [IP](#))

**Exercise 2-2:** Compute the energy of the pulse defined by  $x(t) = 5 \text{rect}\left(\frac{t-2}{6}\right)$ .

**Answer:** 150. (See [IP](#))

## 2-2 Review of 1-D Continuous-Time Systems

A continuous-time system is a device or mathematical model that accepts a signal  $x(t)$  as its input and produces another signal  $y(t)$  at its output. Symbolically, the input-output relationship is expressed as



**Table 2-2** provides a list of important system types and properties.

### 2-2.1 Linear and Time-Invariant Systems

Systems are classified on the basis of two independent properties: (a) linearity and (b) time invariance, which leads to four possible classes:

- (1) **Linear** ([L](#)), but not time-invariant.
- (2) **Linear and time-invariant** ([LTI](#)).
- (3) Nonlinear, but time-invariant ([TI](#)).
- (4) Nonlinear and not time-invariant.

Most practical systems (including 2-D imaging systems such as a camera, ultrasound, radar, etc.) belong to one of the first two

**Table 2-2** Types of systems and associated properties.

Property	Definition
Linear System ( <b>L</b> )	If $x_i(t) \rightarrow \boxed{L} \rightarrow y_i(t)$ , then $\sum_{i=1}^N c_i x_i(t) \rightarrow \boxed{L} \rightarrow \sum_{i=1}^N c_i y_i(t)$
Time-Invariant ( <b>TI</b> )	If $x(t) \rightarrow \boxed{TI} \rightarrow y(t)$ , then $x(t - \tau) \rightarrow \boxed{TI} \rightarrow y(t - \tau)$
Linear Time-Invariant ( <b>LTI</b> )	If $x_i(t) \rightarrow \boxed{LTI} \rightarrow y_i(t)$ , then $\sum_{i=1}^N c_i x_i(t - \tau_i) \rightarrow \boxed{LTI} \rightarrow \sum_{i=1}^N c_i y_i(t - \tau_i)$
Impulse Response of LTI System	$\delta(t - \tau) \rightarrow \boxed{LTI} \rightarrow y(t) = h(t - \tau)$

of the above four classes. If a system is moderately nonlinear, it can be approximated by a linear model, and if it is highly nonlinear, it may be possible to divide its input-output response into a series of quasi-linear regions. *In this book, we limit our treatment to linear (**L**) and linear time-invariant (**LTI**) systems.*

### A. Linear Systems

A system is **linear** (**L**) if its response to a linear combination of input signals acting simultaneously is the same as the linear combination of the responses to each of the input signals acting alone. That is:

If

$$x_i(t) \rightarrow \boxed{L} \rightarrow y_i(t), \quad (2.8a)$$

then for any  $N$  inputs  $\{x_i(t), i = 1 \dots N\}$  and any  $N$  constants  $\{c_i, i = 1 \dots N\}$ ,

$$\sum_{i=1}^N c_i x_i(t) \rightarrow \boxed{L} \rightarrow \sum_{i=1}^N c_i y_i(t). \quad (2.8b)$$

Under mild assumptions (**regularity**) about the system, the finite sum can be extended to infinite sums and integrals.

Linearity also is called the **superposition** property.

### B. Time-Invariant System

A system is **time-invariant** (**TI**) if time shifting (delaying) the input, time shifts the output by exactly the same amount and in

exactly the same direction. That is, if

$$x(t) \rightarrow \boxed{TI} \rightarrow y(t),$$

then it follows that

$$x(t - \tau) \rightarrow \boxed{TI} \rightarrow y(t - \tau). \quad (2.9)$$

for any input signal  $x(t)$  and constant time shift  $\tau$ .

► Systems that are both linear and time-invariant are termed **linear time-invariant (LTI)**. ◀

### 2-2.2 Impulse Response

In general, we use the symbols  $x(t)$  and  $y(t)$  to denote, respectively, the input signal into a system and the resultant output response. The term **impulse response** is used to denote the output response for the specific case when the input is an impulse. For non-time-invariant systems, the impulse response depends on the time at which the impulse is nonzero. The response to the impulse  $\delta(t)$  delayed by  $\tau$ , which is  $\delta(t - \tau)$ , is denoted as  $h(t; \tau)$ . If the system is time-invariant, then delaying the impulse merely delays the impulse response, so  $h(t; \tau) = h(t - \tau)$ , where  $h(t)$  is the response to the impulse  $\delta(t)$ . This can be summarized

in the following two equations:

$$\delta(t - \tau) \rightarrow \boxed{\text{SYSTEM}} \rightarrow h(t; \tau), \quad (2.10\text{a})$$

and for a time-invariant system,

$$\delta(t - \tau) \rightarrow \boxed{\text{TI}} \rightarrow h(t - \tau). \quad (2.10\text{b})$$

### 2-2.3 Convolution

#### A. Linear System

Upon interchanging  $t$  and  $t_0$  in the sifting property given by Eq. (2.4) and replacing  $t_0$  with  $\tau$ , we obtain the relationship

$$\int_{-\infty}^{\infty} x(\tau) \delta(t - \tau) d\tau = x(t). \quad (2.11)$$

Next, if we multiply both sides of Eq. (2.10a) by  $x(\tau)$  and then integrate  $\tau$  over the limits  $(-\infty, \infty)$ , we obtain

$$\int_{-\infty}^{\infty} x(\tau) \delta(t - \tau) d\tau \rightarrow \boxed{\text{L}} \rightarrow y(t) = \int_{-\infty}^{\infty} x(\tau) h(t; \tau) d\tau. \quad (2.12)$$

Upon using Eq. (2.11) to replace the left-hand side of Eq. (2.12) with  $x(t)$ , Eq. (2.12) becomes

$$x(t) \rightarrow \boxed{\text{L}} \rightarrow y(t) = \int_{-\infty}^{\infty} x(\tau) h(t; \tau) d\tau. \quad (2.13)$$

This integral is called the *superposition integral*.

#### B. LTI System

For an LTI system,  $h(t; \tau) = h(t - \tau)$ , in which case the expression for  $y(t)$  in Eq. (2.13) becomes

$$y(t) = \int_{-\infty}^{\infty} x(\tau) h(t - \tau) d\tau, \quad (2.14)$$

**Convolution Integral**

which is known as the *convolution integral*. Often, the convolution integral is represented symbolically by

$$x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau) h(t - \tau) d\tau. \quad (2.15)$$

Combining the previous results leads to the symbolic form

$$x(t) \rightarrow \boxed{\text{LTI}} \rightarrow y(t) = x(t) * h(t). \quad (2.16)$$

The steps leading to Eq. (2.16) are summarized in Fig. 2-2.

The convolution in Eq. (2.15) is realized by time-shifting the impulse response  $h(t)$ . Changing variables from  $\tau$  to  $t - \tau$  shows that convolution has the *commutative property*:

$$x(t) * h(t) = h(t) * x(t). \quad (2.17)$$

The expression for  $y(t)$  can also be derived by time-shifting the input signal  $x(t)$  instead, in which case the result would be

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} x(t - \tau) h(\tau) d\tau. \quad (2.18)$$

**Table 2-3** provides a summary of key properties of convolution that are extendable to 2-D, and **Fig. 2-3** offers a graphical representation of how two of those properties—the *associative* and *distributive* properties—are used to characterize the overall impulse responses of systems composed of multiple systems, when connected in series or in parallel, in terms of the impulse responses of the individual systems.

**Concept Question 2-2:** What is the significance of a system being linear time-invariant?

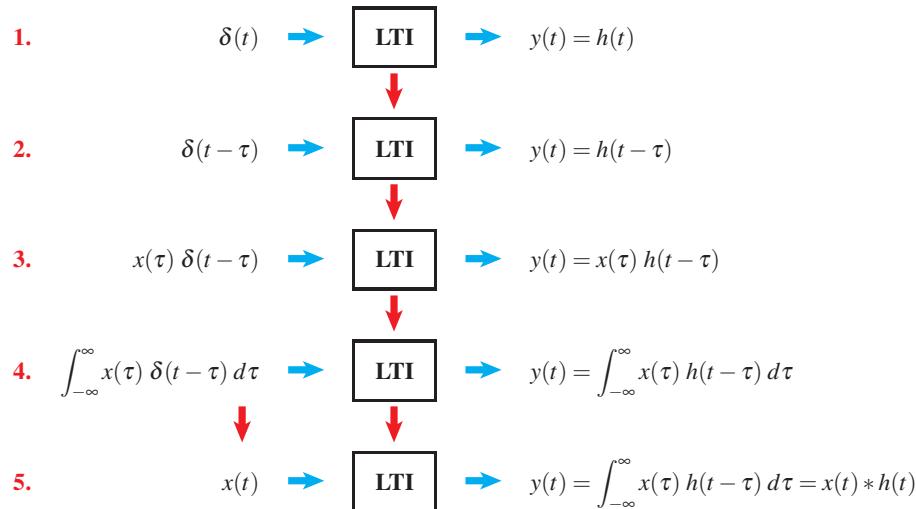
**Concept Question 2-3:** Why does delaying either of two signals delay their convolution?

**Exercise 2-3:** Is the following system linear, time-invariant, both, or neither?

$$\frac{dy}{dt} = 2x(t - 1) + 3tx(t + 1)$$

**Answer:** System is linear but not time-invariant. (See 

### LTI System with Zero Initial Conditions



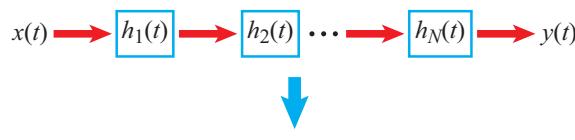
**Figure 2-2** Derivation of the convolution integral for a linear time-invariant system.

**Table 2-3 Convolution properties.**

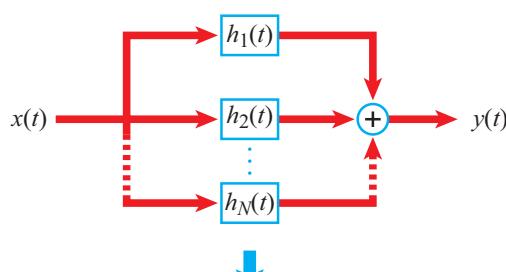
**Convolution Integral**  $y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(\tau) x(t - \tau) d\tau$

• **Causal Systems and Signals:**  $y(t) = h(t) * x(t) = u(t) \int_0^t h(\tau) x(t - \tau) d\tau$

Property	Description	
1. Commutative	$x(t) * h(t) = h(t) * x(t)$	(2.19a)
2. Associative	$[g(t) * h(t)] * x(t) = g(t) * [h(t) * x(t)]$	(2.19b)
3. Distributive	$x(t) * [h_1(t) + \dots + h_N(t)] = x(t) * h_1(t) + \dots + x(t) * h_N(t)$	(2.19c)
4. Causal * Causal = Causal	$y(t) = u(t) \int_0^t h(\tau) x(t - \tau) d\tau$	(2.19d)
5. Time-shift	$h(t - T_1) * x(t - T_2) = y(t - T_1 - T_2)$	(2.19e)
6. Convolution with Impulse	$x(t) * \delta(t - T) = x(t - T)$	(2.19f)



(a) In series



(b) In parallel

**Figure 2-3** (a) The overall impulse response of a system composed of multiple LTI systems connected in series is equivalent to the cumulative convolution of the impulse responses of the individual systems. (b) For LTI systems connected in parallel, the overall impulse response is equal to the sum of the impulse responses of the individual systems.

**Answer:**

$$y(t) = \begin{cases} e^{-2t} - e^{-3t} & \text{for } t > 0, \\ 0 & \text{for } t < 0. \end{cases}$$

(See [IP](#))

## 2-3 1-D Fourier Transforms

The **continuous-time Fourier transform (CTFT)** is a powerful tool for

- computing the spectra of signals, and
- analyzing the frequency responses of LTI systems.

### 2-3.1 Definition of Fourier Transform

The 1-D Fourier transform  $\mathbf{X}(f)$  of  $x(t)$  and the inverse 1-D Fourier transform  $x(t)$  of  $\mathbf{X}(f)$  are defined by the transformations

$$\mathbf{X}(f) = \mathcal{F}\{x(t)\} = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt \quad (2.20a)$$

and

$$x(t) = \mathcal{F}^{-1}\{\mathbf{X}(f)\} = \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi ft} df. \quad (2.20b)$$

Throughout this book, variables written in boldface (e.g.,  $\mathbf{X}(f)$ ) denote vectors or complex-valued quantities.

### A. Alternative Definitions of the Fourier Transform

Note that Eq. (2.20) differs slightly from the usual electrical engineering definition of the Fourier-transform pair:

$$\mathbf{X}_\omega(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \quad (2.21a)$$

and

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathbf{X}_\omega(\omega) e^{j\omega t} d\omega. \quad (2.21b)$$

Whereas Eq. (2.20) uses the oscillation frequency  $f$  (in Hz), the definition given by Eq. (2.21) uses  $\omega$  (in rad/s) instead, where  $\omega = 2\pi f$ . Using Hz makes interpretation of the Fourier

**Exercise 2-4:** Compute the output  $y(t)$  of an LTI system with impulse response  $h(t)$  to input  $x(t)$ , where

$$h(t) = \begin{cases} e^{-3t} & \text{for } t > 0, \\ 0 & \text{for } t < 0, \end{cases}$$

and

$$x(t) = \begin{cases} e^{-2t} & \text{for } t > 0, \\ 0 & \text{for } t < 0. \end{cases}$$

transform as a spectrum—as well as the presentation of the sampling theorem in Section 2-4—easier.

The definition of the Fourier transform used by mathematicians has a different sign for  $\omega$  than the definition used by electrical engineers:

$$\mathbf{X}_{-\omega}(\omega) = \int_{-\infty}^{\infty} x(t) e^{j\omega t} dt \quad (2.22a)$$

and

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathbf{X}_{-\omega}(\omega) e^{-j\omega t} d\omega. \quad (2.22b)$$

Geophysicists use different sign conventions for time and space! In addition, some computer programs, such as Mathematica, split the  $1/(2\pi)$  factor into factors of  $1/\sqrt{2\pi}$  in both the forward and inverse transforms.

- ▶ In this book, we use the definition of the Fourier transform given by Eq. (2.20) exclusively. ◀

## B. Fourier Transform Notation

Throughout this book, we use Eq. (2.20) as the definition of the Fourier transform, we denote the individual transformations by

$$\mathcal{F}\{x(t)\} = \mathbf{X}(f)$$

and

$$\mathcal{F}^{-1}\{\mathbf{X}(f)\} = x(t),$$

and we denote the combined bilateral pair by

$$x(t) \leftrightarrow \mathbf{X}(f).$$

## 2-3.2 Fourier Transform Properties

The major properties of the Fourier transform are summarized in **Table 2-4**, and derived next.

### Scaling:

For  $a \neq 0$ ,

$$\begin{aligned} x(at) &= \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi f(at)} df \\ &= \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi(f/a)t} d(af)/a \\ &= \mathcal{F}^{-1}\left\{\frac{1}{|a|} \mathbf{X}\left(\frac{f}{a}\right)\right\}. \end{aligned} \quad (2.23)$$

### Shifting:

$$\begin{aligned} x(t - \tau) &= \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi f(t-\tau)} df \\ &= \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi ft} e^{-j2\pi f\tau} df \\ &= \mathcal{F}^{-1}\{\mathbf{X}(f) e^{-j2\pi f\tau}\}. \end{aligned} \quad (2.24)$$

### Modulation:

$$\begin{aligned} e^{j2\pi f_0 t} x(t) &= \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi ft} e^{j2\pi f_0 t} df \\ &= \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi(f+f_0)t} df \\ &= \mathcal{F}^{-1}\{\mathbf{X}(f-f_0)\}. \end{aligned} \quad (2.25)$$

### Derivative:

$$\begin{aligned} \frac{dx(t)}{dt} &= \frac{d}{dt} \left[ \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi ft} df \right] \\ &= \int_{-\infty}^{\infty} \mathbf{X}(f) (j2\pi f) e^{j2\pi ft} df \\ &= \mathcal{F}^{-1}\{(j2\pi f) \mathbf{X}(f)\}. \end{aligned} \quad (2.26)$$

### Zero frequency:

Setting  $f = 0$  in Eq. (2.20a) leads to

$$\mathbf{X}(0) = \int_{-\infty}^{\infty} x(t) dt.$$

### Zero time:

Similarly, setting  $t = 0$  in Eq. (2.20b) leads to

$$x(0) = \int_{-\infty}^{\infty} \mathbf{X}(f) df.$$

The other properties in **Table 2-4** follow readily from the definition of the Fourier transform given by Eq. (2.20), except for the convolution property, which requires a few extra steps of algebra.

**Parseval's theorem** states that

$$E = \int_{-\infty}^{\infty} x(t) y^*(t) dt = \int_{-\infty}^{\infty} \mathbf{X}(f) \mathbf{Y}^*(f) df. \quad (2.27)$$

Setting  $y(t) = x(t)$  gives **Rayleigh's theorem** (also commonly known as Parseval's theorem), which states that the energies of

**Table 2-4** Major properties of the Fourier transform.

Property	$x(t)$	$\mathbf{X}(f) = \mathcal{F}[x(t)] = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt$
<b>1. Linearity</b>	$\sum c_i x_i(t)$	$\sum c_i \mathbf{X}_i(f)$
<b>2. Time scaling</b>	$x(at)$	$\frac{1}{ a } \mathbf{X}\left(\frac{f}{a}\right)$
<b>3. Time shift</b>	$x(t - \tau)$	$e^{-j2\pi f \tau} \mathbf{X}(f)$
<b>4. Frequency shift (modulation)</b>	$e^{j2\pi f_0 t} x(t)$	$\mathbf{X}(f - f_0)$
<b>5. Time derivative</b>	$x' = \frac{dx}{dt}$	$j2\pi f \mathbf{X}(f)$
<b>6. Reversal</b>	$x(-t)$	$\mathbf{X}(-f)$
<b>7. Conjugation</b>	$x^*(t)$	$\mathbf{X}^*(-f)$
<b>8. Convolution in <math>t</math></b>	$x(t) * y(t)$	$\mathbf{X}(f) \mathbf{Y}(f)$
<b>9. Convolution in <math>f</math> (multiplication in <math>t</math>)</b>	$x(t) y(t)$	$\mathbf{X}(f) * \mathbf{Y}(f)$
<b>10. Duality</b>	$\mathbf{X}(t)$	$\mathbf{x}(-f)$
<b>Special FT Relationships</b>		
<b>11. Zero frequency</b>	$\mathbf{X}(0) = \int_{-\infty}^{\infty} x(t) dt$	
<b>12. Zero time</b>	$x(0) = \int_{-\infty}^{\infty} \mathbf{X}(f) df$	
<b>13. Parseval's theorem</b>	$\int_{-\infty}^{\infty} x(t) y^*(t) dt = \int_{-\infty}^{\infty} \mathbf{X}(f) \mathbf{Y}^*(f) df$	

$x(t)$  and  $\mathbf{X}(f)$  are equal:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |\mathbf{X}(f)|^2 df. \quad (2.28)$$

where the even component  $x_e(t)$  and the odd component  $x_o(t)$  are formed from their parent signal  $x(t)$  as follows:

$$x_e(t) = [x(t) + x^*(-t)]/2 \quad (2.30a)$$

and

$$x_o(t) = [x(t) - x^*(-t)]/2. \quad (2.30b)$$

A signal is said to have **even symmetry** if  $x(t) = x^*(-t)$ , in which case  $x(t) = x_e(t)$  and  $x_o(t) = 0$ . Similarly, a signal has **odd symmetry** if  $x(t) = -x^*(-t)$ , in which case  $x(t) = x_o(t)$  and  $x_e(t) = 0$ .

## A. Even and Odd Parts of Signals

A signal  $x(t)$  can be decomposed into **even**  $x_e(t)$  and **odd**  $x_o(t)$  components:

$$x(t) = x_e(t) + x_o(t), \quad (2.29)$$

## B. Conjugate Symmetry

If  $x(t)$  is real-valued, then the following **conjugate symmetry** relations hold:

$$\mathbf{X}(-f) = \mathbf{X}^*(f), \quad (2.31a)$$

where  $\mathbf{X}^*(f)$  is the complex conjugate of  $\mathbf{X}(f)$ ,

$$\underline{\mathbf{X}(f)} = -\underline{\mathbf{X}(-f)} \quad (\text{phase is an odd function}), \quad (2.31b)$$

$$|\mathbf{X}(f)| = |\mathbf{X}(-f)| \quad (\text{magnitude is even}), \quad (2.31c)$$

$$\text{real}(\mathbf{X}(f)) = \text{real}(\mathbf{X}(-f)) \quad (\text{real part is even}), \quad (2.31d)$$

$$\text{imag}(\mathbf{X}(f)) = -\text{imag}(\mathbf{X}(-f)) \quad (\text{imaginary part is odd}). \quad (2.31e)$$

The real and imaginary parts of the Fourier transform of a real-valued signal  $x(t)$  are the Fourier transforms of the even and odd parts of  $x(t)$ , respectively. So the Fourier transform of a real-valued and even function is real-valued, and the Fourier transform of a real-valued and odd function is purely imaginary:

$$x(t) \text{ is even} \iff \mathbf{X}(f) \text{ is real},$$

$$x(t) \text{ is odd} \iff \mathbf{X}(f) \text{ is imaginary},$$

$$x(t) \text{ is real and even} \iff \mathbf{X}(f) \text{ is real and even},$$

$$x(t) \text{ is real and odd} \iff \mathbf{X}(f) \text{ is imaginary and odd}.$$

## C. Filtering and Frequency Response

The following is a very important property of the Fourier transform:

► The Fourier transform of a convolution of two functions is equal to the product of their Fourier transforms:

$$x(t) \rightarrow \boxed{\text{LTI}} \rightarrow y(t) = h(t) * x(t)$$

implies that

$$\mathbf{Y}(f) = \mathbf{H}(f) \mathbf{X}(f). \quad (2.32)$$

The function  $\mathbf{H}(f) = \mathcal{F}\{h(t)\}$  is called the **frequency response** of the system. The relationship described by Eq. (2.32) defines the **frequency filtering** process performed by the system. At a given frequency  $f_0$ , frequency component  $\mathbf{X}(f_0)$  of the input is multiplied by  $\mathbf{H}(f_0)$  to obtain the frequency component  $\mathbf{Y}(f_0)$  of the output.

For example, an **ideal lowpass filter** with cutoff frequency  $f_c$

and a frequency response

$$\mathbf{H}_{LP}(f) = \begin{cases} 1 & \text{for } |f| < f_c, \\ 0 & \text{for } |f| > f_c, \end{cases} \quad (2.33)$$

eliminates all frequency components of  $\mathbf{X}(f)$  above  $f_c$ .

## D. Sinc Functions

The impulse response of the ideal lowpass filter characterized by Eq. (2.33) is

$$\begin{aligned} h_{LP}(t) &= \mathcal{F}^{-1}\{\mathbf{H}_{LP}(f)\} \\ &= \int_{-f_c}^{f_c} 1 e^{j2\pi f t} df = \begin{cases} \sin(2\pi f_c t) / \pi t & \text{for } t \neq 0, \\ 2f_c & \text{for } t = 0. \end{cases} \end{aligned} \quad (2.34)$$

The scientific literature contains two different, but both commonly used definitions for the **sinc** function:

$$(1) \text{sinc}(x) = \sin(x)/x, \text{ and}$$

$$(2) \text{sinc}(x) = \sin(\pi x)/\pi x.$$

With either definition,  $\text{sinc}(0) = 1$  since  $\sin(x) \approx x$  for  $x \ll 1$ .

► Throughout this book, we use the sinc function definition

$$\text{sinc}(x) = \begin{cases} \sin(\pi x) / \pi x & \text{for } x \neq 0, \\ 1 & \text{for } x = 0. \end{cases} \quad (2.35)$$

Hence, per the definition given by Eq. (2.35), the impulse response of the ideal lowpass filter is given by

$$h_{LP}(t) = 2f_c \text{sinc}(2f_c t). \quad (2.36)$$

### 2-3-3 Fourier Transform Pairs

Commonly encountered Fourier transform pairs are listed in **Table 2-5**. Note the duality between entries #1 and #2, #4 and #5, and #6 and itself.

### 2-3-4 Interpretation of the Fourier Transform

A Fourier transform can be interpreted in three ways:

(1) as the frequency response of an LTI system,

**Table 2-5** Examples of Fourier transform pairs. Note that constant  $a \geq 0$ .

$ x(t) $	$\mathbf{X}(f) = \mathcal{F}[x(t)]$	$ \mathbf{X}(f) $
<b>BASIC FUNCTIONS</b>		
1a. 	$\delta(t)$ $\leftrightarrow$ 1	
1b. 	$\delta(t - \tau)$ $\leftrightarrow$ $e^{-j2\pi f\tau}$	
2. 	1 $\leftrightarrow$ $\delta(f)$	
3. 	$e^{-a t }, a > 0$ $\leftrightarrow$ $\frac{2a}{(2\pi f)^2 + a^2}$	
4. 	$\text{rect}(t/T)$ $\leftrightarrow$ $T \text{sinc}(fT)$	
5. 	$f_0 \text{sinc}(f_0 t)$ $\leftrightarrow$ $\text{rect}(f/f_0)$	
6. 	$e^{-\pi t^2}$ $\leftrightarrow$ $e^{-\pi f^2}$	

(2) as the spectrum of a signal, and

(3) as the energy spectral density of a signal.

## B. Spectrum

The **spectrum** of  $x(t)$  is  $\mathbf{X}(f)$ . Consider, for example, the eternal sinusoid defined by Eq. (2.1):

### A. Frequency Response

The **frequency response**  $\mathbf{H}(f)$  of an LTI system is the frequency domain equivalent of the system's impulse response  $h(t)$ :

$$\mathbf{H}(f) \leftrightarrow h(t).$$

where we used the relation  $\cos(x) = (e^{jx} + e^{-jx})/2$ . From the properties and pairs listed in Tables 2-4 and 2-5, we have

$$\mathbf{X}(f) = \frac{A}{2} e^{j\theta} \delta(f - f_0) + \frac{A}{2} e^{-j\theta} \delta(f + f_0). \quad (2.38)$$

Strictly speaking, the Fourier transform of an eternal sinusoid is undefined, since an eternal sinusoid is not absolutely integrable. Nevertheless, this example provides a convenient illustration that the spectrum of a sinusoid at  $f_0$  is concentrated entirely at  $\pm f_0$ . By extension, the spectrum of a constant signal is concentrated entirely at  $f = 0$ .

Real signals are more complicated than simple sinusoids, as are their corresponding spectra. **Figure 2-4(a)** displays 7 ms of a trumpet playing note B, and part (b) of the same figure displays the corresponding spectrum. The spectrum is concentrated around narrow spectral lines located at 491 Hz and the next 6 harmonics. In contrast, speech exhibits a much broader spectrum, as illustrated by the examples in **Fig. 2-5**.

### C. Energy Spectral Density

A more rigorous interpretation of  $\mathbf{X}(f)$  that avoids impulses in frequency uses the concept of **energy spectral density**. Let the spectrum of a signal  $x(t)$  be

$$\begin{aligned}\mathbf{X}(f) &= \begin{cases} \text{constant} & \text{for } f_0 < f < f_0 + \varepsilon, \\ 0 & \text{otherwise,} \end{cases} \\ &\approx \begin{cases} \mathbf{X}(f_0) & \text{for } f_0 < f < f_0 + \varepsilon, \\ 0 & \text{otherwise.} \end{cases} \quad (2.39)\end{aligned}$$

The approximation becomes exact in the limit as  $\varepsilon \rightarrow 0$ , provided  $\mathbf{X}(f)$  is continuous at  $f = f_0$ .

Using Rayleigh's theorem, the energy of  $x(t)$  is

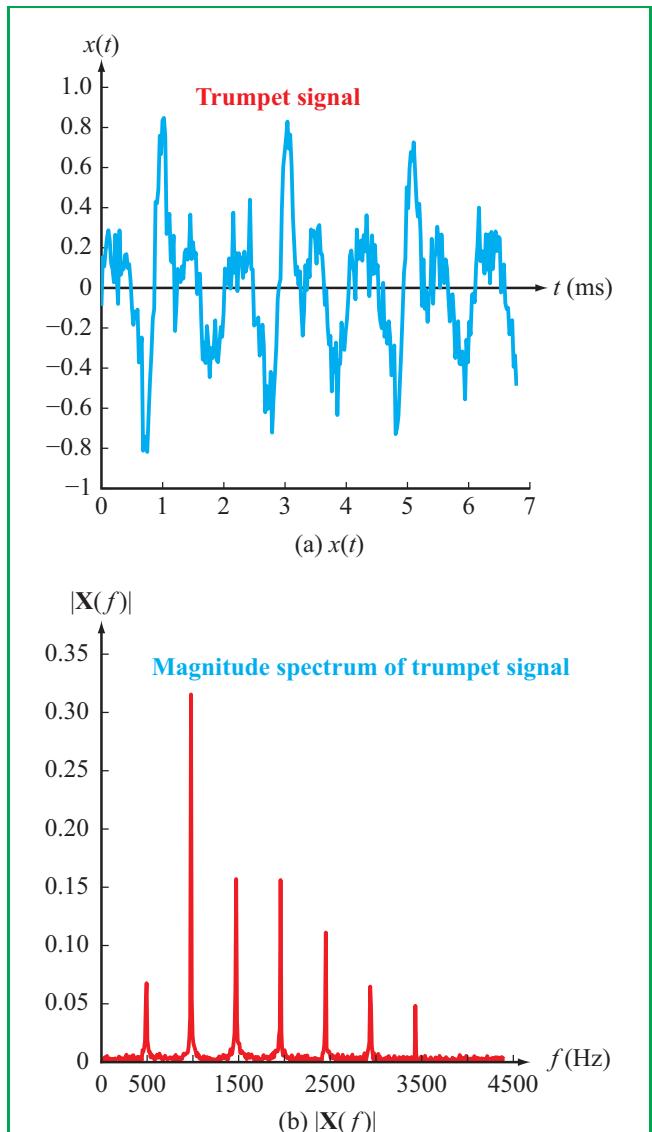
$$E = \int_{-\infty}^{\infty} |\mathbf{X}(f)|^2 df \approx |\mathbf{X}(f_0)|^2 \varepsilon. \quad (2.40)$$

The energy of  $x(t)$  in the interval  $f_0 < f < f_0 + \varepsilon$  is  $|\mathbf{X}(f_0)|^2 \varepsilon$ . The **energy spectral density** at frequency  $f = f_0$  (analogous to probability density or to mass density of a physical object) is  $|\mathbf{X}(f_0)|^2$ .

Note that a real-valued  $x(t)$  will, by conjugate symmetry, also have nonzero  $\mathbf{X}(f)$  in the interval  $-f_0 > f > -f_0 - \delta$ . So the **bilateral** energy spectral density of  $x(t)$  at  $f_0$  is  $2|\mathbf{X}(f_0)|^2$ .

**Concept Question 2-4:** Provide three applications of the Fourier transform.

**Concept Question 2-5:** Provide an application of the sinc function.



**Figure 2-4** Trumpet signal (note B) and its magnitude spectrum.

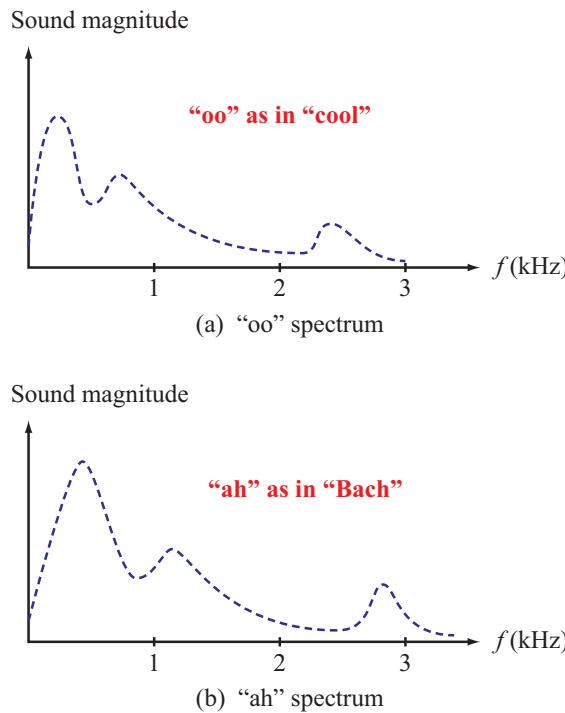


Figure 2-5 Spectra of two vowel sounds.

**Exercise 2-6:** Compute the Fourier transform of  $\frac{d}{dt}[\text{sinc}(t)]$ .

**Answer:**

$$\mathfrak{F}\left\{\frac{d}{dt}\text{sinc}(t)\right\} = \begin{cases} j2\pi f & \text{for } |f| < 0.5, \\ 0 & \text{for } |f| > 0.5. \end{cases}$$

(See [IP](#))

## 2-4 The Sampling Theorem

The sampling theorem is an operational cornerstone of both discrete-time 1-D signal processing and discrete-space 2-D image processing.

### 2-4.1 Sampling Theorem Statement

The **samples**  $\{x(n\Delta)\}$  of a signal  $x(t)$  sampled every  $\Delta$  seconds are

$$\{x(n\Delta), n = \dots, -2, -1, 0, 1, 2, \dots\}. \quad (2.41)$$

The inverse of the **sampling interval**  $\Delta$  is the **sampling rate**  $S = 1/\Delta$  samples per second. The sampling rate has the same dimension as Hz, and is often expressed in “Hz.” For example, the standard sampling rate for CDs is 44100 samples/second, often stated as 44100 Hz. The corresponding sampling interval is  $\Delta = 1/44100 \text{ s} = 22.676 \mu\text{s}$ .

A signal  $x(t)$  is **bandlimited** to a maximum frequency of  $B$  (extending from  $-B$  to  $B$ ), measured in Hz, if its Fourier transform  $\mathbf{X}(f) = 0$  for  $|f| > B$ . Although real-world signals are seldom truly bandlimited, their spectra are often negligible above some frequency  $B$ .

► The **sampling theorem** states that if

$$\mathbf{X}(f) = 0 \text{ for } |f| > B,$$

and if

$x(t)$  is sampled at a sampling rate of  $S$  samples/s,

then

$x(t)$  can be reconstructed exactly from  $\{x(n\Delta), n = \dots, -2, -1, 0, 1, 2, \dots\}$ , provided  $S > 2B$ .

**Exercise 2-5:** A square wave  $x(t)$  has the Fourier series expansion

$$x(t) = \sin(t) + \frac{1}{3} \sin(3t) + \frac{1}{5} \sin(5t) + \frac{1}{7} \sin(7t) + \dots$$

Compute output  $y(t)$  if

$$x(t) \rightarrow \boxed{h(t) = 0.4 \text{sinc}(0.4t)} \rightarrow y(t).$$

**Answer:**  $y(t) = \sin(t)$ . (See [IP](#))

The sampling rate must exceed double the maximum frequency in the spectrum  $\mathbf{X}(f)$  of  $x(t)$ . The minimum (actually an infi-

num) sampling rate  $2B$  samples/second is called the **Nyquist rate**, and the frequency  $2B$  is called the **Nyquist frequency**.

## 2-4.2 Sampling Theorem Derivation

### A. The Sampled Signal $x_s(t)$

Given a signal  $x(t)$ , we construct the **sampled signal**  $x_s(t)$  by multiplying  $x(t)$  by the **impulse train**

$$\delta_s(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\Delta). \quad (2.42)$$

That is,

$$\begin{aligned} x_s(t) &= x(t) \delta_s(t) = \sum_{n=-\infty}^{\infty} x(t) \delta(t - n\Delta) \\ &= \sum_{n=-\infty}^{\infty} x(n\Delta) \delta(t - n\Delta). \end{aligned} \quad (2.43)$$

### B. Spectrum of the sampled signal $x_s(t)$

Using Fourier series, it can be shown that the Fourier transform of the impulse train  $\delta_s(t)$  is itself an impulse train in frequency:

$$\Delta \sum_{n=-\infty}^{\infty} \delta(t - n\Delta) \leftrightarrow \sum_{k=-\infty}^{\infty} \delta(f - k/\Delta). \quad (2.44)$$

This result can be interpreted as follows. A periodic signal has a discrete spectrum (zero except at specific frequencies) given by the signal's Fourier series expansion. By Fourier duality, a discrete signal (zero except at specific times) such as  $x_s(t)$  has a periodic spectrum. So a discrete and periodic signal such as  $\delta_s(t)$  has a spectrum that is both discrete and periodic.

Multiplying Eq. (2.44) by  $x(t)$ , using the definition for  $x_s(t)$  given by Eq. (2.43), and applying property #9 in **Table 2-4** leads to

$$x_s(t) \Delta \leftrightarrow X(f) * \sum_{k=-\infty}^{\infty} \delta(f - k/\Delta), \quad (2.45)$$

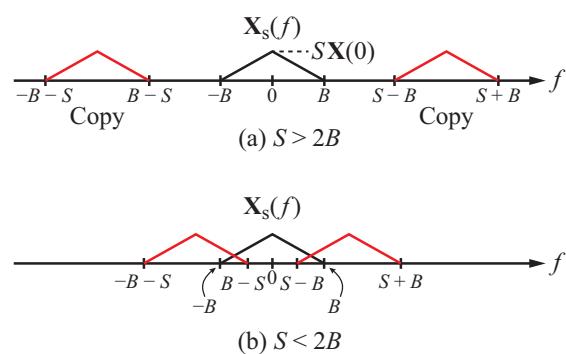
which, using property #6 of **Table 2-3**, simplifies to

$$x_s(t) \Delta \leftrightarrow \sum_{k=-\infty}^{\infty} X(f - k/\Delta). \quad (2.46)$$

Dividing by  $\Delta$  and recalling that  $S = 1/\Delta$  gives

$$x_s(t) \leftrightarrow S \sum_{k=-\infty}^{\infty} X(f - kS). \quad (2.47)$$

The spectrum  $\mathbf{X}_s(f)$  of  $x_s(t)$  consists of a superposition of copies of the spectrum  $\mathbf{X}(f)$  of  $x(t)$ , repeated every  $S = 1/\Delta$  and multiplied by  $S$ . If these copies do not overlap in frequency, we may then recover  $\mathbf{X}(f)$  from  $\mathbf{X}_s(f)$  using a lowpass filter, provided  $S > 2B$  [see **Fig. 2-6(a)**].



**Figure 2-6** Sampling a signal  $x(t)$  with maximum frequency  $B$  at a rate of  $S$  makes  $\mathbf{X}(f)$  change amplitude to  $S \mathbf{X}(f)$  and to repeat in  $f$  with period  $S$ . These copies (a) do not overlap if  $S > 2B$ , but (b) they do if  $S < 2B$ .

## 2-4.3 Aliasing

If the sampling rate  $S$  does not exceed  $2B$ , the copies of  $\mathbf{X}(f)$  will overlap one another, as shown in **Fig. 2-6(b)**. This is called an **aliased** condition, the consequence of which is that the reconstructed signal will no longer match the original signal  $x(t)$ .

### Example 2-1: Two Sinusoids and Aliasing

Two signals, a 2 Hz sinusoid and a 12 Hz sinusoid:

$$x_1(t) = \cos(4\pi t)$$

and

$$x_2(t) = \cos(24\pi t),$$

were sampled at 20 samples/s. Generate plots for

- (a)  $x_1(t)$  and its sampled version  $x_{1s}(t)$ ,
- (b)  $x_2(t)$  and its sampled version  $x_{2s}(t)$ ,
- (c) Spectra  $\mathbf{X}_1(f)$  and  $\mathbf{X}_{1s}(f)$ , and
- (d) Spectra  $\mathbf{X}_2(f)$  and  $\mathbf{X}_{2s}(f)$ .

**Solution:** (a) **Figure 2-7(a)** displays  $x_1(t)$  and  $x_{1s}(t)$ , with the latter generated by sampling  $x_1(t)$  at  $S = 20$  samples/s. The applicable bandwidth of  $x_1(t)$  is  $B_1 = 2$  Hz. Since  $S > 2B_1$ , it should be possible to reconstruct  $x_1(t)$  from  $x_{1s}(t)$ , which we demonstrate in a later subsection.

Similar plots are displayed in **Fig. 2-7(b)** for the 12 Hz sinusoid. In this latter case,  $B = 12$  Hz and  $S = 20$  samples/s. Hence,  $S < 2B$ .

(b) Spectrum  $\mathbf{X}_1(f)$  of  $x_1(t)$  consists of two impulses at  $\pm 2$  Hz, as shown in **Fig. 2-8(a)**. The spectrum of the sampled version consists of the same spectrum  $\mathbf{X}_1(f)$  of  $x_1(t)$ , scaled by the factor  $S$ , plus additional copies repeated every  $\pm S = 20$  Hz (**Fig. 2-8(b)**). Note that the central spectrum in (**Fig. 2-8(b)**), corresponding to  $S \mathbf{X}_1(f)$ , does not overlap with the neighboring copies.

(c) Spectra  $\mathbf{X}_2(f)$  and  $\mathbf{X}_{2s}(f)$  are shown in **Fig. 2-9**. Because  $S < 2B$ , the central spectrum overlaps with its two neighbors.

#### 2-4.4 Sampling Theorem Implementation

##### A. Physical Lowpass Filter

If  $S > 2B$ , the original signal  $x(t)$  can be recovered from the sampled signal  $x_s(t)$  by subjecting the latter to a lowpass filter that passes frequencies below  $B$  with a gain of  $1/S$  (to compensate for the factor of  $S$  induced by sampling (as noted in Eq. (2.47)) and rejects frequencies greater than  $(S - B)$  Hz:

$$\mathbf{H}(f) = \begin{cases} 1/S & \text{for } |f| < B, \\ 0 & \text{for } |f| > S - B. \end{cases} \quad (2.48)$$

This type of filter must be implemented using a physical circuit. For example, a Butterworth filter can be constructed by connecting op-amps, capacitors and resistors in a series of Sallen-Key configurations.<sup>†</sup> This is clearly impractical for image processing.

<sup>†</sup>Ulaby and Yagle, *Signals and Systems: Theory and Applications*, pp. 296–297.

##### B. Sinc Interpolation Formula

Mathematically, we can use an ideal lowpass filter with a cutoff frequency anywhere between  $B$  and  $S - B$ . It is customary to use  $S/2$  as the cutoff frequency, since it is halfway between  $B$  and  $S - B$ , so as to provide a safety margin for avoiding aliasing if the actual maximum frequency of  $\mathbf{X}(f)$  exceeds  $B$  but is less than  $S/2$ . The frequency response of this ideal lowpass filter is, from Eq. (2.48),

$$\mathbf{H}(f) = \begin{cases} 1/S & \text{for } |f| < S/2, \\ 0 & \text{for } |f| > S/2. \end{cases} \quad (2.49)$$

Setting  $f_0 = S$  in entry #4 of **Table 2-5**, the impulse response is found to be

$$h(t) = (1/S)[S \operatorname{sinc}(St)] = \operatorname{sinc}(St). \quad (2.50)$$

Using the convolution property  $x(t) * \delta(t - \tau) = x(t - \tau)$  [see property #6 in **Table 2-3**], we can derive the following **sinc interpolation formula**:

$$\begin{aligned} x(t) &= x_s(t) * h(t) \\ &= \sum_{n=-\infty}^{\infty} x(n\Delta) \delta(t - n\Delta) * h(t) \\ &= \sum_{n=-\infty}^{\infty} x(n\Delta) \operatorname{sinc}(S(t - n\Delta)). \end{aligned} \quad (2.51)$$

In principle, this formula can be used to reconstruct  $x(t)$  for any time  $t$  from its samples  $\{x(n\Delta)\}$ . But since it requires an infinite number of samples  $\{x(n\Delta)\}$  to reconstruct  $x(t)$ , it is of theoretical interest only.

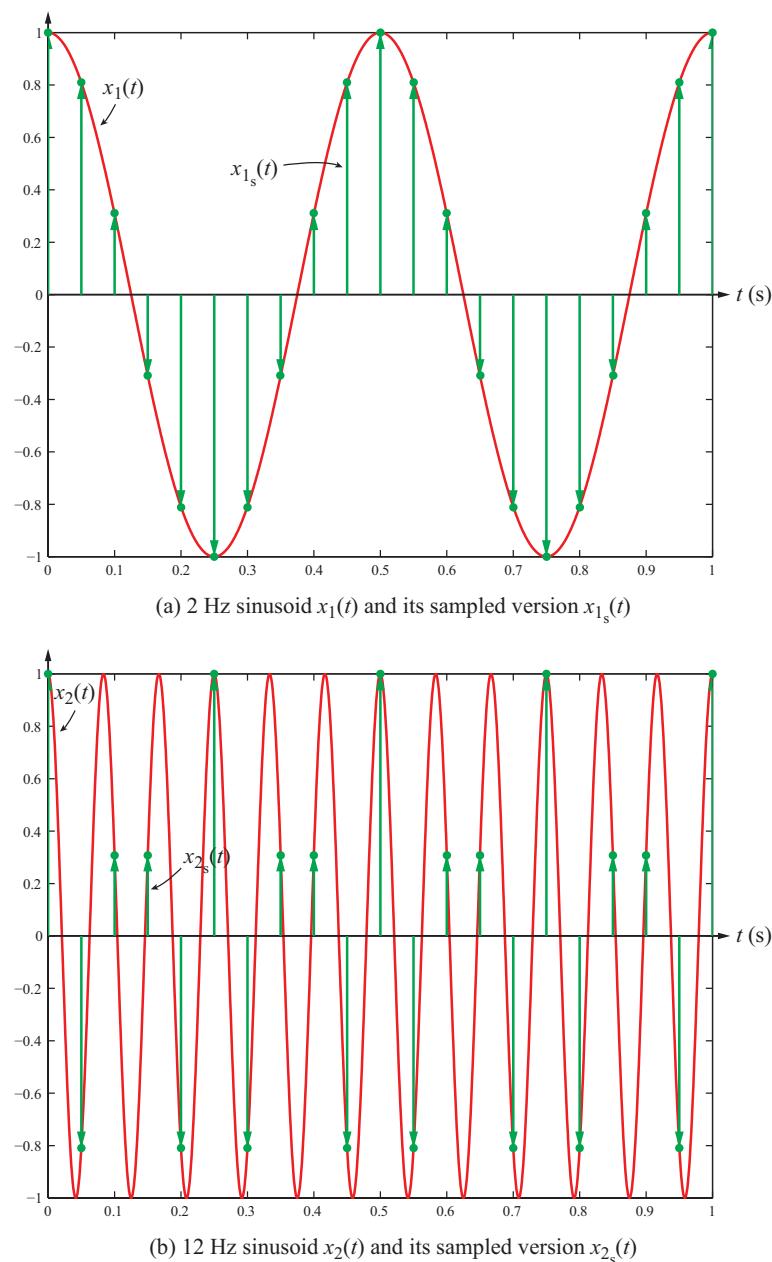
##### C. Reconstruction of $\mathbf{X}(f)$ from Samples $\{x(n\Delta)\}$

According to Eq. (2.43)), the sampled signal  $x_s(t)$  is given by

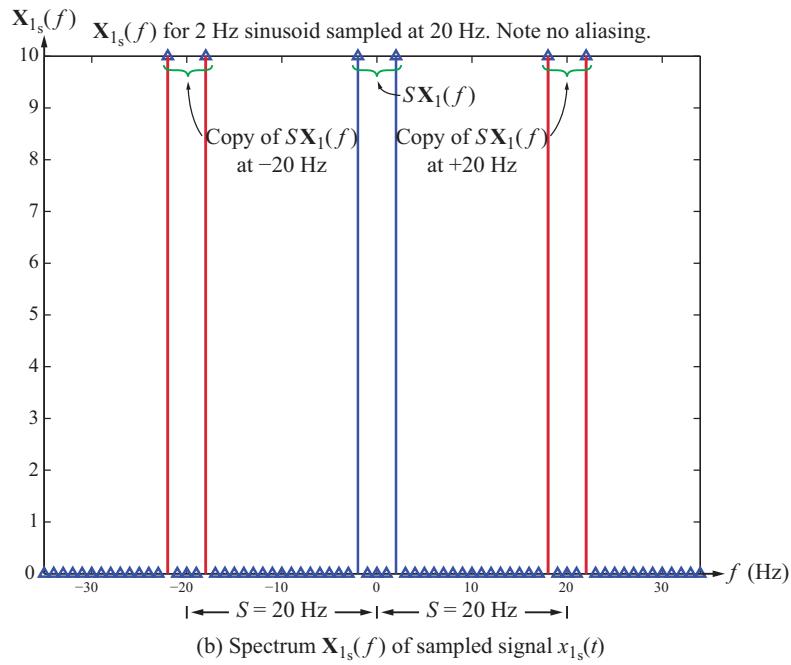
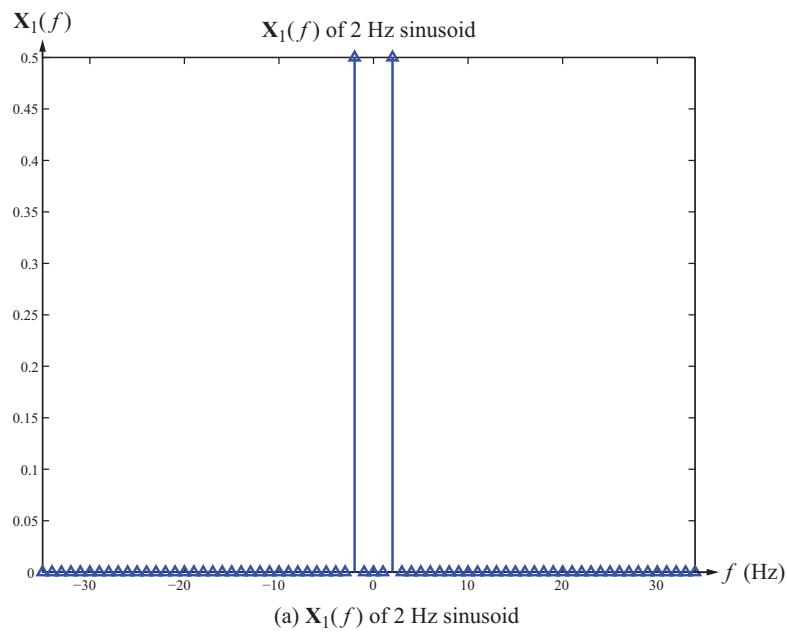
$$x_s(t) = \sum_{n=-\infty}^{\infty} x(n\Delta) \delta(t - n\Delta). \quad (2.52)$$

Application of property #6 in **Table 2-5** yields

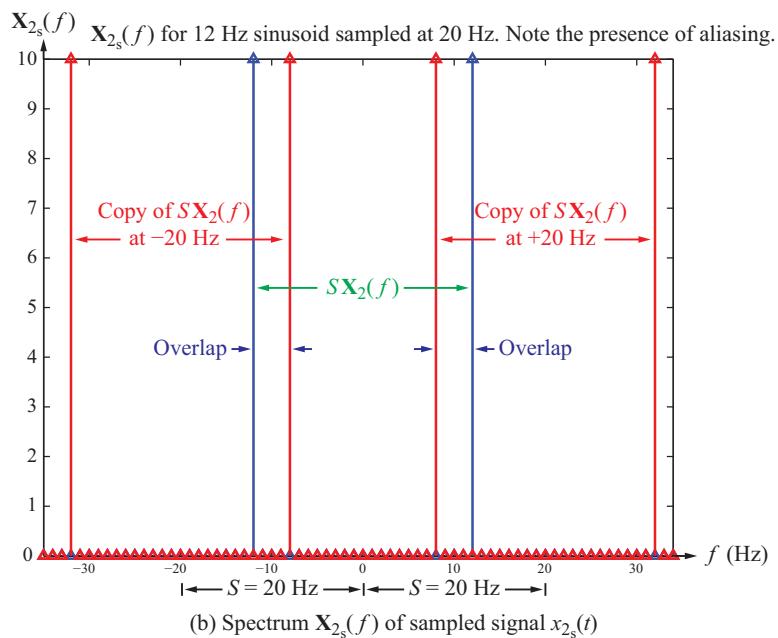
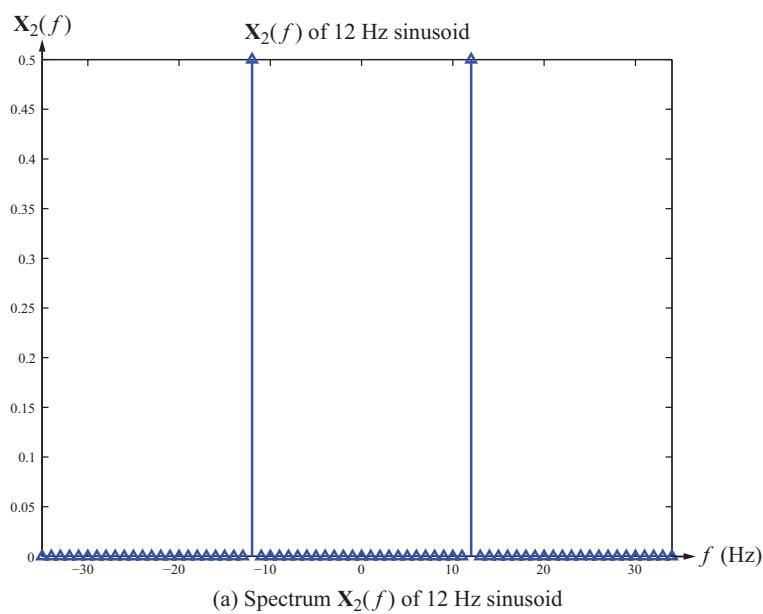
$$\mathbf{X}_s(f) = \sum_{n=-\infty}^{\infty} x(n\Delta) e^{-j2\pi f n\Delta}. \quad (2.53)$$



**Figure 2-7** Plots of (a)  $x_1(t)$  and  $x_{1s}(t)$  and (b)  $x_2(t)$  and  $x_{2s}(t)$ . Sampling rate  $S = 20$  samples/s.



**Figure 2-8** Spectra (a)  $\mathbf{X}_1(f)$  and (b)  $\mathbf{X}_{1s}(f)$  of the 2 Hz sinusoid and its sampled version, respectively. The spectrum  $\mathbf{X}_{1s}(f)$  consists of  $\mathbf{X}(f)$  scaled by  $S = 20$ , plus copies thereof at integer multiples of  $\pm 20$  Hz. The vertical axes denote areas under the impulses.



**Figure 2-9** Spectra (a)  $\mathbf{X}_2(f)$  and (b)  $\mathbf{X}_{2s}(f)$  of the 12 Hz sinusoid and its sampled version. Note the overlap in (b) between the spectrum of  $\mathbf{X}_{2s}(f)$  and its neighboring copies. The vertical axes denote areas under the impulses.

Note that  $\mathbf{X}_s(f)$  is periodic in  $f$  with period  $1/\Delta$ , as it should be. In the absence of aliasing,

$$\mathbf{X}(f) = \frac{1}{S} \mathbf{X}_s(f) \quad \text{for } |f| < S/2. \quad (2.54)$$

The relationship given by Eq. (2.53) still requires an infinite number of samples  $\{x(n\Delta)\}$  to reconstruct  $\mathbf{X}(f)$  at each frequency  $f$ .

## D. Nearest-Neighbor (NN) Interpolation

A common procedure for computing an approximation to  $x(t)$  from its samples  $\{x(n\Delta)\}$  is the nearest neighbor interpolation. The signal  $x(t)$  is approximated by  $\hat{x}(t)$ :

$$\hat{x}(t) = \begin{cases} x(n\Delta) & \text{for } (n - 0.5)\Delta < t < (n + 0.5)\Delta, \\ x((n+1)\Delta) & \text{for } (n + 0.5)\Delta < t < (n + 1.5)\Delta, \\ \vdots & \vdots \end{cases} \quad (2.55)$$

So  $\hat{x}(t)$  is a piecewise-constant approximation to  $x(t)$ , and it is related to the sampled signal  $x_s(t)$  by

$$\hat{x}(t) = x_s(t) * \text{rect}(t/\Delta). \quad (2.56)$$

Using the Fourier transform of a rectangle function (entry #4 in **Table 2-5**), the spectrum  $\hat{\mathbf{X}}(f)$  of  $\hat{x}(t)$  is

$$\hat{\mathbf{X}}(f) = \mathbf{X}_s(f) \Delta \text{sinc}(\Delta f), \quad (2.57)$$

where  $\mathbf{X}_s(f)$  is the spectrum of the sampled signal. The zero-crossings of the sinc function occur at frequencies  $f = k/\Delta = kS$  for integers  $k$ . These are also the centers of the copies of the original spectrum  $\mathbf{X}(f)$  induced by sampling. So these copies are attenuated if the maximum frequency  $B$  of  $\mathbf{X}(f)$  is such that  $B \ll S$ . The factor  $\Delta$  in Eq. (2.57) cancels the factor  $S = 1/\Delta$  in Eq. (2.47).

### Example 2-2: Reconstruction of 2 Hz Sinusoid

For the 2 Hz sinusoid of Example 2-1: (a) plot spectrum  $\hat{\mathbf{X}}_1(f)$  of the approximated reconstruction  $\hat{x}_1(t)$ , and (b) apply nearest-neighbor interpolation to generate  $\hat{x}_1(t)$ .

**Solution:** (a) Spectrum  $\mathbf{X}_{1s}(f)$  of the sampled version of the 2 Hz sinusoid was generated earlier in Example 2-1 and displayed in **Fig. 2-8(b)**. To obtain the spectrum of the approximate

reconstruction  $\hat{x}_1(t)$ , we apply Eq. (2.57) with  $\Delta = 1/S = 0.05$  s:

$$\hat{\mathbf{X}}_1(f) = \mathbf{X}_{1s}(f) \Delta \text{sinc}(\Delta f).$$

The sinc function is displayed in **Fig. 2-10(a)** in red and uses the vertical scale on the right-hand side, and the spectrum  $\hat{\mathbf{X}}_1(f)$  is displayed in blue using the vertical scale on the left-hand side. The sinc function preserves the spectral components at  $\pm 2$  Hz, but attenuates the components centered at  $\pm 20$  Hz by a factor of 10 (approximately).

(b) Application of Eq. (2.56) to  $x_1(n\Delta) = \cos(4\pi n\Delta)$  with  $\Delta = 1/20$  s yields plot  $\hat{x}_1(t)$  shown in **Fig. 2-10(b)**.

**Concept Question 2-6:** Why must the sampling rate of a signal exceed double its maximum frequency, if it is to be reconstructed from its samples?

**Concept Question 2-7:** Why does nearest-neighbor interpolation work as well as it does?

**Exercise 2-7:** What is the Nyquist sampling rate for a signal bandlimited to 4 kHz?

**Answer:** 8000 samples/s. (See [IP](#))

**Exercise 2-8:** A 500 Hz sinusoid is sampled at 900 samples/s. No anti-alias filter is being used. What is the frequency of the reconstructed continuous-time sinusoid?

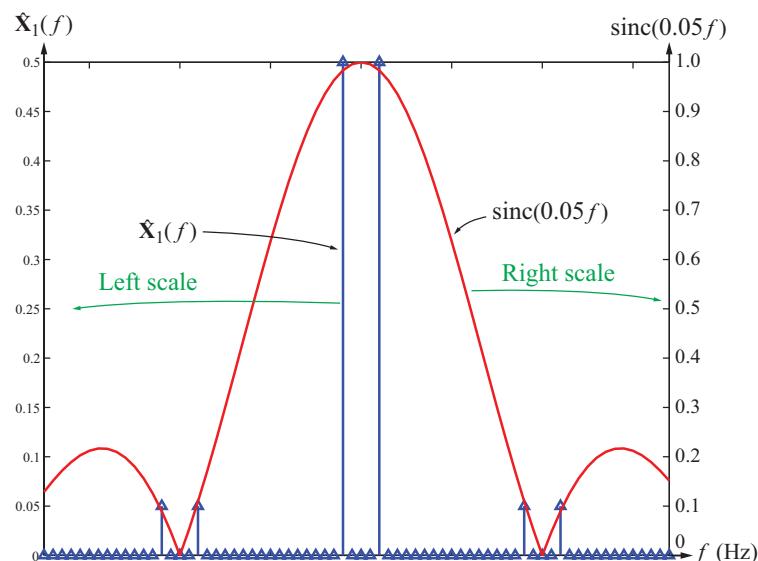
**Answer:** 400 Hz. (See [IP](#))

## 2-5 Review of 1-D Discrete-Time Signals and Systems

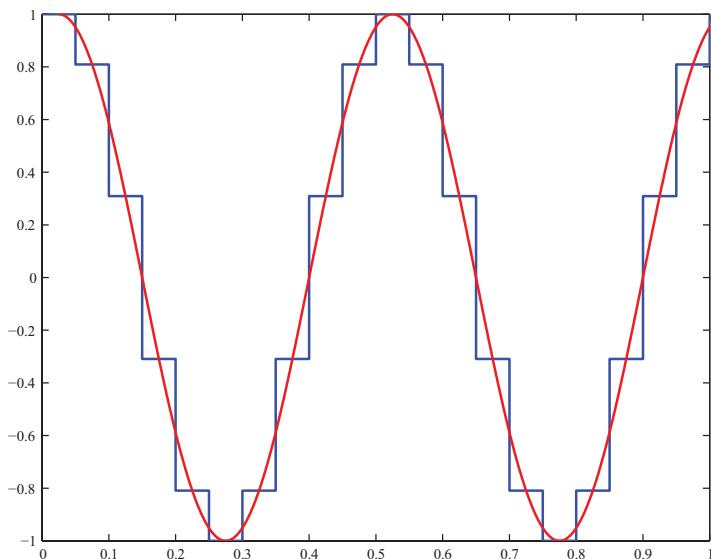
Through direct generalizations of the 1-D continuous-time definitions and properties of signals and systems presented earlier, we now extend our review to their discrete counterparts.

### 2-5.1 Discrete-Time Notation

A discrete-time signal is a physical quantity—such as voltage or acoustic pressure—that varies with discrete time  $n$ , where  $n$  is a dimensionless integer. Mathematically, a **discrete-time signal** is a function  $x[n]$  of **discrete time**  $n$ .

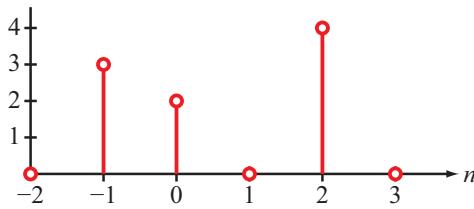


(a) Spectrum  $\hat{X}_1(f)$  of 2 Hz signal  $x_1(t)$ . The sinc function is shown in red.



(b) 2 Hz sinusoidal signal reconstructed from samples at 20 Hz, using nearest neighbor interpolation. The 2 Hz sinusoid is shown in red.

**Figure 2-10** Plots of Example 2-2.



**Figure 2-11** Stem plot representation of  $x[n]$ .

- Discrete-time signals  $x[n]$  use square brackets, whereas continuous-time signals  $x(t)$  use parentheses.
- $t$  has units of seconds, while  $n$  is dimensionless.

Discrete-time signals  $x[n]$  usually result from sampling a continuous-time signal  $x(t)$  at integer multiples of a sampling interval of  $\Delta$  seconds. That is,

$$\bullet \quad x[n] = x(n\Delta) \quad \text{for } n = \{\dots, -2, -1, 0, 1, 2, \dots\}.$$

Discrete-time signals are often represented using bracket notation, and plotted using stem plots. For example, the discrete-time signal  $x[n]$  defined by

$$x[n] = \begin{cases} 3 & \text{for } n = -1, \\ 2 & \text{for } n = 0, \\ 4 & \text{for } n = 2, \\ 0 & \text{for all other } n, \end{cases} \quad (2.58)$$

can be depicted using either the bracket notation

$$x[n] = \{3, \underline{2}, 0, 4\}, \quad (2.59)$$

where the underlined value is the value at time  $n = 0$ , or in the form of the stem plot shown in **Fig. 2-11**.

The **support** of this  $x[n]$  is the **interval**  $[-1, 2]$ , and its **duration** is  $2 - (-1) + 1 = 4$ . In general, a discrete-time signal with support  $[a, b]$  has duration  $b - a + 1$ .

A **discrete-time (Kronecker) impulse**  $\delta[n]$  is defined as

$$\delta[n] = \{\underline{1}\} = \begin{cases} 1 & \text{for } n = 0, \\ 0 & \text{for } n \neq 0. \end{cases} \quad (2.60)$$

Unlike the continuous-time impulse, the discrete-time impulse has no issues about infinite height and zero width. The sifting

property of impulses still holds, with a summation replacing the integral:

$$\sum_{i=-\infty}^{\infty} x[i] \delta[n-i] = x[n]. \quad (2.61)$$

## 2-5.2 Discrete-Time Eternal Sinusoids

A **discrete-time eternal sinusoid** is defined as

$$x[n] = A \cos(\Omega_0 n + \theta), \quad -\infty < n < \infty, \quad (2.62)$$

where  $\Omega_0$  is the **discrete-time frequency** with units of radians per sample, so it is dimensionless.

Comparing the discrete-time eternal sinusoid to the continuous-time eternal sinusoid given by Eq. (2.1), which we repeat here as

$$x(t) = A \cos(2\pi f_0 t + \theta), \quad -\infty < t < \infty, \quad (2.63)$$

it is apparent that a discrete-time sinusoid can be viewed as a continuous-time sinusoid sampled every  $\Delta$  seconds, at a sampling rate of  $S = 1/\Delta$  samples/s. Thus,

$$\Omega_0 = 2\pi f_0 \Delta = 2\pi f_0 / S, \quad (2.64)$$

which confirms that  $\Omega_0$ , like  $n$ , is dimensionless. However, *almost all discrete-time eternal sinusoids are nonperiodic!* In fact,  $x[n]$  is periodic only if

$$\frac{2\pi}{\Omega_0} = \frac{N}{D}, \quad (2.65)$$

with  $N/D$  being a rational number. In such a case, the **fundamental period** of the sinusoid is  $N$ , provided  $N/D$  has been reduced to lowest terms.

### Example 2-3: Discrete Sinusoid

Compute the fundamental period of

$$x[n] = 3 \cos(0.3\pi n + 2).$$

**Solution:** From the expression for  $x[n]$ , we deduce that  $\Omega_0 = 0.3\pi$ . Hence,

$$\frac{2\pi}{\Omega_0} = \frac{2\pi}{0.3\pi} = \frac{20}{3}$$

reduced to lowest terms. Therefore, the period is  $N = 20$ .

Another important property of discrete-time eternal sinusoids is that the discrete-time frequency  $\Omega_0$  is periodic, which is not true for continuous-time sinusoids. For any integer  $k$ , Eq. (2.62) can be rewritten as

$$\begin{aligned} x[n] &= A \cos(\Omega_0 n + \theta), \quad -\infty < n < \infty \\ &= A \cos((\Omega_0 + 2\pi k)n + \theta), \\ &= A \cos(\Omega'_0 n + \theta), \quad -\infty < n < \infty, \end{aligned} \quad (2.66)$$

with

$$\Omega'_0 = \Omega_0 + 2\pi k. \quad (2.67)$$

Also, the nature of the variation of  $x[n]$  with  $n$  has a peculiar dependence on  $\Omega_0$ . Consider, for example, the sinusoid

$$x[n] = \cos(\Omega_0 n), \quad -\infty < n < \infty, \quad (2.68)$$

where, for simplicity, we assigned it an amplitude of 1 and a phase angle  $\theta = 0$ . Next, let us examine what happens as we increase  $\Omega_0$  from a value slightly greater than zero to  $2\pi$ . Initially, as  $\Omega_0$  is increased,  $x[n]$  oscillates faster and faster, until  $x[n]$  reaches a maximum rate of oscillation at  $\Omega_0 = \pi$ , namely

$$x[n] = \cos(\pi n) = (-1)^n \quad \text{at } (\Omega_0 = \pi). \quad (2.69)$$

At  $\Omega_0 = \pi$ ,  $x[n]$  oscillates as a function of  $n$  between  $(-1)$  and  $(+1)$ . As  $\Omega_0$  is increased beyond  $\pi$ , oscillation slows down and then stops altogether when  $\Omega_0$  reaches  $2\pi$ :

$$x[n] = \cos(2\pi n) = 1 \quad \text{at } (\Omega_0 = 2\pi). \quad (2.70)$$

Beyond  $\Omega_0 = 2\pi$ , the oscillatory behavior starts to increase again, and so on. This behavior has no equivalence in the world of continuous-time sinusoids.

### 2-5.3 1-D Discrete-Time Systems

A 1-D discrete-time system accepts an input  $x[n]$  and produces an output  $y[n]$ :



The definition of LTI for discrete-time systems is identical to the definition of LTI for continuous-time systems. If a discrete-time system has impulse response  $h[n]$ , then the output  $y[n]$  can be computed from the input  $x[n]$  using the discrete-time

convolution

$$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} h[i] x[n-i]. \quad (2.71a)$$

#### Discrete-time convolution

Most of the continuous-time properties of convolution also apply in discrete time.

Real-world signals and filters are defined over specified ranges of  $n$  (and set to zero outside those ranges). If  $h[n]$  has support in the interval  $[n_1, n_2]$ , Eq. (2.71a) becomes

$$y[n] = h[n] * x[n] = \sum_{i=n_1}^{n_2} h[i] x[n-i]. \quad (2.71b)$$

Reversing the sequence of  $h[n]$  and  $x[n]$  leads to the same outcome. That is, if  $x[n]$  has support in the interval  $[n_3, n_4]$ , then

$$y[n] = h[n] * x[n] = \sum_{i=n_3}^{n_4} x[i] h[n-i]. \quad (2.71c)$$

► The duration of the convolution of two signals of durations  $N_1$  and  $N_2$  is  $N_c = N_1 + N_2 - 1$ , not  $N_1 + N_2$ . Since  $h[n]$  is of length  $N_1 = n_2 - n_1 + 1$  and  $x[n]$  is of length  $N_2 = n_4 - n_3 + 1$ , the length of the convolution  $y[n]$  is  $N_c = N_1 + N_2 - 1 = (n_2 - n_1) + (n_4 - n_3) + 1$ . ◀

For causal signals ( $x[n]$  and  $h[n]$  equal to zero for  $n < 0$ ),  $y[n]$  assumes the form

$$y[n] = \sum_{i=0}^n x[i] h[n-i], \quad n \geq 0. \quad (2.71d)$$

#### Causal

For example,

$$\{1, 2\} * \{3, 4\} = \{3, 10, 8\}. \quad (2.72)$$

The duration of the output is  $2 + 2 - 1 = 3$ .

### 2-5.4 Discrete-Time Convolution Properties

With one notable difference, the properties of the discrete-time convolution are the same as those for continuous time. If (t)

**Table 2-6** Comparison of convolution properties for continuous-time and discrete-time signals.

Property	Continuous Time	Discrete Time
<b>Definition</b>	$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(\tau) x(t - \tau) d\tau$	$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} h[i] x[n-i]$
<b>1. Commutative</b>	$x(t) * h(t) = h(t) * x(t)$	$x[n] * h[n] = h[n] * x[n]$
<b>2. Associative</b>	$[g(t) * h(t)] * x(t) = g(t) * [h(t) * x(t)]$	$[g[n] * h[n]] * x[n] = g[n] * [h[n] * x[n]]$
<b>3. Distributive</b>	$x(t) * [h_1(t) + \dots + h_N(t)] = x(t) * h_1(t) + \dots + x(t) * h_N(t)$	$x[n] * [h_1[n] + \dots + h_N[n]] = x[n] * h_1[n] + \dots + x[n] * h_N[n]$
<b>4. Causal * Causal = Causal</b>	$y(t) = u(t) \int_0^t h(\tau) x(t - \tau) d\tau$	$y[n] = u[n] \sum_{i=0}^n h[i] x[n-i]$
<b>5. Time-Shift</b>	$h(t - T_1) * x(t - T_2) = y(t - T_1 - T_2)$	$h[n-a] * x[n-b] = y[n-a-b]$
<b>6. Sampling</b>	$x(t) * \delta(t - T) = x(t - T)$	$x[n] * \delta[n-a] = x[n-a]$
<b>7. Width</b>	$\text{width } y(t) = \text{width } x(t) + \text{width } h(t)$	$\text{width } y[n] = \text{width } x[n] + \text{width } h[n] - 1$
<b>8. Area</b>	$\text{area of } y(t) = \text{area of } x(t) \times \text{area of } h(t)$	$\sum_{n=-\infty}^{\infty} y[n] = \left( \sum_{n=-\infty}^{\infty} h[n] \right) \left( \sum_{n=-\infty}^{\infty} x[n] \right)$
<b>9. Convolution with Step</b>	$y(t) = x(t) * u(t) = \int_{-\infty}^t x(\tau) d\tau$	$x[n] * u[n] = \sum_{i=-\infty}^n x[i]$

is replaced with  $[n]$  and integrals are replaced with sums, the convolution properties listed in **Table 2-3** lead to those listed in **Table 2-6**.

The notable difference is associated with property #7. In discrete time, the width (duration) of a signal that is zero-valued outside interval  $[a, b]$  is  $b - a + 1$ , not  $b - a$ . Consider two signals,  $h[n]$  and  $x[n]$ , defined as follows:

Signal	From	To	Duration
$h[n]$	$a$	$b$	$b - a + 1$
$x[n]$	$c$	$d$	$d - c + 1$
$y[n]$	$a+c$	$b+d$	$(b+d) - (a+c) + 1$

where  $y[n] = h[n] * x[n]$ . Note that the duration of  $y[n]$  is

$$(b+d) - (a+c) + 1 = (b-a+1) + (d-c+1) - 1 \\ = \text{duration } h[n] + \text{duration } x[n] - 1.$$

## 2-5.5 Delayed-Impulses Computation Method

For finite-duration signals, computation of the convolution sum can be facilitated by expressing one of the signals as a linear combination of delayed impulses. The process is enabled by the sampling property (#6 in **Table 2-6**).

Consider, for example, the convolution sum of the two signals  $x[n] = \{2, 3, 4\}$  and  $h[n] = \{5, 6, 7\}$ , namely

$$y[n] = x[n] * h[n] = \{2, 3, 4\} * \{5, 6, 7\}.$$

The sampling property allows us to express  $x[n]$  in terms of impulses,

$$x[n] = 2\delta[n] + 3\delta[n-1] + 4\delta[n-2],$$

which leads to

$$y[n] = (2\delta[n] + 3\delta[n-1] + 4\delta[n-2]) * h[n] \\ = 2h[n] + 3h[n-1] + 4h[n-2].$$

Given that both  $x[n]$  and  $h[n]$  are of duration = 3, the duration of their sum is  $3 + 3 - 1 = 5$ , and it extends from  $n = 0$  to  $n = 4$ . Computing  $y[0]$  using the delayed-impulses method (while keeping in mind that  $h[i]$  has a non-zero value for only  $i = 0, 1$ , and 2) leads to

$$\begin{aligned} y[0] &= 2h[0] + 3h[-1] + 4h[-2] \\ &= 2 \times 5 + 3 \times 0 + 4 \times 0 = 10. \end{aligned}$$

The process can then be repeated to obtain the values of  $y[n]$  for  $n = 1, 2, 3$ , and 4.

#### Example 2-4: Discrete-Time Convolution

Given  $x[n] = \{2, 3, 4\}$  and  $h[n] = \{5, 6, 7\}$ , compute

$$y[n] = x[n] * h[n]$$

by (a) applying the sum definition and (b) graphically.

**Solution:** (a) Both signals have a length of 3 and start at time zero. That is,  $x[0] = 2$ ,  $x[1] = 3$ ,  $x[2] = 4$ , and  $x[i] = 0$  for all other values of  $i$ . Similarly,  $h[0] = 5$ ,  $h[1] = 6$ ,  $h[2] = 7$ , and  $h[i] = 0$  for all other values of  $i$ .

By Eq. (2.71d), the convolution sum of  $x[n]$  and  $h[n]$  is

$$y[n] = x[n] * h[n] = \sum_{i=0}^n x[i] h[n-i].$$

Since  $h[i] = 0$  for all values of  $i$  except  $i = 0, 1$ , and 2, it follows that  $h[n-i] = 0$  for all values of  $i$  except for  $i = n$ ,  $n-1$ , and  $n-2$ . With this constraint in mind, we can apply Eq. (2.71d) at discrete values of  $n$ , starting at  $n = 0$ :

$$\begin{aligned} y[0] &= \sum_{i=0}^0 x[i] h[0-i] = x[0] h[0] = 2 \times 5 = 10, \\ y[1] &= \sum_{i=0}^1 x[i] h[1-i] \\ &= x[0] h[1] + x[1] h[0] = 2 \times 6 + 3 \times 5 = 27, \\ y[2] &= \sum_{i=0}^2 x[i] h[2-i] \\ &= x[0] h[2] + x[1] h[1] + x[2] h[0] \\ &= 2 \times 7 + 3 \times 6 + 4 \times 5 = 52, \end{aligned}$$

$$\begin{aligned} y[3] &= \sum_{i=1}^2 x[i] h[3-i] \\ &= x[1] h[2] + x[2] h[1] = 3 \times 7 + 4 \times 6 = 45, \\ y[4] &= \sum_{i=2}^2 x[i] h[4-i] = x[2] h[2] = 4 \times 7 = 28, \\ y[n] &= 0, \text{ otherwise.} \end{aligned}$$

Hence,

$$y[n] = \{10, 27, 52, 45, 28\}.$$

(b) The convolution sum can be computed graphically through a four-step process.

**Step 1:** Replace index  $n$  with index  $i$  and plot  $x[i]$  and  $h[-i]$ , as shown in Fig. 2-12(a). Signal  $h[-i]$  is obtained from  $h[i]$  by reflecting it about the vertical axis.

**Step 2:** Superimpose  $x[i]$  and  $h[-i]$ , as in Fig. 2-12(b), and multiply and sum them. Their product is 10.

**Step 3:** Shift  $h[-i]$  to the right by 1 to obtain  $h[1-i]$ , as shown in Fig. 2-12(c). Multiplication and summation of  $x[i]$  by  $h[1-i]$  generates  $y[1] = 27$ . Shift  $h[1-i]$  by one more unit to the right to obtain  $h[2-i]$ , and then repeat the multiplication and summation process to obtain  $y[2]$ . Continue the shifting and multiplication and summation processes until the two signals no longer overlap.

**Step 4:** Use the values of  $y[n]$  obtained in step 3 to generate a plot of  $y[n]$ , as shown in Fig. 2-12(g);

$$y[n] = \{10, 27, 52, 45, 28\}.$$

**Concept Question 2-8:** Why are most discrete-time sinusoids not periodic?

**Concept Question 2-9:** Why is the length of the convolution of two discrete-time signals not equal to the sum of the lengths of the two signals?

**Exercise 2-9:** A 28 Hz sinusoid is sampled at 100 samples/s. What is  $\Omega_0$  for the resulting discrete-time sinusoid? What is the period of the resulting discrete-time sinusoid?

**Answer:**  $\Omega_0 = 0.56\pi$ ;  $N = 25$ . (See IP)

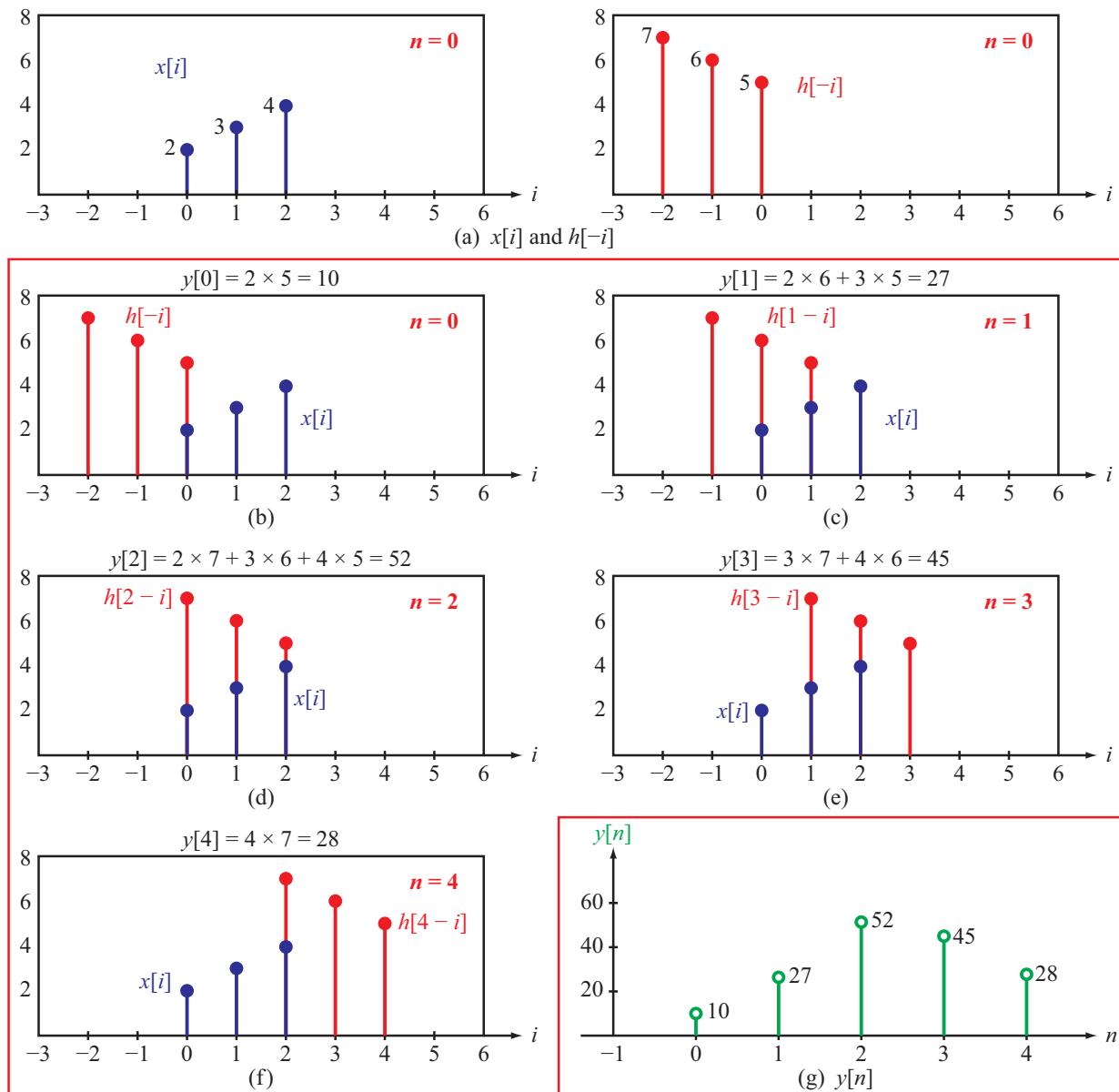


Figure 2-12 Graphical computation of convolution sum.

**Exercise 2-10:** Compute the output  $y[n]$  of a discrete-time LTI system with impulse response  $h[n]$  and input  $x[n]$ , where  $h[n] = \{3, 1\}$  and  $x[n] = \{1, 2, 3, 4\}$ .

**Answer:**  $\{3, 7, 11, 15, 4\}$ . (See [IP](#))

## 2-6 Discrete-Time Fourier Transform (DTFT)

The **discrete-time Fourier transform (DTFT)** is the discrete-time counterpart to the Fourier transform. It has the same two functions: (1) to compute spectra of signals and (2) to analyze the frequency responses of LTI systems.

### 2-6.1 Definition of the DTFT

The DTFT of  $x[n]$ , denoted  $\mathbf{X}(\Omega)$ , and its inverse are defined as

$$\mathbf{X}(\Omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\Omega n} \quad (2.73a)$$

and

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathbf{X}(\Omega) e^{j\Omega n} d\Omega. \quad (2.73b)$$

Readers familiar with the Fourier series will recognize that the DTFT  $\mathbf{X}(\Omega)$  is a Fourier series expansion with  $x[n]$  as the coefficients of the Fourier series. The inverse DTFT is simply the formula used for computing the coefficients  $x[n]$  of the Fourier series expansion of the periodic function  $\mathbf{X}(\Omega)$ .

We note that the DTFT definition given by Eq. (2.73a) is the same as the formula given by Eq. (2.53) for computing the spectrum  $\mathbf{X}_s(f)$  of a continuous-time signal  $x(t)$  directly from its samples  $\{x(n\Delta)\}$ , with  $\Omega = 2\pi f\Delta$ .

The inverse DTFT given by Eq. (2.73b) can be derived as follows. First, we introduce the **orthogonality property**

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\Omega(m-n)} d\Omega = \delta[m-n]. \quad (2.74)$$

To establish the validity of this property, we consider two cases, namely: (1) when  $m \neq n$  and (2) when  $m = n$ .

#### (a) $m \neq n$

Evaluation of the integral in Eq. (2.74) leads to

$$\begin{aligned} \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\Omega(m-n)} d\Omega &= \frac{e^{j\Omega(m-n)}}{2\pi j(m-n)} \Big|_{-\pi}^{\pi} \\ &= \frac{e^{j\pi(m-n)} - e^{-j\pi(m-n)}}{j2\pi(m-n)} \\ &= \frac{(-1)^{m-n} - (-1)^{m-n}}{j2\pi(m-n)} \\ &= 0, \quad (m \neq n). \end{aligned} \quad (2.75)$$

#### (b) $m = n$

If  $m = n$ , the integral reduces to

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\Omega(n-n)} d\Omega = \frac{1}{2\pi} \int_{-\pi}^{\pi} 1 d\Omega = 1. \quad (2.76)$$

The results given by Eqs. (2.75) and (2.76) can be combined into the definition of the orthogonality property given by Eq. (2.74).

Having verified the validity of the orthogonality property, we now use it to derive Eq. (2.73b). Upon multiplying the definition for the DTFT given by Eq. (2.73a) by  $\frac{1}{2\pi} e^{j\Omega m}$  and integrating over  $\Omega$ , we have

$$\begin{aligned} \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathbf{X}(\Omega) e^{j\Omega m} d\Omega &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \sum_{n=-\infty}^{\infty} x[n] e^{j\Omega(m-n)} d\Omega \\ &= \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} x[n] \int_{-\pi}^{\pi} e^{j\Omega(m-n)} d\Omega \\ &= \sum_{n=-\infty}^{\infty} x[n] \delta[m-n] = x[m]. \end{aligned} \quad (2.77)$$

Equation (2.74) was used in the final step leading to Eq. (2.77). Exchanging the order of integration and summation in Eq. (2.77) is acceptable if the summand is absolutely summable; i.e., if the DTFT is defined. Finally, replacing the index  $m$  with  $n$  in the top left-hand side and bottom right-hand side of Eq. (2.77) yields the inverse DTFT expression given by Eq. (2.73b).

## 2-6.2 Properties of the DTFT

The DTFT can be regarded as the Fourier transform of the sampled signal  $x_s(t)$  with a sampling interval  $\Delta = 1$ :

$$\mathbf{X}(\Omega) = \mathcal{F} \left\{ \sum_{n=-\infty}^{\infty} x[n] \delta(t-n) \right\}. \quad (2.78)$$

This statement can be verified by subjecting Eq. (2.73a) to the time-shift property of the Fourier transform (#3 in **Table 2-4**). Consequently, most (but not all) of the Fourier transform properties listed in **Table 2-4** extend directly to the DTFT with  $2\pi f$  replaced with  $\Omega$ , which we list here in **Table 2-7**. The exceptions mostly involve the following property of the DTFT:

► The DTFT  $\mathbf{X}(\Omega)$  is periodic with period  $2\pi$ . ◀

We also note the following special relationships between  $x[n]$  and  $\mathbf{X}(\Omega)$ :

$$\mathbf{X}(0) = \sum_{n=-\infty}^{\infty} x[n], \quad (2.79a)$$

$$x[0] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathbf{X}(\Omega) d\Omega, \quad (2.79b)$$

and

$$\mathbf{X}(\pm\pi) = \sum_{n=-\infty}^{\infty} (-1)^n x[n]. \quad (2.79c)$$

If  $x[n]$  is real-valued, then **conjugate symmetry** holds:

$$\mathbf{X}(\Omega)^* = \mathbf{X}(-\Omega).$$

**Parseval's theorem** for the DTFT states that the energy of  $x[n]$  is identical, whether computed in the discrete-time domain  $n$  or in the frequency domain  $\Omega$ :

$$\sum_{n=-\infty}^{\infty} |x[n]|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |\mathbf{X}(\Omega)|^2 d\Omega \quad (2.80)$$

The **energy spectral density** is now  $\frac{1}{2\pi} |\mathbf{X}(\Omega)|^2$ .

Finally, by analogy to continuous time, a discrete-time ideal lowpass filter with cutoff frequency  $\Omega_0$  has the frequency response for  $|\Omega| < \pi$  (recall that  $\mathbf{H}(\Omega)$  is periodic with period  $\pi$ )

$$\mathbf{H}(\Omega) = \begin{cases} 1 & \text{for } |\Omega| < \Omega_0, \\ 0 & \text{for } \Omega_0 < |\Omega| \leq \pi, \end{cases} \quad (2.81)$$

which eliminates frequency components of  $x[n]$  that lie in the range  $\Omega_0 < |\Omega| \leq \pi$ .

Property	$x[n]$	$\mathbf{X}(\Omega)$
<b>1. Linearity</b>	$\sum c_i x_i[n]$	$\leftrightarrow$
<b>2. Time shift</b>	$x[n - n_0]$	$\leftrightarrow$
<b>3. Modulation</b>	$x[n] e^{j\Omega_0 n}$	$\leftrightarrow$
<b>4. Time reversal</b>	$x[-n]$	$\leftrightarrow$
<b>5. Conjugation</b>	$x^*[n]$	$\leftrightarrow$
<b>6. Time convolution</b>	$h[n] * x[n]$	$\leftrightarrow$
<b>Special DTFT Relationships</b>		
<b>7. Conjugate symmetry</b>	$\mathbf{X}^*(\Omega) = \mathbf{X}(-\Omega)$	
<b>8. Zero frequency</b>	$\mathbf{X}(0) = \sum_{n=-\infty}^{\infty} x[n]$	
<b>9. Zero time</b>	$x[0] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathbf{X}(\Omega) d\Omega$	
<b>10. <math>\Omega = \pm\pi</math></b>	$\mathbf{X}(\pm\pi) = \sum_{n=-\infty}^{\infty} (-1)^n x[n]$	
<b>11. Rayleigh's (often called Parseval's) theorem</b>	$\sum_{n=-\infty}^{\infty}  x[n] ^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi}  \mathbf{X}(\Omega) ^2 d\Omega$	

## 2-6.3 Important DTFT Pairs

For easy access, several DTFT pairs are provided in **Table 2-8**. In all cases, the expressions for  $\mathbf{X}(\Omega)$  are periodic with period  $2\pi$ , as they should be.

Entries #7 and #8 of **Table 2-8** deserve more discussion, which we now present.

**Table 2-8** Discrete-time Fourier transform (DTFT) pairs.

	$x[n]$	$X(\Omega)$	Condition
1.	$\delta[n]$	$\leftrightarrow$	1
1a.	$\delta[n-m]$	$\leftrightarrow$	$e^{-jm\Omega}$
2.	1	$\leftrightarrow$	$2\pi \sum_{k=-\infty}^{\infty} \delta(\Omega - 2\pi k)$
3.	$e^{j\Omega_0 n}$	$\leftrightarrow$	$2\pi \sum_{k=-\infty}^{\infty} \delta(\Omega - \Omega_0 - 2\pi k)$
4.	$\cos(\Omega_0 n)$	$\leftrightarrow$	$\pi \sum_{k=-\infty}^{\infty} [\delta(\Omega - \Omega_0 - 2\pi k) + \delta(\Omega + \Omega_0 - 2\pi k)]$
5.	$\sin(\Omega_0 n)$	$\leftrightarrow$	$\frac{\pi}{j} \sum_{k=-\infty}^{\infty} [\delta(\Omega - \Omega_0 - 2\pi k) - \delta(\Omega + \Omega_0 - 2\pi k)]$
6.	$a^n \cos(\Omega_0 n + \theta) u[n]$	$\leftrightarrow$	$\frac{e^{j2\Omega} \cos \theta - a e^{j\Omega} \cos(\Omega_0 - \theta)}{e^{j2\Omega} - 2a e^{j\Omega} \cos \Omega_0 + a^2}$
7.	$\text{rect}\left[\frac{n}{N}\right]$	$\leftrightarrow$	$\frac{\sin(\Omega(N + \frac{1}{2}))}{\sin(\frac{\Omega}{2})}$
8.	$\frac{\Omega_0}{\pi} \text{sinc}\left(\frac{\Omega_0}{\pi} n\right) = \frac{\sin(\Omega_0 n)}{\pi n}$	$\leftrightarrow$	$\sum_{k=-\infty}^{\infty} \text{rect}\left(\frac{\Omega - 2\pi k}{2\Omega_0}\right)$

## A. Discrete-Time Sinc Functions

by

$$h_{\text{FIR}}[n] = h[n] h_{\text{Ham}}[n]$$

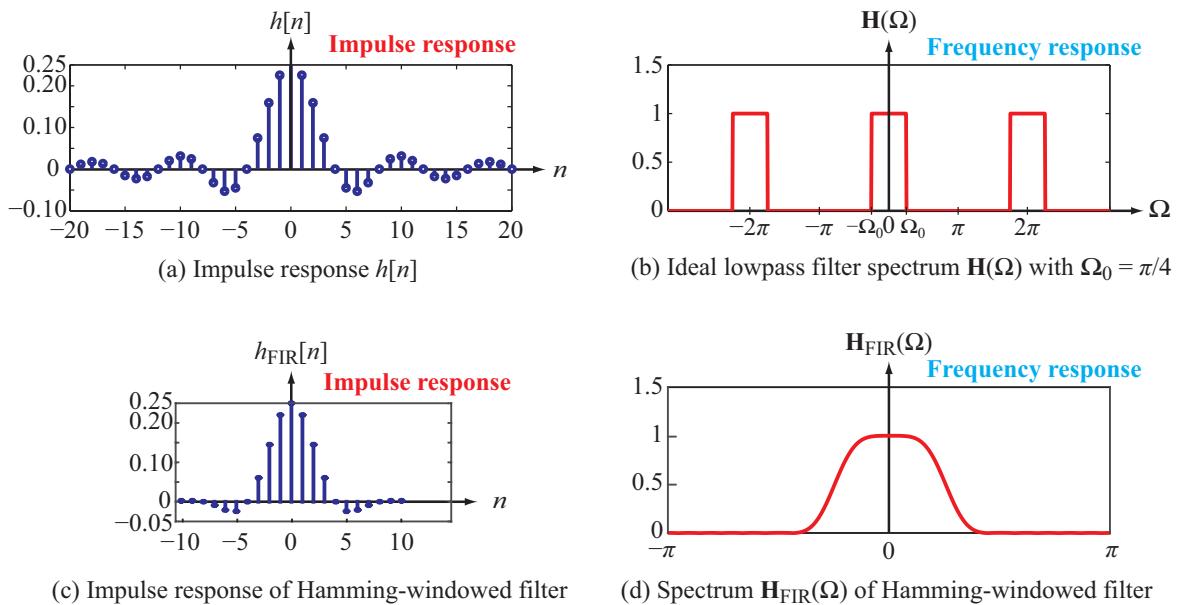
$$= \begin{cases} \underbrace{\frac{\Omega_0}{\pi} \text{sinc}\left(\frac{\Omega_0}{\pi} n\right)}_{h[n]} \underbrace{\left(0.54 + 0.46 \cos\left(\frac{\pi n}{N}\right)\right)}_{h_{\text{Ham}}[n]}, & |n| \leq N, \\ 0, & |n| > N. \end{cases} \quad (2.83)$$

The impulse response of an ideal lowpass filter is

$$h[n] = \int_{-\Omega_0}^{\Omega_0} 1 e^{j\Omega n} d\Omega = \frac{\Omega_0}{\pi} \text{sinc}\left(\frac{\Omega_0}{\pi} n\right). \quad (2.82)$$

This is called a **discrete-time sinc function**. A discrete-time sinc function  $h[n]$  with  $\Omega_0 = \pi/4$  is displayed in **Fig. 2-13(a)**, along with its frequency response  $H(\Omega)$  in **Fig. 2-13(b)**. Such a filter is impractical for real-world applications, because it is unstable and it has infinite duration. To override these limitations, we can multiply  $h[n]$  by a window function, such as a **Hamming window**. The modified impulse response  $h_{\text{FIR}}[n]$  is then given

As can be seen in **Fig. 2-13(c)** and **(d)**,  $h_{\text{FIR}}[n]$  with  $N = 10$  provides a good approximation to an ideal lowpass filter, with a finite duration. The Hamming-windowed filter belongs to a group of filters called **finite-impulse response (FIR)** filters. FIR filters can also be designed using a minimax criterion, resulting in an equiripple filter. This and other FIR filter design procedures are discussed in discrete-time signal processing textbooks.



**Figure 2-13** Parts (a) and (b) are for an ideal lowpass filter with  $\Omega_0 = \pi/4$ , and parts (c) and (d) are for the same filter after multiplying its impulse response with a Hamming window of length  $N = 10$ .

## B. Discrete Sinc Functions

A **discrete-time rectangle function**  $\text{rect}\left[\frac{n}{N}\right]$  is defined as

$$\text{rect}\left[\frac{n}{N}\right] = \begin{cases} 1 & \text{for } |n| \leq N, \\ 0 & \text{for } |n| > N. \end{cases} \quad (2.84)$$

We note that  $\text{rect}\left[\frac{n}{N}\right]$  has duration  $2N + 1$ . This differs from the continuous-time rect function  $\text{rect}(t/T)$ , which has duration  $T$ . The DTFT of  $\text{rect}\left[\frac{n}{N}\right]$  is obtained from Eq. (2.73a) by setting  $x[n] = 1$  and limiting the summation to the range  $(-N, N)$ :

$$\text{DTFT}\{\text{rect}[n/N]\} = \sum_{n=-N}^N e^{-j\Omega n}. \quad (2.85)$$

Using the formula

$$\sum_{k=0}^N r^k = \frac{1 - r^{N+1}}{1 - r} \quad (2.86)$$

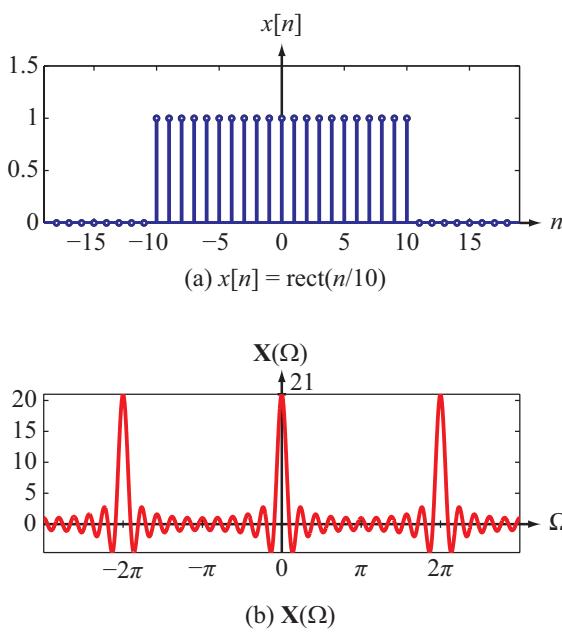
with  $r = e^{j\Omega}$ , the summation in Eq. (2.85) becomes

$$\begin{aligned} \sum_{n=-N}^N e^{-j\Omega n} &= e^{-j\Omega N} \sum_{n=0}^{2N} e^{j\Omega n} \\ &= e^{-j\Omega N} \frac{1 - e^{j\Omega(2N+1)}}{1 - e^{j\Omega}} \\ &= \frac{\sin((2N+1)\Omega/2)}{\sin(\Omega/2)}. \end{aligned} \quad (2.87)$$

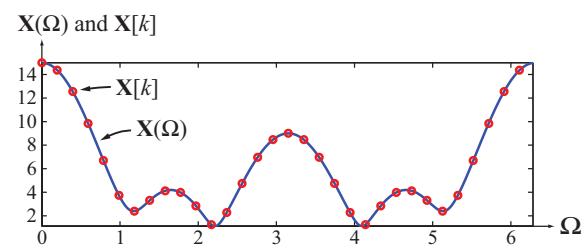
This is called a **discrete (or periodic) sinc function**. A rectangular pulse with  $N = 10$  is shown in Fig. 2-14 along with its DTFT.

**Concept Question 2-10:** Why does the DTFT share so many properties with the CTFT?

**Concept Question 2-11:** Why is the DTFT periodic in frequency?



**Figure 2-14** Discrete-time rectangle function with  $N = 10$  and its DTFT.



**Figure 2-15** The DFT  $\mathbf{X}[k]$  is a sampled version of the DTFT  $\mathbf{X}(\Omega)$ .

**Exercise 2-12:** Compute the inverse DTFT of

$$\text{DTFT}^{-1}[4\cos(2\Omega) + 6\cos(\Omega) + j8\sin(2\Omega) + j2\sin(\Omega)].$$

**Answer:**

$$\begin{aligned} \text{DTFT}^{-1}[4\cos(2\Omega) + 6\cos(\Omega) + j8\sin(2\Omega) + j2\sin(\Omega)] \\ = \{ 6, 4, 0, 2, -2 \}. \end{aligned}$$

(See [IP](#))

## 2-7 Discrete Fourier Transform (DFT)

The **discrete Fourier transform (DFT)** is the numerical bridge between the DTFT and the **fast Fourier transform (FFT)**. For a signal  $\{x[n], n = 0, \dots, M-1\}$  of duration  $M$ , its DFT of order  $N$  is  $\mathbf{X}[k]$ , where  $\mathbf{X}[k]$  is  $\mathbf{X}(\Omega)$  sampled at the  $N$  frequencies  $\Omega = \{2\pi k/N, k = 0, \dots, N-1\}$ :

$$\mathbf{X}[k] = \mathbf{X}(\Omega = 2\pi k/N), \quad k = 0, \dots, N-1. \quad (2.88)$$

An example is shown in **Fig. 2-15**. Usually  $N = M$ ; i.e., the order  $N$  of the DFT is equal to the duration  $M$  of  $x[n]$ . However, in some situations, it is desirable to select  $N$  to be larger than  $M$ . For example, to compute and plot the DTFT of a short-duration signal such as  $x[n] = \{3, 1, 4\}$ , for which  $M = 3$ , we may choose  $N$  to be 256 or 512 so as to produce a smooth plot, such as the blue plot in **Fig. 2-15**. Choosing  $N > M$  will also allow the use of the FFT to compute convolutions quickly (see Section 2-7.2C).

As a result, the properties of the DFT follow those of the DTFT, with some exceptions, such as time reversal and cyclic convolution (discussed later in Section 2-7.2C).

**Exercise 2-11:** Compute the DTFT of  $4\cos(0.15\pi n + 1)$ .

**Answer:**

$$\begin{aligned} \text{DTFT}[4\cos(0.15\pi n + 1)] \\ = \sum_{k=-\infty}^{\infty} 4\pi e^{j1} [\delta(\Omega - 0.15 - 2k\pi)] \\ + \sum_{k=-\infty}^{\infty} 4\pi e^{-j1} [\delta(\Omega + 0.15 - 2k\pi)]. \end{aligned}$$

(See [IP](#))

► To avoid confusion between the DTFT and the DFT, the DFT, being a discrete function of integer  $k$ , uses square brackets, as in  $\mathbf{X}[k]$ , while the DTFT, being a continuous and periodic function of real numbers  $\Omega$ , uses round parentheses, as in  $\mathbf{X}(\Omega)$ . ◀

by  $\frac{1}{N} e^{j2\pi mk/N}$  and summing over  $k$  gives

$$\begin{aligned} \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{X}[k] e^{j2\pi mk/N} &= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{n=0}^{M-1} x[n] e^{j2\pi(m-n)k/N} \\ &= \frac{1}{N} \sum_{n=0}^{M-1} x[n] \sum_{k=0}^{N-1} e^{j2\pi(m-n)k/N} \\ &= \sum_{n=0}^{M-1} x[n] \delta[m-n] \\ &= \begin{cases} x[m] & \text{for } 0 \leq m \leq M-1, \\ 0 & \text{for } M \leq m \leq N-1. \end{cases} \end{aligned} \quad (2.91)$$

Upon changing index  $m$  to  $n$  on the left-hand side of Eq. (2.91) and in the right-hand side in  $x[m]$ , we obtain the formal definition of the inverse DFT given by Eq. (2.89b).

The main use of the DFT is to compute spectra of signals and frequency responses of LTI systems. *All plots of spectra in this book (and all other books on signal and image processing) were made by computing them using the DFT and plotting the results.*

## 2-7.1 Definition of the DFT

The ***N-point*** (or  $N$ th-order) DFT of  $\{x[n], n = 0, \dots, M-1\}$ , denoted  $\mathbf{X}[k]$ , and the inverse DFT of  $\mathbf{X}[k]$ , namely  $x[n]$ , are defined as

$$\mathbf{X}[k] = \sum_{n=0}^{M-1} x[n] e^{-j2\pi nk/N}, \quad k = 0, \dots, N-1, \quad (2.89a)$$

and

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{X}[k] e^{j2\pi nk/N}, \quad n = 0, \dots, M-1. \quad (2.89b)$$

The definition for  $\mathbf{X}[k]$  given by Eq. (2.89a) is obtained by applying Eq. (2.88) to Eq. (2.73a), namely by replacing  $\Omega$  with  $2\pi k/N$  and limiting the range of summation over  $n$  to  $[0, M-1]$ .

For the inverse DFT, the definition given by Eq. (2.89b) can be derived by following a process similar to that we presented earlier in Section 2-6.1 in connection with the inverse DTFT. Specifically, we start with the discrete equivalent of the ***orthogonality property*** given by Eq. (2.74):

$$\frac{1}{N} \sum_{k=0}^{N-1} e^{j2\pi(m-n)k/N} = \delta[m-n]. \quad (2.90)$$

Next, multiplying the definition of the DFT given by Eq. (2.89a)

### Example 2-5: DFT of Periodic Sinusoids

Compute the  $N$ -point DFT of the segment of a periodic discrete-time sinusoid

$$x[n] = A \cos(2\pi(k_0/N)n + \theta), \quad 0 \leq n \leq N-1, \quad (2.92)$$

with  $k_0$  a fixed integer.

**Solution:** We start by rewriting  $x[n]$  as the sum of two exponentials:

$$\begin{aligned} x[n] &= \frac{A}{2} e^{j\theta} e^{j2\pi k_0 n/N} + \frac{A}{2} e^{-j\theta} e^{-j2\pi k_0 n/N} \\ &= \frac{A}{2} e^{j\theta} e^{j2\pi k_0 n/N} + \frac{A}{2} e^{-j\theta} e^{j2\pi(N-k_0)n/N}, \end{aligned} \quad (2.93)$$

where we have multiplied the second term in the first step by  $e^{j2\pi N/N} = 1$ .

Inserting Eq. (2.93) into Eq. (2.89a) with  $M = N$ , we have

$$\begin{aligned}\mathbf{X}[k] &= \sum_{n=0}^{N-1} \frac{A}{2} e^{j\theta} e^{j2\pi k_0 n/N} e^{-j2\pi n k/N} \\ &\quad + \sum_{n=0}^{N-1} \frac{A}{2} e^{-j\theta} e^{j2\pi(N-k_0)n/N} e^{-j2\pi n k/N} \\ &= \frac{A}{2} e^{j\theta} \sum_{n=0}^{N-1} e^{j2\pi n(k_0-k)/N} \\ &\quad + \frac{A}{2} e^{-j\theta} \sum_{n=0}^{N-1} e^{j2\pi n(N-k_0-k)/N}. \quad (2.94a)\end{aligned}$$

In view of the orthogonality property given by Eq. (2.90), the summations simplify to impulses, resulting in

$$\mathbf{X}[k] = \frac{A}{2} e^{j\theta} N \delta[k - k_0] + \frac{A}{2} e^{-j\theta} N \delta[N - k - k_0], \quad (2.94b)$$

which can be restated as

$$\mathbf{X}[k] = \begin{cases} \frac{N}{2} A e^{j\theta} & \text{for } k = k_0, \\ \frac{N}{2} A e^{-j\theta} & \text{for } k = N - k_0, \\ 0 & \text{otherwise.} \end{cases} \quad (2.94c)$$

Thus, the DFT of a segment of a periodic sinusoid with  $\Omega_0 = 2\pi k_0/N$  consists of two discrete-time impulses, at indices  $k = k_0$  and  $k = N - k_0$ .

## 2-7.2 Properties of the DFT

**Table 2-9** provides a summary of the salient properties of the DFT, as well as some of the special relationships between  $x[n]$  and  $\mathbf{X}[k]$ . Of particular note are the three relationships

$$\mathbf{X}[0] = \sum_{n=0}^{N-1} x[n], \quad (2.95)$$

$$x[0] = \frac{1}{N} \sum_{n=0}^{N-1} \mathbf{X}[k], \quad (2.96)$$

and

$$\mathbf{X}[N/2] = \sum_{n=0}^{N-1} (-1)^n x[n] \quad \text{for } N \text{ even.} \quad (2.97)$$

**Table 2-9 Properties of the DFT.** In the time-shift and modulation properties,  $(n - n_0)$  and  $(k - k_0)$  must be reduced mod( $N$ ).

Property	$x[n]$	$\leftrightarrow$	$\mathbf{X}[k]$
<b>1. Linearity</b>	$\sum c_i x_i[n]$	$\leftrightarrow$	$\sum c_i \mathbf{X}_i[k]$
<b>2. Time shift</b>	$x[n - n_0]$	$\leftrightarrow$	$e^{-j2\pi n_0 k/N} \mathbf{X}[k]$
<b>3. Modulation</b>	$e^{j2\pi k_0 n/N} x[n]$	$\leftrightarrow$	$\mathbf{X}[k - k_0]$
<b>4. Time reversal</b>	$x[N - n]$	$\leftrightarrow$	$\mathbf{X}[N - k]$
<b>5. Conjugation</b>	$x^*[n]$	$\leftrightarrow$	$\mathbf{X}^*[N - k]$
<b>6. Convolution</b>	$h[n] \odot x[n]$	$\leftrightarrow$	$\mathbf{H}[k] \mathbf{X}[k]$

### Special DFT Relationships

<b>7. Conjugate symmetry</b>	$\mathbf{X}^*[k] = \mathbf{X}[N - k]$
<b>8. Zero frequency</b>	$\mathbf{X}[0] = \sum_{n=0}^{N-1} x[n]$
<b>9. Zero time</b>	$x[0] = \frac{1}{N} \sum_{n=0}^{N-1} \mathbf{X}[k]$
<b>10. <math>k = N/2</math></b>	$\mathbf{X}[N/2] = \sum_{n=0}^{N-1} (-1)^n x[n]$
<b>11. Parseval's theorem</b>	$\sum_{n=0}^{N-1}  x[n] ^2 = \frac{1}{N} \sum_{k=0}^{N-1}  \mathbf{X}[k] ^2$

### A. Conjugate Symmetry Property of the DFT

If  $x[n]$  is real-valued, then conjugate symmetry holds for the DFT, which takes the form

$$\mathbf{X}^*[k] = \mathbf{X}[N - k], \quad k = 1, \dots, N - 1. \quad (2.98)$$

For example, the 4-point DFT of  $x[n] = \{1, 2, 3, 4\}$  is

$$\mathbf{X}[k] = \{10, -2 + j2, -2, -2 - j2\}$$

and

$$\mathbf{X}^*[1] = -2 - j2 = \mathbf{X}[4 - 1] = \mathbf{X}[3] = -2 - j2.$$

Similarly,

$$\mathbf{X}^*[2] = \mathbf{X}[4 - 2] = \mathbf{X}[2] = -2,$$

which is real-valued. This conjugate-symmetry property follows from the definition of the DFT given by Eq. (2.89a):

$$\mathbf{X}^*[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N} \quad (2.99)$$

and

$$\mathbf{X}[N-k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi n(N-k)/N} = \sum_{n=0}^{N-1} x[n] e^{-j2\pi n} e^{j2\pi k/N}. \quad (2.100)$$

Since  $n$  is an integer,  $e^{-j2\pi n} = 1$  and Eq. (2.100) reduces to

$$\mathbf{X}[N-k] = \sum_{n=0}^{N-1} e^{j2\pi nk/N} = \mathbf{X}^*[k].$$

## B. Use of DFT for Convolution

The convolution property of the DTFT extends to the DFT after some modifications. Consider two signals,  $x_1[n]$  and  $x_2[n]$ , with  $N$ -point DFTs  $\mathbf{X}_1[k]$  and  $\mathbf{X}_2[k]$ . From Eq. (2.89b), the inverse DFT of their product is

$$\begin{aligned} \text{DFT}^{-1}(\mathbf{X}_1[k] \mathbf{X}_2[k]) &= \frac{1}{N} \sum_{k=0}^{N-1} (\mathbf{X}_1[k] \mathbf{X}_2[k]) e^{jk \frac{2\pi}{N} n} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} e^{jk \frac{2\pi}{N} n} \left[ \sum_{n_1=0}^{N-1} x_1[n_1] e^{-jk \frac{2\pi}{N} n_1} \right] \\ &\quad \cdot \left[ \sum_{n_2=0}^{N-1} x_2[n_2] e^{-jk \frac{2\pi}{N} n_2} \right]. \end{aligned} \quad (2.101)$$

Rearranging the order of the summations gives

$$\begin{aligned} \text{DFT}^{-1}(\mathbf{X}_1[k] \mathbf{X}_2[k]) &= \\ \frac{1}{N} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x_1[n_1] x_2[n_2] &\sum_{k=0}^{N-1} e^{jk \frac{2\pi}{N} (n-n_1-n_2)}. \end{aligned} \quad (2.102)$$

In view of the orthogonality property given by Eq. (2.90), Eq. (2.102) reduces to

$$\begin{aligned} \text{DFT}^{-1}(\mathbf{X}_1[k] \mathbf{X}_2[k]) &= \\ \frac{1}{N} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x_1[n_1] x_2[n_2] N \delta[(n-n_1-n_2)_N] & \\ = \sum_{n_1=0}^{N-1} x_1[n_1] x_2[(n-n_1)_N], \end{aligned} \quad (2.103)$$

where  $(n-n_1)_N$  means  $(n-n_1)$  reduced mod  $N$  (i.e., reduced by the largest integer multiple of  $N$  without  $(n-n_1)$  becoming negative).

## C. DFT and Cyclic Convolution

Because of the mod  $N$  reduction cycle, the expression on the right-hand side of Eq. (2.103) is called the **cyclic** or circular convolution of signals  $x_1[n]$  and  $x_2[n]$ . The terminology helps distinguish it from the traditional *linear* convolution of two nonperiodic signals.

*The symbol commonly used to denote cyclic convolution is  $\odot$ .* Combining Eqs. (2.101) and (2.103) leads to

$$\begin{aligned} y_c[n] &= x_1[n] \odot x_2[n] = \sum_{n_1=0}^{N-1} x_1[n_1] x_2[(n-n_1)_N] \\ &= \text{DFT}^{-1}(\mathbf{X}_1[k] \mathbf{X}_2[k]) \\ &= \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{X}_1[k] \mathbf{X}_2[k] e^{jk \frac{2\pi}{N} n}. \end{aligned} \quad (2.104)$$

The cyclic convolution  $y_c[n]$  can certainly be computed by applying Eq. (2.104), but it can also be computed from the linear convolution  $x_1[n] * x_2[n]$  by aliasing the latter. To illustrate, suppose  $x_1[n]$  and  $x_2[n]$  are both of duration  $N$ . The **linear convolution** of the two signals

$$y[n] = x_1[n] * x_2[n] \quad (2.105)$$

is of duration  $2N-1$ , extending from  $n=0$  to  $n=2N-2$ . **Aliasing**  $y[n]$  means defining  $z[n]$ , the aliased version of  $y[n]$ , as

$$\begin{aligned} z[0] &= y[0] + y[0+N] \\ z[1] &= y[1] + y[1+N] \\ &\vdots \\ z[N-2] &= y[N-2] + y[2N-2] \\ z[N-1] &= y[N-1]. \end{aligned} \quad (2.106)$$

The aliasing process leads to the result that  $z[n]$  is the cyclic convolution of  $x_1[n]$  and  $x_2[n]$ :

$$y_c[n] = z[n] = x_1[n] \odot x_2[n]. \quad (2.107)$$

**Example 2-6: Cyclic Convolution**

Given the two signals

$$\begin{aligned}x_1[n] &= \{2, 1, 4, 3\}, \\x_2[n] &= \{5, 3, 2, 1\},\end{aligned}$$

compute the cyclic convolution of the two signals by

- (a) applying the DFT method;
- (b) applying the aliasing of the linear convolution method.

**Solution:**

(a) With  $N = 4$ , application of Eq. (2.89a) to  $x_1[n]$  and  $x_2[n]$  leads to

$$\begin{aligned}\mathbf{X}_1[k] &= \{10, -2 + j2, 2, -2 - j2\}, \\ \mathbf{X}_2[k] &= \{11, 3 - j2, 3, 3 + j2\}.\end{aligned}$$

The point-by-point product of  $\mathbf{X}_1[k]$  and  $\mathbf{X}_2[k]$  is

$$\begin{aligned}\mathbf{X}_1[k] \mathbf{X}_2[k] \\ &= \{10 \times 11, (-2 + j2)(3 - j2), 2 \times 3, (-2 - j2)(3 + j2)\} \\ &= \{110, -2 + j10, 6, -2 - j10\}.\end{aligned}$$

Application of Eq. (2.104) leads to

$$x_1[n] \odot x_2[n] = \{28, 21, 30, 31\}.$$

(b) Per Eq. (2.71d), the linear convolution of

$$x_1[n] = \{2, 1, 4, 3\}$$

and

$$x_2[n] = \{5, 3, 2, 1\}$$

is

$$\begin{aligned}y[n] &= x_1[n] * x_2[n] \\ &= \sum_{i=0}^3 x_1[i] x_2[n-i] \\ &= \{10, 11, 27, 31, 18, 10, 3\}.\end{aligned}$$

Per Eq. (2.106),

$$\begin{aligned}z[n] &= \{y[0] + y[4], y[1] + y[5], y[2] + y[6], y[3]\} \\ &= \{10 + 18, 11 + 10, 27 + 3, 31\} = \{28, 21, 30, 31\}.\end{aligned}$$

Hence, by Eq. (2.107),

$$x_1[n] \odot x_2[n] = z[n] = \{28, 21, 30, 31\},$$

which is the same answer obtained in part (a).

## 2-7.3 DFT and Linear Convolution

In the preceding subsection, we examined how the DFT can be used to compute the cyclic convolution of two discrete-time signals (Eq. (2.104)). The same method can be applied to compute the linear convolution of the two signals, provided a preparatory step of **zero-padding** the two signals is applied first.

Let us suppose that signal  $x_1[n]$  is of duration  $N_1$  and signal  $x_2[n]$  is of duration  $N_2$ , and we are interested in computing their linear convolution

$$y[n] = x_1[n] * x_2[n].$$

The duration of  $y[n]$  is

$$N_c = N_1 + N_2 - 1. \quad (2.108)$$

Next, we zero-pad  $x_1[n]$  and  $x_2[n]$  so that their durations are equal to or greater than  $N_c$ . As we will see in Section 2-8 on how the fast Fourier transform (FFT) is used to compute the DFT, it is advantageous to choose the total length of the zero-padded signals to be  $M$  such that  $M \geq N_c$ , and simultaneously  $M$  is a power of 2.

The zero-padded signals are defined as

$$x'_1[n] = \underbrace{x_1[n]}_{N_1}, \underbrace{0, \dots, 0}_{M-N_1}, \quad (2.109a)$$

$$x'_2[n] = \underbrace{x_2[n]}_{N_2}, \underbrace{0, \dots, 0}_{M-N_2}, \quad (2.109b)$$

and their  $M$ -point DFTs are  $\mathbf{X}'_1[k]$  and  $\mathbf{X}'_2[k]$ , respectively. The linear convolution  $y[n]$  can now be computed by a modified version of Eq. (2.104), namely

$$\begin{aligned}y[n] &= x'_1[n] * x'_2[n] \\ &= \text{DFT}^{-1} \{ \mathbf{X}'_1[k] \mathbf{X}'_2[k] \} \\ &= \frac{1}{M} \sum_{k=0}^{M-1} \mathbf{X}'_1[k] \mathbf{X}'_2[k] e^{j2\pi nk/M}. \quad (2.110)\end{aligned}$$

Note that the DFTs can be computed using  $M$ -point DFTs of  $x'_1[n]$  and  $x'_2[n]$ , since using an  $M$ -point DFT performs the zero-padding automatically.

**Example 2-7: DFT Convolution**

Given signals  $x_1[n] = \{4, 5\}$  and  $x_2[n] = \{1, 2, 3\}$ , (a) compute their convolution in discrete time, and (b) compare the result with the DFT relation given by Eq. (2.110).

**Solution:**

(a) Application of Eq. (2.71a) gives

$$x_1[n] * x_2[n] = \sum_{i=0}^3 x_1[i] x_2[n-i] = \{4, 13, 22, 15\}.$$

(b) Since  $x_1[n]$  is of length  $N_1 = 2$  and  $x_2[n]$  is of length  $N_2 = 3$ , their convolution is of length

$$N_c = N_1 + N_2 - 1 = 2 + 3 - 1 = 4.$$

Hence, we need to zero-pad  $x_1[n]$  and  $x_2[n]$  as

$$x'_1[n] = \{4, 5, 0, 0\}$$

and

$$x'_2[n] = \{1, 2, 3, 0\}.$$

From Eq. (2.89a) with  $N = N_c = 4$ , the 4-point DFT of  $x'_1[n] = \{4, 5, 0, 0\}$  is

$$\mathbf{X}_1[k] = \sum_{n=0}^3 x'_1[n] e^{-j k \pi n / 2}, \quad k = 0, 1, 2, 3,$$

which gives

$$\begin{aligned} \mathbf{X}'_1[0] &= 4(1) + 5(j) + 0(1) + 0(j) = 9, \\ \mathbf{X}'_1[1] &= 4(1) + 5(-j) + 0(-1) + 0(j) = 4 - j5, \\ \mathbf{X}'_1[2] &= 4(1) + 5(-1) + 0(1) + 0(-1) = -1, \end{aligned}$$

and

$$\mathbf{X}'_1[3] = 4(1) + 5(j) + 0(-1) + 0(-j) = 4 + j5.$$

Similarly, the 4-point DFT of  $x'_2[n] = \{1, 2, 3, 0\}$  gives

$$\begin{aligned} \mathbf{X}'_2[0] &= 6, \\ \mathbf{X}'_2[1] &= -2 - j2, \\ \mathbf{X}'_2[2] &= 2, \end{aligned}$$

and

$$\mathbf{X}'_2[3] = -2 + j2.$$

Multiplication of corresponding pairs gives

$$\begin{aligned} \mathbf{X}'_1[0] \mathbf{X}'_2[0] &= 9 \times 6 = 54, \\ \mathbf{X}'_1[1] \mathbf{X}'_2[1] &= (4 - j5)(-2 - j2) = -18 + j2, \\ \mathbf{X}'_1[2] \mathbf{X}'_2[2] &= -1 \times 2 = -2, \end{aligned}$$

and

$$\mathbf{X}'_1[3] \mathbf{X}'_2[3] = (4 + j5)(-2 + j2) = -18 - j2.$$

Application of Eq. (2.110) gives

$$\begin{aligned} y[n] &= x'_1[n] * x'_2[n] = \frac{1}{N_c} \sum_{k=0}^{N_c-1} \mathbf{X}'_1[k] \mathbf{X}'_2[k] e^{j 2 \pi n k / N_c} \\ &= \frac{1}{4} \sum_{k=0}^3 \mathbf{X}'_1[k] \mathbf{X}'_2[k] e^{j k \pi n / 2}. \end{aligned}$$

Evaluating the summation for  $n = 0, 1, 2$  and  $3$  leads to

$$y[n] = x'_1[n] * x'_2[n] = \{4, 13, 22, 15\},$$

which is identical to the answer obtained earlier in part (a). For simple signals like those in this example, the DFT method involves many more steps than does the straightforward convolution method of part (a), but for the type of signals used in practice, the DFT method is computationally superior.

► To summarize, the linear convolution  $y[n] = h[n] * x[n]$  defined in Eq. (2.71) computes the response (output)  $y[n]$  to the input  $x[n]$  for an LTI system with impulse response  $h[n]$ . The cyclic convolution  $y_c[n] = h[n] \odot x[n]$  defined in Eq. (2.104) is what the DFT maps to products, so a cyclic convolution can be computed very quickly using the FFT algorithm to compute the DFTs  $\mathbf{H}[k]$  of  $h[n]$  and  $\mathbf{X}[k]$  of  $x[n]$ , and the inverse DFT of  $\mathbf{H}[k] \mathbf{X}[k]$ . Fortunately, a linear convolution can be zero-padded to a cyclic convolution, as presented in Subsection 2-7.3, so linear convolutions can also be computed quickly using the FFT algorithm. Cyclic convolutions will also be used in Chapter 7 for computing wavelet transforms, because the cyclic convolution of a signal  $x[n]$  of duration  $N$  with an impulse response  $h[n]$  (that has been zero-padded to length  $N$ ) gives an output  $y_c[n]$  of the same length as that of the input  $x[n]$ . For wavelets, the cyclic convolution approach is superior to the linear convolution approach because the latter results in an output longer than the input. ◀

**Exercise 2-13:** Compute the 4-point DFT of  $\{4, 3, 2, 1\}$ .

**Answer:**  $\{10, (2 - j2), 2, (2 + j2)\}$ . (See 

## 2-8 Fast Fourier Transform (FFT)

► The fast Fourier transform (FFT) is a computational algorithm used to compute the discrete Fourier transforms (DFT) of discrete signals. Strictly speaking, the FFT is not a transform, but rather an algorithm for computing the transform. ◀

As was mentioned earlier, the fast Fourier transform (FFT) is a highly efficient algorithm for computing the DFT of discrete time signals. An  $N$ -point DFT performs a linear transformation from an  $N$ -long discrete-time vector, namely  $x[n]$ , into an  $N$ -long frequency domain vector  $\mathbf{X}[k]$  for  $k = 0, 1, \dots, N - 1$ . Computation of each  $\mathbf{X}[k]$  involves  $N$  complex multiplications, so the total number of multiplications required to perform the DFT for all  $\mathbf{X}[k]$  is  $N^2$ . This is in addition to  $N(N - 1)$  complex additions. For  $N = 512$ , for example, direct implementation of the DFT operation requires 262,144 multiplications and 261,632 complex additions. For small  $N$ , these numbers are smaller, since multiplication by any of  $\{1, -1, j, -j\}$  does not count as a true multiplication.

Contrast these large number of **multiplications and additions** (MADs) with the number required using the FFT algorithm: for  $N$  large, the number of complex multiplications is reduced from  $N^2$  to approximately  $(N/2)\log_2 N$ , which is only 2304 complex multiplications for  $N = 512$ . For complex additions, the number is reduced from  $N(N - 1)$  to  $N\log_2 N$ , or 4608 for  $N = 512$ . These reductions, thanks to the efficiency of the FFT algorithm, are on the order of 100 for multiplications and on the order of 50 for addition. The reduction ratios become increasingly more impressive at larger values of  $N$  (**Table 2-10**).

The computational efficiency of the FFT algorithm relies on a “divide and conquer” concept. An  $N$ -point DFT is **decomposed** (divided) into two  $(N/2)$ -point DFTs. Each of the  $(N/2)$ -point DFTs is decomposed further into two  $(N/4)$ -point DFTs. The decomposition process, which is continued until it reaches the 2-point DFT level, is illustrated in the next subsections.

### 2-8.1 2-Point DFT

For notational efficiency, we introduce the symbols

$$W_N = e^{-j2\pi/N}, \quad (2.111a)$$

$$W_N^{nk} = e^{-j2\pi nk/N}, \quad (2.111b)$$

and

$$W_N^{-nk} = e^{j2\pi nk/N}. \quad (2.111c)$$

Using this shorthand notation, the summations for the DFT, and its inverse given by Eq. (2.89), assume the form

$$\mathbf{X}[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \quad k = 0, 1, \dots, N - 1, \quad (2.112a)$$

and

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{X}[k] W_N^{-nk}, \quad n = 0, 1, \dots, N - 1. \quad (2.112b)$$

In this form, the  $N$ -long vector  $\mathbf{X}[k]$  is given in terms of the  $N$ -long vector  $x[n]$ , and vice versa, with  $W_N^{nk}$  and  $W_N^{-nk}$  acting as **weighting coefficients**.

For a 2-point DFT,

$$N = 2,$$

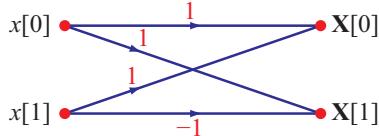
$$W_2^{0k} = e^{-j0} = 1,$$

and

$$W_2^{1k} = e^{-jk\pi} = (-1)^k.$$

**Table 2-10** Comparison of number of complex computations required by a standard DFT and an FFT using the formulas in the bottom row.

<i>N</i>	Multiplication		Additions	
	Standard DFT	FFT	Standard DFT	FFT
2	4	1	2	2
4	16	4	12	8
8	64	12	56	24
16	256	32	240	64
⋮	⋮	⋮	⋮	⋮
512	262,144	2,304	261,632	4,608
1,024	1,048,576	5,120	1,047,552	10,240
2,048	4,194,304	11,264	4,192,256	22,528
<i>N</i>	$N^2$	$\frac{N}{2} \log_2 N$	$N(N - 1)$	$N \log_2 N$



**Figure 2-16** Signal flow graph for a 2-point DFT.

Hence, Eq. (2.112a) yields the following expressions for  $\mathbf{X}[0]$  and  $\mathbf{X}[1]$ :

$$\mathbf{X}[0] = x[0] + x[1] \quad (2.113a)$$

and

$$\mathbf{X}[1] = x[0] - x[1], \quad (2.113b)$$

which can be combined into the compact form

$$\mathbf{X}[k] = x[0] + (-1)^k x[1], \quad k = 0, 1. \quad (2.114)$$

The equations for  $\mathbf{X}[0]$  and  $\mathbf{X}[1]$  can be represented by the *signal flow graph* shown in Fig. 2-16, which is often called a *butterfly diagram*.

## 2-8.2 4-Point DFT

For a 4-point DFT,  $N = 4$  and

$$W_N^{nk} = W_4^{nk} = e^{-jnk\pi/2} = (-j)^{nk}. \quad (2.115)$$

From Eq. (2.112a), we have

$$\begin{aligned} \mathbf{X}[k] &= \sum_{n=0}^3 x[n] W_4^{nk} \\ &= x[0] + x[1] W_4^{1k} + x[2] W_4^{2k} + x[3] W_4^{3k}, \\ &\quad k = 0, 1, 2, 3. \end{aligned} \quad (2.116)$$

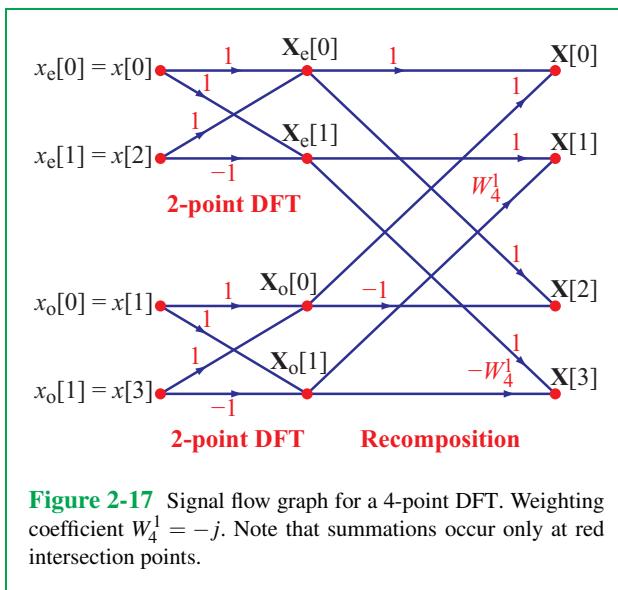
Upon evaluating  $W_4^{1k}$ ,  $W_4^{2k}$ , and  $W_4^{3k}$  and the relationships between them, Eq. (2.116) can be cast in the form

$$\mathbf{X}[k] = \underbrace{[x[0] + (-1)^k x[2]]}_{\text{2-point DFT}} + \underbrace{W_4^{1k} [x[1] + (-1)^k x[3]]}_{\text{2-point DFT}}, \quad (2.117)$$

which consists of two 2-point DFTs: one that includes values of  $x[n]$  for *even values of n*, and another for *odd values of n*. At this point, it is convenient to define  $x_e[n]$  and  $x_o[n]$  as  $x[n]$  at *even* and *odd* times:

$$x_e[n] = x[2n], \quad n = 0, 1, \quad (2.118a)$$

$$x_o[n] = x[2n + 1], \quad n = 0, 1. \quad (2.118b)$$



**Figure 2-17** Signal flow graph for a 4-point DFT. Weighting coefficient  $W_4^1 = -j$ . Note that summations occur only at red intersection points.

Thus,  $x_e[0] = x[0]$  and  $x_e[1] = x[2]$  and, similarly,  $x_o[0] = x[1]$  and  $x_o[1] = x[3]$ . When expressed in terms of  $x_e[n]$  and  $x_o[n]$ , Eq. (2.117) becomes

$$\begin{aligned} \mathbf{X}[k] &= [x_e[0] + (-1)^k x_e[1]] \\ &\quad + W_4^{1k} [x_o[0] + (-1)^k x_o[1]], \end{aligned} \quad (2.119)$$

2-point DFT of  $x_e[n]$   
2-point DFT of  $x_o[n]$

$k = 0, 1, 2, 3.$

The FFT computes the 4-point DFT by computing the two 2-point DFTs, followed by a recomposition step that involves multiplying the even 2-point DFT by  $W_4^{1k}$  and then adding it to the odd 2-point DFT. The entire process is depicted by the signal flow graph shown in **Fig. 2-17**. In the graph, Fourier coefficients  $\mathbf{X}_e[0]$  and  $\mathbf{X}_e[1]$  represent the outputs of the even 2-point DFT, and similarly,  $\mathbf{X}_o[0]$  and  $\mathbf{X}_o[1]$  represent the outputs of the odd 2-point DFT.

### 2-8.3 16-Point DFT

We now show how to compute a 16-point DFT using two 8-point DFTs and 8 **multiplications and additions (MADs)**. This *divides* the 16-point DFT into two 8-point DFTs, which in turn can be

*divided* into four 4-point DFTs, which are just additions and subtractions. This *conquers* the 16-point DFT by *dividing* it into 4-point DFTs and additional MADs.

#### A. Dividing a 16-Point DFT

We now show that the 16-point DFT can be computed for even values of  $k$  using an 8-point DFT of  $(x[n] + x[n+8])$  and for odd values of  $k$  using an 8-point DFT of the **modulated signal**  $(x[n] - x[n+8])e^{-j2\pi n/16}$ . Thus, the 16-point DFT can be computed as an 8-point DFT (for even values of  $k$ ) and as a modulated 8-point DFT (for odd values of  $k$ ).

#### B. Computation at Even Indices

We consider even and odd indices  $k$  separately.

For even values of  $k$ , we can write  $k = 2k'$  and split the 16-point DFT summation into two summations:

$$\begin{aligned} \mathbf{X}[2k'] &= \sum_{n=0}^{15} x[n] e^{-j2\pi(2k'/16)n} \\ &= \sum_{n=0}^7 x[n] e^{-j2\pi(2k'/16)n} + \sum_{n=8}^{15} x[n] e^{-j2\pi(2k'/16)n}. \end{aligned} \quad (2.120)$$

Changing variables from  $n$  to  $n' = n - 8$  in the second summation, and recognizing  $2k'/16 = k'/8$ , gives

$$\begin{aligned} \mathbf{X}[2k'] &= \sum_{n=0}^7 x[n] e^{-j2\pi(k'/8)n} + \sum_{n'=0}^7 x[n'+8] e^{-j2\pi(k'/8)(n'+8)} \\ &= \sum_{n=0}^7 (x[n] + x[n+8]) e^{-j2\pi(k'/8)n} \\ &= \text{DFT}(\{x[n] + x[n+8], n = 0, \dots, 7\}). \end{aligned} \quad (2.121)$$

So for even values of  $k$ , the 16-point DFT of  $x[n]$  is the 8-point DFT of  $\{x[n] + x[n+8], n = 0, \dots, 7\}$ .

### C. Computation at Odd Indices

For odd values of  $k$ , we can write  $k = 2k' + 1$  and split the 16-point DFT summation into two summations:

$$\begin{aligned} \mathbf{X}[2k'+1] &= \sum_{n=0}^{15} x[n] e^{-j2\pi(2k'+1)/16n} \\ &= \sum_{n=0}^7 x[n] e^{-j2\pi(2k'+1)/16n} \\ &\quad + \sum_{n=8}^{15} x[n] e^{-j2\pi(2k'+1)/16n}. \end{aligned} \quad (2.122)$$

Changing variables from  $n$  to  $n' = n - 8$  in the second summation, and recognizing that  $e^{-j2\pi 8/16} = -1$  and

$$\frac{2k'+1}{16} = \frac{k'}{8} + \frac{1}{16},$$

gives

$$\begin{aligned} \mathbf{X}[2k'+1] &= \sum_{n=0}^7 (x[n] e^{-j2\pi(1/16)n}) e^{-j2\pi(k'/8)n} \\ &\quad + \sum_{n'=0}^7 (x[n'+8] e^{-j2\pi(1/16)(n'+8)}) e^{-j2\pi(k'/8)(n'+8)} \\ &= \sum_{n=0}^7 e^{-j2\pi(1/16)n} (x[n] - x[n+8]) e^{-j2\pi(k'/8)n} \\ &= \text{DFT}\{e^{-j2\pi(1/16)n}(x[n] - x[n+8]), n = 0, \dots, 7\}. \end{aligned} \quad (2.123)$$

So for odd values of  $k$ , the 16-point DFT of  $x[n]$  is the 8-point DFT of  $\{e^{-j(2\pi/16)n}(x[n] - x[n+8]), n = 0, \dots, 7\}$ . The signal  $\{x[n] - x[n+8], n = 0, \dots, 7\}$  has been *modulated* through multiplication by  $e^{-j(2\pi/16)n}$ . The multiplications by  $e^{-j(2\pi/16)n}$  are known as **twiddle multiplications** (mults) by the **twiddle factors**  $e^{-j(2\pi/16)n}$ .

#### Example 2-8: Dividing an 8-Point DFT into Two 4-Point DFTs

Divide the 8-point DFT of  $\{7, 1, 4, 2, 8, 5, 3, 6\}$  into two 4-point DFTs and twiddle mults.

**Solution:**

- For even values of index  $k$ , we have

$$\begin{aligned} \mathbf{X}[0, 2, 4, 6] &= \text{DFT}\{7+8, 1+5, 4+3, 2+6\} \\ &= \{36, 8+j2, 8, 8-j2\}. \end{aligned}$$

For odd index values, we need twiddle mults. The twiddle factors are given by  $\{e^{-j2\pi n/8}\}$  for  $n = 0, 1, 2$ , and  $3$ , which reduce to  $\{1, \frac{\sqrt{2}}{2}(1-j), -j, \frac{\sqrt{2}}{2}(-1-j)\}$ .

- Implementing the twiddle mults gives

$$\begin{aligned} &\{7-8, 1-5, 4-3, 2-6\} \\ &\times \left\{1, \frac{\sqrt{2}}{2}(1-j), -j, \frac{\sqrt{2}}{2}(-1-j)\right\} \\ &= \{-1, 2\sqrt{2}(-1+j), -j, 2\sqrt{2}(1+j)\}. \end{aligned}$$

- For odd values of index  $k$ , we have

$$\begin{aligned} \mathbf{X}[1, 3, 5, 7] &= \text{DFT}\{-1+2\sqrt{2}(-1+j), -j, 2\sqrt{2}(1+j)\} \\ &= \{-1+j4.66, -1+j6.66, -1-j6.66, -1-j4.66\}. \end{aligned}$$

- Combining these results for even and odd  $k$  gives

$$\begin{aligned} \text{DFT}\{7, 1, 4, 2, 8, 5, 3, 6\} \\ &= \{36, -1+j4.7, 8+j2, -1+j6.7, 8, \\ &\quad -1-j6.7, 8-j2, -1-j4.7\}. \end{aligned}$$

Note the conjugate symmetry in the second and third lines:  $\mathbf{X}[7] = \mathbf{X}^*[1]$ ,  $\mathbf{X}[6] = \mathbf{X}^*[2]$ , and  $\mathbf{X}[5] = \mathbf{X}^*[3]$ .

- This result agrees with direct MATLAB computation using

`fft ([7 1 4 2 8 5 3 6]).`

### 2-8.4 Dividing Up a $2N$ -Point DFT

We now generalize the procedure to a  $2N$ -point DFT by dividing it into two  $N$ -point DFTs and  $N$  twiddle mults.

- (1) For even indices  $k = 2k'$  we have:

$$\begin{aligned} \mathbf{X}[2k'] &= \sum_{n=0}^{N-1} (x[n] + x[n+N]) e^{-j2\pi(k'/N)n} \\ &= \text{DFT}\{x[n] + x[n+N], n = 0, 1, \dots, N-1\}. \end{aligned} \quad (2.124)$$

(2) For odd indices  $k = 2k' + 1$  we have:

$$\begin{aligned}\mathbf{X}[2k'+1] &= \sum_{n=0}^{N-1} e^{-j2\pi(1/(2N))n} (x[n] - x[n+N]) e^{-j2\pi(k'/N)n} \\ &= \text{DFT}\{e^{-j2\pi(1/(2N))n} (x[n] - x[n+N])\}. \quad (2.125)\end{aligned}$$

Thus, a  $2N$ -point DFT can be divided into

- Two  $N$ -point DFTs,
- $N$  multiplications by twiddle factors  $e^{-j2\pi(1/(2N))n}$ , and
- $2N$  additions and subtractions.

## 2-8.5 Dividing and Conquering

Now suppose  $N$  is a power of two; e.g.,  $N = 1024 = 2^{10}$ . In that case, we can apply the algorithm of the previous subsection recursively to divide an  $N$ -point DFT into two  $N/2$ -point DFTs, then into four  $N/4$ -point DFTs, then into eight  $N/8$ -point DFTs, and so on until we reach the following 4-point DFTs:

$$\begin{aligned}\mathbf{X}[0] &= x[0] + x[1] + x[2] + x[3], \\ \mathbf{X}[1] &= x[0] - jx[1] - x[2] + jx[3], \\ \mathbf{X}[2] &= x[0] - x[1] + x[2] - x[3], \\ \mathbf{X}[3] &= x[0] + jx[1] - x[2] - jx[3]. \quad (2.126)\end{aligned}$$

At each stage, half of the DFTs are modulated, requiring  $N/2$  multiplications. So if  $N$  is a power of 2, then an  $N$ -point DFT computed using the FFT will require approximately  $(N/2)\log_2(N)$  multiplications and  $N\log_2(N)$  additions. These can be reduced slightly by recognizing that some multiplications are simply multiplications by  $\pm 1$  and  $\pm j$ .

To illustrate the computational significance of the FFT, suppose we wish to compute a 32768-point DFT. Direct computation using Eq. (2.89a) would require  $(32768)^2 \approx 1.1 \times 10^9$  MADs. In contrast, computation using the FFT would require less than  $\frac{32768}{2} \log_2(32768) \approx 250,000$  MADs, representing a computational saving of a factor of 4000!

## 2-9 Deconvolution Using the DFT

Recall that the objective of deconvolution is to reconstruct the input  $x[n]$  of a system from measurements of its output  $y[n]$  and knowledge of its impulse response  $h[n]$ . That is, we seek to solve  $y[n] = h[n] * x[n]$  for  $x[n]$ , given  $y[n]$  and  $h[n]$ .

### 2-9.1 Deconvolution Procedure

If  $x[n]$  has duration  $M$  and  $h[n]$  has duration  $L$ , then  $y[n]$  has duration  $N = L + M - 1$ . Let us define the **zero-padded functions**

$$\tilde{h}[n] = \{h[n], \underbrace{0, \dots, 0}_{N-L \text{ zeros}}\}, \quad (2.127a)$$

$$\tilde{x}[n] = \{x[n], \underbrace{0, \dots, 0}_{N-M \text{ zeros}}\}. \quad (2.127b)$$

(1) With  $\tilde{x}[n]$ ,  $\tilde{h}[n]$ , and  $y[n]$  all now of duration  $N$ , we can obtain their respective  $N$ -point DFTs,  $\tilde{\mathbf{X}}[k]$ ,  $\tilde{\mathbf{H}}[k]$ , and  $\mathbf{Y}[k]$ , which are interrelated by

$$\mathbf{Y}[k] = \tilde{\mathbf{H}}[k]\tilde{\mathbf{X}}[k]. \quad (2.128)$$

Upon dividing by  $\tilde{\mathbf{H}}[k]$  and taking an  $N$ -point inverse DFT, we have

$$\begin{aligned}\tilde{x}[n] &= \text{DFT}^{-1}\{\tilde{\mathbf{X}}[k]\} \\ &= \text{DFT}^{-1}\left\{\frac{\mathbf{Y}[k]}{\tilde{\mathbf{H}}[k]}\right\} \\ &= \text{DFT}^{-1}\left\{\frac{\text{DFT}\{y[n]\}}{\text{DFT}\{\tilde{h}[n]\}}\right\}. \quad (2.129)\end{aligned}$$

(2) Discarding the  $(N - M)$  final zeros in  $\tilde{x}[n]$  gives  $x[n]$ . The zero-padding and unpadding processes allow us to perform the deconvolution problem for any system, provided  $\tilde{\mathbf{H}}[k]$  is nonzero for all  $0 \leq k \leq N - 1$ .

### 2-9.2 FFT Implementation Issues

(a) To use the FFT algorithm (Section 2-8) to compute the three DFTs,  $N$  should be rounded up to the next power of 2 because the FFT can be computed more rapidly.

(b) In some cases, some of the values of  $\tilde{\mathbf{H}}[k]$  may be zero, which is problematic because the computation of  $\mathbf{Y}[k]/\tilde{\mathbf{H}}[k]$  would involve dividing by zero. A possible solution to the division-by-zero problem is to change the value of  $N$ . Suppose  $\tilde{\mathbf{H}}[k] = 0$  for some value of index  $k$ , such as  $k = 3$ . This corresponds to  $\mathbf{H}(\Omega)$  having a zero at  $2\pi k/N$  for  $k = 3$ , because by definition, the DFT is the DTFT sampled at  $\Omega = 2\pi k/N$  for integers  $k$ . Changing  $N$  to, say,  $N + 1$  (or some other suitable integer) means that the DFT is now the DTFT  $\mathbf{H}(\Omega)$  sampled at  $\Omega = 2\pi k/(N + 1)$ , so the zero at  $k = 3$  when the order was  $N$  may now get missed with the sampling at the new order  $N + 1$ . Changing  $N$  to  $N + 1$  may

avoid one or more zeros in  $\tilde{\mathbf{H}}[k]$ , but it may also introduce new ones. It may be necessary to try multiple values of  $N$  to satisfy the condition that  $\tilde{\mathbf{H}}[k] \neq 0$  for all  $k$ .

### Example 2-9: DFT Deconvolution

In response to an input  $x[n]$ , an LTI system with an impulse response  $h[n] = \{1, 2, 3\}$  generated an output

$$y[n] = \{6, 19, 32, 21\}.$$

Determine  $x[n]$ , given that it is of finite duration.

**Solution:** The output is of duration  $N = 4$ , so we should zero-pad  $h[n]$  to the same duration by defining

$$\tilde{h}[n] = \{1, 2, 3, 0\}. \quad (2.130)$$

From Eq. (2.89a), the 4-point DFT of  $\tilde{h}[n]$  is

$$\tilde{\mathbf{H}}[k] = \sum_{n=0}^3 \tilde{h}[n] e^{-j2\pi kn/4}, \quad k = 0, 1, 2, 3, \quad (2.131)$$

which yields

$$\begin{aligned}\tilde{\mathbf{H}}[0] &= 1(1) + 2(1) + 3(1) + 0(1) = 6, \\ \tilde{\mathbf{H}}[1] &= 1(1) + 2(-j) + 3(-1) + 0(j) = -2 - j2, \\ \tilde{\mathbf{H}}[2] &= 1(1) + 2(-1) + 3(1) + 0(-1) = 2,\end{aligned}$$

and

$$\tilde{\mathbf{H}}[3] = 1(1) + 2(j) + 3(-1) + 0(-j) = -2 + j2.$$

Similarly, the 4-point DFT of  $y[n] = \{6, 19, 32, 21\}$  is

$$\mathbf{Y}[k] = \sum_{n=0}^3 y[n] e^{-j2\pi kn/4}, \quad k = 0, 1, 2, 3,$$

which yields

$$\begin{aligned}\mathbf{Y}[0] &= 6(1) + 19(1) + 32(1) + 21(1) = 78, \\ \mathbf{Y}[1] &= 6(1) + 19(-j) + 32(-1) + 21(j) = -26 + j2, \\ \mathbf{Y}[2] &= 6(1) + 19(-1) + 32(1) + 21(-1) = -2,\end{aligned}$$

and

$$\mathbf{Y}[3] = 6(1) + 19(j) + 32(-1) + 21(-j) = -26 - j2.$$

The 4-point DFT of  $x[\tilde{n}]$  is, therefore,

$$\begin{aligned}\tilde{\mathbf{X}}[0] &= \frac{\mathbf{Y}[0]}{\tilde{\mathbf{H}}[0]} = \frac{78}{6} = 13, \\ \tilde{\mathbf{X}}[1] &= \frac{\mathbf{Y}[1]}{\tilde{\mathbf{H}}[1]} = \frac{-26 + j2}{-2 - j2} = 6 - j7, \\ \tilde{\mathbf{X}}[2] &= \frac{\mathbf{Y}[2]}{\tilde{\mathbf{H}}[2]} = \frac{-2}{2} = -1,\end{aligned}$$

and

$$\tilde{\mathbf{X}}[3] = \frac{\mathbf{Y}[3]}{\tilde{\mathbf{H}}[3]} = \frac{-26 - j2}{-2 + j2} = 6 + j7.$$

By Eq. (2.89b), the inverse DFT of  $\tilde{\mathbf{X}}[k]$  is

$$\tilde{x}[n] = \frac{1}{4} \sum_{k=0}^3 \tilde{\mathbf{X}}[k] e^{j2\pi kn/4}, \quad n = 0, 1, 2, 3, \quad (2.132)$$

which yields

$$\tilde{x}[n] = \{6, 7, 0, 0\}.$$

Given that  $y[n]$  is of duration  $N = 4$  and  $h[n]$  is of duration  $L = 3$ , it follows that  $x[n]$  must be of duration  $M = N - L + 1 = 4 - 3 + 1 = 2$ , if its duration is finite. Deletion of the zero-pads from  $\tilde{x}[n]$  leads to

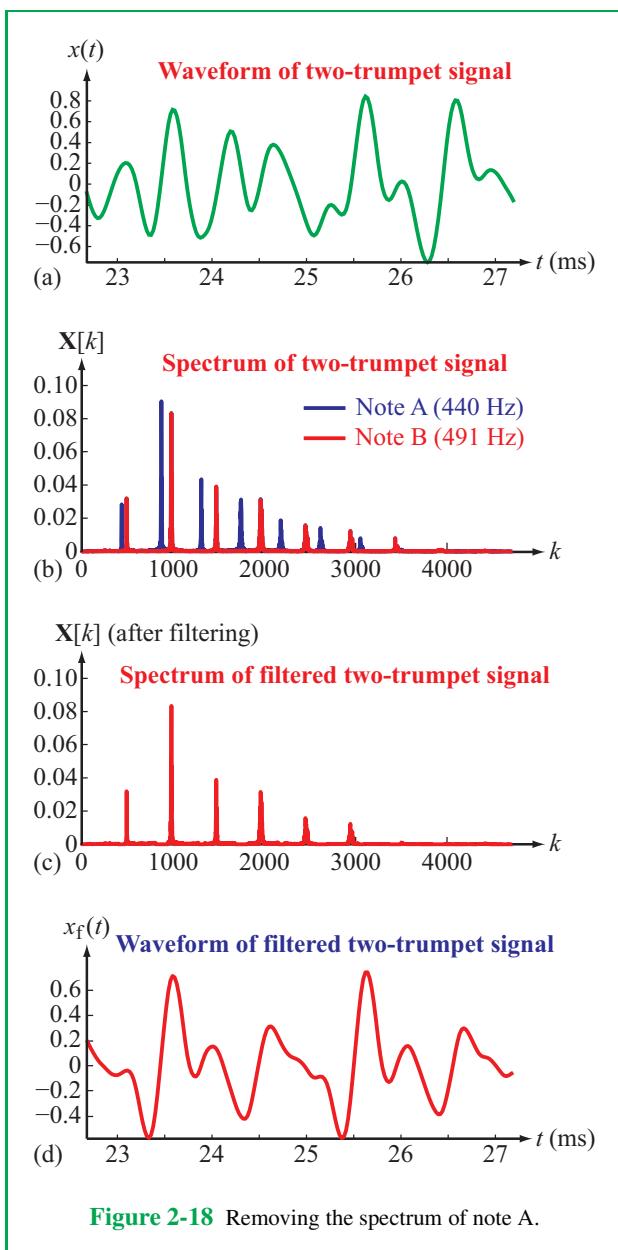
$$x[n] = \{6, 7\}, \quad (2.133)$$

whose duration is indeed 2.

### Example 2-10: Removal of Periodic Interference

We are given the signal of two actual trumpets playing simultaneously notes A and B. The goal is to use the DFT to eliminate the trumpet playing note A, while preserving the trumpet playing note B. We only need to know that note B is at a higher frequency than note A.

**Solution:** The two-trumpets signal time-waveform is shown in Fig. 2-18(a), and the corresponding spectrum is shown in Fig. 2-18(b). We note that the spectral lines occur in pairs of harmonics with the lower harmonic of each pair associated with



**Figure 2-18** Removing the spectrum of note A.

note A and the higher harmonic of each pair associated with note B.

Since we wish to eliminate note A, we set the lower compo-

nent of each pair of spectral lines to zero. The modified spectrum is shown in **Fig. 2-18(c)**. The inverse DFT of this spectrum, followed by reconstruction to continuous time, is shown in **Fig. 2-18(d)**.

The filtering process eliminated the signal due to the trumpet playing note A, while preserving the signal due to note B, almost completely. This can be confirmed by listening to the signals before and after filtering.

Whereas it is easy to distinguish between the harmonics of note A and those of note B at lower frequencies, this is not the case at higher frequencies, particularly when they overlap. Hence, neither note can be eliminated without affecting the other slightly. Fortunately, the overlapping high-frequency harmonics contain very little power compared with the non-overlapping, low-frequency harmonics, and therefore, their role is quite insignificant.

## 2-10 Computation of Continuous-Time Fourier Transform (CTFT) Using the DFT

Let us consider a continuous-time signal  $x(t)$  with support  $[-T/2, T/2]$ , which means that

$$x(t) = 0 \quad \text{for } |t| > \frac{T}{2}.$$

Also, let us assume (for the time being) that the signal spectrum  $\mathbf{X}(f)$  is bandwidth-limited to a maximum frequency  $F/2$  in Hz, which means that

$$\mathbf{X}(f) = 0 \quad \text{for } |f| > \frac{F}{2}.$$

Our goal is to compute samples  $\{\mathbf{X}(k\Delta_f)\}$  of  $\mathbf{X}(f)$  at a frequency spacing  $\Delta_f$  from signal samples  $\{x(n\Delta_t)\}$  of  $x(t)$  recorded at time interval  $\Delta_t$ , and to do so using the DFT.

### 2-10.1 Derivation Using Sampling Theorem Twice

Whereas it is impossible for a signal to be simultaneously bandlimited and time-limited, as presumed earlier, many real-world signals are approximately band- and time-limited.

According to the sampling theorem (Section 2-4),  $x(t)$  can be reconstructed from its samples  $\{x(n\Delta_t)\}$  if the sampling rate  $S_t = 1/\Delta_t > 2\frac{F}{2} = F$ . Applying the sampling theorem with  $t$  and  $f$  exchanged,  $\mathbf{X}(f)$  can be reconstructed from its samples

$\{\mathbf{X}(k\Delta_f)\}$  if its sampling rate  $S_f = 1/\Delta_f > 2\frac{T}{2} = T$ .

In the sequel, we use the minimum sampling intervals  $\Delta_t = 1/F$  and  $\Delta_f = 1/T$ . Finer discretization can be achieved by simply increasing  $F$  and/or  $T$ . In practice,  $F$  and/or  $T$  is (are) increased slightly so that  $N = FT$  is an odd integer, which makes the factor  $M = (N - 1)/2$  also an integer (but not necessarily an odd integer). The factor  $M$  is related to the order of the DFT, which has to be an integer.

The Fourier transform of the synthetic sampled signal  $x_s(t)$ , defined in Eq. (2.43) and repeated here as

$$x_s(t) = \sum_{n=-\infty}^{\infty} x(n\Delta_t) \delta(t - n\Delta_t), \quad (2.134)$$

was computed in Eq. (2.53), and also repeated here as

$$\mathbf{X}_s(f) = \sum_{n=-\infty}^{\infty} x(n\Delta_t) e^{-j2\pi f n\Delta_t}. \quad (2.135)$$

Setting  $f = k\Delta_f$  gives

$$\mathbf{X}_s(k\Delta_f) = \sum_{n=-\infty}^{\infty} x(n\Delta_t) e^{-j2\pi(k\Delta_f)(n\Delta_t)}. \quad (2.136)$$

Noting that  $x(t) = 0$  for  $|t| > \frac{T}{2}$  and  $\mathbf{X}(f) = 0$  for  $|f| > \frac{F}{2}$ , we restrict the ranges of  $n$  and  $k$  to

$$|n| \leq \frac{T/2}{\Delta_t} = \frac{FT}{2} = \frac{N}{2} \quad (2.137a)$$

and

$$|k| \leq \frac{F/2}{\Delta_f} = \frac{FT}{2} = \frac{N}{2}. \quad (2.137b)$$

Next, we introduce factor  $M$  defined as

$$M = \frac{N-1}{2}, \quad (2.138)$$

and we note that if  $N$  is an odd integer,  $M$  is guaranteed to be an integer. In view of Eq. (2.137), the ranges of  $n$  and  $k$  become  $n, k = -M, \dots, M$ . Upon substituting

$$\Delta_t \Delta_f = \frac{1}{F} \frac{1}{T} = \frac{1}{FT} = \frac{1}{N} = \frac{1}{2M+1}. \quad (2.139)$$

In the exponent of Eq. (2.136), the expression becomes

$$\begin{aligned} \mathbf{X}_s(k\Delta_f) &= \sum_{n=-M}^M x(n\Delta_t) e^{-j2\pi(k\Delta_f)(n\Delta_t)}, \quad |k| \leq M \\ &= \sum_{n=-M}^M x(n\Delta_t) e^{-j2\pi nk/(2M+1)}, \quad |k| \leq M. \end{aligned} \quad (2.140)$$

This expression looks like a DFT of order  $2M+1$ . Recall from the statement in connection with Eq. (2.47) that the spectrum  $\mathbf{X}_s(f)$  of the sampled signal includes the spectrum  $\mathbf{X}(f)$  of the continuous-time signal (multiplied by the sampling rate  $S_t$ ) plus additional copies repeated every  $\pm S_t$  along the frequency axis. With  $S_t = 1/\Delta_t = F$  in the present case,

$$\mathbf{X}_s(f) = F\mathbf{X}(f), \quad \text{for } |f| < F, \quad (2.141)$$

from which we deduce that

$$\mathbf{X}(k\Delta_f) = \mathbf{X}_s(f) \Delta_t = \sum_{n=-M}^M x(n\Delta_t) e^{-j2\pi nk/(2M+1)} \Delta_t, \quad |k| \leq M. \quad (2.142)$$

Ironically, this is the same result that would be obtained by simply discretizing the definition of the continuous-time Fourier transform! But this derivation shows that discretization gives the exact result if  $x(t)$  is time- and bandlimited.

### Example 2-11: Computing CTFT by DFT

Use the DFT to compute the Fourier transform of the continuous Gaussian signal

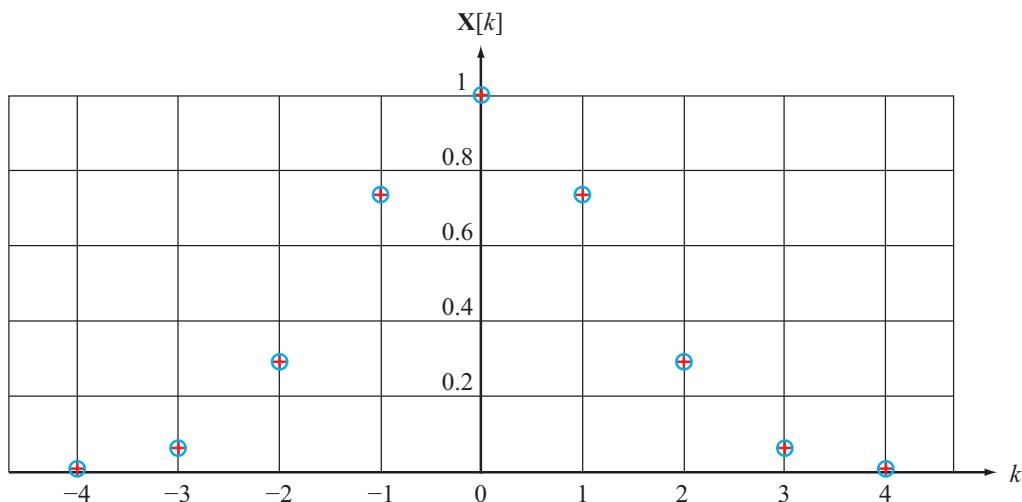
$$x(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}.$$

**Solution:** Our first task is to assign realistic values for the signal duration  $T$  and the width of its spectrum  $F$ . It is an “educated” trial-and-error process. At  $t = 4$ ,  $x(t) = 0.00013$ , so we will assume that  $x(t) \approx 0$  for  $|t| > 4$ . Since  $x(t)$  is symmetrical with respect to the vertical axis, we assign

$$T = 2 \times 4 = 8 \text{ s.}$$

The Fourier transform of  $x(t)$  is  $\mathbf{X}(f) = e^{-2\pi^2 f^2}$ . By trial and error, we determine that  $F = 1.2 \text{ Hz}$  is sufficient to characterize  $\mathbf{X}(f)$ . The combination gives

$$N = TF = 8 \times 1.2 = 9.6.$$



**Figure 2-19** Comparison of exact (blue circles) and DFT-computed (red crosses) of the continuous-time Fourier transform of a Gaussian signal.

To increase the value of  $N$  to an odd integer, we increase  $F$  to 1.375 Hz, which results in  $N = 11$  and  $M = 5$ . In Fig. 2-19 computed values of the discretized spectrum of  $x(t)$  are compared with exact values based on evaluating the analytical expression for  $\mathbf{X}(f) = e^{-2\pi^2 f^2}$ . The comparison provides an excellent demonstration of the power of the sampling theorem; representing  $x(t)$  by only 11 equally spaced samples is sufficient to capture its information content and generate its Fourier transform with high fidelity.

**Concept Question 2-13:** The DFT is often used to compute the CTFT numerically. Why does this often work as well as it does?

## 2-10.2 Practical Computation of $\mathbf{X}(f)$ Using the DFT

Example 2-11 shows that simple discretization of the continuous-time Fourier transform works very well, provided that the discretization lengths in time and frequency are chosen properly. The integer  $N$  was odd for clarity of derivation. In practice, we would increase  $T$  and  $F$  so that  $N = TF$  is a power of two, permitting the fast Fourier transform (FFT) algorithm to be used to compute the DFT quickly.

**Concept Question 2-12:** Why do we need a discrete Fourier transform (DFT)?

## Summary

### Concepts

- Many 2-D concepts can be understood more easily by reviewing their 1-D counterparts. These include: LTI systems, convolution, sampling, continuous-time; discrete-time; and discrete Fourier transforms.
- The DTFT is periodic with period  $2\pi$ .
- Continuous-time signals can be sampled to discrete-time signals, on which discrete-time signal processing can be performed.
- The response of an LTI system with impulse response

$h(t)$  to input  $x(t)$  is output  $y(t) = h(t) * x(t)$ , and similarly in discrete time.

- The response of an LTI system with impulse response  $h(t)$  to input  $A \cos(2\pi f_0 t + \theta)$  is

$$A|\mathbf{H}(f)| \cos(2\pi f_0 t + \theta + \underline{\mathbf{H}(f)}),$$

where  $\mathbf{H}(f)$  is the Fourier transform of  $h(t)$ , and similarly in discrete time.

### Mathematical Formulae

#### Impulse

$$\delta(x) = \lim_{\epsilon \rightarrow 0} \frac{1}{2\epsilon} \operatorname{rect}\left(\frac{t}{2\epsilon}\right)$$

#### Energy of $x(t)$

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt$$

#### Convolution

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(\tau)x(t - \tau) d\tau$$

#### Convolution

$$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} h[i] x[n-i]$$

#### Fourier transform

$$\mathbf{X}(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

#### Inverse Fourier transform

$$x(t) = \int_{-\infty}^{\infty} \mathbf{X}(f) e^{j2\pi ft} df$$

#### Sinc function

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

#### Ideal lowpass filter impulse response

$$h(t) = 2f_c \operatorname{sinc}(2f_c t)$$

#### Sampling theorem

$$\text{Sampling rate } S = \frac{1}{\Delta} > 2B \text{ if } \mathbf{X}(f) = 0 \text{ for } |f| > B$$

#### Sinc interpolation formula

$$x(t) = \sum_{n=-\infty}^{\infty} x(n\Delta) \operatorname{sinc}(S(t - n\Delta))$$

#### Discrete-time Fourier transform (DTFT)

$$\mathbf{X}(\Omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\Omega n}$$

#### Inverse DTFT

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathbf{X}(\Omega) e^{j\Omega n} d\Omega$$

#### Discrete-time sinc

$$h[n] = \frac{\Omega_0}{\pi} \operatorname{sinc}\left(\frac{\Omega_0 n}{\pi}\right)$$

#### Discrete sinc

$$\mathbf{X}(\Omega) = \frac{\sin((2N+1)\Omega/2)}{\sin(\Omega/2)}$$

#### Discrete Fourier Transform (DFT)

$$\mathbf{X}[k] = \sum_{n=0}^{M-1} x[n] e^{-j2\pi nk/M}$$

#### Inverse DFT

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{X}[k] e^{j2\pi nk/N}$$

#### Cyclic convolution

$$y_c[n] = x_1[n] \odot x_2[n] = \sum_{n_1=0}^{N-1} x_1[n_1] x_2[(n - n_1)_N]$$

**Important Terms**

Provide definitions or explain the meaning of the following terms:

aliasing	FFT
cyclic convolution	Fourier transform
deconvolution	frequency response function
DFT	impulse response convolution
DTFT	linear time-invariant (LTI)

Parseval's theorem
Rayleigh's theorem
sampled signal
sampling theorem
sinc function

spectrum
zero padding

**PROBLEMS****Section 2-2: Review of Continuous-Time Systems****2.1** Compute the following convolutions:

- (a)  $e^{-t} u(t) * e^{-2t} u(t)$
- (b)  $e^{-2t} u(t) * e^{-3t} u(t)$
- (c)  $e^{-3t} u(t) * e^{-3t} u(t)$

**Section 2-3: 1-D Fourier Transforms****2.2** Show that the spectrum of

$$\frac{\sin(20\pi t)}{\pi t} \frac{\sin(10\pi t)}{\pi t}$$

is zero for  $|f| > 15$  Hz.**2.3** Using only Fourier transform properties, show that

$$\frac{\sin(10\pi t)}{\pi t} [1 + 2\cos(20\pi t)] = \frac{\sin(30\pi t)}{\pi t}.$$

**2.4** If  $x(t) = \sin(2t)/(\pi t)$ , compute the energy of  $d^2x/dt^2$ .**2.5** Compute the energy of  $e^{-t} u(t) * \sin(t)/(\pi t)$ .**2.6** Show that

$$\int_{-\infty}^{\infty} \frac{\sin^2(at)}{(\pi t)^2} dt = \frac{a}{\pi}$$

if  $a > 0$ .**Section 2-4: The Sampling Theorem****2.7** The spectrum of the trumpet signal for note G (784 Hz) is negligible above its ninth harmonic. What is the Nyquist sampling rate required for reconstructing the trumpet signal from its samples?**2.8** Compute a Nyquist sampling rate for reconstructing signal

$$x(t) = \frac{\sin(40\pi t) \sin(60\pi t)}{\pi^2 t^2}$$

from its samples.

**2.9** Signal

$$x(t) = \frac{\sin(2\pi t)}{\pi t} [1 + 2\cos(4\pi t)]$$

is sampled every 1/6 second. What is the spectrum of the sampled signal?

**2.10** Signal  $x(t) = \cos(14\pi t) - \cos(18\pi t)$  is sampled at 16 sample/s. The result is passed through an ideal brick-wall lowpass filter with a cutoff frequency of 8 Hz. What is the spectrum of the output signal?**2.11** Signal  $x(t) = \sin(30\pi t) + \sin(70\pi t)$  is sampled at 50 sample/s. The result is passed through an ideal brick-wall lowpass filter with a cutoff frequency of 25 Hz. What is the spectrum of the output signal?**Section 2-5: Review of Discrete-Time Signals and Systems****2.12** Compute the following convolutions:

- (a)  $\{\underline{1}, 2\} * \{\underline{3}, 4, 5\}$
- (b)  $\{\underline{1}, 2, 3\} * \{\underline{4}, 5, 6\}$
- (c)  $\{\underline{2}, 1, 4\} * \{\underline{3}, 6, 5\}$

**2.13** If  $\{\underline{1}, 2, 3\} * x[n] = \{\underline{5}, 16, 34, 32, 21\}$ , compute  $x[n]$ .**2.14** Given the two systems connected in series as

$$x[n] \rightarrow \boxed{h_1[n]} \rightarrow w[n] = 3x[n] - 2x[n-1],$$

and

$$w[n] \rightarrow \boxed{h_2[n]} \rightarrow y[n] = 5w[n] - 4w[n-1],$$

compute the overall impulse response.

**2.15** The two systems

$$y[n] = 3x[n] - 2x[n-1]$$

and

$$y[n] = 5x[n] - 4x[n-1]$$

are connected in parallel. Compute the overall impulse response.

## Section 2-6: Discrete-Time Frequency Response

**2.16** Given

$$\cos\left(\frac{\pi}{2}n\right) \rightarrow \boxed{y[n] = x[n] + 0.5x[n-1] + x[n-2]} \rightarrow y[n],$$

- (a) Compute the frequency response  $\mathbf{H}(\Omega)$ .
- (b) Compute the output  $y[n]$ .

**2.17** Given

$$\cos\left(\frac{\pi}{2}n\right) \rightarrow \boxed{y[n] = 8x[n] + 3x[n-1] + 4x[n-2]} \rightarrow y[n],$$

- (a) Compute the frequency response  $\mathbf{H}(\Omega)$ .
- (b) Compute the output  $y[n]$ .

**2.18** If input  $x[n] = \cos\left(\frac{\pi}{2}n\right) + \cos(\pi n)$ , and

$$x[n] \rightarrow \boxed{y[n] = x[n] + x[n-1] + x[n-2] + x[n-3]} \rightarrow y[n],$$

- (a) Compute the frequency response  $\mathbf{H}(\Omega)$ .
- (b) Compute the output  $y[n]$ .

**2.19** If input  $x[n] = 1 + 2\cos\left(\frac{\pi}{2}n\right) + 3\cos(\pi n)$ , and

$$x[n] \rightarrow \boxed{y[n] = x[n] + 4x[n-1] + 3x[n-3]} \rightarrow y[n],$$

- (a) Compute the frequency response  $\mathbf{H}(\Omega)$ .
- (b) Compute the output  $y[n]$ .

## Section 2-6: Discrete-Time Fourier Transform (DTFT)

**2.20** Compute the DTFTs of the following signals (simplify answers to sums of sines and cosines).

- (a)  $\{1, 1, \underline{1}, 1, 1\}$
- (b)  $\{3, \underline{2}, 1\}$

**2.21** Compute the inverse DTFT of

$$\mathbf{X}(\Omega) = [3 + 2\cos(\Omega) + 4\cos(2\Omega)] + j[6\sin(\Omega) + 8\sin(2\Omega)].$$

**2.22** Compute the inverse DTFT of

$$\mathbf{X}(\Omega) = [7 + 5\cos(\Omega) + 3\cos(2\Omega)] + j[\sin(\Omega) + \sin(2\Omega)].$$

## Section 2-7: Discrete Fourier Transform (DFT)

**2.23** Compute the DFTs of each of the following signals:

- (a)  $\{\underline{12}, 8, 4, 8\}$
- (b)  $\{\underline{16}, 8, 12, 4\}$

**2.24** Determine the DFT of a single period of each of the following signals:

- (a)  $\cos\left(\frac{\pi}{4}n\right)$
- (b)  $\frac{1}{4}\sin\left(\frac{3\pi}{4}n\right)$

**2.25** Compute the inverse DFTs of the following:

- (a)  $\{0, 0, 3, 0, 4, 0, 3, 0\}$
- (b)  $\{0, 3 + j4, 0, 0, 0, 0, 0, 3 - j4\}$

## Section 2-9: Deconvolution Using the DFT

**2.26** Use DFTs to compute the convolution

$$\{\underline{1}, 3, 5\} * \{\underline{7}, 9\}$$

by hand.

**2.27** Solve each of the following deconvolution problems for input  $x[n]$ . Use MATLAB.

- (a)  $x[n] * \{\underline{1}, 2, 3\} = \{\underline{7}, 15, 27, 13, 24, 27, 34, 15\}$ .
- (b)  $x[n] * \{\underline{1}, 3, 5\} = \{\underline{3}, 10, 22, 18, 28, 29, 52, 45\}$ .
- (c)  $x[n] * \{\underline{1}, 4, 2, 6, 5, 3\} = \{\underline{2}, 9, 11, 31, 48, 67, 76, 78, 69, 38, 12\}$ .

**2.28** Solve each of the following deconvolution problems for input  $x[n]$ . Use MATLAB.

- (a)  $x[n] * \{3, 1, 4, 2\} = \{\underline{6}, 23, 18, 57, 35, 37, 28, 6\}$ .  
 (b)  $x[n] * \{\underline{1}, 7, 3, 2\} = \{\underline{2}, 20, 53, 60, 53, 54, 21, 10\}$ .  
 (c)  $x[n] * \{2, 2, 3, 6\} = \{\underline{12}, 30, 42, 71, 73, 43, 32, 45, 42\}$ .

### Section 2-10: Computation of CTFT Using the DFT

**2.29** Use a 40-point DFT to compute the inverse Fourier transform of

$$\mathbf{X}(f) = \left( \frac{\sin(\pi f)}{\pi f} \right)^2.$$

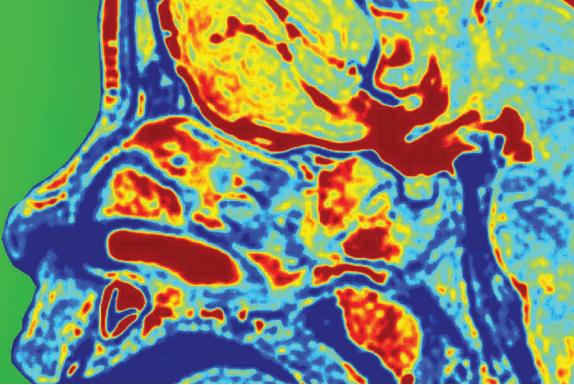
Assume that  $\mathbf{X}(f) \approx 0$  for  $|f| > 10$  Hz and  $x(t) \approx 0$  for  $|t| > 1$  s. Plot the actual and computed inverse Fourier transforms on the same plot to show the close agreement between them.

**2.30** Use an 80-point DFT to compute the inverse Fourier transform of

$$\mathbf{X}(f) \mathbf{H}(f) = \left( \frac{\sin(\pi f)}{\pi f} \right)^2 (1 + e^{-2j\pi f}).$$

Assume that  $\mathbf{X}(f) \approx 0$  for  $|f| > 10$  Hz and  $x(t) \approx 0$  for  $|t| > 2$  s. Plot the actual and computed inverse Fourier transforms on the same plot to show the close agreement between them.

# CHAPTER 3



## 3

## 2-D Images and Systems

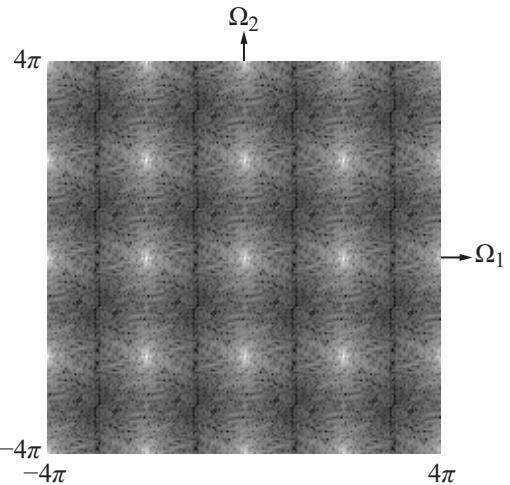
### Contents

- Overview, 90
- 3-1** Displaying Images, 90
- 3-2** 2-D Continuous-Space Images, 91
- 3-3** Continuous-Space Systems, 93
- 3-4** 2-D Continuous-Space Fourier Transform (CSFT), 94
- 3-5** 2-D Sampling Theorem, 107
- 3-6** 2-D Discrete Space, 113
- 3-7** 2-D Discrete-Space Fourier Transform (DSFT), 118
- 3-8** 2-D Discrete Fourier Transform (2-D DFT), 119
- 3-9** Computation of the 2-D DFT Using MATLAB, 126
- Problems, 86

### Objectives

Learn to:

- Compute the output image from an LSI system to a given input image using convolution.
- Compute the continuous-space Fourier transform of an image or point-spread function.
- Use the 2-D sampling theorem to convert a continuous-space image to a discrete-space image.
- Perform the two tasks listed above for continuous-space images on discrete-space images.



This chapter presents the 2-D versions, suitable for image processing, of the 1-D concepts presented in Chapter 2. These include: linear shift-invariant (LSI) systems, 2-D convolution, spatial frequency response, filtering, Fourier transforms for continuous and discrete-space images, and the 2-D sampling theorem. It also covers concepts that do not arise in 1-D, such as separability, image scaling and rotation, and representation of discrete-space images using various coordinate systems.

## Overview

A 2-D image is a signal that varies as a function of the two spatial dimensions,  $x$  and  $y$ , instead of time  $t$ . A 3-D image, such as a CT scan (Fig. 1-24), is a signal that varies as a function of  $(x, y, z)$ .

In 1-D, it is common practice to assign the symbol  $x(t)$  to represent the signal at the input to a system, and to assign  $y(t)$  to the output signal [or  $x[n]$  and  $y[n]$  if the signals are discrete]. Because  $x$ ,  $y$ , and  $z$  are the standard symbols of the Cartesian coordinate system, a different symbolic representation is used with 2-D images:

- ▶ Image intensity is represented by  $f(x, y)$ , where  $(x, y)$  are two orthogonal spatial dimensions. ◀

This chapter extends the 1-D definitions, properties, and transformations covered in the previous chapter into their 2-D equivalents. It also presents certain 2-D properties that have no counterparts in 1-D.

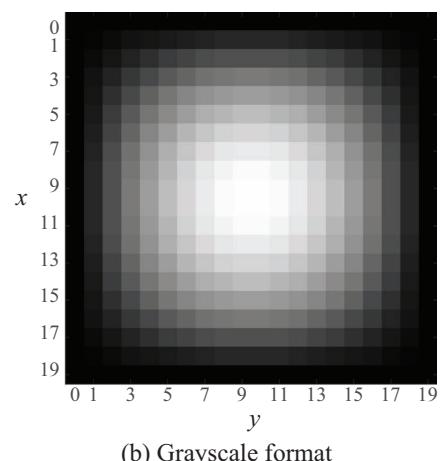
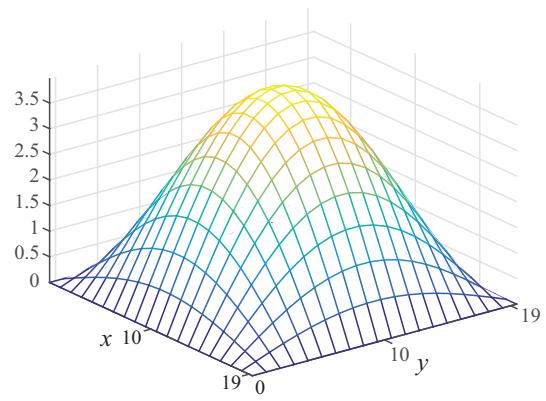
### 3-1 Displaying Images

In 1-D, a continuous-time signal  $x(t)$  is displayed by plotting  $x(t)$  versus  $t$ . A discrete-time signal  $x[n]$  is displayed using a stem plot of  $x[n]$  versus  $n$ . Clearly such plots are not applicable for 2-D images.

Image intensity  $f(x, y)$  of a 2-D image can be displayed either as a 3-D **mesh plot** (Fig. 3-1(a)), which hides some features of the image and is difficult to create and interpret, or as a **grayscale image** (Fig. 3-1(b)). In a grayscale image, the image intensity is scaled so that the minimum value of  $f(x, y)$  is depicted in black and the maximum value of  $f(x, y)$  is depicted in white. If the image is non-negative ( $f(x, y) \geq 0$ ), as is often the case, black in the grayscale image denotes zero values of  $f(x, y)$ . If the image is not non-negative, zero values of  $f(x, y)$  appear as a shade of gray.

MATLAB's `imagesc(X)`, `colormap(gray)` displays the 2-D array  $X$  as a grayscale image in which black depicts the minimum value of  $X$  and white depicts the maximum value of  $X$ .

It is also possible to display an image  $f(x, y)$  as a **false-color image**, in which case different colors denote different values of  $f(x, y)$ . The relation between color and values of  $f(x, y)$  is denoted using a colorbar, to the side or bottom of the image. An example of a false-color display was shown earlier in Fig. 1-11,



**Figure 3-1** An image displayed in (a) mesh plot format and (b) grayscale format.

depicting the infrared intensity emitted by a hot air balloon. We should not confuse a false-color image with a true-color image. Whereas a false-color image is a single grayscale image (1 channel) displayed in color, a true-color image actually is a set of three images (3 channels):

$$\{f_{\text{red}}(x, y), f_{\text{green}}(x, y), f_{\text{blue}}(x, y)\},$$

representing (here) the three primary colors: red, green and blue. Other triplets of colors, such as yellow, cyan and magenta, can also be used. Hence, image processing of color images

encompasses 3-channel image processing (see Chapter 10).

A grayscale image can be regarded as a still of a black-and-white TV image, while a color image can be regarded as a still of a color TV image. Before flat-panel displays were invented, color images on TVs and monitors were created from three separate signals using three different electron guns in picture tubes, while black-and-white TV images were created using a single electron gun in a picture tube. Modern solid-state color display, such as *liquid crystal displays (LCDs)*, are composed of three interleaved displays (similar to the APS display in **Fig. 1-4**), each driven by one of the three signal channels.

**Concept Question 3-1:** What is the difference between a true-color image and a false-color image?

**Exercise 3-1:** How can you tell whether an image displayed in color is a color image or a false-color image?

**Answer:** False-color images should have colorbars to identify the numerical value associated with each color.

## B. Box Image

The *box image*  $f_{\text{Box}}(x, y)$  is the 2-D pulse-equivalent of the rectangular function  $\text{rect}(t)$  and is defined as

$$f_{\text{Box}}(x, y) = \text{rect}(x) \text{ rect}(y) \\ = \begin{cases} 1 & \text{for } |x| < 1/2 \text{ and } |y| < 1/2, \\ 0 & \text{otherwise.} \end{cases}$$

By extension, a box image of widths  $(\ell_x, \ell_y)$  and centered at  $(x_0, y_0)$  is defined as

$$f_{\text{Box}}\left(\frac{x - x_0}{\ell_x}, \frac{y - y_0}{\ell_y}\right) = \text{rect}\left(\frac{x - x_0}{\ell_x}\right) \text{rect}\left(\frac{y - y_0}{\ell_y}\right) \\ = \begin{cases} 1 & \text{for } |x - x_0| < \ell_x/2 \text{ & } |y - y_0| < \ell_y/2, \\ 0 & \text{otherwise,} \end{cases} \quad (3.2)$$

and shown in **Fig. 3-2(a)**.

## 3-2 2-D Continuous-Space Images

A continuous-space image is a physical quantity, such as temperature or pressure, that varies with spatial position in 2-D. Mathematically a continuous-space image is a function  $f(x, y)$  of spatial position  $(x, y)$ , where  $x$  and  $y$  have units of length (meters).

### 3-2.1 Fundamental 2-D Images

#### A. Impulse

A *2-D impulse*  $\delta(x, y)$  is simply

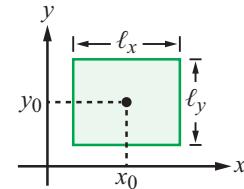
$$\delta(x, y) = \delta(x) \delta(y),$$

and a 2-D impulse shifted by  $\xi$  along  $x$  and by  $\eta$  along  $y$  is

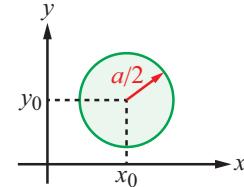
$$\delta(x - \xi, y - \eta) = \delta(x - \xi) \delta(y - \eta).$$

The *sifting property* generalizes directly from 1-D to 2-D. In 2-D

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi, \eta) \delta(x - \xi, y - \eta) d\xi d\eta = f(x, y). \quad (3.1)$$



$$(a) f_{\text{Box}}\left(\frac{x - x_0}{\ell_x}, \frac{y - y_0}{\ell_y}\right) = \text{rect}\left(\frac{x - x_0}{\ell_x}\right) \text{rect}\left(\frac{y - y_0}{\ell_y}\right)$$



$$(b) f_{\text{Disk}}\left(\frac{x - x_0}{a}, \frac{y - y_0}{a}\right)$$

**Figure 3-2** (a) Box image of widths  $(\ell_x, \ell_y)$  and centered at  $(x_0, y_0)$  and (b) disk image of radius  $a/2$  and centered at  $(x_0, y_0)$ .

### C. Disk Image

Being rectangular in shape, the box-image function is suitable for applications involving Cartesian coordinates, such as shifting the box sideways or up and down across the image. Some applications, however, require the use of polar coordinates, in which case the *disk-image* function is more suitable. The disk image  $f_{\text{Disk}}(x, y)$  of radius  $1/2$  is defined as

$$f_{\text{Disk}}(x, y) = \begin{cases} 1 & \text{for } \sqrt{x^2 + y^2} < 1/2, \\ 0 & \text{for } \sqrt{x^2 + y^2} > 1/2. \end{cases} \quad (3.3)$$

The expression given by Eq. (3.3) pertains to a circular disk centered at the origin and of radius  $1/2$ . For the more general case of a circular disk of radius  $a/2$  and centered at  $(x_0, y_0)$ ,

$$f_{\text{Disk}}\left(\frac{x - x_0}{a}, \frac{y - y_0}{a}\right) = \begin{cases} 1 & \text{for } \sqrt{(x - x_0)^2 + (y - y_0)^2} < a/2, \\ 0 & \text{for } \sqrt{(x - x_0)^2 + (y - y_0)^2} > a/2. \end{cases} \quad (3.4)$$

An example is displayed in [Fig. 3-2\(b\)](#).

### 3-2.2 Properties of Images

#### A. Generalizations of 1-D Properties

##### 1. Spatial shift

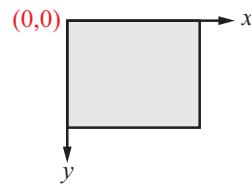
When shifted spatially by  $(x_0, y_0)$ , image  $f(x, y)$  becomes  $f(x - x_0, y - y_0)$ . Assuming the  $x$  axis is pointing to the right, image  $f(x - x_0, y - y_0)$  is shifted to the right by  $x_0$ , relative to  $f(x, y)$ , if  $x_0$  is positive, and to the left by the same amount if  $x_0$  is negative.

Whereas it is customary to define the direction of the  $x$  axis as pointing to the right, there is less uniformity with regard to the definition of the  $y$  axis; sometimes the  $y$  axis is defined along the upward direction, and in other cases it is defined to be along the downward direction. Hence, if  $y_0$  is positive,  $f(x - x_0, y - y_0)$  is shifted by  $y_0$  either upwards or downwards, depending on the direction of the  $y$  axis.

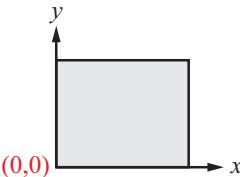
As to the location of the origin  $(x, y) = (0, 0)$ , it is usually defined to be at any one of the following three locations: upper left corner, lower left corner, or the center of the image ([Fig. 3-3](#)).

In this book:

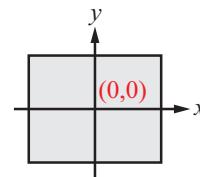
- **Images**  $f(x, y)$  are usually displayed with the origin  $(0, 0)$  at the upper-left corner, as in [Fig. 3-3\(a\)](#).



(a) Origin at top left and  $y$  axis downward



(b) Origin at bottom left and  $y$  axis upward



(c) Origin at center of image

**Figure 3-3** Three commonly used image coordinate systems.

- **Point-spread functions (PSFs)**—introduced in Section 3-3.2—are usually displayed with the origin at the center, as in [Fig. 3-3\(c\)](#).

- **Image spectra**, defined in Section 3-4, are usually displayed as in [Fig. 3-3\(c\)](#).

##### 2. Spatial scaling

When spatially scaled by  $(a_x, a_y)$ , image  $f(x, y)$  becomes  $f(a_x x, a_y y)$ . If  $a_x > 1$ , the image is shrunk in the  $x$  direction by a factor  $a_x$ , and if  $0 < a_x < 1$ , the image is magnified in size by  $1/a_x$ . So  $a_x$  represents a shrinkage factor. If  $a_x < 0$ , the image is reversed in the  $x$  direction, in addition to being shrunk by a factor of  $|a_x|$ . The same comments apply to  $a_y$ .

### 3. Image energy

Extending the expression for signal energy given by Eq. (3.7) from 1-D to 2-D leads to

$$E = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |f(x,y)|^2 dx dy. \quad (3.5)$$

### 4. Even-odd decomposition

A real-valued image  $f(x,y)$  can be decomposed into its **even**  $f_e(x,y)$  and **odd**  $f_o(x,y)$  components:

$$f(x,y) = f_e(x,y) + f_o(x,y), \quad (3.6)$$

where

$$f_e(x,y) = [f(x,y) + f(-x,-y)]/2, \quad (3.7a)$$

$$f_o(x,y) = [f(x,y) - f(-x,-y)]/2. \quad (3.7b)$$

## B. Non-Generalizations of 1-D Properties

We now introduce two new properties of images, separability and rotation, neither one of which has a 1-D counterpart.

### 1. Separability

An image  $f(x,y)$  is **separable** if it can be written as a product of separate functions  $f_1(x)$  and  $f_2(y)$ :

$$f(x,y) = f_1(x) f_2(y). \quad (3.8)$$

As we will see later, the 2-D Fourier transform of a separable image can be computed as a product of the 1-D Fourier transforms of  $f_1(x)$  and  $f_2(y)$ . 2-D impulses and box images are separable, whereas disk images are not.

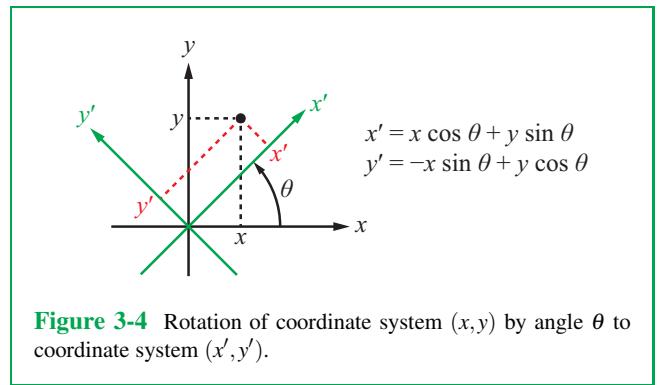
### 2. Rotation

To rotate an image by an angle  $\theta$ , we define rectangular coordinates  $(x',y')$  as the rectangular coordinates  $(x,y)$  rotated by angle  $\theta$ . Using the sine and cosine addition formulae, the rotated coordinates  $(x',y')$  are related to coordinates  $(x,y)$  by (Fig. 3-4)

$$x' = x \cos \theta + y \sin \theta \quad (3.9)$$

and

$$y' = -x \sin \theta + y \cos \theta. \quad (3.10)$$



**Figure 3-4** Rotation of coordinate system  $(x,y)$  by angle  $\theta$  to coordinate system  $(x',y')$ .

These can be combined into

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \quad (3.11)$$

Hence, after rotation by angle  $\theta$ , image  $f(x,y)$  becomes transformed into a new image  $g(x,y)$  given by

$$g(x,y) = f(x',y') = f(x \cos \theta + y \sin \theta, -x \sin \theta + y \cos \theta) \quad (3.12)$$

Note that rotating an image usually assumes that the image has been shifted so that the center of the image is at the origin  $(0,0)$ , as in Fig. 3-3(c).

**Exercise 3-2:** Which of the following images is separable:  
(a) 2-D impulse; (b) box; (c) disk?

**Answer:** 2-D impulse and box are separable; disk is not separable.

**Exercise 3-3:** Which of the following images is invariant to rotation: (a) 2-D impulse; (b) box; (c) disk with center at the origin  $(0,0)$ ?

**Answer:** 2-D impulse and disk are invariant to rotation; box is not invariant to rotation.

## 3-3 Continuous-Space Systems

A continuous-space system is a device or mathematical model that accepts as an input an image  $f(x,y)$  and produces as an

output an image  $g(x,y)$ .



The image rotation transformation described by Eq. (3.12) is a good example of such a 2-D system.

### 3-3.1 Linear and Shift-Invariant (LSI) Systems

The definition of the linearity property of 1-D systems (Section 2-2.1) extends directly to 2-D spatial systems, as does the definition for the invariance, except that time invariance in 1-D systems becomes *shift invariance* in 2-D systems. Systems that are both linear and shift-invariant are termed *linear shift-invariant (LSI)*.

and if the system also is shift-invariant, the 2-D superposition integral simplifies to the **2-D convolution** given by

$$\begin{aligned} g(x,y) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi, \eta) h(x - \xi, y - \eta) d\xi d\eta \\ &= f(x,y) * * h(x,y), \end{aligned} \quad (3.15a)$$

where the “double star” in  $f(x,y) * * h(x,y)$  denotes the 2-D convolution of the PSF  $h(x,y)$  with the input image  $f(x,y)$ . In symbolic form, the 2-D convolution is written as

$$f(x,y) \rightarrow \boxed{\text{LSI}} \rightarrow g(x,y) = f(x,y) * * h(x,y). \quad (3.15b)$$

► A 2-D convolution consists of a convolution in the  $x$  direction, followed by a convolution in the  $y$  direction, or vice versa. Consequently, the 1-D convolution properties listed in **Table 2-3** generalize to 2-D. ◀

### 3-3.2 Point-Spread Function (PSF)

The *point-spread function (PSF)* of a 2-D system is essentially its 2-D impulse response. The PSF  $h(x,y; x_0, y_0)$  of an image system is its response to a 2-D impulse  $\delta(x,y)$  shifted by  $(x_0, y_0)$ :

$$\delta(x - x_0, y - y_0) \rightarrow \boxed{\text{SYSTEM}} \rightarrow h(x,y; x_0, y_0). \quad (3.13a)$$

If the system is also *shift-invariant (SI)*, Eq. (3.13a) becomes

$$\delta(x - x_0, y - y_0) \rightarrow \boxed{\text{SI}} \rightarrow h(x - x_0, y - y_0). \quad (3.13b)$$

### 3-3.3 2-D Convolution

For a linear system, extending the 1-D superposition integral expression given by Eq. (2.13) to 2-D gives

$$\begin{aligned} f(x,y) \rightarrow \boxed{\text{L}} \rightarrow g(x,y) &= \\ &\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi, \eta) h(x,y; \xi, \eta) d\xi d\eta, \end{aligned} \quad (3.14)$$

**Concept Question 3-2:** Why do so many 1-D convolution properties generalize directly to 2-D?

**Exercise 3-4:** The operation of a 2-D mirror is described by  $g(x,y) = f(-x, -y)$ . Find the PSF of the mirror.

**Answer:**  $h(x,y; \xi, \eta) = \delta(x + \xi; y + \eta)$ .

**Exercise 3-5:** If  $g(x,y) = h(x,y) * * f(x,y)$ , what is  $4h(x,y) * * f(x-3, y-2)$  in terms of  $g(x,y)$ ?

**Answer:**  $4g(x-3, y-2)$ , using the shift and scaling properties of 1-D convolutions.

### 3-4 2-D Continuous-Space Fourier Transform (CSFT)

The 2-D *continuous-space Fourier transform (CSFT)* operates between the spatial domain  $(x,y)$  and the *spatial frequency* domain  $(\mu, v)$ , where  $\mu$  and  $v$  are called *spatial frequencies* or *wavenumbers*, with units of cycles/meter, analogous to the units of cycles/s (i.e., Hz) for the time frequency  $f$ .

The 2-D CSFT is denoted  $\mathbf{F}(\mu, v)$  and it is related to  $f(x,y)$

by

$$\mathbf{F}(\mu, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(\mu x + vy)} dx dy. \quad (3.16a)$$

The inverse operation is given by

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbf{F}(\mu, v) e^{j2\pi(\mu x + vy)} d\mu dv, \quad (3.16b)$$

and the combination of the two operations is represented symbolically by

$$f(x, y) \leftrightarrow \mathbf{F}(\mu, v).$$

In the 1-D continuous-time domain, we call the output of an LTI system its impulse response  $h(t)$  when the input is an impulse:



and we call the Fourier transform of  $h(t)$  the frequency response of the system,  $\mathbf{H}(f)$ :

$$h(t) \leftrightarrow \mathbf{H}(f).$$

The analogous relationships for a 2-D LSI system are

$$\delta(x, y) \rightarrow \boxed{\text{LSI}} \rightarrow h(x, y), \quad (3.17a)$$

$$h(x, y) \leftrightarrow \mathbf{H}(\mu, v). \quad (3.17b)$$

The CSFT  $\mathbf{H}(\mu, v)$  is called the *spatial frequency response* of the LSI system.

- As in 1-D, the 2-D Fourier transform of a convolution of two functions is equal to the product of their Fourier transforms:

$$f(x, y) \rightarrow \boxed{\text{LTI}} \rightarrow g(x, y) = h(x, y) * * f(x, y)$$

implies that

$$\mathbf{G}(\mu, v) = \mathbf{H}(\mu, v) \mathbf{F}(\mu, v).$$

All of the 1-D Fourier transform properties listed in **Table 2-4** and all of the 1-D transform pairs listed in **Table 2-5** generalize readily to 2-D. The 2-D version of the two tables is available in **Table 3-1**.

- The spectrum  $\mathbf{F}(\mu, v)$  of an image  $f(x, y)$  is its 2-D CSFT. The spatial frequency response  $\mathbf{H}(\mu, v)$  of an LSI 2-D system is the 2-D CSFT of its PSF  $h(x, y)$ . ◀

### 3-4.1 Notable 2-D CSFT Pairs and Properties

#### A. Conjugate Symmetry

The 2-D CTFT of a real-valued image  $f(x, y)$  obeys the *conjugate symmetry* property

$$\mathbf{F}^*(\mu, v) = \mathbf{F}(-\mu, -v), \quad (3.18)$$

which states that the 2-D Fourier transform  $\mathbf{F}(\mu, v)$ , must be reflected across both the  $\mu$  and  $v$  axes to produce its complex conjugate.

#### B. Separable Images

- The CSFT of a separable image  $f(x, y) = f_1(x) f_2(y)$  is itself separable in the spatial frequency domain:

$$f_1(x) f_2(y) \leftrightarrow \mathbf{F}_1(\mu) \mathbf{F}_2(v). \quad (3.19)$$

This assertion follows directly from the definition of the CSFT given by Eq. (3.16a):

$$\begin{aligned} \mathbf{F}(\mu, v) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(\mu x + vy)} dx dy \\ &= \int_{-\infty}^{\infty} f_1(x) e^{-j2\pi\mu x} dx \int_{-\infty}^{\infty} f_2(y) e^{-j2\pi vy} dy \\ &= \mathbf{F}_1(\mu) \mathbf{F}_2(v). \end{aligned} \quad (3.20)$$

- The CSFT pairs listed in **Table 3-1** are all separable functions, and can be obtained by applying Eq. (3.20) to the 1-D Fourier transform pairs listed in **Table 2-5**. CSFT pairs for non-separable functions are listed later in **Table 3-2**. ◀

#### C. Sinusoidal Image

Consider the sinusoidal image described by

$$f(x, y) = \cos(2\pi\mu_0 x) \cos(2\pi\nu_0 y),$$

where  $\mu_0 = 1.9$  cycles/cm and  $\nu_0 = 0.9$  cycles/cm are the frequencies of the spatial variations along  $x$  and  $y$  in the spatial

**Table 3-1** 2-D Continuous-space Fourier transform (CSFT).

Selected Properties			
1. Linearity	$\sum c_i f_i(x,y)$	$\leftrightarrow$	$\sum c_i \mathbf{F}_i(\mu,v)$
2. Spatial scaling	$f(a_x x, a_y y)$	$\leftrightarrow$	$\frac{1}{ a_x a_y } \mathbf{F}\left(\frac{\mu}{a_x}, \frac{v}{a_y}\right)$
3. Spatial shift	$f(x - x_0, y - y_0)$	$\leftrightarrow$	$e^{-j2\pi\mu x_0} e^{-j2\pi v y_0} \mathbf{F}(\mu,v)$
4. Reversal	$f(-x, -y)$	$\leftrightarrow$	$\mathbf{F}(-\mu, -v)$
5. Conjugation	$f^*(x,y)$	$\leftrightarrow$	$\mathbf{F}^*(-\mu, -v)$
6. Convolution in space	$f(x,y) * h(x,y)$	$\leftrightarrow$	$\mathbf{F}(\mu,v) \mathbf{H}(\mu,v)$
7. Convolution in frequency	$f(x,y) h(x,y)$	$\leftrightarrow$	$\mathbf{F}(\mu,v) * \mathbf{H}(\mu,v)$
CSFT Pairs			
8.	$\delta(x,y)$	$\leftrightarrow$	1
9.	$\delta(x - x_0, y - y_0)$	$\leftrightarrow$	$e^{-j2\pi\mu x_0} e^{-j2\pi v y_0}$
10.	$e^{j2\pi\mu_0 x} e^{j2\pi v_0 y}$	$\leftrightarrow$	$\delta(\mu - \mu_0, v - v_0)$
11.	$\text{rect}\left(\frac{x}{\ell_x}\right) \text{rect}\left(\frac{y}{\ell_y}\right)$	$\leftrightarrow$	$\ell_x \ell_y \text{sinc}(\ell_x \mu) \text{sinc}(\ell_y v)$
12.	$\mu_0 v_0 \text{sinc}(\mu_0 x) \text{sinc}(v_0 y)$	$\leftrightarrow$	$\text{rect}\left(\frac{\mu}{\mu_0}\right) \text{rect}\left(\frac{v}{v_0}\right)$
13.	$e^{-\pi x^2} e^{-\pi y^2}$	$\leftrightarrow$	$e^{-\pi \mu^2} e^{-\pi v^2}$
14.	$\cos(2\pi\mu_0 x) \cos(2\pi v_0 y)$	$\leftrightarrow$	$\frac{1}{4} [\delta(\mu \pm \mu_0) \delta(v \pm v_0)]$

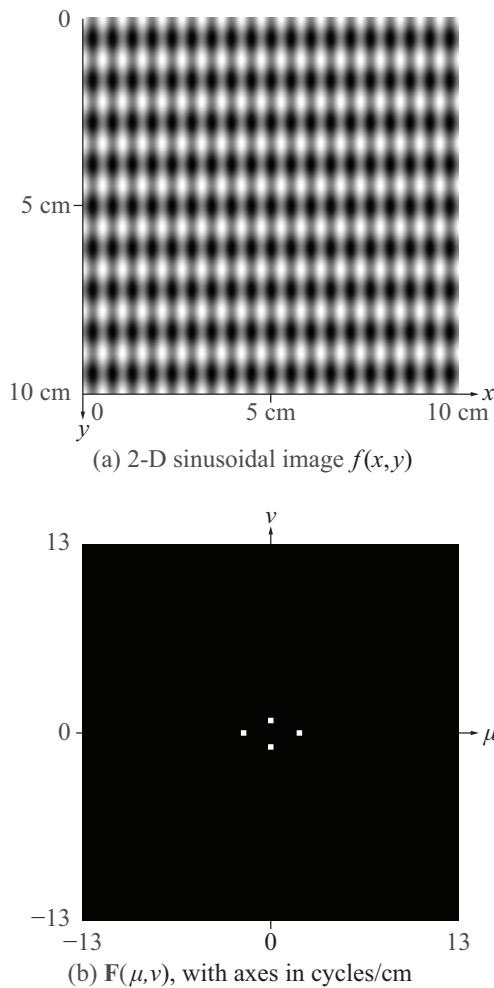
domain, respectively. A grayscale image-display of  $f(x,y)$  is shown in **Fig. 3-5(a)**, with pure black representing  $f(x,y) = -1$  and pure white representing  $f(x,y) = +1$ . As expected, the image exhibits a repetitive pattern along both  $x$  and  $y$ , with 19 cycles in 10 cm in the  $x$  direction and 9 cycles in 10 cm in the  $y$  direction, corresponding to spatial frequencies of  $\mu = 1.9$  cycles/cm and  $v = 0.9$  cycles/cm, respectively.

By Eq. (3.20), the CSFT of  $f(x,y)$  is

$$\begin{aligned} \mathbf{F}(\mu,v) &= \mathbf{F}_1(\mu) \mathbf{F}_2(v) \\ &= \mathcal{F}\{\cos(2\pi\mu_0 x)\} \mathcal{F}\{\cos(2\pi v_0 y)\} \\ &= \frac{1}{4} [\delta(\mu - \mu_0) + \delta(\mu + \mu_0)] \\ &\quad \times [\delta(v - v_0) + \delta(v + v_0)], \end{aligned} \quad (3.21)$$

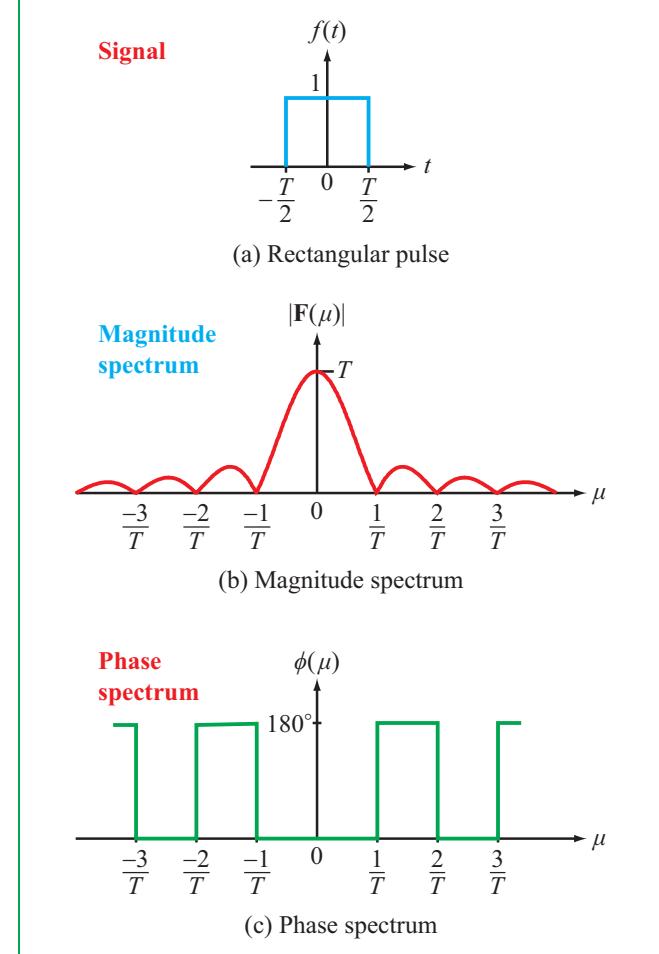
where we derived entry #14 in **Table 3-1**. The CSFT consists of four impulses at spatial frequency locations  $\{\mu, v\} = \{\pm\mu_0, \pm v_0\}$ , as is also shown in **Fig. 3-5(b)**.

We should note that the images displayed in **Fig. 3-5** are



**Figure 3-5** (a) Sinusoidal image  $f(x,y)=\cos(2\pi\mu_0x)\cos(2\pi\nu_0y)$  with  $\mu_0 = 1.9$  cycles/cm and  $\nu_0 = 0.9$  cycles/cm, and (b) the corresponding Fourier transform  $\mathbf{F}(\mu, v)$ , which consists of four impulses (4 white dots) at  $\{\pm\mu_0, \pm\nu_0\}$ .

not truly continuous functions; function  $f(x,y)$  was discretized into  $256 \times 256$  pixels and then a discrete form of the Fourier transform called the DFT (Section 3-8) was used to generate  $\mathbf{F}(\mu, v)$ , also in the form of a  $256 \times 256$  image.



**Figure 3-6** (a) Rectangular pulse, and corresponding (b) magnitude spectrum and (c) phase spectrum.

## D. Box Image

As a prelude to presenting the CSFT of a 2-D box image, let us examine the 1-D case of the rectangular pulse  $f(t) = \text{rect}(t/T)$  shown in Fig. 3-6(a). The pulse is centered at the origin and extends between  $-T/2$  and  $+T/2$ , and the corresponding Fourier transform is, from entry #3 in Table 2-5,

$$\mathbf{F}(\mu) = T \text{sinc}(\mu T), \quad (3.22)$$

where  $\text{sinc}(\theta)$  is the sinc function defined by Eq. (2.35) as  $\text{sinc } \theta = [\sin(\pi\theta)]/(\pi\theta)$ . By defining  $\mathbf{F}(\mu)$  as

$$\mathbf{F}(\mu) = |\mathbf{F}(\mu)| e^{j\phi(\mu)}, \quad (3.23)$$

we determine that the **phase spectrum**  $\phi(\mu)$  can be ascertained from

$$e^{j\phi(\mu)} = \frac{\mathbf{F}(\mu)}{|\mathbf{F}(\mu)|} = \frac{\text{sinc}(\mu T)}{|\text{sinc}(\mu T)|}. \quad (3.24)$$

The quantity on the right-hand side of Eq. (3.24) is always equal to +1 or -1. Hence,  $\phi(\mu) = 0^\circ$  when  $\text{sinc}(\mu T)$  is positive and  $180^\circ$  when  $\text{sinc}(\mu T)$  is negative. The magnitude and phase spectra of the rectangular pulse are displayed in **Figs. 3-6(b)** and **(c)**, respectively.

Next, let us consider the white square shown in **Fig. 3-7(a)**. If we assign an amplitude of 1 to the white part of the image and 0 to the black part, the variation across the image along the  $x$  direction is analogous to that representing the time-domain pulse of **Fig. 3-6(a)**, and the same is true along  $y$ . Hence, the white square represents the product of two pulses, one along  $x$  and another along  $y$ , and is given by

$$f(x, y) = \text{rect}\left(\frac{x}{\ell}\right) \text{rect}\left(\frac{y}{\ell}\right), \quad (3.25)$$

where  $\ell$  is the length of the square sides. In analogy with Eq. (3.22),

$$\mathbf{F}(\mu, v) = \ell^2 \text{sinc}(\mu\ell) \text{sinc}(v\ell). \quad (3.26)$$

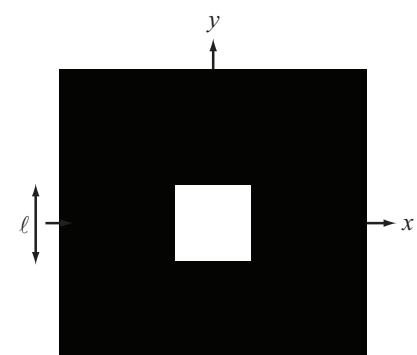
The magnitude and phase spectra associated with the expression given by Eq. (3.26) are displayed in grayscale format in **Fig. 3-7(b)** and **(c)**, respectively. For the magnitude spectrum, white represents the peak value of  $|\mathbf{F}_{\text{Box}}(\mu, v)|$  and black represents  $|\mathbf{F}_{\text{Box}}(\mu, v)| = 0$ . The phase spectrum  $\phi(\mu, v)$  varies between  $0^\circ$  and  $180^\circ$ , so the grayscale was defined such that white corresponds to  $+180^\circ$  and black to  $0^\circ$ . The tonal variations along  $\mu$  and  $v$  are equivalent to the patterns depicted in **Figs. 3-6(b)** and **(c)** for the rectangular pulse.

In the general case of a box image of widths  $\ell_x$  along  $x$  and  $\ell_y$  along  $y$ , and centered at  $(x_0, y_0)$ ,

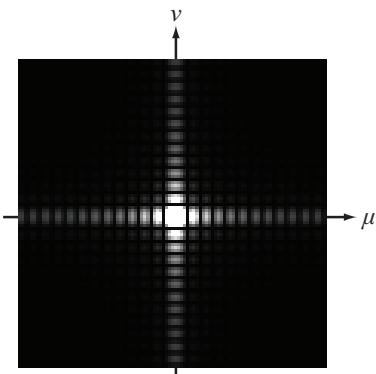
$$f(x, y) = \text{rect}\left(\frac{x-x_0}{\ell_x}\right) \text{rect}\left(\frac{y-y_0}{\ell_y}\right). \quad (3.27)$$

In view of properties #3 and 11 in **Table 3-1**, the corresponding CSFT is

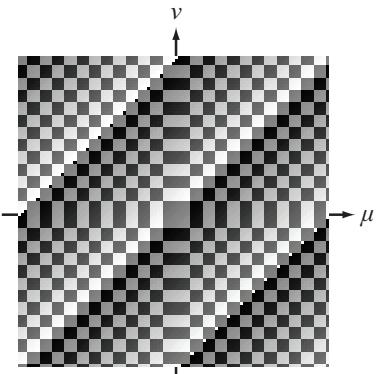
$$\mathbf{F}(\mu, v) = \ell_x e^{-j2\pi\mu x_0} \text{sinc}(\mu\ell_x) \ell_y e^{-j2\pi v y_0} \text{sinc}(v\ell_y), \quad (3.28)$$



(a) White square image



(b) Magnitude image  $|\mathbf{F}(\mu, v)|$



(c) Phase image  $\phi(\mu, v)$

**Figure 3-7** (a) Grayscale image of a white square in a black background, (b) magnitude spectrum, and (c) phase spectrum.

and the associated magnitude and phase spectra are given by

$$|\mathbf{F}(\mu, v)| \quad (3.29a)$$

and

$$\phi(\mu, v) = \tan^{-1} \left\{ \frac{\Im[\mathbf{F}(\mu, v)]}{\Re[\mathbf{F}(\mu, v)]} \right\}. \quad (3.29b)$$

A visual example is shown in **Fig. 3-8(a)** for a square box of sides  $\ell_x = \ell_y = \ell$ , and shifted to the right by  $L$  and also downward by  $L$ . Inserting  $x_0 = L$ ,  $y_0 = -L$ , and  $\ell_x = \ell_y = \ell$  in Eq. (3.28) leads to

$$\mathbf{F}(\mu, v) = \ell^2 e^{-j2\pi\mu L} \text{sinc}(\mu\ell) e^{j2\pi v L} \text{sinc}(v\ell). \quad (3.30)$$

The magnitude and phase spectra associated with the CSFT of the box image defined by Eq. (3.30) are displayed in **Fig. 3-8(b)** and **(c)**, respectively. The magnitude spectrum of the shifted box is similar to that of the unshifted box (**Fig. 3-7**), but the phase spectra of the two boxes are considerably different.

### E. 2-D Ideal Brickwall Lowpass Filter

An ideal *lowpass filter* is characterized by a spatial frequency response  $\mathbf{H}_{LP}(\mu, v)$  with a specified cutoff frequency  $\mu_0$  along both the  $\mu$  and  $v$  axes:

$$\mathbf{H}_{LP}(\mu, v) = \begin{cases} 1 & \text{for } 0 \leq |\mu|, |v| \leq \mu_0, \\ 0 & \text{otherwise.} \end{cases} \quad (3.31)$$

Mathematically,  $\mathbf{H}_{LP}(\mu, v)$  can be expressed in terms of rectangle functions centered at the origin and of width  $2\mu_0$ :

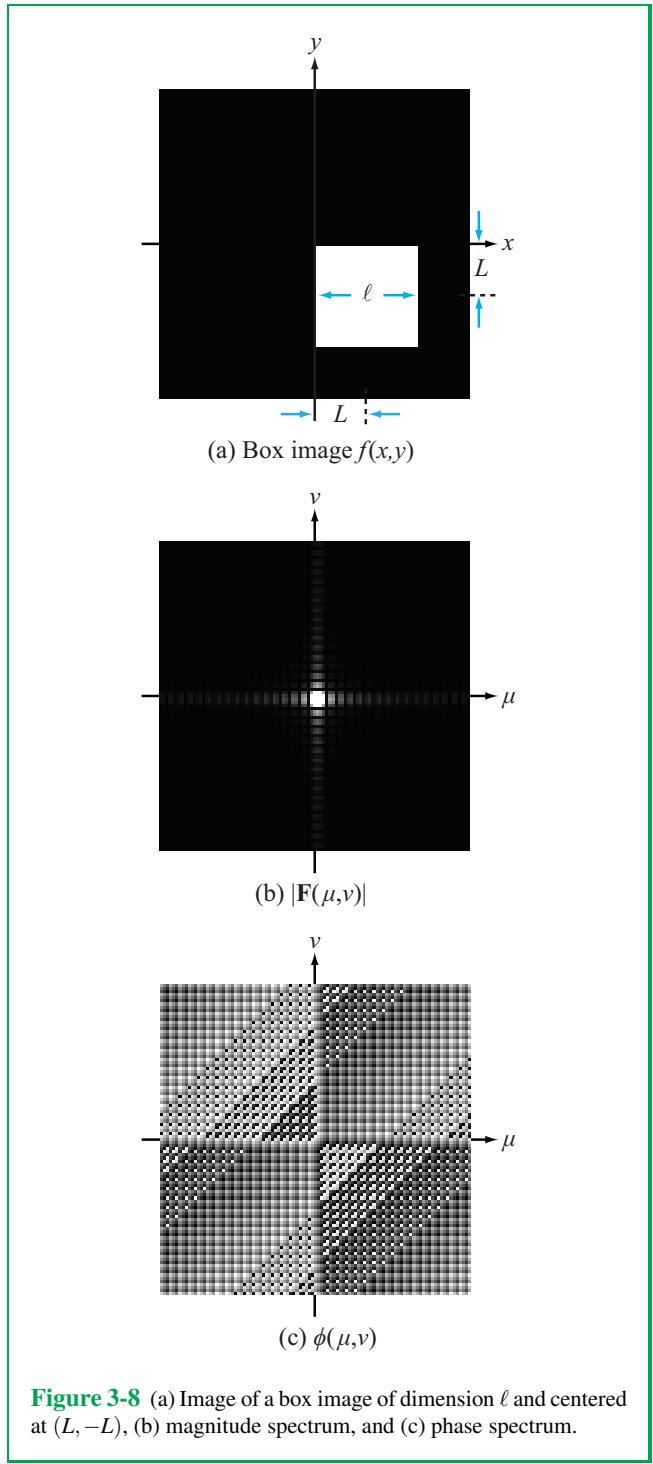
$$\mathbf{H}_{LP}(\mu, v) = \text{rect}\left(\frac{\mu}{2\mu_0}\right) \text{rect}\left(\frac{v}{2\mu_0}\right). \quad (3.32)$$

The inverse 2-D Fourier transfer of  $\mathbf{H}_{LP}(\mu, v)$  is the PSF  $h_{LP}(x, y)$ . Application of property #12 in **Table 3-1** yields

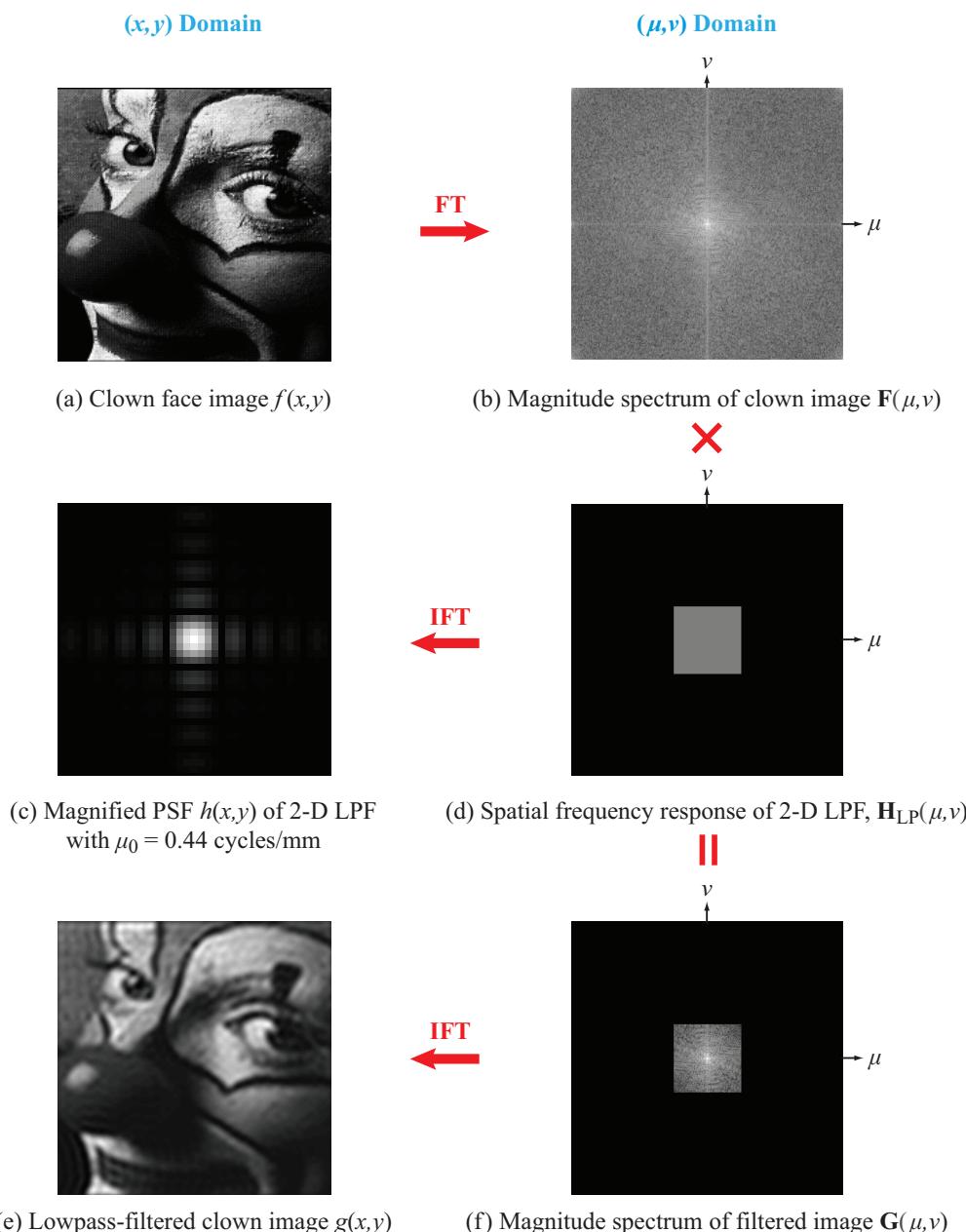
$$\begin{aligned} h_{LP}(x, y) &= \frac{\sin(2\pi x\mu_0)}{\pi x} \frac{\sin(2\pi y\mu_0)}{\pi y} \\ &= 4\mu_0^2 \text{sinc}(2x\mu_0) \text{sinc}(2y\mu_0). \end{aligned} \quad (3.33)$$

### F. Example: Lowpass-Filtering of Clown Image

**Figure 3-9(a)** displays an image of a clown's face. Our goal is to lowpass-filter the clown image using an ideal lowpass filter with a cutoff frequency  $\mu_0 = 0.44$  cycles/mm. To do so, we perform



**Figure 3-8** (a) Image of a box image of dimension  $\ell$  and centered at  $(L, -L)$ , (b) magnitude spectrum, and (c) phase spectrum.



**Figure 3-9** Lowpass filtering the clown image in (a) to generate the image in (e). Image  $f(x,y)$  is 40 mm × 40 mm and the magnitude spectra extend between -2.5 cycles/mm and +2.5 cycles/mm in both directions.

the following steps:

(1) We denote the intensity distribution of the clown image as  $f(x,y)$ , and we apply the 2-D Fourier transform to obtain the spectrum  $\mathbf{F}(\mu,v)$ , whose magnitude is displayed in **Fig. 3-9(b)**.

(2) The spatial frequency response of the lowpass filter, shown in **Fig. 3-9(d)**, consists of a white square representing the passband of the filter. Its functional form is given by Eq. (3.32) with  $\mu_0 = 0.44$  cycles/mm. The corresponding PSF given by Eq. (3.33) is displayed in **Fig. 3-9(c)**.

(3) Multiplication of  $\mathbf{F}(\mu,v)$  by  $\mathbf{H}_{LP}(\mu,v)$  yields the spectrum of the filtered image,  $\mathbf{G}(\mu,v)$ :

$$\mathbf{G}(\mu,v) = \mathbf{F}(\mu,v) \mathbf{H}_{LP}(\mu,v). \quad (3.34)$$

The magnitude of the result is displayed in **Fig. 3-9(f)**. Upon performing an inverse Fourier transform on  $\mathbf{G}(\mu,v)$ , we obtain  $g(x,y)$ , the lowpass-filtered image of the clown face shown in **Fig. 3-9(e)**. Image  $g(x,y)$  looks like a blurred version of the original image  $f(x,y)$  because the lowpass filtering smooths out rapid variations in the image.

Alternatively, we could have obtained  $g(x,y)$  directly by performing a convolution in the spatial domain:

$$g(x,y) = f(x,y) * h_{LP}(x,y). \quad (3.35)$$

Even though the convolution approach is direct and conceptually straightforward, it is computationally much easier to perform the filtering by transforming to the angular frequency domain, multiplying the two spectra, and then inverse transforming back to the spatial domain. The actual computation was performed using discretized (pixelated) images and the Fourier transformations were realized using the 2-D DFT introduced later in Section 3-8.

## 3-4.2 Image Rotation

► Rotating an image by angle  $\theta$  (**Fig. 3-10**) in the 2-D spatial domain  $(x,y)$  causes its Fourier transform to also rotate by the same angle in the frequency domain  $(\mu,v)$ . ◀

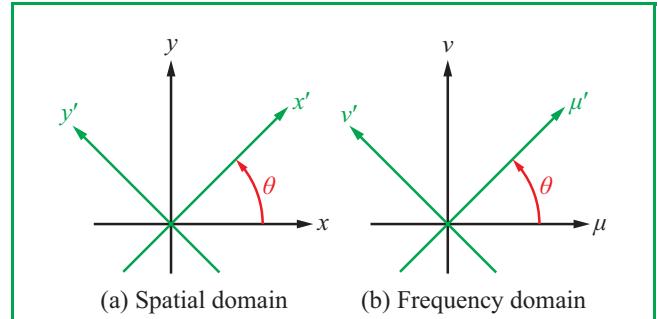
To demonstrate the validity of the assertion, we start with the relationships given by Eqs. (3.11) and (3.12):

$$g(x,y) = f(x',y') \quad (3.36)$$

and

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R_\theta \begin{bmatrix} x \\ y \end{bmatrix}, \quad (3.37)$$

where  $f(x,y)$  is the original image,  $(x',y')$  are the coordinates



**Figure 3-10** Rotation of axes by  $\theta$  in (a) spatial domain causes rotation by the same angle in the (b) spatial frequency domain.

of the rotated image  $g(x,y) = f(x',y')$ , and  $R_\theta$  is the **rotation matrix** relating  $(x,y)$  to  $(x',y')$ :

$$R_\theta = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}. \quad (3.38)$$

The inverse relationship between  $(x',y')$  and  $(x,y)$  is given in terms of the inverse of matrix  $R_\theta$ :

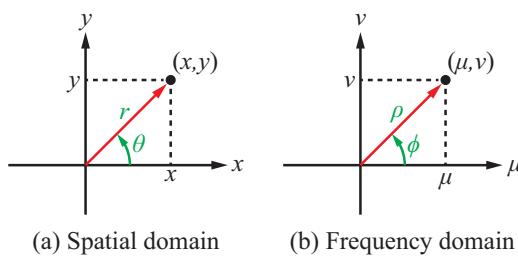
$$\begin{bmatrix} x \\ y \end{bmatrix} = R_\theta^{-1} \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix}. \quad (3.39)$$

The 2-D Fourier transform of  $g(x,y)$  is given by

$$\begin{aligned} \mathbf{G}(\mu,v) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x,y) e^{-j2\pi(\mu x + vy)} dx dy \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x',y') e^{-j2\pi(\mu x + vy)} dx dy. \end{aligned} \quad (3.40)$$

Using the relationships between  $(x,y)$  and  $(x',y')$  defined by Eq. (3.39), while also recognizing that  $dx dy = dx' dy'$  because a differential element of area is the same in either coordinate system, Eq. (3.40) becomes

$$\begin{aligned} \mathbf{G}(\mu,v) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x',y') e^{-j2\pi[\mu(x' \cos \theta - y' \sin \theta) + v(x' \sin \theta + y' \cos \theta)]} dx' dy' \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x',y') e^{-j2\pi[\mu'x' + v'y']} dx' dy', \end{aligned} \quad (3.41)$$



**Figure 3-11** Relationships between Cartesian and polar coordinates in (a) spatial domain and (b) spatial frequency domain.

where we define

$$\begin{bmatrix} \mu' \\ v' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} \mu \\ v \end{bmatrix} = R_\theta \begin{bmatrix} \mu \\ v \end{bmatrix}. \quad (3.42)$$

The newly defined spatial-frequency coordinates  $(\mu', v')$  are related to the original frequency coordinates  $(\mu, v)$  by exactly the same rotation matrix  $R_\theta$  that was used to rotate image  $f(x, y)$  to  $g(x, y)$ . The consequence of using Eq. (3.42) is that Eq. (3.38) now assumes the standard form for the definition of the Fourier transform for  $f(x', y')$ :

$$\mathbf{G}(\mu, v) = \mathbf{F}(\mu', v'). \quad (3.43)$$

In conclusion, we have demonstrated that rotation of image  $f(x, y)$  by angle  $\theta$  in the  $(x, y)$  plane leads to rotation of  $\mathbf{F}(\mu, v)$  by exactly the same angle in the spatial frequency domain.

### 3-4.3 2-D Fourier Transform in Polar Coordinates

In the spatial domain, the location of a point can be specified by its  $(x, y)$  coordinates in a Cartesian coordinate system or by its  $(r, \theta)$  in the corresponding polar coordinate system (**Fig. 3-11(a)**). The two pairs of variables are related by

$$\begin{bmatrix} x = r \cos \theta \\ y = r \sin \theta \end{bmatrix} \leftrightarrow \begin{bmatrix} r = \sqrt{x^2 + y^2} \\ \theta = \tan^{-1}(y/x) \end{bmatrix}. \quad (3.44)$$

Similarly, in the spatial frequency domain (**Fig. 3-11(b)**), we can use Cartesian coordinates  $(\mu, v)$  or their corresponding polar

coordinates  $(\rho, \phi)$ , with

$$\begin{bmatrix} \mu = \rho \cos \phi \\ v = \rho \sin \phi \end{bmatrix} \leftrightarrow \begin{bmatrix} \rho = \sqrt{\mu^2 + v^2} \\ \phi = \tan^{-1}(v/\mu) \end{bmatrix}. \quad (3.45)$$

The Fourier transform of  $f(x, y)$  is given by Eq. (3.16a) as

$$\mathbf{F}(\mu, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(\mu x + vy)} dx dy. \quad (3.46)$$

We wish to transform  $\mathbf{F}(\mu, v)$  into polar coordinates so we may apply it to circularly symmetric images or to use it in filtering applications where the filter's frequency response is defined in terms of polar coordinates. To that end, we convert the differential area  $dx dy$  in Eq. (3.46) to  $r dr d\theta$ , and we use the relations given by Eqs. (3.44) and (3.45) to transform the exponent in Eq. (3.46):

$$\begin{aligned} \mu x + vy &= (\rho \cos \phi)(r \cos \theta) + (\rho \sin \phi)(r \sin \theta) \\ &= \rho r [\cos \phi \cos \theta + \sin \phi \sin \theta] \\ &= \rho r \cos(\phi - \theta). \end{aligned} \quad (3.47)$$

The cosine addition formula was used in the last step. Conversion to polar coordinates leads to

$$\mathbf{F}(\rho, \phi) = \int_{r=0}^{\infty} \int_{\theta=0}^{2\pi} f(r, \theta) e^{-j2\pi\rho r \cos(\phi - \theta)} r dr d\theta. \quad (3.48a)$$

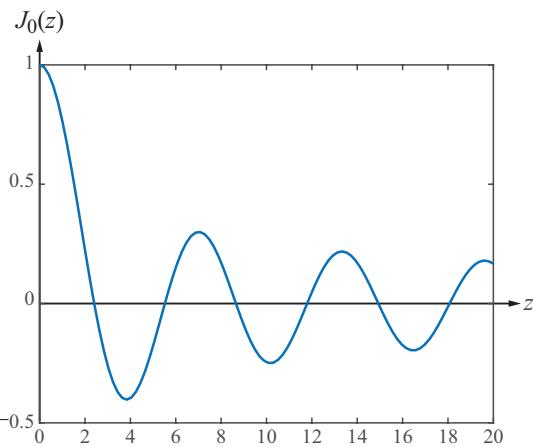
The inverse transform is given by

$$f(r, \theta) = \int_{\rho=0}^{\infty} \int_{\phi=0}^{2\pi} \mathbf{F}(\rho, \phi) e^{j2\pi\rho r \cos(\phi - \theta)} \rho d\rho d\phi. \quad (3.48b)$$

### 3-4.4 Rotationally Invariant Images

A **rotationally invariant** image is a circularly symmetric image, which means that  $f(r, \theta)$  is a function of  $r$  only. According to Section 3-4.2, rotation of  $f(r, \theta)$  by a fixed angle  $\theta_0$  causes the transform  $\mathbf{F}(\rho, \phi)$  to rotate by exactly the same angle  $\theta_0$ . Hence, if  $f(r, \theta)$  is independent of  $\theta$ , it follows that  $\mathbf{F}(\rho, \phi)$  is independent of  $\phi$ , in which case Eq. (3.48a) can be rewritten as

$$\begin{aligned} \mathbf{F}(\rho) &= \int_{r=0}^{\infty} \int_{\theta=0}^{2\pi} f(r) e^{-j2\pi\rho r \cos(\phi - \theta)} r dr d\theta \\ &= \int_{r=0}^{\infty} r f(r) \left[ \int_{\theta=0}^{2\pi} e^{-j2\pi\rho r \cos(\phi - \theta)} d\theta \right] dr. \end{aligned} \quad (3.49)$$



**Figure 3-12** Plot of  $J_0(z)$  the Bessel function of order zero, as a function of  $z$ .

Because the integration over  $\theta$  extends over the range  $(0, 2\pi)$ , the integrated value is the same for any fixed value of  $\phi$ . Hence, for simplicity we set  $\phi = 0$ , in which case Eq. (3.49) simplifies to

$$\begin{aligned} \mathbf{F}(\rho) &= \int_{r=0}^{\infty} r f(r) \left[ \int_{\theta=0}^{2\pi} e^{-j2\pi\rho r \cos\theta} d\theta \right] dr \\ &= 2\pi \int_{r=0}^{\infty} r f(r) J_0(2\pi\rho r) dr, \end{aligned} \quad (3.50)$$

where  $J_0(z)$  is the **Bessel function** of order zero:

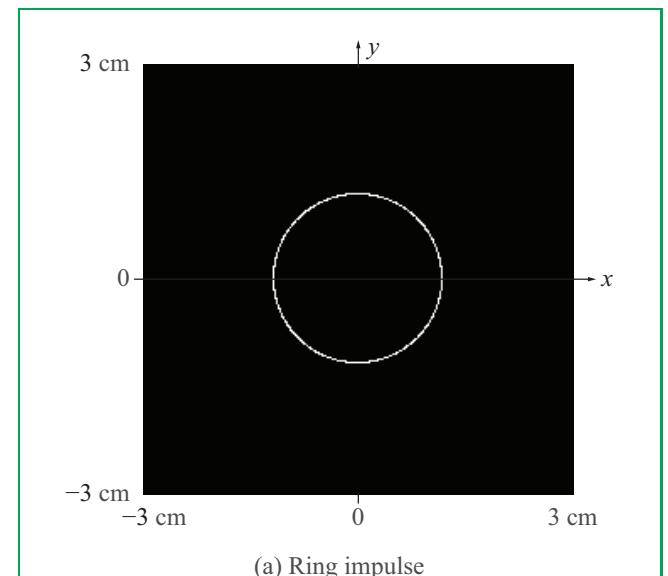
$$J_0(z) = \frac{1}{2\pi} \int_0^{2\pi} e^{-jz\cos\theta} d\theta. \quad (3.51)$$

A plot of  $J_0(z)$  versus  $z$  is shown in **Fig. 3-12**.

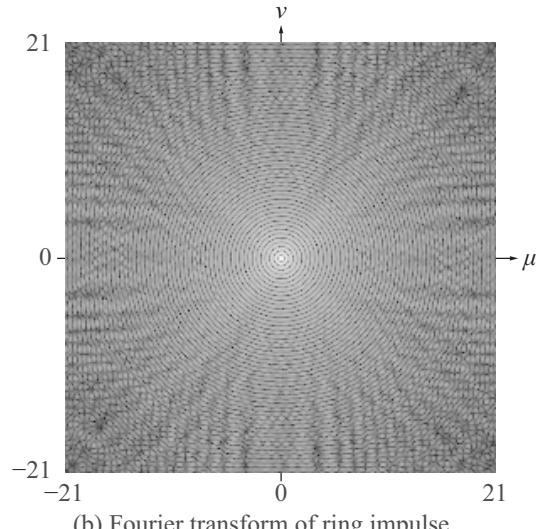
The integral expression on the right-hand side of Eq. (3.50) is known as the **Hankel transform** of order zero. Hence, the Fourier transform of a circularly symmetric image  $f(r)$  is given by its Hankel transform of order zero. An example is the **ring impulse**

$$f(r) = \delta(r - a), \quad (3.52a)$$

which defines a unit-intensity circle of radius  $a$  (**Fig. 3-13(a)**) in the spatial coordinate system  $(r, \theta)$ . The corresponding Fourier



(a) Ring impulse

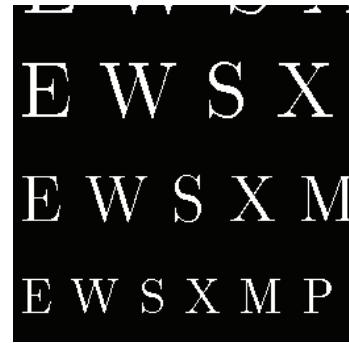


(b) Fourier transform of ring impulse

**Figure 3-13** (a) Image of ring impulse of radius  $a = 1$  cm and (b) the logarithm of its 2-D CTFT. [In display mode, the images are  $256 \times 256$  pixels.]

**Table 3-2** 2-D Fourier transforms of rotationally invariant images.

$f(r)$	$\leftrightarrow$	$F(\rho)$
$\frac{\delta(r)}{\pi r}$	$\leftrightarrow$	1
$\text{rect}(r)$	$\leftrightarrow$	$\frac{J_1(\pi\rho)}{2\rho}$
$\frac{J_1(\pi r)}{2r}$	$\leftrightarrow$	$\text{rect}(\rho)$
$\frac{1}{r}$	$\leftrightarrow$	$\frac{1}{\rho}$
$e^{-\pi r^2}$	$\leftrightarrow$	$e^{-\pi\rho^2}$
$\delta(r - r_0)$	$\leftrightarrow$	$2\pi r_0 J_0(2\pi r_0 \rho)$



(a) Letters image  $f(x,y)$



(b) Letters image  $f(x',y')$  with  $x' = ax$  and  $y' = ay$ , spatially scaled by  $a = 4$

**Figure 3-14** (a) Letters image and (b) a scaled version.

transform is

$$\mathbf{F}(\rho) = 2\pi \int_{r=0}^{\infty} r \delta(r-a) J_0(2\pi\rho r) dr = 2\pi a J_0(2\pi\rho a). \quad (3.52b)$$

The image in **Fig. 3-13(b)** displays the variation of  $\mathbf{F}(\rho)$  as a function of  $\rho$  in the spatial frequency domain for a ring with  $a = 1$  cm (image size is 6 cm  $\times$  6 cm).

**Table 3-2** provides a list of Fourier transform pairs of rotationally symmetric images.

### 3-4.5 Image Examples

#### A. Scaling

**Figure 3-14** compares image  $f(x,y)$ , representing an image of letters, to a scaled-down version  $f(x',y')$  with  $x' = ax$ ,  $y' = ay$ , and  $a = 4$ . The area of the scaled-down image is  $1/16$  of the area of the original. To enlarge the image, the value of  $a$  should be smaller than 1.

#### B. Image Rotation

The image displayed in **Fig. 3-15(a)** is a sinusoidal image that oscillates along only the  $y$  direction. Its 2-D spectrum consists of two impulse functions along the  $v$  direction, as shown in **Fig. 3-15(b)**. Rotating the sinusoidal image by  $45^\circ$  to the image in **Fig. 3-15(c)** leads to a corresponding rotation of the spectrum, as shown in **Fig. 3-15(d)**.

### C. Gaussian Image

A 2-D Gaussian image is characterized by

$$f(x,y) = e^{-\pi(x^2+y^2)}. \quad (3.53a)$$

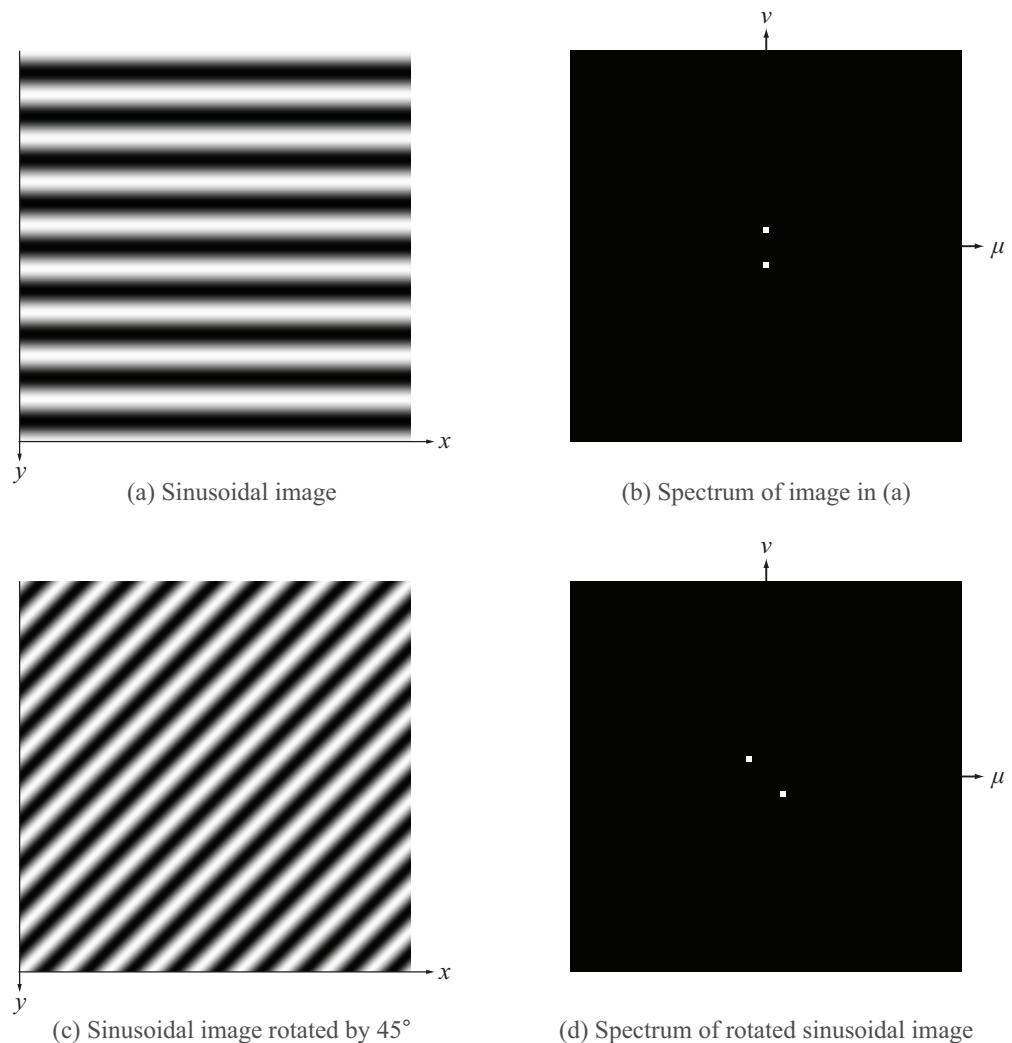
**Gaussian image**

Since  $x^2 + y^2 = r^2$ ,  $f(x,y)$  is rotationally invariant, so we can rewrite it as

$$f(r) = e^{-\pi r^2}. \quad (3.53b)$$

To obtain the Fourier transform  $\mathbf{F}(\rho)$ , we can apply Eq. (3.50), the Fourier transform for a rotationally invariant image:

$$\begin{aligned} \mathbf{F}(\rho) &= 2\pi \int_0^{\infty} r f(r) J_0(2\pi\rho r) dr \\ &= 2\pi \int_0^{\infty} r e^{-\pi r^2} J_0(2\pi\rho r) dr. \end{aligned} \quad (3.54)$$



**Figure 3-15** (a) Sinusoidal image and (b) its 2-D spectrum; (c) rotated image and (d) its rotated spectrum.

From standard tables of integrals, we borrow the following identity for any real variable  $t$ :

$$\int_0^\infty te^{-a^2t^2} J_0(bt) dt = \frac{1}{2a^2} e^{-b^2/4a^2}, \quad \text{for } a^2 > 0. \quad (3.55)$$

The integrals in Eq. (3.54) and Eq. (3.55) become identical if we set  $t = r$ ,  $a^2 = \pi$ , and  $b = 2\pi\rho$ , which leads to

$$\mathbf{F}(\rho) = e^{-\pi\rho^2}. \quad (3.56)$$

**Gaussian spectrum**

► Hence, the Fourier transform of a 2-D Gaussian image is itself 2-D Gaussian. ◀

## D. Disk Image

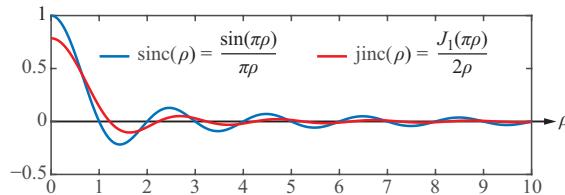
A disk image has a value of 1 inside the disk area and zero outside it. A disk image centered at the origin and of radius  $1/2$  is characterized by Eq. (3.3) in  $(x, y)$  coordinates. Conversion to polar coordinates gives

$$f_{\text{Disk}}(r) = \text{rect}(r) = \begin{cases} 1 & \text{for } 0 \leq r < 1/2, \\ 0 & \text{otherwise.} \end{cases} \quad (3.57)$$

After much algebra, it can be shown that the corresponding Fourier transform is given by

$$\mathbf{F}_{\text{Disk}}(\rho) = \frac{J_1(\pi\rho)}{2\rho} = \text{jinc}(\rho), \quad (3.58)$$

where  $J_1(x)$  is the Bessel function of order 1, and  $\text{jinc}(x) = J_1(\pi x)/(2x)$  is called the **jinc function**, which comes from its resemblance in both waveform and purpose to the sinc function defined by Eq. (2.35), except that the numerator changes from a sine to a Bessel function. **Figure 3-16** displays a plot of  $\text{jinc}(\rho)$ , as well as a plot of the sinc function  $\text{sinc}(\rho)$ , included here for comparison. In one dimension, the Fourier transform of  $\text{rect}(x)$  is  $\text{sinc}(\mu)$ ; in two dimensions, the Fourier transform of a disk image  $f_{\text{Disk}}(r) = \text{rect}(r)$  is given by  $\text{jinc}(\rho)$ , which resembles the variation exhibited by the sinc function.



**Figure 3-16** Sinc function (blue) and jinc function (red).

## E. PSF of Radial Brickwall Lowpass Filter

In the spatial frequency domain, the frequency response of a **radial brickwall lowpass filter** with cutoff spatial frequency  $\rho_0$

extends between 0 and  $\rho_0$ , and is given by

$$\mathbf{H}_{\text{LP}}(\rho) = \text{rect}\left(\frac{\rho}{2\rho_0}\right) = \begin{cases} 1 & \text{for } 0 < |\rho| < \rho_0, \\ 0 & \text{otherwise.} \end{cases}$$

By Fourier duality, or equivalently by application of the inverse transformation given by Eq. (3.48b), we obtain the PSF of the lowpass filter as

$$h_{\text{LP}}(r) = \frac{J_1(2\pi\rho_0 r)(2\rho_0)}{2r} = (2\rho_0)^2 \text{jinc}(2\rho_0 r). \quad (3.59)$$

► We should always remember the scaling property of the Fourier transform, namely if

$$f(r) \leftrightarrow \mathbf{F}(\rho),$$

then for any real-valued scaling factor  $a$ ,

$$f(ar) \leftrightarrow \frac{1}{|a|^2} \mathbf{F}\left(\frac{\rho}{a}\right).$$

The scaling property allows us to use available expressions, such as those in Eqs. (3.58) and (3.59), and to easily convert them into the expressions appropriate (for example) to disks of different sizes or filters with different cutoff frequencies. ◀

**Concept Question 3-3:** Why do so many 1-D Fourier transform properties generalize directly to 2-D?

**Concept Question 3-4:** Where does the jinc function get its name?

**Exercise 3-6:** Why do  $f(x, y)$  and  $f(x - x_0, y - y_0)$  have the same magnitude spectrum  $|\mathbf{F}(\mu, v)|$ ?

**Answer:** Let  $g(x, y) = f(x - x_0, y - y_0)$ . Then, from entry #3 in **Table 3-1**,

$$\begin{aligned} |\mathbf{G}(\mu, v)| &= |e^{-j2\pi\mu x_0} e^{-j2\pi v y_0} \mathbf{F}(\mu, v)| \\ &= |e^{-j2\pi\mu x_0} e^{-j2\pi v y_0}| |\mathbf{F}(\mu, v)| = |\mathbf{F}(\mu, v)|. \end{aligned}$$

**Exercise 3-7:** Compute the 2-D CSFT of  $f(x,y) = e^{-\pi r^2}$ , where  $r^2 = x^2 + y^2$ , without using Bessel functions. Hint:  $f(x,y)$  is separable.

**Answer:**  $f(x,y) = e^{-\pi r^2} = e^{-\pi x^2} e^{-\pi y^2}$  is separable, so Eq. (3.19) and entry #5 of **Table 2-5** (see also entry #13 of **Table 3-1**) give

$$F(\mu, \nu) = e^{-\pi \mu^2} e^{-\pi \nu^2} = e^{-\pi(\mu^2 + \nu^2)} = e^{-\pi \rho^2}.$$

**Exercise 3-8:** The 1-D phase spectrum  $\phi(\mu)$  in **Fig. 3-6(c)** is either 0 or  $180^\circ$  for all  $\mu$ . Yet the phase of the 1-D CTFT of a real-valued function must be an odd function of frequency. How can these two statements be reconciled?

**Answer:** A phase of  $180^\circ$  is equivalent to a phase of  $-180^\circ$ . Replacing  $180^\circ$  with  $-180^\circ$  for  $\mu < 0$  in **Fig. 3-6(c)** makes the phase  $\phi(\mu)$  an odd function of  $\mu$ .

## 3-5 2-D Sampling Theorem

The sampling theorem generalizes directly from 1-D to 2-D using rectangular sampling:

$$f[n,m] = f(n\Delta, m\Delta) = f(n/S, m/S) \quad (3.60)$$

where  $\Delta$  is the **sampling length** (instead of interval) and  $S_x = 1/\Delta$  is the **sampling rate** in samples/meter.

If the spectrum of image  $f(x,y)$  is **bandlimited** to  $B$ —that is,  $F(\mu, \nu) = 0$  outside the square region defined by

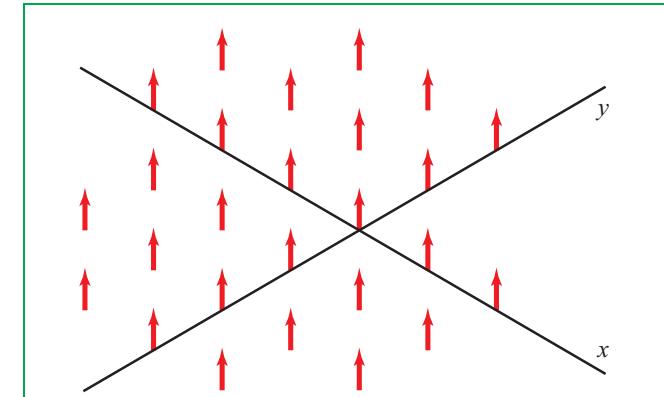
$$\{(\mu, \nu) : 0 \leq |\mu|, |\nu| \leq B\},$$

then the image  $f(x,y)$  can be reconstructed from its samples  $f[m,n]$ , provided the sampling rate is such that  $S > 2B$ . As in 1-D,  $2B$  is called the **Nyquist sampling rate**, although the units are now samples/meter instead of samples/second.

The sampled signal  $x_s(t)$  defined by Eq. (2.43) generalizes directly to the **sampled image**:

$$f_s(x,y) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f(n\Delta, m\Delta) \times [\delta(x - n\Delta) \delta(y - m\Delta)]. \quad (3.61)$$

The term inside the square brackets (product of two impulse trains) is called the **bed of nails** function, because it consists of a 2-D array of impulses, as shown in **Fig. 3-17**.



**Figure 3-17** The “bed of nails” function

$$\sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \delta(x - n\Delta) \delta(y - m\Delta).$$

Conceptually, image  $f(x,y)$  can be reconstructed from its discretized version  $f(n\Delta, m\Delta)$  by applying the 2-D version of the **sinc interpolation formula**. Generalizing Eq. (2.51) to 2-D gives

$$f(x,y) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f(n\Delta, m\Delta) \times \frac{\sin(\pi S(x - n\Delta))}{\pi S(x - n\Delta)} \frac{\sin(\pi S(y - m\Delta))}{\pi S(y - m\Delta)}. \quad (3.62)$$

As noted earlier in Section 2-4.4 in connection with Eq. (2.51), accurate reconstruction using the sinc interpolation formula is not practical because it requires summations over infinite number of samples.

### 3-5.1 Sampling/Reconstruction Examples

The following image examples are designed to illustrate the important role of the Nyquist rate when sampling an **image**  $f(x,y)$  (for storage or digital transmission) and then reconstructing it from its **sampled version**  $f_s(x,y)$ . We will use the term **image reconstruction fidelity** as a qualitative measure of how well the **reconstructed image**  $f_{rec}(x,y)$  resembles the original image  $f(x,y)$ .

Reconstruction of  $f_{rec}(x,y)$  from the sampled image  $f_s(x,y)$  can be accomplished through either of two approaches:

(a) Application of **nearest-neighbor (NN)** interpolation (which is a 2-D version of the 1-D nearest-neighbor interpola-

tion), implemented directly on image  $f_s(x, y)$ .

(b) Transforming image  $f_s(x, y)$  to the frequency domain, applying 2-D lowpass filtering (LPF) to simultaneously preserve the central spectrum of  $f(x, y)$  and remove all copies thereof (generated by the sampling process), and then inverse transforming to the spatial domain.

Both approaches will be demonstrated in the examples that follow, and in each case we will compare an image reconstructed from an image sampled at the Nyquist rate with an aliased image reconstructed from an image sampled at a rate well below the Nyquist rate. In all cases, the following parameters apply:

- **Size** of original (clown) image  $f(x, y)$  and reconstructed image  $f_{\text{rec}}(x, y)$ :  $40 \text{ mm} \times 40 \text{ mm}$
- **Sampling interval**  $\Delta$  (and corresponding sampling rate  $S = 1/\Delta$ ) and number of samples  $N$ :
  - **Nyquist-sampled version**:  $\Delta = 0.2 \text{ mm}$ ,  $S = 5 \text{ samples/mm}$ ,  $N = 200 \times 200$
  - **Sub-Nyquist-sampled version**:  $\Delta = 0.4 \text{ mm}$ ,  $S = 2.5 \text{ samples/mm}$ ,  $N = 100 \times 100$
- **Spectrum** of original image  $f(x, y)$  is bandlimited to  $B = 2.5 \text{ cycles/mm}$
- **Display**
  - Images  $f(x, y)$ ,  $f_s(x, y)$ ,  $f_{\text{rec}}(x, y)$ : **Linear scale**
  - Image magnitude spectra: **Logarithmic scale** (for easier viewing; magnitude spectra extend over a wide range).

### Reconstruction Example 1: Image Sampled at Nyquist Rate

Our first step is to create a bandlimited image  $f(x, y)$ . This was done by transforming an available clown image to the spatial frequency domain and then applying a lowpass filter with a cutoff frequency of 2.5 cycles/mm. The resultant image and its corresponding spectrum are displayed in [Figs. 3-18\(a\)](#) and [\(b\)](#), respectively.

#### A. LPF Reconstruction

Given that image  $f(x, y)$  is bandlimited to  $B = 2.5 \text{ cycles/mm}$ , the Nyquist rate is  $2B = 5 \text{ cycles/mm}$ . [Figure 3-18\(c\)](#) displays  $f_s(x, y)$ , a sampled version of  $f(x, y)$ , sampled at the Nyquist rate, so it should be possible to reconstruct the original image with good fidelity. The spectrum of  $f_s(x, y)$  is displayed

in part (d). The spectrum of the sampled image contains the spectrum of the original image (namely, the spectrum in [Fig. 3-18\(b\)](#)), plus periodic copies spaced at an interval  $S$  along both directions in the spatial frequency domain. To preserve the central spectrum and simultaneously remove all of the copies, a lowpass filter is applied in step (f) of [Fig. 3-18](#). Finally, application of the 2-D inverse Fourier transform to the spectrum in part (f) leads to the reconstructed image  $f_{\text{rec}}(x, y)$  in part (e). We note that the process yields a reconstructed image with high-fidelity resemblance to the original image  $f(x, y)$ .

#### B. NN Reconstruction

[Figure 3-19](#) displays image  $f(x, y)$ , sampled image  $f_s(x, y)$ , and the NN reconstructed image  $f(x, y)$ . The last step was realized using a 2-D version of the nearest-neighbor interpolation technique described in Section 2-4.4D. NN reconstruction provides image

$$\hat{f}(x, y) = f_s(x, y) * * \text{rect}(x/\Delta) \text{ rect}(y/\Delta), \quad (3.63)$$

which is a 2-D convolution of the 2-D sampled image  $f_s(x, y)$  with a box function. The spectrum of the NN interpolated signal is

$$\hat{\mathbf{F}}(\mu, v) = \mathbf{F}(\mu, v) \frac{\sin(\pi \Delta \mu)}{\pi \mu} \frac{\sin(\pi \Delta v)}{\pi v}. \quad (3.64)$$

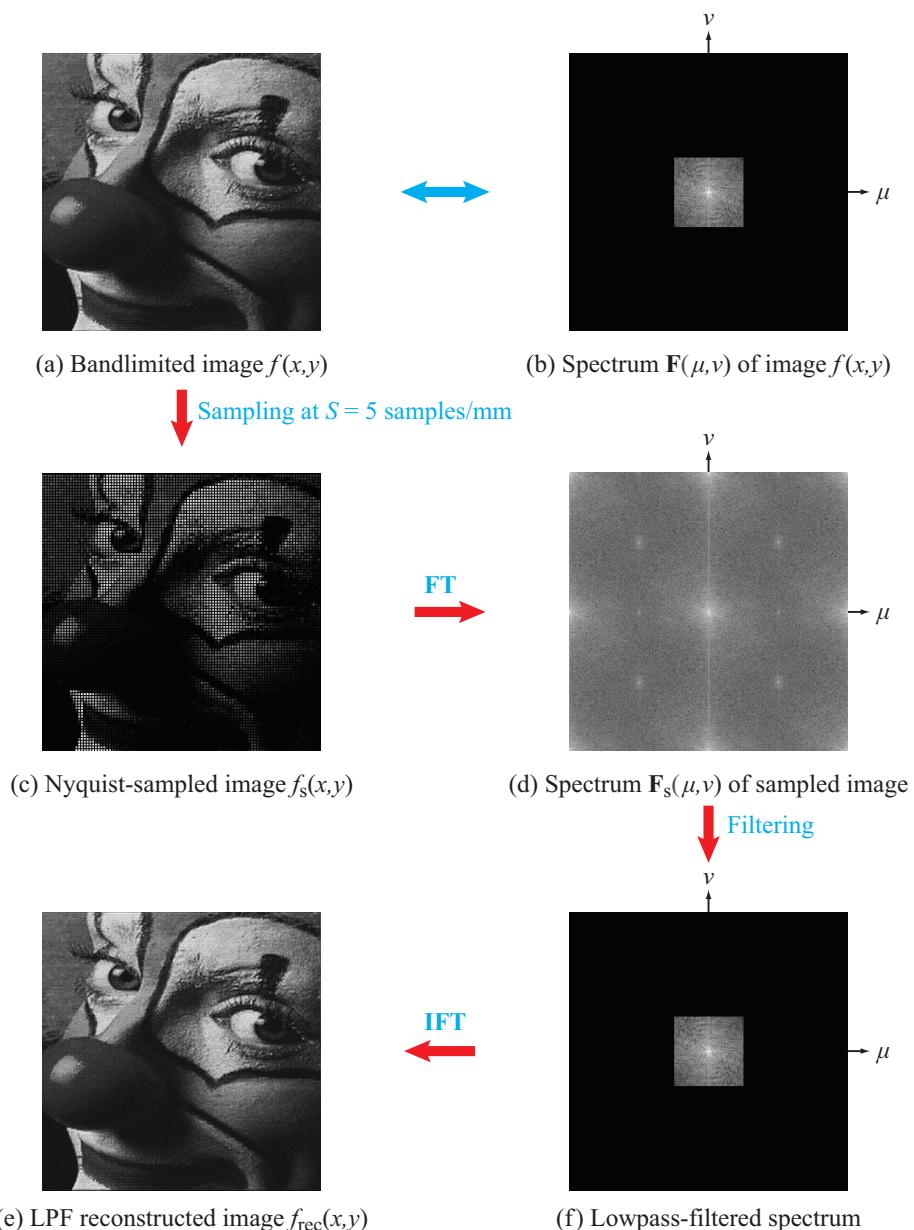
As in 1-D, the zero crossings of the 2-D sinc functions coincide with the centers of the copies of  $\mathbf{F}(\mu, v)$  induced by sampling. Consequently, the 2-D sinc functions act like lowpass filters along  $\mu$  and  $v$ , serving to eliminate the copies almost completely.

Comparison of the NN-interpolated image in [Fig. 3-19\(c\)](#) with the original image in part (a) of the figure leads to the conclusion that the NN technique works quite well for images sampled at or above the Nyquist rate.

### Reconstruction Example 2: Image Sampled below the Nyquist Rate

#### A. LPF Reconstruction

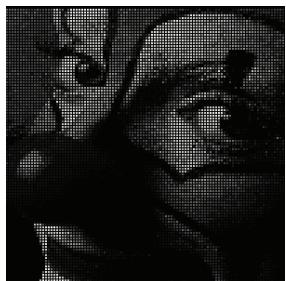
The sequence in this example ([Fig. 3-20](#)) is identical with that described earlier in Example 1A, except for one very important difference: in the present case the sampling rate is  $S = 2.5 \text{ cycles/mm}$ , which is one-half of the Nyquist rate. Consequently, the final reconstructed image in [Fig. 3-20\(e\)](#) bears a poor resemblance to the original image in part (a) of the figure.



**Figure 3-18** Reconstruction Example 1A: After sampling image  $f(x,y)$  in (a) to generate  $f_s(x,y)$  in (c), the sampled image is Fourier transformed [(c) to (d)], then lowpass-filtered [(d) to (f)] to remove copies of the central spectrum) and inverse Fourier transform [(f) to (e)] to generate the reconstructed image  $f_{\text{rec}}(x,y)$ . All spectra are displayed in log scale.

(a) Bandlimited image  $f(x,y)$ 

↓  
Sampled at  
 $S = 5$  samples/mm

(b) Nyquist-sampled image  $f_s(x,y)$ 

↓  
NN

(c) NN reconstructed image  $f_{\text{rec}}(x,y)$ 

**Figure 3-19** Reconstruction Example 1B: Nearest-neighbor (NN) interpolation for Nyquist-sampled image. Sampling rate is  $S = 5$  samples/mm.

## B. NN Reconstruction

The sequence in **Fig. 3-21** parallels the sequence in **Fig. 3-19**, except that in the present case we are working with the sub-Nyquist sampled image. As expected, the NN interpolation technique generates a poor-fidelity reconstruction, just like the LPF reconstructed version.

### 3-5.2 Hexagonal Sampling

The transformation defined by Eq. (3.48a) converts image  $f(r, \theta)$ —expressed in terms of spatial polar coordinates  $(r, \theta)$ —to its spectrum  $\mathbf{F}(\rho, \phi)$ —expressed in terms of **radial frequency**  $\rho$  and associated azimuth angle  $\phi$ . Spectrum  $\mathbf{F}(\rho, \phi)$  is said to be **radially bandlimited** to radial frequency  $\rho_0$  if:

$$\mathbf{F}(\rho, \phi) = 0 \quad \text{for } \rho > \rho_0.$$

If  $f(r, \theta)$  is sampled along a rectangular grid—the same as when sampling  $f(x, y)$  in rectangular coordinates—at a sampling spacing  $\Delta_{\text{rect}}$  (**Fig. 3-22(a)**) and corresponding sampling rate  $S = 1/\Delta_{\text{rect}}$  such that  $S \geq 2\rho_0$  (to satisfy the Nyquist rate), then the spectrum  $\mathbf{F}_s(\rho, \phi)$  of the sampled image  $f_s(r, \theta)$  would consist of a central disk of radius  $\rho_0$ , as shown in **Fig. 3-22(b)**, plus additional copies at a spacing  $S$  along both the  $\mu$  and  $\nu$  directions. The term commonly used to describe the sampling in  $(x, y)$  space is **tiling**; the image space in **Fig. 3-22(a)** is tiled with square pixels.

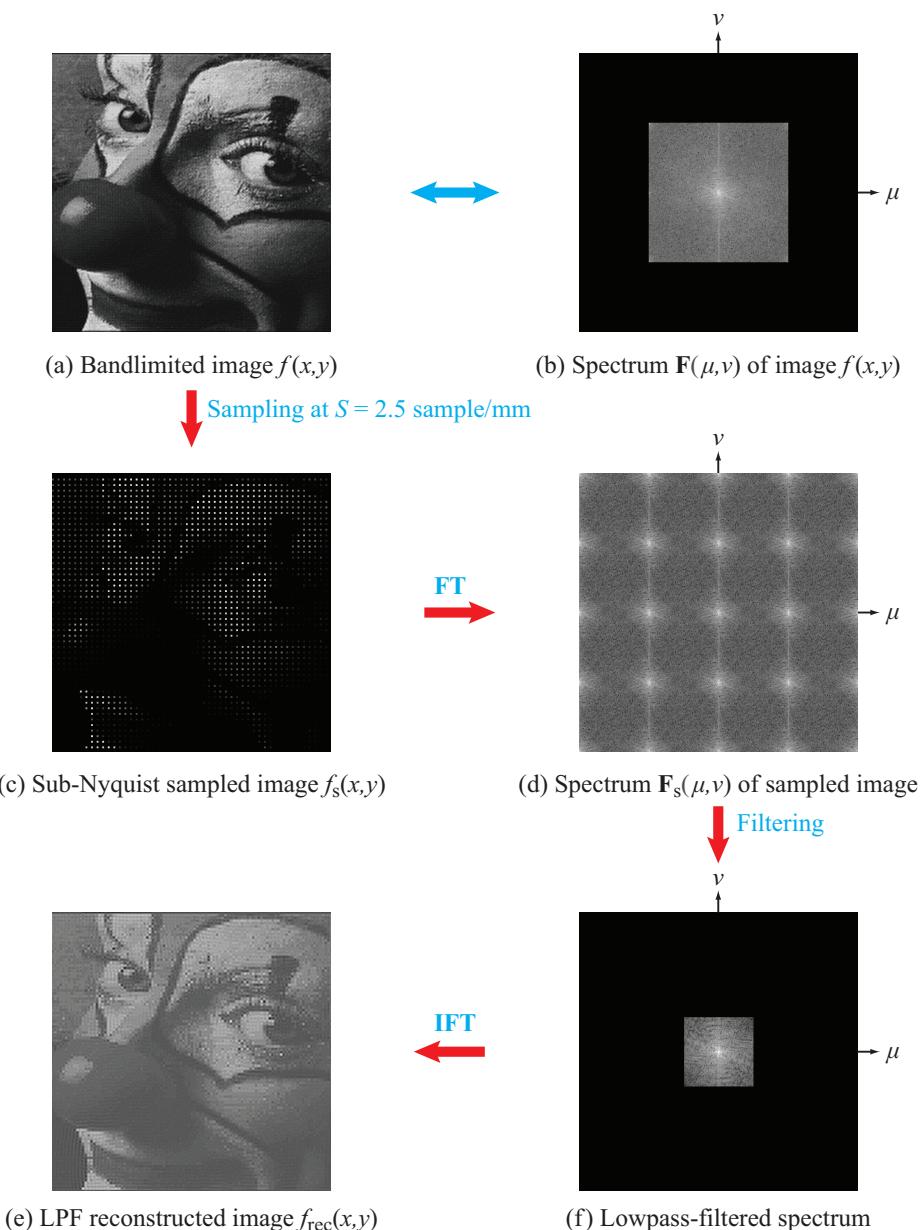
Square tiling is not the only type of tiling used to sample 2-D images. A more efficient arrangement in terms of data rate (or total number of samples per image) is to tile the image space using hexagons instead of squares. Such an arrangement is shown in **Fig. 3-23(a)** and is called **hexagonal sampling**. The image space is tiled with hexagons. The spacing along  $y$  is unchanged (**Fig. 3-23(a)**), but the spacing along  $x$  has changed to

$$\Delta_{\text{hex}} = \frac{2}{\sqrt{3}} \Delta_{\text{rect}} = 1.15\Delta_{\text{rect}}.$$

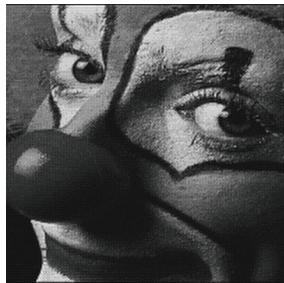
The modest wider spacing along  $x$  translates into fewer samples needed to tile the image, and more efficient utilization of the spatial frequency space (**Fig. 3-23(b)**).

Hexagonal sampling is integral to how the human vision system functions, in part because our photoreceptors are arranged along a hexagonal lattice. The same is true for other mammals as well.

Reconstruction of  $f(r, \theta)$  from its hexagonal samples entails the application of a radial lowpass filter with cutoff frequency  $\rho_0$  to  $\mathbf{F}_s(\rho, \phi)$ , followed by an inverse Fourier transformation (using Eq. (3.48b)) to the  $(r, \theta)$  domain. A clown image ex-



**Figure 3-20** Reconstruction Example 2A: Image  $f(x,y)$  is sampled at half the Nyquist rate ( $S = 2.5$  sample/mm compared with  $2B = 5$  samples/mm). Consequently, the reconstructed image in (e) bears a poor resemblance to the original image in (a). All spectra are displayed in log scale.

(a) Bandlimited image  $f(x,y)$ 

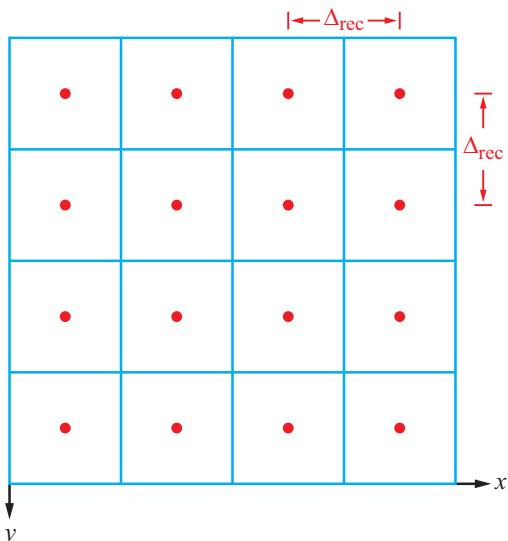
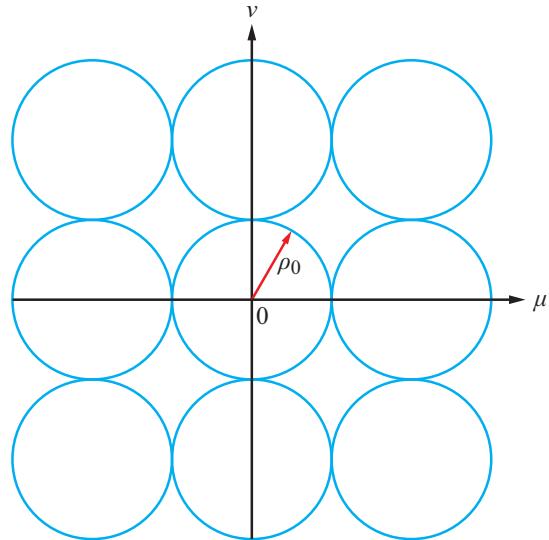
↓  
Sampled at  
 $S = 2.5$  samples/mm

(b) Sub-Nyquist sampled image  $f_s(x,y)$ 

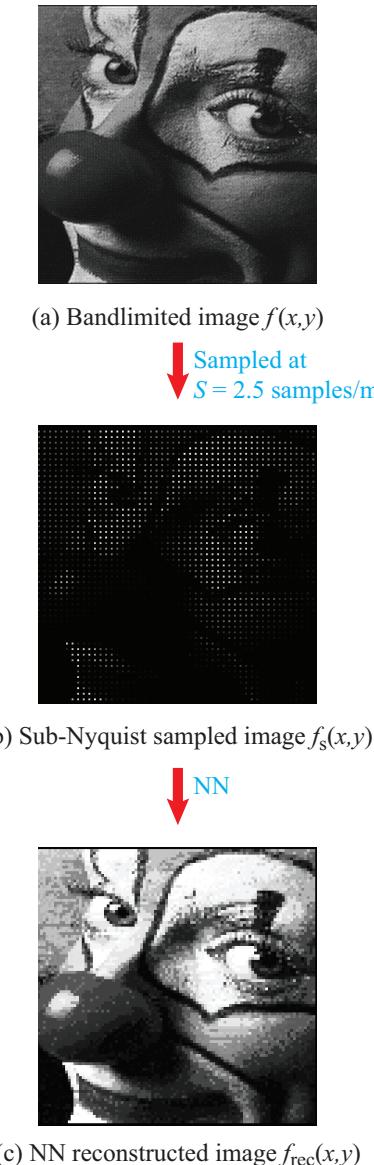
↓ NN

(c) NN reconstructed image  $f_{\text{rec}}(x,y)$ 

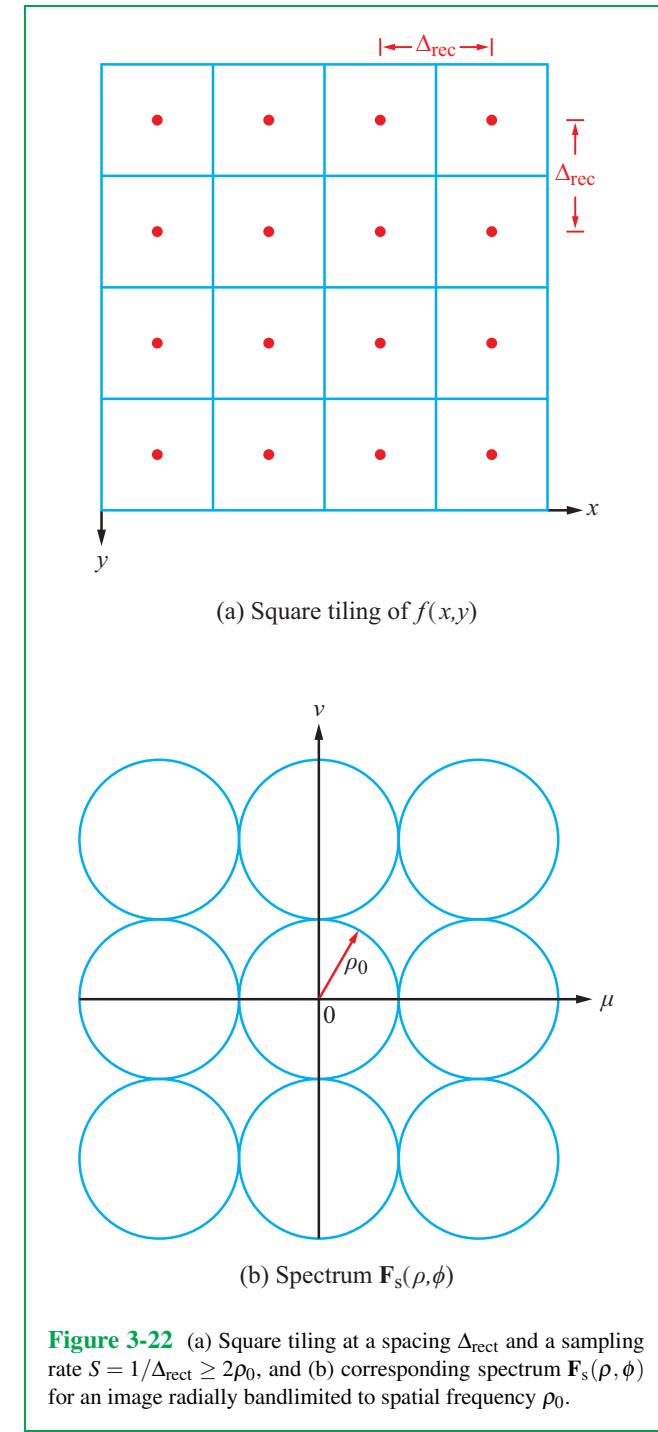
**Figure 3-21** Reconstruction Example 2B: Nearest-neighbor interpolation for sub-Nyquist-sampled image.

(a) Square tiling of  $f(x,y)$ (b) Spectrum  $F_s(\rho, \phi)$ 

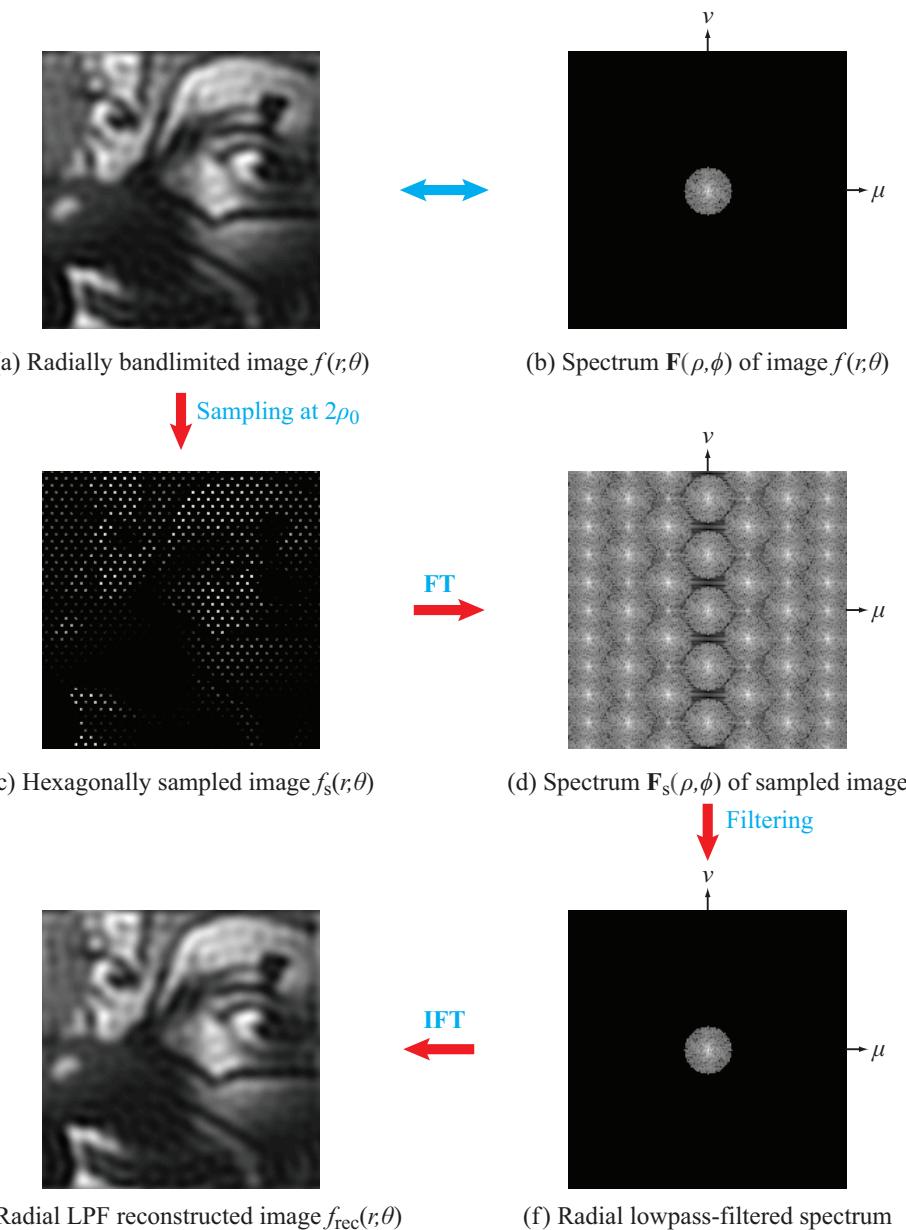
**Figure 3-22** (a) Square tiling at a spacing  $\Delta_{\text{rect}}$  and a sampling rate  $S = 1/\Delta_{\text{rect}} \geq 2\rho_0$ , and (b) corresponding spectrum  $F_s(\rho, \phi)$  for an image radially bandlimited to spatial frequency  $\rho_0$ .



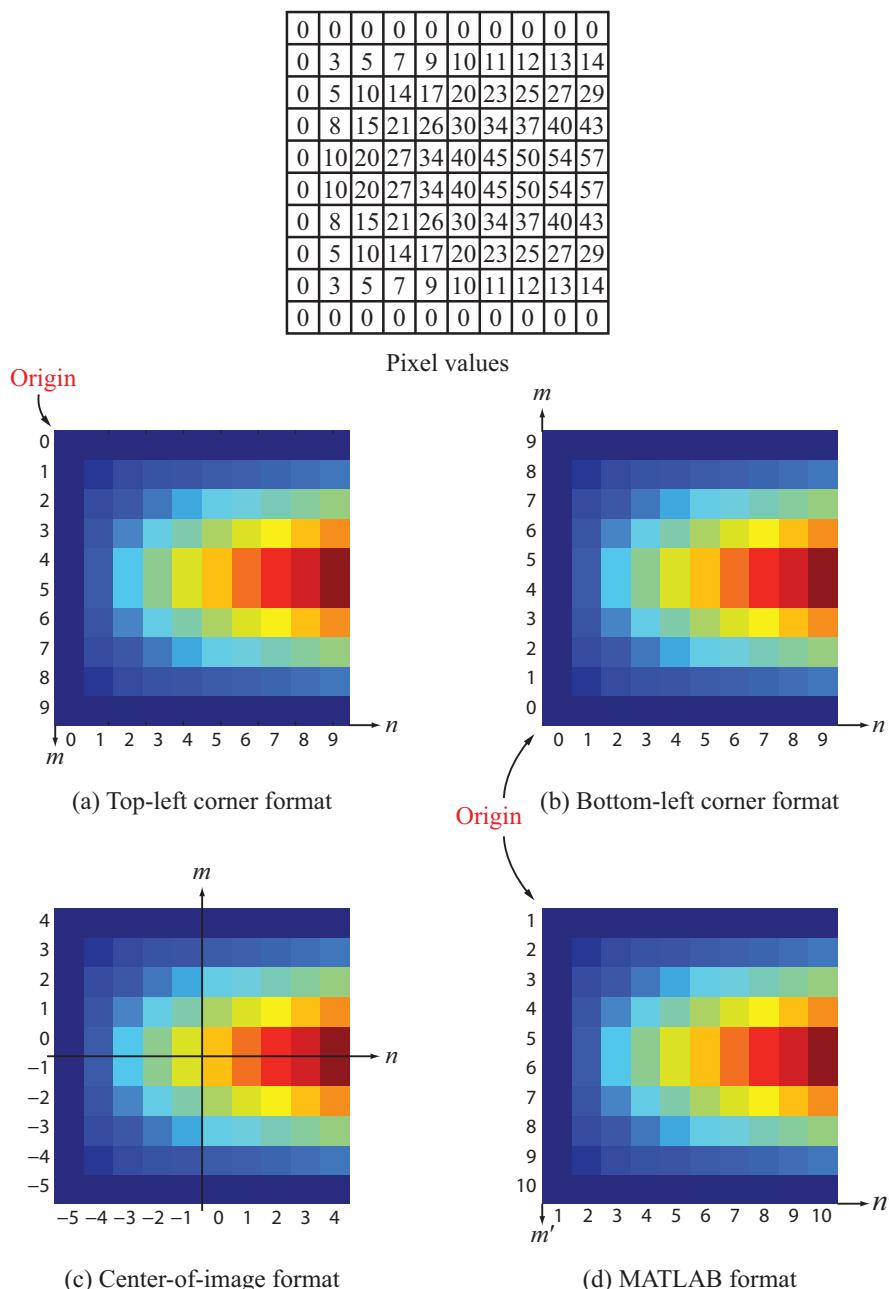
**Figure 3-21** Reconstruction Example 2B: Nearest-neighbor interpolation for sub-Nyquist-sampled image.



**Figure 3-22** (a) Square tiling at a spacing  $\Delta_{\text{rect}}$  and a sampling rate  $S = 1/\Delta_{\text{rect}} \geq 2\rho_0$ , and (b) corresponding spectrum  $\mathbf{F}_s(\rho, \phi)$  for an image radially bandlimited to spatial frequency  $\rho_0$ .



**Figure 3-24** Hexagonal sampling and reconstruction example.



**Figure 3-25** The four color images are identical in pixel values, but they use different formats for the location of the origin and for coordinate directions for image  $f[n, m]$ . The MATLAB format in (d) represents  $\mathbb{X}(m', n')$ .

through (d), we display the same color maps corresponding to the pixel array, except that the  $[n, m]$  coordinates are defined differently, namely:

#### Top-left corner format

**Figure 3-25(a):**  $[n, m]$  starts at  $[0, 0]$  and both integers extend to 9, the origin is located at the upper left-hand corner,  $m$  increases downward, and  $n$  increases to the right. For a  $(M \times N)$  image,  $f[n, m]$  is defined as

$$\{f[n, m], 0 \leq n \leq N-1, 0 \leq m \leq M-1\} \quad (3.66a)$$

or, equivalently,

$$f[n, m] = \begin{bmatrix} f[0, 0] & f[1, 0] \dots f[N-1, 0] \\ f[0, 1] & f[1, 1] \dots f[N-1, 1] \\ \vdots & \vdots \\ f[0, M-1] & f[1, M-1] \dots f[N-1, M-1] \end{bmatrix}. \quad (3.66b)$$

#### Bottom-left corner format

**Figure 3-25(b):**  $[n, m]$  starts at  $[0, 0]$  and both integers extend to 9, the origin is located at the bottom left-hand corner,  $m$  increases upward, and  $n$  increases to the right. This is a vertically flipped version of the image in **Fig. 3-25(a)**, and  $f[n, m]$  has the same definition given by Eq. (3.66a).

#### Center-of-image format

**Figure 3-25(c):** The axes directions are the same as in the image of **Fig. 3-25(b)**, except that the origin of the coordinate system is now located at the center of the image. The ranges of  $n$  and  $m$  depend on whether  $M$  and  $N$  are odd or even integers. If pixel  $[0, 0]$  is to be located in the center in the  $[n, m]$  coordinate system, then the range of  $m$  is

$$-\frac{M-1}{2} \leq m \leq \frac{M-1}{2}, \quad \text{if } M \text{ is odd,} \quad (3.67a)$$

$$-\frac{M}{2} \leq m \leq \frac{M}{2}-1, \quad \text{if } M \text{ is even.} \quad (3.67b)$$

A similar definition applies to index  $n$ .

#### MATLAB format

In MATLAB, an  $(M \times N)$  image is defined as

$$\{X(m', n'), 1 \leq m' \leq M, 1 \leq n' \leq N\}. \quad (3.68)$$

MATLAB uses the top-left corner format, except that its indices  $n'$  and  $m'$  start at 1 instead of 0. Thus, the top-left corner is  $(1, 1)$  instead of  $[0, 0]$ . Also,  $m'$ , the first index in  $X(m', n')$ , represents the vertical axis and the second index,  $n'$ , represents the horizontal axis, which is the reverse of the index notation represented in  $f[n, m]$ . The two notations are related as follows:

$$f[n, m] = X(m', n'), \quad (3.69a)$$

with

$$m' = m + 1, \quad (3.69b)$$

$$n' = n + 1. \quad (3.69c)$$

#### Common-Image vs. MATLAB Format

The format used by MATLAB to store images is different from the conventional matrix format given in Eq. (3.66b) in two important ways:

(1) Whereas the pixel at the upper left-hand corner of image  $f[n, m]$  is  $f[0, 0]$ , at location  $(0, 0)$ , that pixel is denoted  $X(1, 1)$  in MATLAB.

(2) In  $f[n, m]$ , the value of  $n$  denotes the column that  $f[n, m]$  resides within (relative to the most-left column, which is denoted by  $n = 0$ ), and  $m$  denotes the row that  $f[n, m]$  resides within (relative to the top row, which is denoted by  $m = 0$ ). In MATLAB, the notation is reversed: the first index of  $X(m', n')$  denotes the row and the second index denotes the column. Hence,

$$f[n, m] = X(m', n')$$

with  $m'$  and  $n'$  related to  $n$  and  $m$  by Eq. (3.69). To distinguish between the two formats,  $f[n, m]$  uses square brackets whereas  $X(m', n')$  uses curved brackets.

► From here on forward, the top-left-corner format will be used for images, the center-of-image format will be used for image spectra and PSFs, and the MATLAB format will be used in MATLAB arrays. ◀

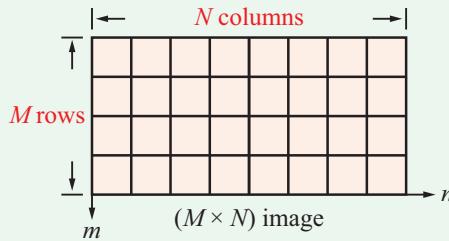
## B. Impulses and Shifts

In 2-D discrete-space, impulse  $\delta[n, m]$  is defined as

$$\delta[n, m] = \delta[n] \delta[m] = \begin{cases} 1 & \text{if } n = m = 0, \\ 0 & \text{otherwise.} \end{cases} \quad (3.70)$$

In upper-left-corner image format, shifting an image  $f[n, m]$  by  $m_0$  downward and  $n_0$  rightward generates image  $f[n - n_0, m - m_0]$ . An example is shown in Fig. 3-26.

**Image Size**



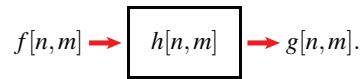
As noted earlier in Chapter 1, the size of an image is denoted by (# of rows  $\times$  # of columns) =  $M \times N$ .

## 3-6.2 Discrete-Space Systems

For discrete-space systems, linearity and shift invariance follow analogously from their continuous-space counterparts. When a **linear shift-invariant (LSI)** system characterized by a point spread function  $h[n, m]$  is subjected to an input  $f[n, m]$ , it generates an output  $g[n, m]$  given by the 2-D convolution of  $f[n, m]$  and  $h[n, m]$ :

$$\begin{aligned} g[n, m] &= h[n, m] * * f[n, m] \\ &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h[i, j] f[n - i, m - j]. \end{aligned} \quad (3.71)$$

Symbolically, the 2-D convolution is represented by



The properties of 1-D convolution are equally applicable in 2-D discrete-space. Convolution of images of sizes  $(L_1 \times L_2)$  and  $(M_1 \times M_2)$  yields an image of size  $(N_1 \times N_2)$ , where

$$N_1 = L_1 + M_1 - 1, \quad (3.72a)$$

$$N_2 = L_2 + M_2 - 1. \quad (3.72b)$$

The 2-D convolution process is illustrated next through a simple example.

### Example 3-1: 2-D Convolution

Compute the 2-D convolution

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * * \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}.$$

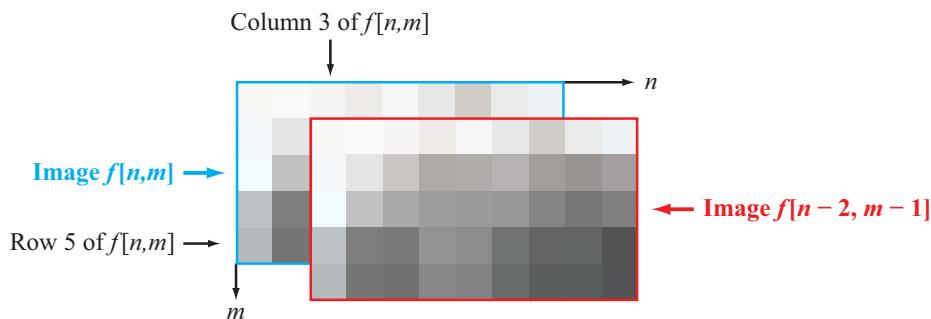
**Solution:** By using entries in the first image as weights, we have

$$\begin{aligned} 1 \begin{bmatrix} 5 & 6 & 0 \\ 7 & 8 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2 \begin{bmatrix} 0 & 5 & 6 \\ 0 & 7 & 8 \\ 0 & 0 & 0 \end{bmatrix} + 3 \begin{bmatrix} 0 & 0 & 0 \\ 5 & 6 & 0 \\ 7 & 8 & 0 \end{bmatrix} + 4 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 6 \\ 0 & 7 & 8 \end{bmatrix} \\ = \begin{bmatrix} 5 & 16 & 12 \\ 22 & 60 & 40 \\ 21 & 52 & 32 \end{bmatrix}. \end{aligned}$$

**Concept Question 3-6:** Why do we bother studying discrete-space images and systems, when almost all real-world systems and images are defined in continuous space?

**Exercise 3-11:** If  $g[n, m] = h[n, m] * * f[n, m]$ , what is  $4h[n, m] * * f[n - 3, m - 2]$  in terms of  $g[n, m]$ ?

**Answer:**  $4g[n - 3, m - 2]$ , using the shift and scaling properties of 1-D convolutions.



**Figure 3-26** Image  $f[n - 2, m - 1]$  is image  $f[n, m]$  shifted down by 1 and to the right by 2.

**Exercise 3-12:** Compute  $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \ast \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ .

**Answer:**  $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ .

Conjugate symmetry for real-valued images  $f[m, n]$  implies that

$$\mathbf{F}^*(\Omega_1, \Omega_2) = \mathbf{F}(-\Omega_1, -\Omega_2). \quad (3.74)$$

As in the 2-D continuous-space Fourier transform,  $\mathbf{F}(\Omega_1, \Omega_2)$  must be reflected across both spatial frequency axes to produce its complex conjugate.

The DSFT is doubly periodic in  $(\Omega_1, \Omega_2)$  with periods  $2\pi$  along each axis, as demonstrated by Example 3-2.

## 3-7 2-D Discrete-Space Fourier Transform (DSFT)

The 2-D **discrete-space Fourier transform (DSFT)** is obtained via direct generalization of the 1-D DTFT (Section 2-6) to 2-D. The DSFT consists of a DTFT applied first along  $m$  and then along  $n$ , or vice versa. By extending the 1-D DTFT definition given by Eq. (2.73a) (as well as the properties listed in Table 2-7) to 2-D, we obtain the following definition for the DSFT  $\mathbf{F}(\Omega_1, \Omega_2)$  and its inverse  $f[n, m]$ :

$$\mathbf{F}(\Omega_1, \Omega_2) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n, m] e^{-j(\Omega_1 n + \Omega_2 m)}, \quad (3.73a)$$

$$f[n, m] = \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \mathbf{F}(\Omega_1, \Omega_2) e^{j(\Omega_1 n + \Omega_2 m)} d\Omega_1 d\Omega_2. \quad (3.73b)$$

The properties of the DSFT are direct 2-D generalizations of the properties of the DTFT, and discrete-time generalizations of the properties of the 2-D continuous-space Fourier transform.

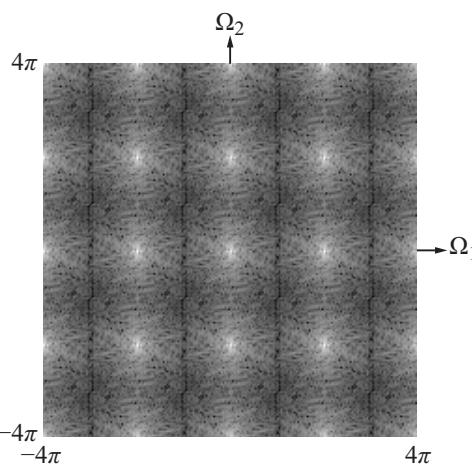
► The spectrum of an image  $f[n, m]$  is its DSFT  $\mathbf{F}(\Omega_1, \Omega_2)$ . The discrete-space frequency response  $\mathbf{H}(\Omega_1, \Omega_2)$  of an LSI system is the DSFT of its point spread function (PSF)  $h[n, m]$ . ◀

### Example 3-2: DSFT of Clown Image

Use MATLAB to obtain the magnitude image of the DSFT of the clown image.

**Solution:** The magnitude part of the DSFT is displayed in Fig. 3-27. As expected, the spectrum is periodic with period  $2\pi$  along both  $\Omega_1$  and  $\Omega_2$ .

**Concept Question 3-7:** What is the DSFT used for? Give three applications.



**Figure 3-27** DSFT magnitude of clown image (log scale).

**Exercise 3-13:** An LSI system has a PSF given by

$$h[n, m] = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Compute its spatial frequency response  $\mathbf{H}(\Omega_1, \Omega_2)$ . Hint:  $h[n, m]$  is separable.

**Answer:** The DTFT was defined in Eq. (2.73). Recognizing that  $h[n, m] = h_1[n] h_1[m]$  with  $h_1[n] = \{1, 2, 1\}$ , the DSFT is the product of two 1-D DTFTs, each of the form

$$\mathbf{H}_1(\Omega) = e^{j\Omega} + 2 + e^{-j\Omega} = 2 + 2\cos(\Omega),$$

and therefore

$$\mathbf{H}(\Omega_1, \Omega_2) = [2 + 2\cos(\Omega_1)][2 + 2\cos(\Omega_2)].$$

## 3-8 2-D Discrete Fourier Transform (2-D DFT)

According to Eq. (3.73), while  $f[n, m]$  is a discrete function, its DSFT  $\mathbf{F}(\Omega_1, \Omega_2)$  is a continuous function of  $\Omega_1$  and  $\Omega_2$ . Numerical computation using the *fast Fourier transform (FFT)* requires an initial step of converting  $\mathbf{F}(\Omega_1, \Omega_2)$  into discrete

format. That conversion is called the *discrete Fourier transform (DFT)*. With the DFT, both  $f[n, m]$  and its 2-D Fourier transform operate in discrete domains. For an  $(M \times N)$  image  $f[n, m]$ , generalizing Eq. (2.89) to 2-D leads to the 2-D DFT of order  $(\mathcal{K}_2 \times \mathcal{K}_1)$ :

$$\mathbf{F}[k_1, k_2] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m] \exp \left[ -j2\pi \left( \frac{nk_1}{\mathcal{K}_1} + \frac{mk_2}{\mathcal{K}_2} \right) \right],$$

$$k_1 = \{0, \dots, \mathcal{K}_1 - 1\}, \quad k_2 = \{0, \dots, \mathcal{K}_2 - 1\}, \quad (3.75)$$

where we have converted  $(\Omega_1, \Omega_2)$  into discrete indices  $(k_1, k_2)$  by setting

$$\Omega_1 = \frac{2\pi}{\mathcal{K}_1} k_1$$

and

$$\Omega_2 = \frac{2\pi}{\mathcal{K}_2} k_2.$$

Array  $\mathbf{F}[k_1, k_2]$  is given by the  $(\mathcal{K}_2 \times \mathcal{K}_1)$  array

$$\mathbf{F}[k_1, k_2] = \begin{bmatrix} \mathbf{F}[0, 0] & \mathbf{F}[1, 0] \dots \mathbf{F}[\mathcal{K}_1 - 1, 0] \\ \mathbf{F}[0, 1] & \mathbf{F}[1, 1] \dots \mathbf{F}[\mathcal{K}_1 - 1, 1] \\ \vdots & \vdots \\ \mathbf{F}[0, \mathcal{K}_2 - 1] & \mathbf{F}[1, \mathcal{K}_2 - 1] \dots \mathbf{F}[\mathcal{K}_1 - 1, \mathcal{K}_2 - 1] \end{bmatrix}. \quad (3.76)$$

Note that the indexing is the same as that used for  $f[n, m]$ . The inverse DFT is

$$f[n, m] = \frac{1}{\mathcal{K}_1 \mathcal{K}_2} \sum_{k_1=0}^{\mathcal{K}_1-1} \sum_{k_2=0}^{\mathcal{K}_2-1} \mathbf{F}[k_1, k_2] \exp \left[ j2\pi \left( \frac{nk_1}{\mathcal{K}_1} + \frac{mk_2}{\mathcal{K}_2} \right) \right],$$

$$n = \{0, \dots, N - 1\}, \quad m = \{0, \dots, M - 1\}. \quad (3.77)$$

► Note that if  $N < \mathcal{K}_1$  and  $M < \mathcal{K}_2$ , then the reconstructed image  $f[n, m] = 0$  for  $N \leq n \leq \mathcal{K}_1 - 1$  and  $M \leq m \leq \mathcal{K}_2 - 1$ .

### 3-8.1 Properties of the 2-D DFT

The 2-D DFT, like the 2-D CSFT and 2-D DSFT, consists of a 1-D transform along either the horizontal or vertical direction, followed by another 1-D transform along the other direction. Accordingly, the 2-D DFT has the following properties listed in **Table 3-3**, which are direct generalizations of the 1-D DFT properties listed earlier in **Table 2-9**.

The cyclic convolution  $h[n] \odot x[n]$  was defined in Eq. (2.104). The 2-D DFT maps 2-D cyclic convolutions to products, and

**Table 3-3 Properties of the  $(\mathcal{K}_2 \times \mathcal{K}_1)$  2-D DFT. In the time-shift and modulation properties,  $(k_1 - k'_1)$  and  $(n - n_0)$  must be reduced mod( $\mathcal{K}_1$ ), and  $(k_2 - k'_2)$  and  $(m - m_0)$  must be reduced mod( $\mathcal{K}_2$ ).**

Selected Properties		
<b>1. Linearity</b>	$\sum c_i f_i[n, m]$	$\leftrightarrow$
<b>2. Shift</b>	$f[(n - n_0), (m - m_0)]$	$\leftrightarrow$
<b>3. Modulation</b>	$e^{j2\pi k'_1 n / \mathcal{K}_1} e^{j2\pi k'_2 m / \mathcal{K}_2} f[n, m]$	$\leftrightarrow$
<b>4. Reversal</b>	$f[(N - n), (M - m)]$	$\leftrightarrow$
<b>5. Convolution</b>	$h[n, m] \odot \odot f[n, m]$	$\leftrightarrow$

#### Special DFT Relationships

**6. Conjugate Symmetry for  $f[n, m]$  real**

$$\mathbf{F}^*[k_1, k_2] = \mathbf{F}[(\mathcal{K}_1 - k_1), (\mathcal{K}_2 - k_2)]$$

**7. Zero spatial frequency**

$$\mathbf{F}[0, 0] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m]$$

**8. Spatial origin**

$$f[0, 0] = \frac{1}{\mathcal{K}_1 \mathcal{K}_2} \sum_{k_1=0}^{\mathcal{K}_1-1} \sum_{k_2=0}^{\mathcal{K}_2-1} \mathbf{F}[k_1, k_2]$$

**9. Rayleigh's theorem**

$$\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} |f[n, m]|^2 = \frac{1}{\mathcal{K}_1 \mathcal{K}_2} \sum_{k_1=0}^{\mathcal{K}_1-1} \sum_{k_2=0}^{\mathcal{K}_2-1} |\mathbf{F}[k_1, k_2]|^2$$

linear 2-D convolutions  $h[n, m] * * f[n, m]$  can be zero-padded to cyclic convolutions, just as in 1-D.

#### 3-8.2 Conjugate Symmetry for the 2-D DFT

**Concept Question 3-8:** Why is the 2-D DFT defined only over a finite region, while the DSFT is defined over all spatial frequency space?

**Exercise 3-14:** Compute an expression for the  $(256 \times 256)$  2-D DFT of  $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ . Use the result of Exercise 3-11.

**Answer:** The 2-D DFT is the DSFT sampled at  $\Omega_i = 2\pi k_i / 256$  for  $i = 1, 2$ . Substituting in the answer to Exercise 3-11 gives

$$\mathbf{X}[k_1, k_2] = \left[ 2 + 2 \cos\left(2\pi \frac{k_1}{256}\right) \right] \left[ 2 + 2 \cos\left(2\pi \frac{k_2}{256}\right) \right], \quad 0 \leq k_1, k_2 \leq 255.$$

Given an  $(M \times N)$  image  $f[n, m]$ , the expression given by Eq. (3.75) allows us to compute the 2-D DFT of  $f[n, m]$  for any order  $(\mathcal{K}_2 \times \mathcal{K}_1)$ . If  $f[n, m]$  is real, then **conjugate symmetry** holds:

$$\mathbf{F}^*[k_1, k_2] = \mathbf{F}[\mathcal{K}_1 - k_1, \mathcal{K}_2 - k_2], \quad (3.78) \\ 1 \leq k_1 \leq \mathcal{K}_1 - 1; \quad 1 \leq k_2 \leq \mathcal{K}_2 - 1.$$

[Compare this statement with the conjugate symmetry of the 1-D DFT  $\mathbf{X}[k]$  of a real-valued signal  $x[n]$ , as given by Eq. (2.98).]

The conjugate-symmetry relation given by Eq. (3.78) states that an array element  $\mathbf{F}[k_1, k_2]$  is equal to the complex conjugate of array element  $\mathbf{F}[\mathcal{K}_1 - k_1, \mathcal{K}_2 - k_2]$ , and vice versa. To demonstrate the validity of conjugate symmetry, we start by rewriting Eq. (3.75) with  $k_1$  and  $k_2$  replaced with  $(\mathcal{K}_1 - k_1)$  and

$(\mathcal{K}_2 - k_2)$ , respectively:

$$\begin{aligned} \mathbf{F}[\mathcal{K}_1 - k_1, \mathcal{K}_2 - k_2] &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m] \exp \left[ -j2\pi \left( \frac{n(\mathcal{K}_1 - k_1)}{\mathcal{K}_1} + \frac{m(\mathcal{K}_2 - k_2)}{\mathcal{K}_2} \right) \right] \\ &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m] \exp \left[ j2\pi \left( \frac{nk_1}{\mathcal{K}_1} + \frac{mk_2}{\mathcal{K}_2} \right) \right] \\ &\quad \times \exp \left[ -j2\pi \left( \frac{n\mathcal{K}_1}{\mathcal{K}_1} + \frac{m\mathcal{K}_2}{\mathcal{K}_2} \right) \right] \\ &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m] \exp \left[ j2\pi \left( \frac{nk_1}{\mathcal{K}_1} + \frac{mk_2}{\mathcal{K}_2} \right) \right] \\ &\quad \times e^{-j2\pi(n+m)} \\ &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m] \exp \left[ j2\pi \left( \frac{nk_1}{\mathcal{K}_1} + \frac{mk_2}{\mathcal{K}_2} \right) \right], \end{aligned} \quad (3.79)$$

where we used  $e^{-j2\pi(n+m)} = 1$  because  $n$  and  $m$  are integers. The expression on the right-hand side of Eq. (3.79) is identical to the expression for  $\mathbf{F}[k_1, k_2]$  given by Eq. (3.75) except for the minus sign ahead of  $j$ . Hence, for a real-valued image  $f[n, m]$ ,

$$\begin{aligned} \mathbf{F}^*[k_1, k_2] &= \mathbf{F}[\mathcal{K}_1 - k_1, \mathcal{K}_2 - k_2], \\ 1 \leq k_1 \leq \mathcal{K}_1 - 1; \quad 1 \leq k_2 \leq \mathcal{K}_2 - 1, \end{aligned} \quad (3.80)$$

where  $(\mathcal{K}_2 \times \mathcal{K}_1)$  is the order of the 2-D DFT.

### 3-8.3 Special Cases

#### A. $f[n, m]$ is real

If  $f[n, m]$  is a real-valued image, the following special cases hold:

- (1)  $\mathbf{F}[0, 0] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m]$  is real-valued.
- (2)  $\mathbf{F}[0, k_2] = \sum_{m=0}^{M-1} (\sum_{n=0}^{N-1} f[n, m]) e^{-j(2\pi/\mathcal{K}_2)mk_2}$ , which is the  $\mathcal{K}_2$ -point 1-D DFT of  $\sum_{n=0}^{N-1} f[n, m]$ .
- (3)  $\mathbf{F}[k_1, 0] = \sum_{n=0}^{N-1} (\sum_{m=0}^{M-1} f[n, m]) e^{-j(2\pi/\mathcal{K}_1)nk_1}$ , which is the  $\mathcal{K}_1$ -point 1-D DFT of  $\sum_{m=0}^{M-1} f[n, m]$ .

#### B. $f[n, m]$ is real and $\mathcal{K}_1$ and $\mathcal{K}_2$ are even

If also  $\mathcal{K}_1$  and  $\mathcal{K}_2$  are even, then the following relations apply:

(4)  $\mathbf{F}[\mathcal{K}_1/2, k_2] = \sum_{m=0}^{M-1} (\sum_{n=0}^{N-1} f[n, m] (-1)^n) e^{-j(2\pi/\mathcal{K}_2)mk_2}$ , which is the  $\mathcal{K}_2$ -point 1-D DFT of  $\sum_{n=0}^{N-1} f[n, m] (-1)^n$  because  $e^{-j(2\pi/\mathcal{K}_1)n(\mathcal{K}_1/2)} = e^{j\pi n} = (-1)^n$ .

(5)  $\mathbf{F}[k_1, \mathcal{K}_2/2] = \sum_{n=0}^{N-1} (\sum_{m=0}^{M-1} f[n, m] (-1)^m) e^{-j(2\pi/\mathcal{K}_1)nk_1}$ , which is the  $\mathcal{K}_1$ -point 1-D DFT of  $\sum_{m=0}^{M-1} f[n, m] (-1)^m$  because  $e^{-j(2\pi/\mathcal{K}_2)m(\mathcal{K}_2/2)} = e^{j\pi m} = (-1)^m$ .

(6)  $\mathbf{F}[\mathcal{K}_1/2, \mathcal{K}_2/2] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m] (-1)^{m+n}$ .

### 3-9 Computation of the 2-D DFT Using MATLAB

We remind the reader that the notation used in this book represents images  $f[n, m]$  defined in Cartesian coordinates, with the origin at the upper left corner, the first element  $n$  of the coordinates  $[n, m]$  increasing horizontally rightward from the origin, and the second element  $m$  of the coordinates  $[n, m]$  increasing vertically *downward* from the origin. To illustrate with an example, let us consider the  $(3 \times 3)$  image given by

$$f[n, m] = \begin{bmatrix} f[0, 0] & f[1, 0] & f[2, 0] \\ f[0, 1] & f[1, 1] & f[2, 1] \\ f[0, 2] & f[1, 2] & f[2, 2] \end{bmatrix} = \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix}. \quad (3.81)$$

When stored in MATLAB as array  $\mathbf{X}(m', n')$ , the content remains the same, but the indices swap roles and their values start at  $(1, 1)$ :

$$\mathbf{X}(m', n') = \begin{bmatrix} \mathbf{X}(1, 1) & \mathbf{X}(1, 2) & \mathbf{X}(1, 3) \\ \mathbf{X}(2, 1) & \mathbf{X}(2, 2) & \mathbf{X}(2, 3) \\ \mathbf{X}(3, 1) & \mathbf{X}(3, 2) & \mathbf{X}(3, 3) \end{bmatrix} = \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix}. \quad (3.82)$$

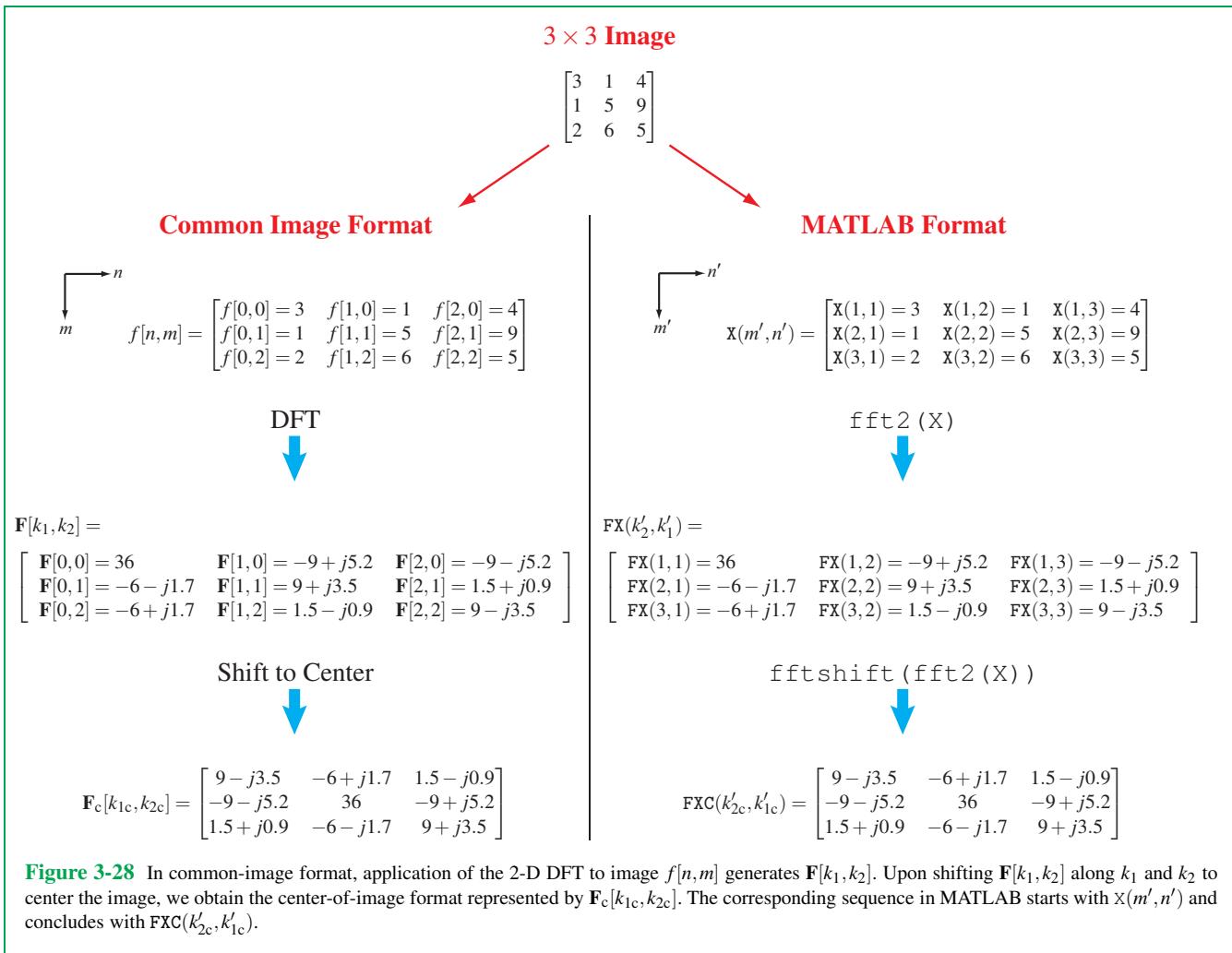
Arrays  $f[n, m]$  and  $\mathbf{X}(m', n')$  are displayed in Fig. 3-28.

Application of Eq. (3.75) with  $N = M = 3$  and  $\mathcal{K}_1 = \mathcal{K}_2 = 3$  to the  $3 \times 3$  image defined by Eq. (3.81) leads to

$$\mathbf{F}[k_1, k_2] = \begin{bmatrix} 36 & -9 + j5.2 & -9 - j5.2 \\ -6 - j1.7 & 9 + j3.5 & 1.5 + j0.9 \\ -6 + j1.7 & 1.5 - j0.9 & 9 - j3.5 \end{bmatrix}. \quad (3.83)$$

► In MATLAB, the command `FX=fft2(X, M, N)` computes the  $(M \times N)$  2-D DFT of array  $\mathbf{X}$  and stores it in array  $\mathbf{FX}$ .

The corresponding array in MATLAB, designated `FX(k'_2, k'_1)`



and displayed in [Fig. 3-28](#), has the same content but with MATLAB indices  $(k'_2, k'_1)$ . Also,  $k'_2$  increases downward and  $k'_1$  increases horizontally. The relationships between MATLAB indices  $(k'_2, k'_1)$  and common-image format indices  $(k_1, k_2)$  are identical in form to those given by Eq. (3.69), namely

$$k'_2 = k_2 + 1, \quad (3.84)$$

$$k'_1 = k_1 + 1. \quad (3.85)$$

### 3-9.1 Center-of-Image Format

In some applications, it is more convenient to work with the 2-D DFT array when arranged in a center-of-image format ([Fig. 3-25\(c\)](#) but with the vertical axis pointing downward) than in the top-left corner format. To convert array  $\mathbf{F}[k_1, k_2]$  to a center-of-image format, we need to shift the array elements to the right and downward by an appropriate number of steps so as to locate  $\mathbf{F}[0, 0]$  in the center of the array. If we denote the 2-D

DFT in the center-of-image format as  $\mathbf{F}_c[k_{1c}, k_{2c}]$ , then its index  $k_{1c}$  extends over the range

$$-\left(\frac{\mathcal{K}_i - 1}{2}\right) \leq k_{1c} \leq \left(\frac{\mathcal{K}_i - 1}{2}\right), \quad \text{for } \mathcal{K}_i = \text{odd}, \quad (3.86)$$

and

$$-\frac{\mathcal{K}_i}{2} \leq k_{1c} \leq \frac{\mathcal{K}_i}{2} - 1, \quad \text{for } \mathcal{K}_i = \text{even}. \quad (3.87)$$

To obtain  $\mathbf{F}_c[k_{1c}, k_{2c}]$  from  $\mathbf{F}[k_1, k_2]$  for the array given by Eq. (3.83), we *circularly shift* the array by one unit to the right and one unit downward, which yields

$$\begin{aligned} \mathbf{F}_c[k_{1c}, k_{2c}] = & \\ \begin{bmatrix} \mathbf{F}_c[-1, 1] = 9 - j3.5 & \mathbf{F}_c[0, 1] = -6 + j1.7 & \mathbf{F}_c[1, 1] = 1.5 - j0.9 \\ \mathbf{F}_c[-1, 0] = -9 - j5.2 & \mathbf{F}_c[0, 0] = 36 & \mathbf{F}_c[1, 0] = -9 + j5.2 \\ \mathbf{F}_c[-1, -1] = 1.5 + j0.9 & \mathbf{F}_c[0, -1] = -6 - j1.7 & \mathbf{F}_c[1, -1] = 9 + j3.5 \end{bmatrix}. \end{aligned} \quad (3.88)$$

► In MATLAB, the command

`FXC=fftshift(fft2(FX))`

shifts array  $\mathbf{F}$  to center-image format and stores it in array  $\mathbf{FXC}$ . ◀

In the general case for any integers  $\mathcal{K}_1$  and  $\mathcal{K}_2$ , transforming the 2-D DFT  $\mathbf{F}[k_1, k_2]$  into the center-of-image format  $\mathbf{F}_c[k_{1c}, k_{2c}]$  entails the following recipe:

For

$$\mathcal{K}'_i = \begin{cases} \mathcal{K}_i/2 - 1 & \text{if } \mathcal{K}_i \text{ is even,} \\ (\mathcal{K}_i - 1)/2 & \text{if } \mathcal{K}_i \text{ is odd,} \end{cases} \quad (3.89)$$

and  $0 \leq k_{1c}, k_{2c} \leq \mathcal{K}'_i$ :

(a) First Quadrant

$$\mathbf{F}_c[k_{1c}, k_{2c}] = \mathbf{F}[k_1, k_2], \quad (3.90a)$$

with  $k_1 = k_{1c}$  and  $k_2 = k_{2c}$ ,

(b) Second Quadrant

$$\mathbf{F}_c[-k_{1c}, k_{2c}] = \mathbf{F}[k_1, k_2], \quad (3.90b)$$

with  $k_1 = \mathcal{K}_1 - k_{1c}$ ,  $k_2 = k_{2c}$ ,

(c) Third Quadrant

$$\mathbf{F}_c[-k_{1c}, -k_{2c}] = \mathbf{F}[k_1, k_2], \quad (3.90c)$$

with  $k_1 = \mathcal{K}_1 - k_{1c}$ ,  $k_2 = \mathcal{K}_2 - k_{2c}$ ,

(d) Fourth Quadrant

$$\mathbf{F}_c[k_{1c}, -k_{2c}] = \mathbf{F}[k_1, k_2], \quad (3.90d)$$

with  $k_1 = k_{1c}$ ,  $k_2 = \mathcal{K}_2 - k_{2c}$ .

To demonstrate the recipe, we use a numerical example with  $\mathcal{K}_1 = \mathcal{K}_2 = 3$ , and again for  $\mathcal{K}_1 = \mathcal{K}_2 = 4$  because the recipe is different for odd and even integers.

## 3-9.2 Odd and Even Image Examples

### A. $N = M$ and $\mathcal{K}_1 = \mathcal{K}_2 = \text{odd}$

The  $(3 \times 3)$  image shown in Fig. 3-28 provides an example of an  $(M \times M)$  image with  $M$  being an odd integer. As noted earlier, when the 2-D DFT is displayed in the center-of-image format, the conjugate symmetry about the center of the array becomes readily apparent.

### A. $N = M$ and $\mathcal{K}_1 = \mathcal{K}_2 = \text{even}$

Let us consider the  $(4 \times 4)$  image

$$f[n, m] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 5 & 3 \\ 3 & 4 & 6 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix}. \quad (3.91)$$

The  $(4 \times 4)$  2-D DFT  $\mathbf{F}[k_1, k_2]$  of  $f[n, m]$ , displayed in the upper-left corner format, is

$$\begin{aligned} \mathbf{F} &= \begin{bmatrix} \mathbf{F}[0, 0] & \mathbf{F}[1, 0] & \mathbf{F}[2, 0] & \mathbf{F}[3, 0] \\ \mathbf{F}[0, 1] & \mathbf{F}[1, 1] & \mathbf{F}[2, 1] & \mathbf{F}[3, 1] \\ \mathbf{F}[0, 2] & \mathbf{F}[1, 2] & \mathbf{F}[2, 2] & \mathbf{F}[3, 2] \\ \mathbf{F}[0, 3] & \mathbf{F}[1, 3] & \mathbf{F}[2, 3] & \mathbf{F}[3, 3] \end{bmatrix} \\ &= \begin{bmatrix} 49 & -6 - j3 & 3 & -6 + j3 \\ -5 - j4 & 2 + j9 & -5 + j2 & j \\ 1 & -4 + j3 & -1 & -4 - j3 \\ -5 + j4 & -j & -5 - j2 & 2 - j9 \end{bmatrix}. \end{aligned} \quad (3.92)$$

Application of the recipe given by Eq. (3.90) leads to

$$\begin{aligned}
 & \mathbf{F}_c(k_{1c}, k_{2c}) \\
 &= \begin{bmatrix} \mathbf{F}'[-2, -2] & \mathbf{F}'[-1, -2] & \mathbf{F}'[0, -2] & \mathbf{F}'[1, -2] \\ \mathbf{F}'[-2, -1] & \mathbf{F}'[-1, -1] & \mathbf{F}'[0, -1] & \mathbf{F}'[1, -1] \\ \mathbf{F}'[-2, 0] & \mathbf{F}'[-1, 0] & \mathbf{F}'[0, 0] & \mathbf{F}'[1, 0] \\ \mathbf{F}'[-2, 1] & \mathbf{F}'[-1, 1] & \mathbf{F}'[0, 1] & \mathbf{F}'[1, 1] \end{bmatrix} \\
 &= \begin{bmatrix} -1 & -4-j3 & 1 & -4+j3 \\ -5-j2 & 2-j9 & -5+j4 & -j \\ 3 & -6+j3 & 49 & -6-j3 \\ -5+j2 & j & -5-j4 & 2+j9 \end{bmatrix}. \quad (3.93)
 \end{aligned}$$

Now conjugate symmetry  $\mathbf{F}_c[-k_{1c}, -k_{2c}] = \mathbf{F}_c^*[k_{1c}, k_{2c}]$  applies, but only after omitting the first row and column of  $\mathbf{F}_c[k_{1c}, k_{2c}]$ . This is because the dc value (49) is not at the center of the array. Indeed, there is no center pixel in an  $M \times M$  array if  $M$  is even. It is customary in center-of-image depictions to place the origin at array coordinates  $[\frac{M}{2} + 1, \frac{M}{2} + 1]$ . The first row and column are not part of the mirror symmetry about the origin. This is not noticeable in  $M \times M$  arrays if  $M$  is even and large.

Conjugate symmetry applies within the first row and within the first column, since these are 1-D DFTs with  $\frac{M}{2} = 2$ .

**Exercise 3-15:** Why did this book not spend more space on computing the DSFT?

**Answer:** Because in practice, the DSFT is computed using the 2-D DFT.

## Summary

### Concepts

- Many 2-D concepts are generalizations of 1-D counterparts. These include: LSI systems, convolution, sampling, 2-D continuous-space; 2-D discrete-space; and 2-D discrete Fourier transforms.
- The 2-D DSFT is doubly periodic in  $\Omega_1$  and  $\Omega_2$  with periods  $2\pi$ .
- Rotating an image rotates its 2-D continuous-space Fourier transform (CSFT). The CSFT of a radially symmetric image is radially symmetric.
- Continuous-space images can be sampled to discrete-

space images, on which discrete-space image processing can be performed.

- Nearest-neighbor interpolation often works well for interpolating sampled images to continuous space. Hexagonal sampling can also be used.
- Discrete-space images can be displayed in several different formats (the location of the origin differs).
- The response of an LSI system with point-spread function  $h(x,y)$  to image  $f(x,y)$  is output  $g(x,y) = h(x,y) * f(x,y)$ , and similarly in discrete space.

### Mathematical Formulae

#### Impulse

$$\delta(x,y) = \delta(x) \delta(y) = \frac{\delta(r)}{\pi r}$$

#### Energy of $f(x,y)$

$$E = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |f(x,y)|^2 dx dy$$

#### Convolution

$$h(x,y) * f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi, \eta) h(x-\xi, y-\eta) d\xi d\eta$$

#### Convolution

$$h[n,m] * f[n,m] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h[i,j] f[n-i, m-j]$$

#### Fourier transform (CSFT)

$$\mathbf{F}(\mu, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(\mu x + vy)} dx dy$$

#### Inverse CSFT

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbf{F}(\mu, v) e^{j2\pi(\mu x + vy)} d\mu dv$$

#### Ideal square lowpass filter PSF

$$h(x,y) = 4\mu_0^2 \operatorname{sinc}(2\mu_0 x) \operatorname{sinc}(2v_0 y)$$

#### Ideal radial lowpass filter PSF

$$h(r) = 4\rho_0^2 \operatorname{jinc}(2\rho_0 r)$$

#### 2-D Sampling

Sampling interval  $\frac{1}{\Delta} > 2B$  if  $\mathbf{F}(\mu, v) = 0$  for  $|\mu|, |\nu| > B$

#### 2-D Sinc interpolation formula

$$f(x,y) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f(n\Delta, m\Delta) \operatorname{sinc}\left(\frac{x-n\Delta}{\Delta}\right) \operatorname{sinc}\left(\frac{y-m\Delta}{\Delta}\right)$$

#### Discrete-space Fourier transform (DSFT)

$$\mathbf{F}(\Omega_1, \Omega_2) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n,m] e^{-j(\Omega_1 n + \Omega_2 m)}$$

#### $(\mathcal{K}_2 \times \mathcal{K}_1)$ 2-D DFT of $(M \times N)$ image

$$\mathbf{F}[k_1, k_2] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n,m] e^{-j2\pi(nk_1/\mathcal{K}_1 + mk_2/\mathcal{K}_2)}$$

#### Inverse 2-D DFT

$$f[n,m] = \frac{1}{\mathcal{K}_1 \mathcal{K}_2} \sum_{k_1=0}^{\mathcal{K}_1-1} \sum_{k_2=0}^{\mathcal{K}_2-1} \mathbf{F}[k_1, k_2] e^{j2\pi(nk_1/\mathcal{K}_1 + mk_2/\mathcal{K}_2)}$$

### Important Terms

Provide definitions or explain the meaning of the following terms:

aliasing	CSFT	DSFT	linear shift-invariant (LSI)	point-spread function	sampling theorem
convolution	DFT	FFT	nearest-neighbor interpolation	sampled image	sinc function

## PROBLEMS

### Section 3-2: 2-D Continuous-Space Images

**3.1** Let  $f(x,y)$  be an annulus (ring) with inner radius 3 and outer radius 5, with center at the point (2,4). Express  $f(x,y)$  in terms of  $f_{\text{Disk}}(x,y)$ .

### Section 3-3: Continuous-Space Systems

**3.2** Compute the autocorrelation

$$r(x,y) = f_{\text{Box}}(x,y) \ast \ast f_{\text{Box}}(-x,-y)$$

of  $f_{\text{Box}}(x,y)$ .

### Section 3-4: 2-D Continuous-Space Fourier Transform (CSFT)

**3.3** Compute the 2-D CSFT  $\mathbf{F}(\mu, \nu)$  of a  $10 \times 6$  ellipse  $f(x,y)$ .

**3.4** A 2-D Gaussian function has the form

$$f_g(r) = \frac{e^{-r^2/(2\sigma^2)}}{2\pi\sigma^2}.$$

Compute the 2-D CSFT  $\mathbf{f}_g(\rho)$  of  $f_g(r)$  using the scaling property of the 2-D CSFT.

**3.5** Compute the CSFT of an annulus (ring)  $f(x,y)$  with inner radius 3 and outer radius 5, with center at:

- (a) the origin (0,0);
- (b) the point (2,4).

### Section 3-5: 2-D Sampling Theorem

**3.6**  $f(x,y) = \cos(2\pi 3x)\cos(2\pi 4y)$  is sampled every  $\Delta = 0.2$ . What is reconstructed by a brick-wall lowpass filter with cutoff = 5 in  $\mu$  and  $\nu$  and passband gain  $\Delta = 0.2$ ?

**3.7**  $f(x,y) = \cos(2\pi 12x)\cos(2\pi 15y)$  is sampled every  $\Delta = 0.1$ . What is reconstructed by a brick-wall lowpass filter with cutoff = 10 in  $\mu$  and  $\nu$  and passband gain  $\Delta = 0.1$ ?

**3.8** Antialias filtering: Run MATLAB program P38.m. This lowpass-filters the clown image before sampling it below its Nyquist rate. Explain why the image reconstructed from its samples has no aliasing.

**3.9** Nearest-Neighbor interpolation: Run MATLAB program P39.m. This samples the clown image and reconstructs from

its samples using NN interpolation, but displays spectra at each stage. Explain why the image reconstructed from its samples matches the clown image.

### Section 3-6: 2-D Discrete Space

**3.10** Compute the 2-D convolution

$$\begin{bmatrix} 3 & 1 \\ 4 & 1 \end{bmatrix} \ast \ast \begin{bmatrix} 5 & 9 \\ 2 & 6 \end{bmatrix}$$

by hand. Check your answer using MATLAB's conv2.

**3.11** An LTI system is described by the equation

$$\begin{aligned} g[n,m] = & \frac{1}{9}f[n-1, m-1] + \frac{1}{9}f[n-1, m] + \frac{1}{9}f[n-1, m+1] \\ & + \frac{1}{9}f[n, m-1] + \frac{1}{9}f[n, m] + \frac{1}{9}f[n, m+1] \\ & + \frac{1}{9}f[n+1, m-1] + \frac{1}{9}f[n+1, m] + \frac{1}{9}f[n+1, m+1]. \end{aligned}$$

What is the PSF  $h[n,m]$  of the system? Describe in words what it does to its input.

**3.12** An LTI system is described by the equation

$$\begin{aligned} g[n,m] = & 9f[n-1, m-1] + 8f[n-1, m] + 7f[n-1, m+1] \\ & + 6f[n, m-1] + 5f[n, m] + 4f[n, m+1] \\ & + 3f[n+1, m-1] + 2f[n+1, m] + f[n+1, m+1]. \end{aligned}$$

What is the PSF  $h[n,m]$  of the system? Use center-of-image notation (Fig. 3-25(c)).

### Section 3-7: Discrete-Space Fourier Transform (DSFT)

**3.13** Prove the shift property of the DSFT:

$$f[n-a, m-b] \leftrightarrow e^{-j(a\Omega_1+b\Omega_2)} \mathbf{F}(\Omega_1, \Omega_2).$$

**3.14** Compute the spatial frequency response of the system in Problem 3.11.

**3.15** Compute the spatial frequency response of the system in Problem 3.12.

**3.16** Show that the 2-D spatial frequency response of the PSF using  $(2 \times 2)$  2-D DFTs.

$$h[m, n] = \begin{bmatrix} 1 & 2 & 1 \\ 2 & \frac{4}{4} & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

is close to circularly symmetric, making it a circularly symmetric lowpass filter, by:

(a) displaying the spatial frequency response as an image with dc at the center;

(b) using

$$\cos(\Omega) = 1 - \frac{\Omega^2}{2!} + \frac{\Omega^4}{4!} - \dots$$

and neglecting all terms of degree four or higher.

### Section 3-8: 2-D Discrete Fourier Transform (DFT)

**3.17** Compute by hand the  $(2 \times 2)$  2-D DFT of

$$F = f[n, m] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

Check your answer using MATLAB using `FX=fft2(F, 2, 2)`.

**3.18** Prove that the  $(M \times M)$  2-D DFT of a separable image  $f[n, m] = f_1[n] f_2[m]$  is the product of the 1-D  $M$ -point DFTs of  $f_1[n]$  and  $f_2[n]$ :  $\mathbf{F}[k_1, k_2] = \mathbf{f}_1[k_1] \mathbf{f}_2[k_2]$ .

**3.19** Show that we can extend the definition of the  $(M \times M)$  2-D DFT to negative values of  $k_1$  and  $k_2$  using

$$\mathbf{F}[-k_1, -k_2] = \mathbf{F}[M - k_1, M - k_2]$$

for indices  $0 < k_1, k_2 < M$ . Conjugate symmetry for real  $f[n, m]$  is then

$$\mathbf{F}[-k_1, -k_2] = \mathbf{F}[M - k_1, M - k_2] = \mathbf{F}^*[k_1, k_2].$$

**3.20** Compute the 2-D cyclic convolution

$$y[n, m] = x_1[n, m] \odot \odot x_2[n, m],$$

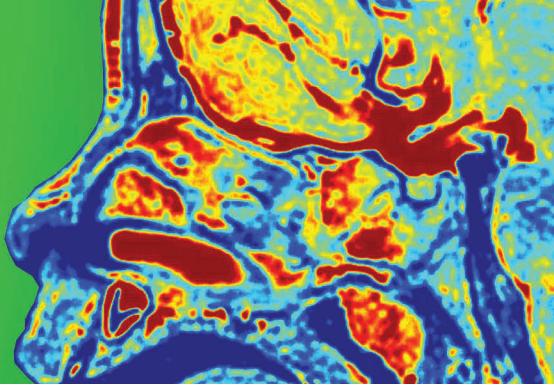
where

$$x_1[n, m] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

and

$$x_2[n, m] = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

# CHAPTER 4



## 4

## Image Interpolation

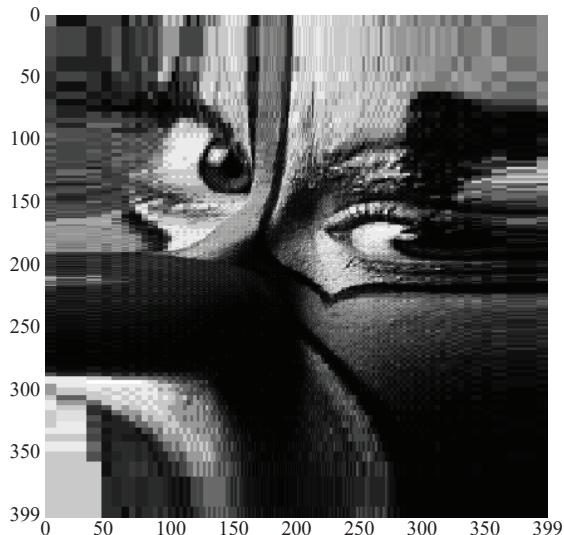
### Contents

- Overview, 129
- 4-1** Interpolation Using Sinc Functions, 129
- 4-2** Upsampling and Downsampling Modalities, 130
- 4-3** Upsampling and Interpolation, 133
- 4-4** Implementation of Upsampling Using 2-D DFT in MATLAB, 137
- 4-5** Downsampling, 140
- 4-6** Antialias Lowpass Filtering, 141
- 4-7** B-Splines Interpolation, 143
- 4-8** 2-D Spline Interpolation, 149
- 4-9** Comparison of 2-D Interpolation Methods, 150
- 4-10** Examples of Image Interpolation
  - Applications, 152
  - Problems, 156

### Objectives

Learn to:

- Use sinc and Lanczos functions to interpolate a bandlimited image.
- Perform upsampling and interpolation using the 2-D DFT and MATLAB.
- Perform downsampling for thumbnails using the 2-D DFT and MATLAB.
- Use B-spline functions of various orders to interpolate non-bandlimited images.
- Rotate, magnify, and morph images using interpolation.



In many image processing applications, such as magnification, thumbnails, rotation, morphing, and reconstruction from samples, it is necessary to interpolate (roughly, fill in gaps between given samples). This chapter presents three approaches to interpolation: using sinc or Lanczos functions; upsampling using the 2-D DFT; and use of B-splines. These methods are compared and used on each of the applications listed above.

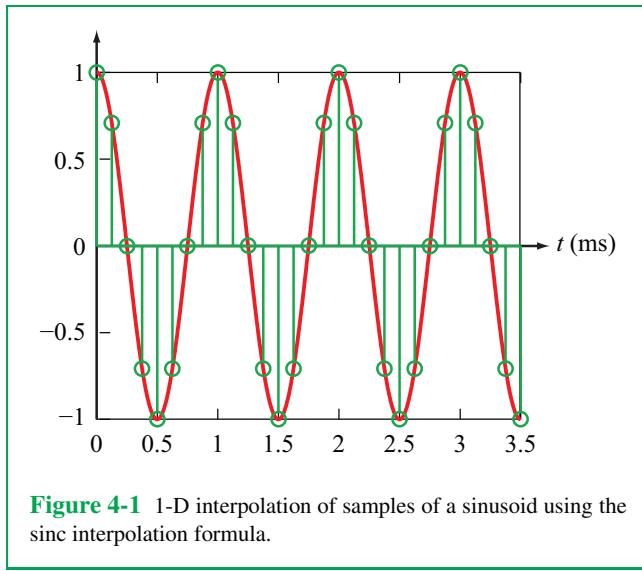
## Overview

Suppose some unknown signal  $x(t)$  had been sampled at a sampling rate  $S$  to generate a **sampled signal**  $x[n] = \{x(n\Delta), n = \dots, -1, 0, 1, \dots\}$ , sampled at times  $t = n\Delta$ , where  $\Delta = 1/S$  is the **sampling interval**. **Interpolation** entails the use of a recipe or formula to compute a continuous-time interpolated version  $x_{\text{int}}(t)$  that takes on the given values  $\{x(n\Delta)\}$  and interpolates between them. In 1-D, interpolation is akin to **connecting the dots**, represented here by  $\{x(n\Delta)\}$ , to obtain  $x_{\text{int}}(t)$ . The degree to which the interpolated signal is identical to or a close rendition of the original signal  $x(t)$  depends on two factors:

- (1) Whether or not the **sampling rate**  $S$  used to generate  $x_{\text{int}}(t)$  from  $x(t)$  satisfies the Nyquist criterion, namely  $S > 2B$ , where  $B$  is the maximum frequency in the spectrum of signal  $x(t)$ , and
- (2) the specific interpolation method used to obtain  $x_{\text{int}}(t)$ .

If the spectrum  $X(f)$  of  $x(t)$  is bandlimited to  $B$  and  $S > 2B$ , it should be possible to use the sinc interpolation formula given by Eq. (2.51) to reconstruct  $x(t)$  exactly. A simple example is illustrated by the finite-duration sinusoid shown in **Fig. 4-1**. In practice, however, it is computationally more efficient to perform the interpolation in the frequency domain by lowpass-filtering the spectrum of the sampled signal.

Oftentimes, the sampling rate does not satisfy the Nyquist criterion, as a consequence of which the signal obtained by



applying the sinc interpolation formula may be an aliased version of  $x(t)$ . In such cases, different interpolation methods should be used, several of which are examined in this chapter.

In 2-D, interpolation seeks to **fill in the gaps** between the image values  $f[n, m] = \{f(n\Delta, m\Delta)\}$  defined at discrete locations  $(x = n\Delta, y = m\Delta)$  governed by the spatial sampling rate  $S = S_x = S_y = 1/\Delta$ . In analogy with the 1-D case, if the spectrum  $\mathbf{F}(\mu, v)$  of the original image  $f(x, y)$  is bandlimited to  $B_x = B_y = B$  and if the sampling rate satisfies the Nyquist criterion (i.e.,  $S > 2B$ ), then it should be possible to interpolate  $f[n, m] = f(n\Delta, m\Delta)$  to generate  $f(x, y)$  exactly.

This chapter explores several types of recipes for interpolating a sampled image  $f[n, m] = \{f(n\Delta, m\Delta)\}$  into a continuous-space image  $f_{\text{int}}(x, y)$ . Some of these recipes are extensions of the sinc interpolation formula, while others rely on the use of **B-spline** functions. As discussed later in Section 4-7, B-splines are polynomial functions that can be designed to perform nearest-neighbor, linear, quadratic, and cubic interpolation of images and signals. We will also explore how interpolation is used to realize image zooming (magnification), rotation, and warping.

## 4-1 Interpolation Using Sinc Functions

### 4-1.1 Sinc Interpolation Formula

An image  $f(x, y)$  is said to be bandlimited to a maximum spatial frequency  $B$  if its spectrum  $\mathbf{F}(\mu, v)$  is such that

$$\mathbf{F}(\mu, v) = 0 \quad \text{for } |\mu|, |v| \geq B.$$

If such an image is sampled uniformly along  $x$  and  $y$  at sampling rates  $S_x = S_y = S = 1/\Delta$ , and the sampling interval  $\Delta$  satisfies the Nyquist rate, namely

$$\Delta < \frac{1}{2B},$$

then we know from the 2-D sampling theorem (Section 3-5) that  $f(x, y)$  can be reconstructed from its samples  $f[n, m] = \{f(n\Delta, m\Delta)\}$  using the **sinc interpolation formula** given by Eq. (3.62), which we repeat here as

$$f(x, y) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f(n\Delta, m\Delta) \operatorname{sinc}\left(\frac{x}{\Delta} - n\right) \operatorname{sinc}\left(\frac{y}{\Delta} - m\right), \quad (4.1)$$

where for any argument  $z$ , the sinc function is defined as

$$\operatorname{sinc}(z) = \frac{\sin(\pi z)}{\pi z}. \quad (4.2)$$

In reality, an image is finite in size, and so is the number of samples  $\{f(n\Delta, m\Delta)\}$ . Consequently, for a square image, the ranges of indices  $n$  and  $m$  are limited to finite lengths  $M$ , in which case the infinite sums in Eq. (4.1) become finite sums:

$$f_{\text{sinc}}(x, y) = \sum_{n=0}^{M-1} \sum_{m=0}^{M-1} f(n\Delta, m\Delta) \text{sinc}\left(\frac{x}{\Delta} - n\right) \text{sinc}\left(\frac{y}{\Delta} - m\right). \quad (4.3)$$

The summations start at  $n = 0$  and  $m = 0$ , consistent with the image display format shown in **Fig. 3-3(a)**, wherein location  $(0, 0)$  is at the top-left corner of the image.

The sampled image consists of  $M \times M$  values—denoted here by  $f(n\Delta, m\Delta)$ —each of which is multiplied by the product of two sinc functions, one along  $x$  and another along  $y$ . The value  $f_{\text{sinc}}(x, y)$  at a specified location  $(x, y)$  on the image consists of the sum of  $M \times M$  terms. In the limit for an image of infinite size, and correspondingly an infinite number of samples along  $x$  and  $y$ , the infinite summations lead to  $f_{\text{sinc}}(x, y) = f(x, y)$ , where  $f(x, y)$  is the original image. That is, the interpolated image is identical to the original image, assuming all along that the sampled image is in compliance with the Nyquist criterion of the sampling theorem. When applying the sinc interpolation formula to a finite-size image, the interpolated image should be a good match to the original image, but it is computationally inefficient when compared with the spatial-frequency domain interpolation technique described later in Section 4-3.2.

where the rectangle function, defined earlier in Eq. (2.2), is given by

$$\text{rect}\left(\frac{x}{2a}\right) = \begin{cases} 1 & \text{for } -a < x < a, \\ 0 & \text{otherwise.} \end{cases}$$

Parameter  $a$  usually is assigned a value of 2 or 3. In MATLAB, the Lanczos interpolation formula can be exercised using the command `imresize` and selecting Lanczos from the menu.

In addition to the plot for  $\text{sinc}(x)$ , **Fig. 4-2** also contains plots of the sinc function multiplied by the windowed sinc function, with  $a = 2$  and also with  $a = 3$ . The rectangle function is zero beyond  $|x| = a$ , so the windowed function stops at  $|x| = 2$  for  $a = 2$  and at  $|x| = 3$  for  $a = 3$ . The Lanczos interpolation method provides significant computational improvement over the simple sinc interpolation formula.

**Concept Question 4-1:** What is the advantage of Lanczos interpolation over sinc interpolation?

**Exercise 4-1:** If sinc interpolation is applied to the samples  $\{x(0.1n) = \cos(2\pi(6)0.1n)\}$ , what will be the result?

**Answer:**  $\{x(0.1n)\}$  are samples of a 6 Hz cosine sampled at a rate of 10 samples/second, which is below the Nyquist frequency of 12 Hz. The sinc interpolation will be a cosine with frequency aliased to  $(10 - 6) = 4$  Hz (see Section 2-4.3).

## 4-1.2 Lanczos Interpolation

To reduce the number of terms involved in the computation of the interpolated image, the sinc function in Eq. (4.3) can be modified by truncating the sinc pattern along  $x$  and  $y$  so that it is zero beyond a certain multiple of  $\Delta$ , such as  $|x| = 2\Delta$  and  $|y| = 2\Delta$ . Such a truncation is offered by the **Lanczos interpolation formula** which replaces each of the sinc functions in Eq. (4.3) by **windowed sinc functions**, thereby assuming the form

$$f_{\text{Lanczos}}(x, y) = \sum_{n=0}^{M-1} \sum_{m=0}^{M-1} f(n\Delta, m\Delta) \times \text{sinc}\left(\frac{x}{\Delta} - n\right) \text{sinc}\left(\frac{x}{a\Delta} - n\right) \text{rect}\left(\frac{x}{2a}\right) \times \text{sinc}\left(\frac{y}{\Delta} - m\right) \text{sinc}\left(\frac{y}{a\Delta} - m\right) \text{rect}\left(\frac{y}{2a}\right), \quad (4.4)$$

## 4-2 Upsampling and Downsampling Modalities

As a prelude to the material presented in forthcoming sections, we present here four examples of image upsampling and downsampling applications. **Figure 4-3** depicts five image configurations, each consisting of a square array of square pixels. The central image is the *initial* image from which the other four were generated. We define this initial image as the discrete version of a continuous image  $f(x, y)$ , sampled at  $(M \times M)$  locations at a sampling interval  $\Delta_0$  along both dimensions:

$$f[n, m] = \{f(n\Delta_0, m\Delta_0), 0 \leq n, m \leq M - 1\}. \quad (4.5)$$

We assume that the sampling rate  $S$  is such that  $\Delta_0 = 1/S$  satisfies the Nyquist rate  $S > 2B$  or, equivalently,  $\Delta_0 < 1/2B$ , where  $B$  is the maximum spatial frequency of  $f(x, y)$ .

When displayed on a computer screen, the initial (central

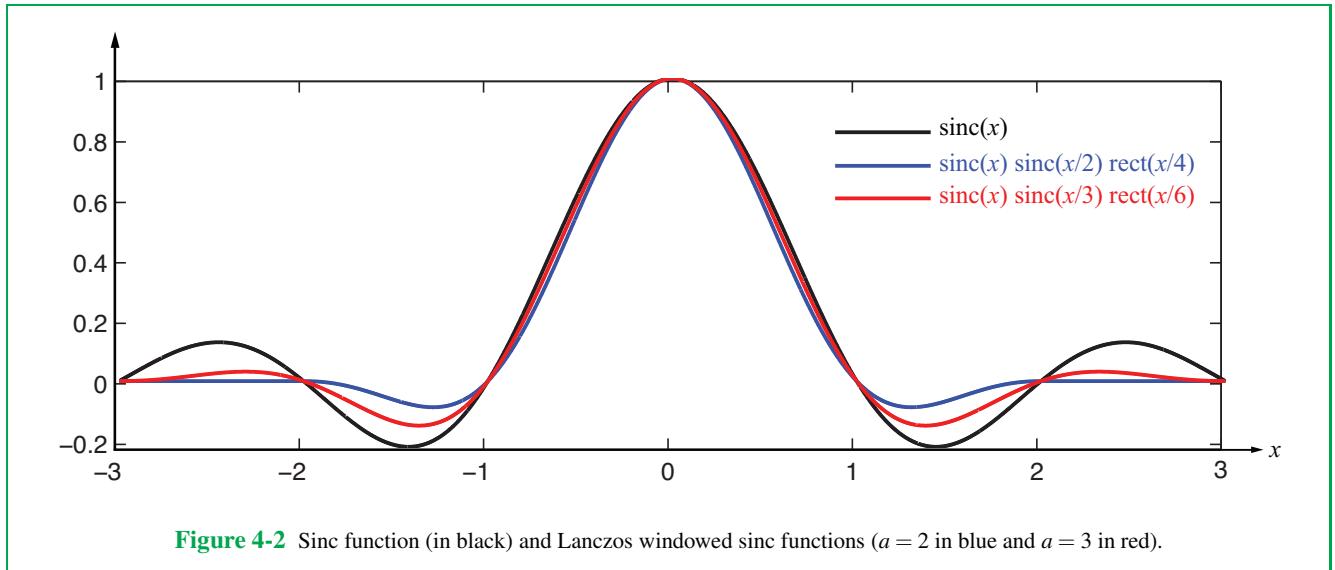


image is characterized by four parameters:

$$M_o \times M_o = M^2 = \text{image array size},$$

$$T \times T = T^2 = \text{image physical size on computer screen},$$

$$w \times w = w^2 = \text{image pixel area},$$

$$\Delta_o \times \Delta_o = \Delta_o^2 = \text{true resolution area}.$$

If the displayed image bears a one-to-one correspondence to the array  $f[n, m]$ , then the brightness of a given pixel corresponds to the magnitude  $f[n, m]$  of the corresponding element in the array, and the **pixel dimensions**  $w \times w$  are directly proportional to  $\Delta_o \times \Delta_o$ . For simplicity, we set  $w = \Delta_o$ , which means that the image displayed on the computer screen has the same physical dimensions as the original image  $f(x, y)$ . The area  $\Delta_o^2$  is the **true resolution area** of the image.

### 4-2.1 Upsampling Modalities

#### A. Enlarging Physical Size While Keeping Pixel Size Unchanged

The image configuration in part (a) of **Fig. 4-3** depicts what happens when the central image  $f[n, m]$  is enlarged in size from  $(T \times T)$  to  $(T' \times T')$  while keeping the pixel size the same. The process, accomplished by an upsampling operation, leads to an

$(M_u \times M_u)$  image  $g[n, m]$ , with  $M_u > M_o$  and

$$T' = \frac{M_u}{M_o} T, \quad w' = w, \quad \text{and} \quad \Delta_u = \frac{M_o}{M_u} \Delta_o. \quad (4.6)$$

Since the sampling interval  $\Delta_u$  in the enlarged upsampled image is shorter than the sampling interval in the initial image,  $\Delta_o$ , the Nyquist requirement continues to be satisfied.

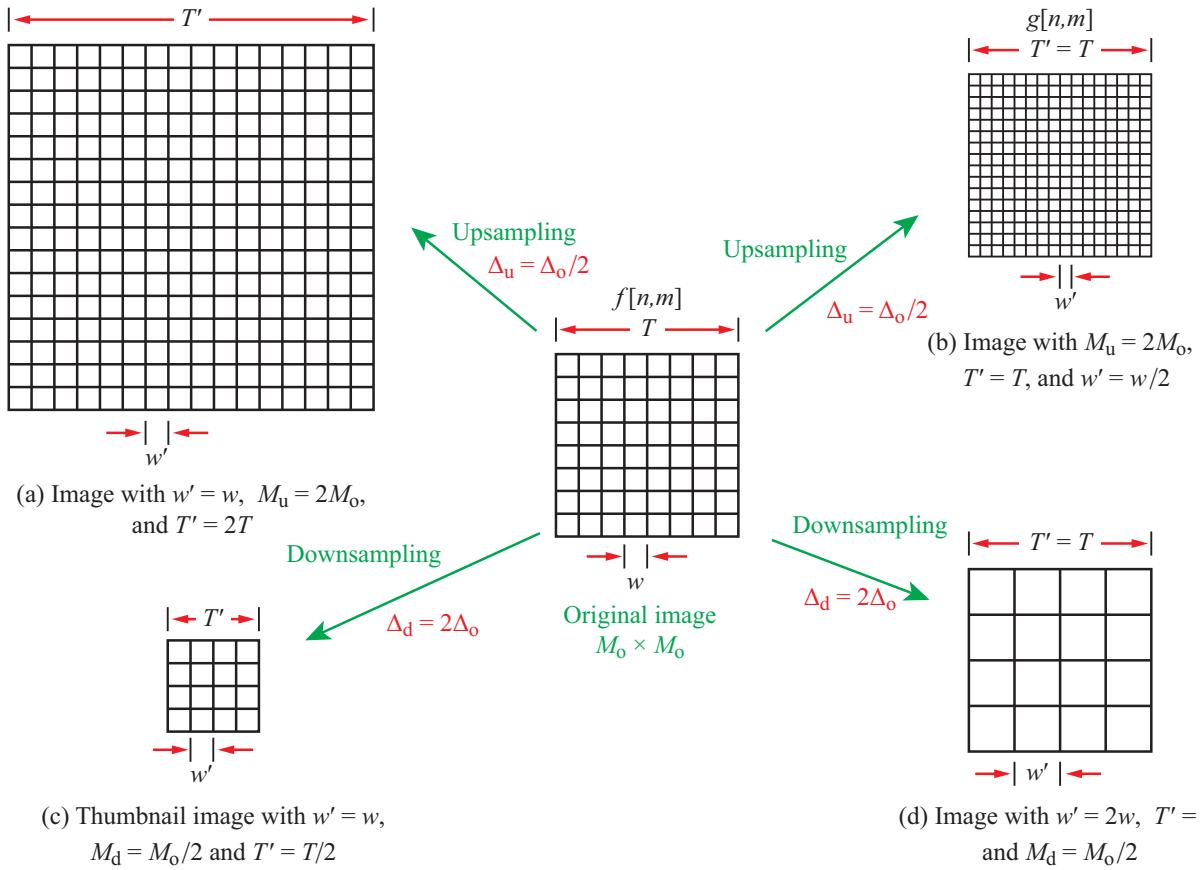
#### B. Increasing Image Array Size While Keeping Physical Size Unchanged

The image displayed in **Fig. 4-3(b)** is an upsampled version of  $f[n, m]$  in which the array size was increased from  $(M \times M)$  to  $(M_u \times M_u)$  and the physical size of the image remained the same, but the pixel size is smaller. In the image  $g[n, m]$ ,

$$T = T', \quad w' = w \frac{M_o}{M_u}, \quad \text{and} \quad \Delta_u = \frac{M_o}{M_u} \Delta_o. \quad (4.7)$$

► The two images in **Fig. 4-3(a)** and **(b)** have identical arrays  $g[n, m]$ , but they are artificially displayed on computer screens with different pixel sizes. ◀

### 4-2.2 Downsampling Modalities



**Figure 4-3** Examples of image upsampling and downsampling.

### A. Thumbnail Image

If we apply downsampling to reduce the array size from  $(M_o \times M_o)$  to  $(M_d \times M_d)$ , we end up with the downsampled images depicted in Figs. 4-3(c) and (d). In the thumbnail image shown in Fig. 4-3(c), the pixel size of the computer display is the same as that of the original image. Hence,

$$T' = \frac{M_d}{M_o} T, \quad w' = w, \quad \text{and} \quad \Delta_d = \frac{M_o}{M_d} \Delta_o. \quad (4.8)$$

### B. Reduce Array Size While Keeping Physical Size Unchanged

The final of the transformed images, shown in Fig. 4-3(d), has the same identical content of the thumbnail image, but the pixel size on the computer screen has been enlarged so that

$$T = T', \quad \Delta_d = \frac{M_o}{M_d} \Delta_o, \quad \text{and} \quad w' = w \frac{M_o}{M_d}. \quad (4.9)$$

## 4-3 Upsampling and Interpolation

Let  $f(x, y)$  be a continuous-space image of size  $T$  (meters) by  $T$  (meters) whose 2-D Fourier transform  $\mathbf{F}(\mu, v)$  is bandlimited to  $B$  (cycles/m). That is,

$$F(\mu, v) = 0 \quad \text{for } |\mu|, |v| > B. \quad (4.10)$$

Image  $f(x, y)$  is not available to us, but an  $(M_o \times M_o)$  sampled version of  $f(x, y)$  is available. We define it as the **original sampled image**

$$f[n, m] = \{ f(n\Delta_o, m\Delta_o), 0 \leq n, m \leq M_o - 1 \}, \quad (4.11)$$

where  $\Delta_o$  is the associated sampling interval. Moreover, the sampling had been performed at a rate exceeding the Nyquist rate, which requires the choice of  $\Delta_o$  to satisfy the condition

$$\Delta_o < \frac{1}{2B}. \quad (4.12)$$

Given that the sampled image is  $T \times T$  in size, the number of samples  $M_o$  along each direction is

$$M_o = \frac{T}{\Delta_o}. \quad (4.13)$$

Next, we introduce a new (yet to be created) higher-density sampled image  $g[n, m]$ , also  $T \times T$  in physical dimensions but containing  $M_u \times M_u$  samples—instead of  $M_o \times M_o$  samples—with  $M_u > M_o$  (which corresponds to the scenario depicted in **Fig. 4-3(b)**). We call  $g[n', m']$  the **upsampled version** of  $f[n, m]$ . The goal of **upsampling and interpolation**, which usually is abbreviated to just “upsampling,” is to compute  $g[n', m']$  from  $f[n, m]$ . Since  $M_u > M_o$ ,  $g[n', m']$  is more finely discretized than  $f[n, m]$ , and the narrower sampling interval  $\Delta_u$  of the upsampled image is

$$\Delta_u = \frac{T}{M_u} = \frac{M_o}{M_u} \Delta_o. \quad (4.14)$$

Since  $\Delta_u < \Delta_o$ , it follows that the sampling rate associated with  $g[n', m']$  also satisfies the Nyquist rate.

The upsampled image is given by

$$g[n', m'] = \{ f(n'\Delta_u, m'\Delta_u), 0 \leq n', m' \leq M_u - 1 \}. \quad (4.15)$$

In a later section of this chapter (Section 4-6) we demonstrate how the finer discretization provided by upsampling is used to compute a rotated or warped version of image  $f[n, m]$ . Another application is image magnification, but in that case the primary goal is to increase the image size (from  $T_1 \times T_1$  to  $T_2 \times T_2$ ), as

illustrated in **Fig. 4-3(a)**, rather than to decrease the sampling interval.

Upsampling image  $f[n, m]$  to image  $g[n', m']$  can be accomplished either directly in the discrete spatial domain or indirectly in the spatial frequency domain. We examine both approaches in the subsections that follow.

### 4-3.1 Upsampling in the Spatial Domain

In practice, image upsampling is performed using the 2-D DFT in the spatial frequency domain because it is much faster and easier computationally than performing the upsampling directly in the spatial domain. Nevertheless, for the sake of completeness, we now provide a succinct presentation of how upsampling is performed in the spatial domain using the sinc interpolation formula.

We start by repeating Eq. (4.3) after replacing  $\Delta$  with  $\Delta_o$  and  $M$  with  $M_o$ :

$$f(x, y) = \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] \operatorname{sinc}\left(\frac{x}{\Delta_o} - n\right) \operatorname{sinc}\left(\frac{y}{\Delta_o} - m\right). \quad (4.16)$$

Here,  $f[n, m]$  is the **original**  $(M_o \times M_o)$  sampled image available to us, and the goal is to upsample it to an  $(M_u \times M_u)$  image  $g[n', m']$ , with  $M_u = LM_o$ , where  $L$  is an **upsampling factor**. In the upsampled image, the sampling interval  $\Delta_u$  is related to the sampling interval  $\Delta_o$  of the original sampled image by

$$\Delta_u = \frac{M_o}{M_u} \Delta_o = \frac{\Delta_o}{L}. \quad (4.17)$$

To obtain  $g[n', m']$ , we sample  $f(x, y)$  at  $x = n'\Delta_u$  and  $y = m'\Delta_u$ :

$$\begin{aligned} g[n', m'] &= f(n'\Delta_u, m'\Delta_u) \quad (0 \leq n', m' \leq M_u - 1) \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} f[n, m] \\ &\quad \times \operatorname{sinc}\left(n' \frac{\Delta_u}{\Delta_o} - n\right) \operatorname{sinc}\left(m' \frac{\Delta_u}{\Delta_o} - m\right) \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} f[n, m] \\ &\quad \times \operatorname{sinc}\left(\frac{n'}{L} - n\right) \operatorname{sinc}\left(\frac{m'}{L} - m\right). \end{aligned} \quad (4.18)$$

If greater truncation is desired, we can replace the product of sinc functions with the product of the Lanczos functions defined in Eq. (4.4). In either case, application of Eq. (4.18) generates

the upsampled version  $g[n',m']$  directly from the original sampled version  $f[n,m]$ . If  $\Delta_u = \Delta_o/L$  and  $L$  is an integer, then the process preserves the values of  $f[n,m]$  while adding new ones in between them. To demonstrate that the upsampling process does indeed preserve  $f[n,m]$ , let us consider the expression given by Eq. (4.15) for the specific case where  $n' = Ln$  and  $m' = Lm$ :

$$\begin{aligned} g[Ln, Lm] &= \{ f(n'\Delta_u, m'\Delta_u), 0 \leq n', m' \leq M_u - 1 \} \\ &= \{ f(Ln\Delta_u, Lm\Delta_u), 0 \leq n, m \leq M_o - 1 \} \\ &= \{ f(n\Delta_o, m\Delta_o), 0 \leq n, m \leq M_o - 1 \} = f[n, m], \end{aligned}$$

where we used the relationships given by Eqs. (4.11) and (4.14).

Hence, upsampling by an integer  $L$  using the sinc interpolation formula does indeed preserve the existing values of  $f[n,m]$ , in addition to adding interpolated values between them.

### 4-3.2 Upsampling in the Spatial Frequency Domain

Instead of using the sinc interpolation formula given by Eq. (4.18), upsampling can be performed much more easily, and with less computation, using the 2-D DFT in the spatial frequency domain. From Eqs. (4.11) and (4.18), the  $(M_o \times M_o)$  original  $f[n,m]$  image and the  $(M_u \times M_u)$  upsampled image  $g[n',m']$  are defined as

$$f[n, m] = \{ f(n\Delta_o, m\Delta_o), 0 \leq n, m \leq M_o - 1 \}, \quad (4.19a)$$

$$g[n', m'] = \{ g(n'\Delta_u, m'\Delta_u), 0 \leq n', m' \leq M_u - 1 \}. \quad (4.19b)$$

Note that whereas in the earlier section it proved convenient to distinguish the indices of the upsampled image from those of the original image—so we used  $[n,m]$  for the original image and  $[n',m']$  for the upsampled image—the distinction is no longer needed in the present section, so we will now use indices  $[n,m]$  for both images.

From Eq. (3.75), the 2-D DFT of  $f[n,m]$  of order  $(M_o \times M_o)$  and the 2-D DFT of  $g[n,m]$  of order  $(M_u \times M_u)$  are given by

$$\begin{aligned} \mathbf{F}[k_1, k_2] &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j(2\pi/M_o)(nk_1+mk_2)}, \\ 0 \leq k_1, k_2 \leq M_o - 1, \end{aligned} \quad (4.20a)$$

$$\begin{aligned} \mathbf{G}[k_1, k_2] &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} f[n, m] e^{-j(2\pi/M_u)(nk_1+mk_2)}, \\ 0 \leq k_1, k_2 \leq M_u - 1. \end{aligned} \quad (4.20b)$$

The summations are identical in form, except that the summa-

tion for  $\mathbf{F}[k_1, k_2]$  extends to  $(M_o - 1)$  whereas the summation for  $\mathbf{G}[k_1, k_2]$  extends to  $(M_u - 1)$ .

The goal is to compute  $g[n,m]$  by (1) transforming  $f[n,m]$  to obtain  $\mathbf{F}[k_1, k_2]$ , (2) transforming  $\mathbf{F}[k_1, k_2]$  to  $\mathbf{G}[k_1, k_2]$ , and (3) then transforming  $\mathbf{G}[k_1, k_2]$  back to the spatial domain to form  $g[n,m]$ . Despite the seeming complexity of having to execute a three-step process, the process is computationally more efficient than performing upsampling entirely in the spatial domain.

Upsampling in the discrete frequency domain  $[k_1, k_2]$  entails increasing the number of discrete frequency components from  $(M_o \times M_o)$  for  $\mathbf{F}[k_1, k_2]$  to  $(M_u \times M_u)$  for  $\mathbf{G}[k_1, k_2]$ , with  $M_u > M_o$ . As we will demonstrate shortly,  $\mathbf{G}[k_1, k_2]$  includes all of the elements of  $\mathbf{F}[k_1, k_2]$ , but it also includes some additional rows and columns filled with zeros.

#### A. Original Image

Let us start with the sampled image  $f_o(x,y)$  of continuous image  $f(x,y)$  sampled at a sampling interval  $\Delta_o$  and resulting in  $(M_o \times M_o)$  samples. Per Eq. (3.61), adapted to a finite sum that starts at  $(0,0)$  and ends at  $(M_o - 1, M_o - 1)$ ,

$$\begin{aligned} f_o(x, y) &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f(n\Delta_o, m\Delta_o) \delta(x - n\Delta_o) \delta(y - m\Delta_o) \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] \delta(x - n\Delta_o) \delta(y - m\Delta_o), \end{aligned} \quad (4.21)$$

where we used the definition for  $f[n,m]$  given by Eq. (4.19a).

Using entry #9 in **Table 3-1**, the 2-D CSFT  $\mathbf{F}_o(\mu, v)$  of  $f_o(x,y)$  can be written as

$$\begin{aligned} \mathbf{F}_o(\mu, v) &= \mathcal{F}\{f_o(x, y)\} \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] \mathcal{F}\{\delta(x - n\Delta_o) \delta(y - m\Delta_o)\} \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j2\pi\mu n\Delta_o} e^{-j2\pi v m\Delta_o}. \end{aligned} \quad (4.22)$$

The spectrum  $\mathbf{F}_o(\mu, v)$  of sampled image  $f_o(x,y)$  is doubly periodic in  $\mu$  and  $v$  with period  $1/\Delta_o$ , as expected.

Extending the relations expressed by Eqs. (2.47) and (2.54) from 1-D to 2-D, the spectrum  $\mathbf{F}_o(\mu, v)$  of the sampled image is related to the spectrum  $\mathbf{F}(\mu, v)$  of continuous image  $f(x,y)$  by

$$\mathbf{F}_o(\mu, v) = \frac{1}{\Delta_o^2} \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \mathbf{F}(\mu - k_1/\Delta_o, v - k_2/\Delta_o). \quad (4.23)$$

The spectrum  $\mathbf{F}_o(\mu, v)$  of the sampled image consists of copies of the spectrum  $\mathbf{F}(\mu, v)$  of  $f(x, y)$  repeated every  $1/\Delta_o$  in both  $\mu$  and  $v$ , and also scaled by  $1/\Delta_o^2$ .

In  $(\mu, v)$  space,  $\mu$  and  $v$  can be both positive or negative. The relation between the spectrum  $\mathbf{F}_o(\mu, v)$  of the sampled image  $f_o(x, y)$  and the spectrum  $\mathbf{F}(\mu, v)$  of the original continuous image  $f(x, y)$  assumes different forms for the four quadrants of  $(\mu, v)$  space.

For ease of presentation, let  $M_o$  be odd. If  $M_o$  is even, then simply replace  $(M_o - 1)/2$  with  $M_o/2$  (see Section 4-4.2).

Next, we sample  $\mu$  and  $v$  by setting them to

$$\begin{aligned}\mu &= \frac{k_1}{M_o \Delta_o} \quad \text{and} \quad v = \frac{k_2}{M_o \Delta_o}, \\ 0 &\leq |k_1|, |k_2| \leq \frac{M_o - 1}{2}.\end{aligned}\quad (4.24)$$

## 1. Quadrant 1: $\mu \geq 0$ and $v \geq 0$

At these values of  $\mu$  and  $v$ ,  $\mathbf{F}_o(\mu, v)$  becomes

$$\begin{aligned}\mathbf{F}_o\left(\frac{k_1}{M_o \Delta_o}, \frac{k_2}{M_o \Delta_o}\right) \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j2\pi n k_1 / M_o} e^{-j2\pi m k_2 / M_o} \\ &= \mathbf{F}[k_1, k_2], \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2}.\end{aligned}\quad (4.25)$$

## 2. Quadrant 2: $\mu \leq 0$ and $v \geq 0$

In quadrants 2–4, we make use of the relation

$$\begin{aligned}e^{-j2\pi n(-k_1)/M_o} &= e^{-j2\pi n M_o / M_o} e^{-j2\pi n(-k_1)/M_o} \\ &= e^{-j2\pi n(M_o - k_1)/M_o},\end{aligned}$$

where we used  $e^{-j2\pi n M_o / M_o} = 1$ . A similar relation applies to  $k_2$ .

In quadrant 2,  $\mu$  is negative and  $v$  is positive, so keeping

$k_1, k_2 \geq 0$  and redefining  $\mu$  as  $\mu = -k_1/(M_o \Delta_o)$  leads to

$$\begin{aligned}\mathbf{F}_o\left(\frac{-k_1}{M_o \Delta_o}, \frac{k_2}{M_o \Delta_o}\right) &\quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2} \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j2\pi n(-k_1)/M_o} e^{-j2\pi m k_2 / M_o} \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j2\pi n(M_o - k_1)/M_o} e^{-j2\pi m k_2 / M_o} \\ &= \mathbf{F}[M_o - k_1, k_2], \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2},\end{aligned}\quad (4.26)$$

## 3. Quadrant 3: $\mu \leq 0$ and $v \leq 0$

Redefining  $\mu$  as  $\mu = -k_1/(M_o \Delta_o)$  and  $v$  as  $v = -k_2/(M_o \Delta_o)$  leads to

$$\begin{aligned}\mathbf{F}_o\left(\frac{-k_1}{M_o \Delta_o}, \frac{-k_2}{M_o \Delta_o}\right), &\quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2} \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j2\pi n(-k_1)/M_o} e^{-j2\pi m(-k_2)/M_o} \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j2\pi n(M_o - k_1)/M_o} e^{-j2\pi m(M_o - k_2)/M_o} \\ &= \mathbf{F}[M_o - k_1, M_o - k_2], \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2}.\end{aligned}\quad (4.27)$$

## 4. Quadrant 4: $\mu \geq 0$ and $v \leq 0$

Upon defining  $\mu$  as in Eq. (4.24) and redefining  $v$  as  $v = -k_2/(M_o \Delta_o)$ ,

$$\begin{aligned}\mathbf{F}_o\left(\frac{k_1}{M_o \Delta_o}, \frac{-k_2}{M_o \Delta_o}\right), &\quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2} \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j2\pi n k_1 / M_o} e^{-j2\pi m(-k_2)/M_o} \\ &= \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] e^{-j2\pi n k_1 / M_o} e^{-j2\pi m(M_o - k_2)/M_o} \\ &= \mathbf{F}[k_1, M_o - k_2], \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2},\end{aligned}\quad (4.28)$$

The result given by Eqs. (4.25)–(4.28) states that the 2-D CSFT  $\mathbf{F}_o(\mu, v)$  of the sampled image  $f_o(x, y)$ —when sampled at the discrete spatial frequency values defined by Eq. (4.24)—is the 2-D DFT of the sampled image  $f[n, m]$ . Also, the spectrum  $\mathbf{F}_o(\mu, v)$  of the sampled image consists of copies of the

spectrum  $\mathbf{F}(\mu, \nu)$  repeated every  $1/\Delta_o$  in both  $\mu$  and  $\nu$ , and also scaled by  $1/\Delta_o^2$ . Hence, generalizing Eq. (2.54) from 1-D to 2-D gives

$$\mathbf{F}_o\left(\frac{k_1}{M_o\Delta_o}, \frac{k_2}{M_o\Delta_o}\right) = \frac{1}{\Delta_o^2} \mathbf{F}\left(\frac{k_1}{M_o\Delta_o}, \frac{k_2}{M_o\Delta_o}\right), \quad (4.29)$$

where  $\mathbf{F}(\mu, \nu)$  is the 2-D CSFT of the continuous-space image  $f(x, y)$  and  $0 \leq |k_1|, |k_2| \leq (M_o - 1)/2$ .

upon sampling  $\mathbf{F}_u(\mu, \nu)$  at the rates defined by Eq. (4.24), we obtain

$$\begin{aligned} & \mathbf{F}_u\left(\frac{k_1}{M_u\Delta_u}, \frac{k_2}{M_u\Delta_u}\right) \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] e^{-j2\pi n k_1 / M_u} e^{-j2\pi m k_2 / M_u} \\ &= \mathbf{G}[k_1, k_2], \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2}, \end{aligned} \quad (4.32)$$

## A. Upsampled Image

Now we repeat this entire derivation using a sampling interval  $\Delta_u$  instead of  $\Delta_o$ , and replacing  $M_o$  with  $M_u$ , but keeping the form of the relations given by Eq. (4.24) the same. Since  $\Delta_u < \Delta_o$ , the sampled image is now  $(M_u \times M_u)$  instead of  $(M_o \times M_o)$ . Hence, the  $(M_u \times M_u)$  sampled image is

$$\begin{aligned} f_u(x, y) &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} f(n\Delta_u, m\Delta_u) \delta(x - n\Delta_u) \delta(y - m\Delta_u) \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] \delta(x - n\Delta_u) \delta(y - m\Delta_u), \end{aligned} \quad (4.30)$$

with  $g[n, m]$  as defined in Eq. (4.19b). The associated 2-D CSFT of the sampled image  $f_u(x, y)$  is

$$\begin{aligned} \mathbf{F}_u(\mu, \nu) &= \mathcal{F}\{f_u(x, y)\} \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] \mathcal{F}\{\delta(x - n\Delta_u) \delta(y - m\Delta_u)\} \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] e^{-j2\pi \mu n \Delta_u} e^{-j2\pi \nu m \Delta_u}. \end{aligned} \quad (4.31)$$

## 1. Quadrant 1: $\mu \geq 0$ and $\nu \geq 0$

In view of the relationship (from Eq. (4.14))

$$\frac{\Delta_u}{M_o\Delta_o} = \frac{\Delta_u}{M_u\Delta_u} = \frac{1}{M_u},$$

## 2. Quadrant 2: $\mu \leq 0$ and $\nu \geq 0$

$$\begin{aligned} & \mathbf{F}_u\left(\frac{-k_1}{M_o\Delta_o}, \frac{k_2}{M_o\Delta_o}\right) \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2} \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] e^{-j2\pi n(-k_1) / M_u} e^{-j2\pi m k_2 / M_u} \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] e^{-j2\pi n(M_u - k_1) / M_u} e^{-j2\pi m k_2 / M_u} \\ &= \mathbf{G}[M_u - k_1, k_2], \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2}, \end{aligned} \quad (4.33)$$

## 3. Quadrant 3: $\mu \leq 0$ and $\nu \leq 0$

$$\begin{aligned} & \mathbf{F}_u\left(\frac{-k_1}{M_o\Delta_o}, \frac{-k_2}{M_o\Delta_o}\right), \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2} \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] e^{-j2\pi n(-k_1) / M_u} e^{-j2\pi m(-k_2) / M_u} \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] e^{-j2\pi n(M_u - k_1) / M_u} e^{-j2\pi m(M_u - k_2) / M_u} \\ &= \mathbf{G}[M_u - k_1, M_u - k_2], \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2}. \end{aligned} \quad (4.34)$$

#### 4. Quadrant 4: $\mu \geq 0$ and $\nu \leq 0$

$$\begin{aligned} & \mathbf{F}_u\left(\frac{k_1}{M_o\Delta_o}, \frac{-k_2}{M_o\Delta_o}\right), \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2} \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] e^{-j2\pi n k_1 / M_u} e^{-j2\pi m (-k_2) / M_u} \\ &= \sum_{n=0}^{M_u-1} \sum_{m=0}^{M_u-1} g[n, m] e^{-j2\pi n k_1 / M_u} e^{-j2\pi m (M_u - k_2) / M_u} \\ &= \mathbf{G}[k_1, M_u - k_2], \quad 0 \leq k_1, k_2 \leq \frac{M_o - 1}{2}, \end{aligned} \quad (4.35)$$

The result given by Eqs. (4.32)–(4.35) states that the 2-D CSFT  $\mathbf{F}_u(\mu, \nu)$  of the upsampled image  $f_u(x, y)$ —when sampled at the discrete spatial frequency values defined by Eq. (4.24)—is the 2-D DFT of the sampled image  $g[n, m]$ . Also, the spectrum  $\mathbf{F}_u(\mu, \nu)$  of the sampled image consists of copies of the spectrum  $\mathbf{F}(\mu, \nu)$  of  $f(x, y)$  repeated every  $1/\Delta_u$  in both  $\mu$  and  $\nu$ , and also scaled by  $1/\Delta_u^2$ . Thus,

$$\mathbf{F}_u\left(\frac{k_1}{M_o\Delta_o}, \frac{k_2}{M_o\Delta_o}\right) = \frac{1}{\Delta_u^2} \mathbf{F}\left(\frac{k_1}{M_o\Delta_o}, \frac{k_2}{M_o\Delta_o}\right), \quad (4.36)$$

From Eq. (4.14),  $M_o\Delta_o = M_u\Delta_u$ . Combining Eq. (4.29) and Eq. (4.36) shows that

$$\mathbf{F}_u\left(\frac{k_1}{M_o\Delta_o}, \frac{k_2}{M_o\Delta_o}\right) = \frac{M_u^2}{M_o^2} \mathbf{F}_o\left(\frac{k_1}{M_o\Delta_o}, \frac{k_2}{M_o\Delta_o}\right). \quad (4.37)$$

Hence, the  $(M_o \times M_o)$  2-D DFT  $\mathbf{F}[k_1, k_2]$  of  $f[n, m]$  and the  $(M_u \times M_u)$  2-D DFT  $\mathbf{G}[k_1, k_2]$  of  $g[n, m]$  are related by

$$\begin{aligned} \mathbf{G}[k_1, k_2] &= \frac{M_u^2}{M_o^2} \mathbf{F}[k_1, k_2], \\ \mathbf{G}[M_u - k_1, k_2] &= \frac{M_u^2}{M_o^2} \mathbf{F}[M_o - k_1, k_2], \\ \mathbf{G}[k_1, M_u - k_2] &= \frac{M_u^2}{M_o^2} \mathbf{F}[k_1, M_o - k_2], \\ \mathbf{G}[M_u - k_1, M_u - k_2] &= \frac{M_u^2}{M_o^2} \mathbf{F}[M_o - k_1, M_o - k_2], \\ 0 \leq k_1, k_2 &\leq \frac{M_o - 1}{2}. \end{aligned} \quad (4.38)$$

This leaves  $\mathbf{G}[k_1, k_2]$  for  $M_o \leq k_1, k_2 \leq M_u - 1$  to be determined. But Eq. (4.38) shows that these values of  $\mathbf{G}[k_1, k_2]$  are samples of  $\mathbf{F}_u(\mu, \nu)$  at values of  $\mu, \nu$  for which this spectrum of the

sampled signal is zero, since sampling the original signal  $f(x, y)$  at above its Nyquist rate separates the copies of its spectrum, leaving bands of zero between copies. Thus

$$\mathbf{G}[k_1, k_2] = 0, \quad M_o \leq k_1, k_2 \leq M_u - 1. \quad (4.39)$$

#### 4-4 Implementation of Upsampling Using 2-D DFT in MATLAB

In MATLAB, both the image  $f[n, m]$  and its 2-D DFT are stored and displayed using the format shown in Fig. 3-25(d), wherein the origin is at the upper left-hand corner of the image, and the indices of the corner pixel are  $(1, 1)$ .

##### Image and 2-D DFT Notation

To avoid confusion between the common-image format (CIF) and the MATLAB format, we provide the following list of symbols and definitions:

CIF	MATLAB
Original image	$f[n, m]$
Upsampled image	$\mathbf{g}[n, m]$
2-D DFT of $f[n, m]$	$\mathbf{F}[k_1, k_2]$
2-D DFT of $g[n, m]$	$\mathbf{G}[k_1, k_2]$

As noted earlier in Section 3-9, when an image  $f[n, m]$  is stored in MATLAB as array  $\mathbf{x}(m', n')$ , the two sets of indices are related by

$$m' = m + 1, \quad (4.40a)$$

$$n' = n + 1. \quad (4.40b)$$

The indices get interchanged in orientation ( $n$  represents row number, whereas  $n'$  represents column number) and are shifted by 1. For example,  $f[0, 0] = \mathbf{x}(1, 1)$ , and  $f[0, 1] = \mathbf{x}(2, 1)$ . While the indices of the two formats are different, the contents of array  $\mathbf{x}(m', n')$  is identical with that of  $f[n, m]$ . That is, for an

$(M_o \times M_o)$  image

$$\begin{aligned} X(m', n') &= f[n, m] = \\ &\left[ \begin{array}{cccc} f[0, 0] & f[1, 0] & \cdots & f[M_o - 1, 0] \\ f[0, 1] & f[1, 1] & \cdots & f[M_o - 1, 1] \\ \vdots & \vdots & \vdots & \vdots \\ f[0, M_o - 1] & f[1, M_o - 1] & \cdots & f[M_o - 1, M_o - 1] \end{array} \right]. \end{aligned} \quad (4.41)$$

The MATLAB command  $\text{FX} = \text{fft}(X, M_o, M_o)$  computes the 2-D DFT  $\mathbf{F}[k_1, k_2]$  and stores it in array  $\text{FX}(k'_2, k'_1)$ :

$$\begin{aligned} \text{FX}(k'_2, k'_1) &= \mathbf{F}[k_1, k_2] = \\ &\left[ \begin{array}{cccc} \mathbf{F}[0, 0] & \mathbf{F}[1, 0] & \cdots & \mathbf{F}[M_o - 1, 0] \\ \mathbf{F}[0, 1] & \mathbf{F}[1, 1] & \cdots & \mathbf{F}[M_o - 1, 1] \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{F}[0, M_o - 1] & \mathbf{F}[1, M_o - 1] & \cdots & \mathbf{F}[M_o - 1, M_o - 1] \end{array} \right]. \end{aligned} \quad (4.42)$$

The goal of upsampling using the spatial-frequency domain is to compute  $g[n, m]$  from  $f[n, m]$  by computing  $\mathbf{G}[k_1, k_2]$  from  $\mathbf{F}[k_1, k_2]$  and then applying the inverse DFT to obtain  $g[n, m]$ . The details of the procedure are somewhat different depending on whether the array size parameter  $M$  is an odd integer or an even integer. Hence, we consider the two cases separately.

#### 4-4.1 $M_o = \text{Odd Integer}$

The recipe for upsampling using the 2-D DFT is as follows:

1. **Given:** image  $\{f[n, m], 0 \leq n, m \leq M_o - 1\}$ , as represented by Eq. (4.41).
2. **Compute:** the 2-D DFT  $\mathbf{F}[k_1, k_2]$  of  $f[n, m]$  using Eq. (4.20a) to obtain the array represented by Eq. (4.42).
3. **Create:** an upsampled  $(M_u \times M_u)$  array  $\mathbf{G}[k_1, k_2]$ , and then set its entries per the rules of Eq. (4.38).
4. **Compute:** the  $(M_u \times M_u)$  2-D inverse DFT of  $\mathbf{G}[k_1, k_2]$  to obtain  $g[n, m]$ .

In MATLAB, the array  $\text{FY}$  containing the 2-D DFT  $\mathbf{G}[k_1, k_2]$  is obtained from the array  $\text{FX}$  given by Eq. (4.42) by inserting  $(M_u - M_o)$  rows of zeros and an equal number of columns of

zeros in the “middle” of the array  $\text{FX}$ . The result is

$$\begin{aligned} \text{FY} &= \frac{M_u^2}{M_o^2} \times \\ &\left[ \begin{array}{ccccc} & & & (M_u - M_o) \text{ columns} & \\ & \mathbf{F}[0, 0] \dots \mathbf{F}\left[\frac{M_o-1}{2}, 0\right] & \overbrace{0 \dots 0 \dots 0} & \mathbf{F}\left[\frac{M_o+1}{2}, 0\right] \dots \mathbf{F}[M_o - 1, 0] & \\ & \vdots & \vdots & \vdots & \vdots \\ \mathbf{F}\left[0, \frac{M_o-1}{2}\right] \dots \mathbf{F}\left[\frac{M_o-1}{2}, \frac{M_o-1}{2}\right] & & \mathbf{F}\left[\frac{M_o+1}{2}, \frac{M_o-1}{2}\right] \dots \mathbf{F}\left[M_o - 1, \frac{M_o-1}{2}\right] & & \\ 0 & & 0 & & 0 \\ \{ & 0 & \vdots & 0 & \} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & & 0 & & 0 \\ \mathbf{F}\left[0, \frac{M_o+1}{2}\right] \dots \mathbf{F}\left[\frac{M_o-1}{2}, \frac{M_o+1}{2}\right] & & \mathbf{F}\left[\frac{M_o+1}{2}, \frac{M_o+1}{2}\right] \dots \mathbf{F}\left[M_o - 1, \frac{M_o+1}{2}\right] & & \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{F}[0, M_o - 1] \dots \mathbf{F}\left[\frac{M_o-1}{2}, M_o - 1\right] & \underbrace{0 \quad 0 \quad 0} & \mathbf{F}\left[M_o - 1, \frac{M_o+1}{2}\right] \dots \mathbf{F}\left[M_o - 1, M_o - 1\right] & & \end{array} \right]. \end{aligned} \quad (4.43)$$

► Note that the  $(M_u - M_o)$  columns of zeros start after entry  $\mathbf{F}[(M_o - 1)/2, 0]$ , and similarly the  $(M_u - M_o)$  rows of zeros start after  $\mathbf{F}[0, (M_o - 1)/2]$ . ◀

Once array  $\text{FY}$  has been established, the corresponding upsampled image  $g[n, m]$  is obtained by applying the MATLAB command  $\text{Y} = \text{real}(\text{iifft2}(\text{FY}, N, N))$ , where  $N = M_u$  and the “real” is needed to eliminate the imaginary part of  $\text{Y}$ , which may exist because of round-off error in the `iifft2`.

As a simple example, consider the  $(3 \times 3)$  array

$$\text{FX} = \begin{bmatrix} \mathbf{F}[0, 0] & \mathbf{F}[1, 0] & \mathbf{F}[2, 0] \\ \mathbf{F}[0, 1] & \mathbf{F}[1, 1] & \mathbf{F}[2, 1] \\ \mathbf{F}[0, 2] & \mathbf{F}[1, 2] & \mathbf{F}[2, 2] \end{bmatrix}. \quad (4.44a)$$

To generate a  $5 \times 5$  array  $\text{FY}$  we insert  $M_u - M_o = 5 - 3 = 2$  columns of zeros after element  $\mathbf{F}[(M_o - 1)/2, 0] = \mathbf{F}[1, 0]$ , and also 2 rows of zeros after  $\mathbf{F}[0, (M_o - 1)/2] = \mathbf{F}[0, 1]$ . The result is

$$\text{FY} = \frac{5^2}{3^2} \begin{bmatrix} \mathbf{F}[0, 0] & \mathbf{F}[1, 0] & 0 & 0 & \mathbf{F}[2, 0] \\ \mathbf{F}[0, 1] & \mathbf{F}[1, 1] & 0 & 0 & \mathbf{F}[2, 1] \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \mathbf{F}[0, 2] & \mathbf{F}[1, 2] & 0 & 0 & \mathbf{F}[2, 2] \end{bmatrix}. \quad (4.44b)$$

Application of the inverse 2-D DFT to  $\text{FY}$  generates array  $\text{Y}$  in MATLAB, which is equivalent in content to image  $g[n, m]$  in common-image format.

#### 4-4.2 $M_o = \text{Even Integer}$

For a real-valued  $(M_o \times M_o)$  image  $f[n, m]$  with  $M_o$  = an odd integer, conjugate symmetry is automatically satisfied for both  $\mathbf{F}[k_1, k_2]$ , the 2-D DFT of the original image, as well as for  $\mathbf{G}[k_1, k_2]$ , the 2-D DFT of the upsampled image. However, if  $M_o$  is an even integer, application of the recipe outlined in the preceding subsection will violate conjugate symmetry, so we need to modify it. Recall from Eq. (3.80) that for a real-valued image  $f[n, m]$ , conjugate symmetry requires that

$$\mathbf{F}^*[k_1, k_2] = \mathbf{F}[M_o - k_1, M_o - k_2], \quad 1 \leq k_1, k_2 \leq M_o - 1. \quad (4.45)$$

Additionally, in view of the definition for  $\mathbf{F}[k_1, k_2]$  given by Eq. (4.20a), the following two conditions should be satisfied:

$$\mathbf{F}[0, 0] = \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} f[n, m] = \text{real-valued}, \quad (4.46a)$$

$$\mathbf{F}\left[\frac{M_o}{2}, \frac{M_o}{2}\right] = \sum_{n=0}^{M_o-1} \sum_{m=0}^{M_o-1} (-1)^{n+m} f[n, m] = \text{real-valued}. \quad (4.46b)$$

In the preceding subsection, we inserted the appropriate number of rows and columns of zeros to obtain  $\mathbf{G}[k_1, k_2]$  from  $\mathbf{F}[k_1, k_2]$ . For  $M_o$  equal to an odd integer, the conditions represented by Eqs. (4.45) and (4.46) are satisfied for both  $\mathbf{F}[k_1, k_2]$  and  $\mathbf{G}[k_1, k_2]$ , but they are not satisfied for  $\mathbf{G}[k_1, k_2]$  when  $M_o$  is an even integer. To demonstrate why the simple zero-insertion procedure is problematic, let us consider the  $(4 \times 4)$  image

$$f[n, m] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 5 & 3 \\ 3 & 4 & 6 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix}. \quad (4.47a)$$

The  $(4 \times 4)$  2-D DFT  $\mathbf{F}[k_1, k_2]$  of  $f[n, m]$  is

$$\mathbf{F}[k_1, k_2] = \begin{bmatrix} 49 & -6-j3 & 3 & -6+j3 \\ -5-j4 & 2+j9 & -5+j2 & j \\ 1 & -4+j3 & -1 & -4-j3 \\ -5+j4 & -j & -5-j2 & 2-j9 \end{bmatrix}. \quad (4.47b)$$

This  $\mathbf{F}[k_1, k_2]$  has conjugate symmetry.

Let us suppose that we wish to upsample the  $(4 \times 4)$  image  $f[n, m]$  to a  $(5 \times 5)$  image  $g[n, m]$ . Inserting one row of zeros and

one column of zeros and multiplying by  $(5^2/3^2)$  would generate

$$\mathbf{G}[k_1, k_2] = \frac{5^2}{3^2} \begin{bmatrix} 49 & -6-j3 & 3 & -6+j3 \\ -5-j4 & 2+j9 & 0 & -5+j2 & j \\ 0 & 0 & 0 & 0 & 0 \\ 1 & -4+j3 & 0 & -1 & -4-j3 \\ -5+j4 & -j & 0 & -5-j2 & 2-j9 \end{bmatrix}. \quad (4.48)$$

This  $\mathbf{G}[k_1, k_2]$  array does not satisfy conjugate symmetry. Applying the inverse 2-D DFT to  $\mathbf{G}[k_1, k_2]$  generates the upsampled image  $g[n, m]$ :

$$g[n, m] = (5^2/3^2) \times$$

$$\begin{bmatrix} 0.64 & 1.05 + j0.19 & 1.64 - j0.30 & 2.39 + j0.30 & 2.27 - j0.19 \\ 1.05 + j0.19 & 2.01 + j0.22 & 2.85 - j0.20 & 2.8 - j0.13 & 1.82 - j0.19 \\ 1.64 - j0.3 & 2.38 - j0.44 & 3.9 + j0.46 & 3.16 + j0.08 & 1.31 + j0.39 \\ 2.39 + j0.30 & 2.37 - j0.066 & 2.88 + j0.36 & 2.04 - j0.9 & 0.84 + j0.12 \\ 2.27 - j0.19 & 1.89 - j0.25 & 1.33 + j0.26 & 0.83 + j0.08 & 1.18 + j0.22 \end{bmatrix},$$

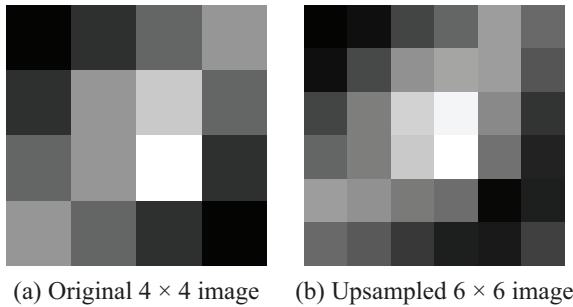
which is clearly incorrect; all of its elements should be real-valued because the original image  $f[n, m]$  is real-valued. Obviously, the upsampling recipe needs to be modified.

A simple solution is to split row  $\mathbf{F}[k_1, M_o/2]$  into 2 rows and to split column  $\mathbf{F}[M_o/2, k_2]$  into 2 columns, which also means that  $\mathbf{F}[M_o/2, M_o/2]$  gets split into 4 entries. The recipe preserves conjugate symmetry in  $\mathbf{G}[k_1, k_2]$ .

When applied to  $\mathbf{F}[k_1, k_2]$ , the recipe yields the  $(6 \times 6)$  array

$$G[k_1, k_2] = (6^2/4^2) \times \begin{bmatrix} \mathbf{F}[0, 0] & \mathbf{F}[1, 0] & \mathbf{F}[2, 0]/2 & 0 & \mathbf{F}[2, 0]/2 & \mathbf{F}[3, 0] \\ \mathbf{F}[0, 1] & \mathbf{F}[1, 1] & \mathbf{F}[2, 1]/2 & 0 & \mathbf{F}[2, 1]/2 & \mathbf{F}[3, 1] \\ \mathbf{F}[0, 2]/2 & \mathbf{F}[1, 2]/2 & \mathbf{F}[2, 2]/4 & 0 & \mathbf{F}[2, 2]/4 & \mathbf{F}[3, 2]/2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{F}[0, 2]/2 & \mathbf{F}[1, 2]/2 & \mathbf{F}[2, 2]/4 & 0 & \mathbf{F}[2, 2]/4 & \mathbf{F}[3, 2]/2 \\ \mathbf{F}[0, 3] & \mathbf{F}[1, 3] & \mathbf{F}[2, 3]/2 & 0 & \mathbf{F}[2, 3]/2 & \mathbf{F}[3, 3] \end{bmatrix} = \begin{bmatrix} 49 & -6-j3 & 1.5 & 0 & 1.5 & -6+j3 \\ -5-j4 & 2+j9 & -2.5+j1 & 0 & -2.5+j1 & j \\ 0.5 & -2+j1.5 & -0.25 & 0 & -0.25 & -2-j1.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & -2+j1.5 & -0.25 & 0 & -0.25 & -2-j1.5 \\ -5-j4 & -j & -j0.5 & 0 & -2.5-j1 & 2-j9 \end{bmatrix}. \quad (4.49)$$

Application of the inverse 2-D DFT to  $\mathbf{G}[k_1, k_2]$  yields the



**Figure 4-4** Comparison of original and upsampled images.

**Answer:** The 2-point DFT  $\mathbf{X}[k] = \{8+4, 8-4\} = \{12, 4\}$ . Since  $M_o = 2$  is even, we split 4 and insert a zero in the middle, and multiply by 4/2 to get  $\mathbf{Y}[k] = \{12, 2, 0, 2\}$ . The inverse 4-point DFT is

$$\begin{aligned}y[0] &= \frac{2}{4}(12 + 2 + 0 + 2) = 8, \\y[1] &= \frac{2}{4}(12 + 2j - 0 - 2j) = 6, \\y[2] &= \frac{2}{4}(12 - 2 + 0 - 2) = 4, \\y[3] &= \frac{2}{4}(12 - 2j + 0 + 2j) = 6.\end{aligned}$$

$$y[n] = \{\underline{8}, 6, 4, 6\}.$$

## 4-5 Downsampling

Downsampling is the inverse operation to upsampling. The objective of downsampling is to reduce the array size  $f[n, m]$  of an image from  $(M_o \times M_o)$  samples down to  $(M_d \times M_d)$ , with  $M_d < M_o$ . The original image  $f[n, m]$  and the downsampled image  $g[n, m]$  are defined as:

$$\begin{aligned}\text{Original image } f[n, m] &= \{f(n\Delta_o, m\Delta_o), 0 \leq n, m \leq M_o - 1\}, \\ \text{Downsampled image } g[n, m] &= \{f(n\Delta_d, m\Delta_d), 0 \leq n, m \leq M_d - 1\}.\end{aligned}$$

Both images are sampled versions of some continuous-space image  $f(x, y)$ , with image  $f[n, m]$  sampled at a sampling interval  $\Delta_o$  that satisfies the Nyquist rate, and the downsampled image  $g[n, m]$  is sampled at  $\Delta_d$ , with  $\Delta_d > \Delta_o$ , so it is unlikely that  $g[n, m]$  satisfies the Nyquist rate.

The goal of downsampling is to compute  $g[n, m]$  from  $f[n, m]$ . That is, we compute a coarser-discretized  $M_d \times M_d$  image  $g[n, m]$  from the finer-discretized  $M_o \times M_o$  image  $f[n, m]$ . Applications of downsampling include computation of “thumbnail” versions of images, as demonstrated in Example 4-1, and shrinking images to fit into a prescribed space, such as columns in a textbook.

### 4-5.1 Aliasing

It might seem that downsampling by, say, two, meaning that  $M_d = M_o/2$  (assuming  $M_o$  is even), could be easily accomplished by simply deleting every even-indexed (or odd-indexed) row and column of  $f[n, m]$ . Deleting every other row and column of

upsampled image

$$\begin{aligned}g[n, m] &= \frac{6^2}{4^2} \begin{bmatrix} 0.44 & 0.61 & 1.06 & 1.33 & 1.83 & 1.38 \\ 0.61 & 1.09 & 1.73 & 1.91 & 1.85 & 1.21 \\ 1.06 & 1.55 & 2.31 & 2.58 & 1.66 & 0.90 \\ 1.33 & 1.55 & 2.22 & 2.67 & 1.45 & 0.78 \\ 1.83 & 1.72 & 1.52 & 1.42 & 0.54 & 0.73 \\ 1.38 & 1.25 & 0.94 & 0.76 & 0.72 & 1.04 \end{bmatrix} \\ &= \begin{bmatrix} 1.0 & 1.37 & 2.39 & 3.0 & 4.12 & 3.11 \\ 1.37 & 2.46 & 3.90 & 4.30 & 4.16 & 2.72 \\ 2.39 & 3.49 & 5.20 & 5.81 & 3.74 & 2.03 \\ 3.0 & 3.49 & 5.00 & 6.01 & 3.26 & 1.76 \\ 4.12 & 3.87 & 3.42 & 3.20 & 1.22 & 1.64 \\ 3.11 & 2.81 & 2.12 & 1.71 & 1.62 & 2.34 \end{bmatrix}, \quad (4.50)\end{aligned}$$

which is entirely real-valued, as it should be. The original  $(4 \times 4)$  image given by Eq. (4.47a) and the upsampled  $(6 \times 6)$  image given by Eq. (4.50) are displayed in Fig. 4-4. The two images, which bear a close resemblance, have the same physical size but different-sized pixels.

**Exercise 4-2:** Upsample the length-2 signal  $x[n] = \{\underline{8}, 4\}$  to a length-4 signal  $y[n]$ .

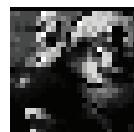
$f[n,m]$  is called **decimation** by two. Decimation by two would give the result of sampling  $f(x,y)$  every  $2\Delta_o$  instead of every  $\Delta_o$ . But if sampling at  $S = 1/2\Delta_o$  is below the Nyquist rate, the decimated image  $g[n,m]$  is **aliased**. The effect of aliasing on the spectrum of  $g[n,m]$  can be understood in 1-D from [Fig. 2-6\(b\)](#). The copies of the spectrum of  $f(x,y)$  produced by sampling overlap one another, so the high-frequency parts of the signal become distorted. Example 4-1 gives an illustration of aliasing in 2-D.

### Example 4-1: Aliasing

The  $200 \times 200$  clown image shown in [Fig. 4-5\(a\)](#) was decimated to the  $25 \times 25$  image shown in [Fig. 4-5\(b\)](#). The decimated image is a poor replica of the original image and could not function as a thumbnail image.



(a) Original



(b) Decimated  $25 \times 25$  version

**Figure 4-5** Clown image: (a) original  $200 \times 200$  and (b) decimated  $25 \times 25$  version.

## 4-6 Antialias Lowpass Filtering

Clearly decimation alone is not sufficient to obtain a downsampled image that looks like a demagnified original image. To avoid aliasing, it is necessary to lowpass filter the image before decimation, eliminating the high spatial frequency components of the image, so that when the filtered image is decimated, the copies of the spectra do not overlap. In 1-D, in [Fig. 2-6\(b\)](#), had the spectrum been previously lowpass filtered with cutoff frequency  $S/2$  Hz, the high-frequency parts of the spectrum would no longer overlap after sampling. This is called **antialias filtering**.

The same concept applies in discrete space (and time). The periodicity of spectra induced by sampling becomes the periodicity of the DSFT and DTFT with periods of  $2\pi$  in  $(\Omega_1, \Omega_2)$  and  $\Omega$ , respectively. Lowpass filtering can be accomplished by setting certain high-spatial-frequency portions of the 2-D DFT to zero. The purpose of this lowpass filtering is now to eliminate the high-spatial-frequency parts of the discrete-space spectrum, prior to decimation. This eliminates aliasing, as demonstrated in Example 4-2.

### Example 4-2: Antialiasing

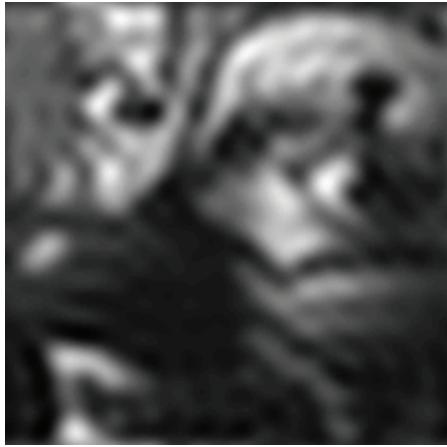
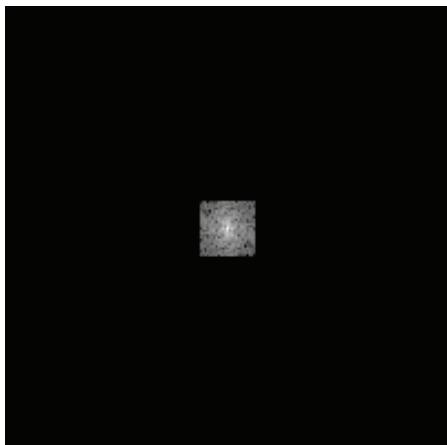
The  $200 \times 200$  clown image shown in [Fig. 4-5\(a\)](#) was first lowpass-filtered to the image shown in [Fig. 4-6\(a\)](#), with the spectrum shown in [Fig. 4-6\(b\)](#), then decimated to the  $25 \times 25$  image shown in [Fig. 4-6\(c\)](#). The decimated image is now a good replica of the original image and could function as a thumbnail image. The decimated image pixel values in [Fig. 4-6\(c\)](#) are all equal to certain pixel values in the lowpass-filtered image in [Fig. 4-6\(a\)](#).

► A MATLAB code for this example is available on the book website. ◀

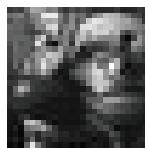
### 4-6.1 Downsampling in the 2-D DFT Domain

The antialiasing approach works well when downsampling by an integer factor  $L$ , since decimation can be easily performed by keeping only every  $L$ th row and column of the antialiased-filtered image. But downsampling by a non-integer factor  $M_d/M_o$  must be performed entirely in the 2-D DFT domain, as follows.

Antialias lowpass filtering is performed by setting to zero the  $M_o - M_d$  center rows and columns of the 2-D DFT, in

(a) Lowpass-filtered  $200 \times 200$  image

(b) Spectrum of lowpass-filtered image

(c) The decimated  $25 \times 25$  image is not aliased

**Figure 4-6** Clown image: (a) lowpass-filtered  $200 \times 200$  image, (b) spectrum of image in (a), and (c) unaliased, decimated version of (a).

MATLAB depiction. Decimation is then performed by deleting those  $(M_o - M_d)$  center rows and columns of the 2-D DFT. Of course, there is no reason to set to zero rows and columns that will be deleted anyways, so in practice the first step need not be performed. By “center rows and columns” we mean those rows and columns with DFT indices  $k$  (horizontal or vertical; add one to all DFT indices to get MATLAB indices) in the  $(M_o \times M_o)$  2-D DFT of the  $(M \times M)$  original image  $f[n, m]$ :

- For  $M_d$  odd:  $(M_d + 1)/2 \leq k \leq M_o - (M_d + 1)/2$ .
- For  $M_d$  even:  $M_d/2 \leq k \leq M_o - M_d/2$ , then insert a row or column of zeros at index  $k = M_d/2$ .

Note that there is no need to subdivide some values of  $\mathbf{F}[k_1, k_2]$  to preserve conjugate symmetry.

The procedure is best illustrated by an example.

### Example 4-3: Downsampling

The goal is to downsample the MATLAB  $(6 \times 6)$  image array:

$$\mathbf{x}(m, n) = \begin{bmatrix} 3 & 1 & 4 & 1 & 5 & 9 \\ 2 & 6 & 5 & 3 & 5 & 8 \\ 9 & 7 & 9 & 3 & 2 & 3 \\ 8 & 4 & 6 & 2 & 6 & 4 \\ 3 & 3 & 8 & 3 & 2 & 7 \\ 9 & 5 & 0 & 2 & 8 & 8 \end{bmatrix} \quad (4.51)$$

The corresponding magnitudes of the  $(6 \times 6)$  2-D DFT of this array in MATLAB is

$$|\mathbf{FX}(k_1, k_2)| = \begin{bmatrix} 173.00 & 23.81 & 20.66 & 15.00 & 20.66 & 23.81 \\ 6.93 & 25.00 & 7.55 & 14.00 & 7.94 & 21.93 \\ 11.14 & 19.98 & 16.09 & 15.10 & 14.93 & 4.58 \\ 9.00 & 16.09 & 19.98 & 1.00 & 19.98 & 16.09 \\ 11.14 & 4.58 & 14.93 & 15.10 & 16.09 & 19.98 \\ 6.93 & 21.93 & 7.94 & 14.00 & 7.55 & 25.00 \end{bmatrix}. \quad (4.52)$$

Setting to zero the middle three columns of the 2-D DFT magnitudes in MATLAB depiction gives the magnitudes

$$|\mathbf{FG}(k_1, k_2)| = \begin{bmatrix} 173.00 & 23.81 & 0 & 0 & 0 & 23.81 \\ 6.93 & 25.00 & 0 & 0 & 0 & 21.93 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 6.93 & 21.93 & 0 & 0 & 0 & 25.00 \end{bmatrix}. \quad (4.53)$$

Deleting the zero-valued rows and columns in the 2-D DFT magnitudes in MATLAB depiction gives the magnitudes

$$|FG(k_1, k_2)| = \begin{bmatrix} 173.00 & 23.82 & 23.82 \\ 6.93 & 25.00 & 21.93 \\ 6.93 & 21.93 & 25.00 \end{bmatrix}. \quad (4.54)$$

Multiplying by  $M_d^2/M_o^2 = 3^2/6^2 = 1/4$  and taking the inverse 2-D DFT gives

$$g[m, n] = \begin{bmatrix} 5.83 & 1.58 & 6.00 \\ 6.08 & 6.33 & 3.00 \\ 6.25 & 3.5 & 4.67 \end{bmatrix}. \quad (4.55)$$

This is the result we would have gotten by decimating the antialiased lowpass-filtered original image, but it was performed entirely in the 2-D DFT domain.

► A MATLAB code for this example is available on the book website. ◀

**Concept Question 4-2:** Why must we delete rows and columns of the 2-D DFT array to perform downsampling?

## 4-7 B-Splines Interpolation

In the preceding sections we examined several different image interpolation methods, some of which perform the interpolation directly in the spatial domain, and others that perform the interpolation in the spatial frequency domain. Now, we introduce yet another method, known as the **B-splines interpolation** method, with the distinguishing feature that it is the method most commonly used for image interpolation. Unlike with downsampling, B-spline interpolation has no aliasing issues when the sampling interval  $\Delta$  is too large. Moreover, unlike with upsampling, B-spline interpolation need not result in blurred images.

B-splines are a family of piecewise polynomial functions, with each polynomial piece having a degree  $N$ , where  $N$  is a non-negative integer. As we will observe later on in this section, a B-spline of order zero is equivalent to the nearest-neighbor interpolation method of Section 3-5.1, but it is simpler to implement than the sinc interpolation formula. Interpolation with B-splines of order  $N = 1$  generates linear interpolation, which is used in computer graphics. Another popular member of the B-spline interpolation family is cubic interpolation, corresponding to  $N = 3$ . Cubic spline interpolation is used in Adobe®

Photoshop® and through the command `imresize`.

### 4-7.1 B-Splines

Splines are piecewise-polynomial functions whose polynomial coefficients change at half-integer or integer values of the independent variable, called **knots**, so that the function and some of its derivatives are continuous at each knot. In 1-D, a B-spline  $\beta_N(t)$  of order  $N$  is a piecewise polynomial of degree  $N$ , centered at  $t = 0$ . The **support** of  $\beta_N(t)$ , which is the interval outside of which  $\beta_N(t) = 0$ , extends between  $-(N+1)/2$  and  $+(N+1)/2$ .

► Hence, the **duration** of  $\beta_N(t)$  is  $(N+1)$ . ◀

Formally, the B-spline function  $\beta_N(t)$  is defined as

$$\beta_N(t) = \int_{-\infty}^{\infty} \left[ \frac{\sin(\pi f)}{\pi f} \right]^{N+1} e^{j2\pi ft} df, \quad (4.56)$$

which is equivalent to the inverse Fourier transform of  $\text{sinc}^{N+1}(f)$ . Recognizing that (a) the inverse Fourier transform of  $\text{sinc}(t)$  is a rectangle function and (b) multiplication in the frequency domain is equivalent to convolution in the time domain, it follows that

$$\beta_N(t) = \underbrace{\text{rect}(t) * \dots * \text{rect}(t)}_{N+1 \text{ times}}, \quad (4.57)$$

with

$$\text{rect}(t) = \begin{cases} 1 & \text{for } |t| < 1/2, \\ 0 & \text{for } |t| > 1/2. \end{cases} \quad (4.58)$$

Application of Eq. (4.57) for  $N = 0, 1, 2$ , and  $3$  leads to:

$$\beta_0(t) = \text{rect}(t) = \begin{cases} 1 & \text{for } |t| < 1/2, \\ 0 & \text{for } |t| > 1/2, \end{cases} \quad (4.59)$$

$$\beta_1(t) = \beta_0(t) * \beta_0(t) = \begin{cases} 1 - |t| & \text{for } |t| < 1, \\ 0 & \text{for } |t| > 1, \end{cases} \quad (4.60)$$

$$\begin{aligned} \beta_2(t) &= \beta_1(t) * \beta_0(t) \\ &= \begin{cases} 3/4 - t^2 & \text{for } 0 \leq |t| \leq 1/2, \\ \frac{1}{2} (3/2 - |t|)^2 & \text{for } 1/2 \leq |t| \leq 3/2, \\ 0 & \text{for } |t| > 3/2, \end{cases} \end{aligned} \quad (4.61)$$

$$\beta_3(t) = \beta_2(t) * \beta_0(t)$$

$$= \begin{cases} 2/3 - t^2 + |t|^3/2 & \text{for } |t| \leq 1, \\ (2 - |t|)^3/6 & \text{for } 1 \leq |t| \leq 2, \\ 0 & \text{for } |t| > 2. \end{cases} \quad (4.62)$$

Note that in all cases,  $\beta_N(t)$  is continuous over its full duration. For  $N \geq 1$ , the B-spline function  $\beta_N(t)$  is continuous and differentiable  $(N-1)$  times at all times  $t$ . For  $\beta_2(t)$ , the function is continuous across its full interval  $(-3/2, 3/2)$ , including at  $t = 1/2$ . Similarly,  $\beta_3(t)$  is continuous over its interval  $(-2, 2)$ , including at  $t = 1$ .

Plots of the B-splines of order  $N = 0, 1, 2$ , and  $3$  are displayed in Fig. 4-7. From the **central limit theorem** in the field of probability, we know that convolving a function with itself repeatedly makes the function resemble a Gaussian. This is evident in the present case as well.

From the standpoint of 1-D and 2-D interpolation of signals and images, the significance of B-splines is in how we can use them to express a signal or image. To guide us through the process, let us assume we have 6 samples  $x(n\Delta)$ , as shown in Fig. 4-8, extending between  $t = 0$  and  $t = 5\Delta$ . Our objective is to interpolate between these 6 points so as to obtain a continuous function  $x(t)$ . An important constraint is to ensure that  $x(t) = x(n\Delta)$  at the 6 discrete times  $n\Delta$ .

For a B-spline of a specified order  $N$ , the interpolation is realized by expressing the desired interpolated signal  $x(t)$  as a linear combination of time-shifted B-splines, all of order  $N$ :

$$x(t) = \sum_{m=-\infty}^{\infty} c[m] \beta_N \left( \frac{t}{\Delta} - m \right). \quad (4.63)$$

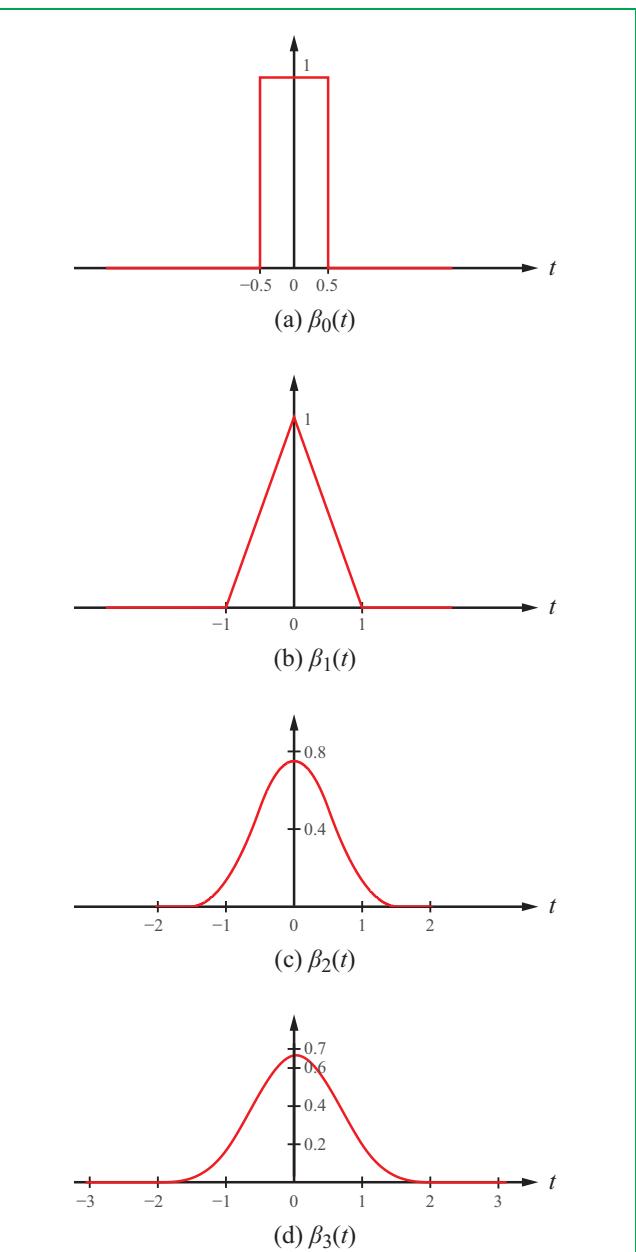
Here,  $\beta_N \left( \frac{t}{\Delta} - m \right)$  is the B-spline function  $\beta_N(t)$ , with  $t$  scaled by the sampling interval  $\Delta$  and delayed by a scaled time integer  $m$ .

► The support of  $\beta_N \left( \frac{t}{\Delta} - m \right)$  is

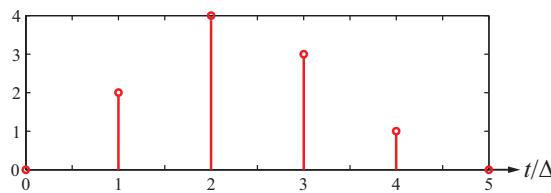
$$\left( m - \frac{N+1}{2} \right) < \frac{t}{\Delta} < \left( m + \frac{N+1}{2} \right). \quad (4.64)$$

That is,  $\beta_N \left( \frac{t}{\Delta} - m \right) = 0$  outside that interval. ◀

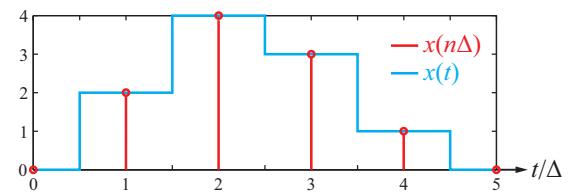
Associated with each value of  $m$  is a constant coefficient  $c[m]$  whose value is related to the sampled values  $x(n\Delta)$  and the order  $N$  of the B-spline. More specifically, the values of  $c[m]$  have to be chosen such that the aforementioned constraint requiring that  $x(t) = x(n\Delta)$  at discrete times  $t = n\Delta$  is satisfied. The process is



**Figure 4-7** Plots of  $\beta_N(t)$  for  $N = 0, 1, 2$ , and  $3$ .



**Figure 4-8** Samples  $x(n\Delta)$  to be interpolated into  $x(t)$ .



**Figure 4-9** B-spline interpolation for  $N=0$ .

described in forthcoming subsections.

Since each B-spline is a piecewise polynomial of order  $N$ , continuous and differentiable  $(N-1)$  times, any linear combination of time-shifted B-splines also constitutes a piecewise polynomial of order  $N$ , and will also be continuous and differentiable. Thus, the B-splines form a basis—hence, the “B” in their name—and where the basis is used to express  $x(t)$ , as in Eq. (4.63),  $x(t)$  is continuous and differentiable  $(N-1)$  times at the knots  $t = m\Delta$  if  $N$  is odd and at the knots  $t = (m+1/2)\Delta$  if  $N$  is even. This feature of B-splines makes them suitable for interpolation, as well as for general representation of signals and images.

**Exercise 4-3:** Show that the area under  $\beta_N(t)$  is 1.

**Answer:**  $\int_{-\infty}^{\infty} \beta_N(t) dt = \mathbf{B}(0)$  by entry #11 in **Table 2-4**. Set  $f = 0$  in  $\text{sinc}^N(f)$  and use  $\text{sinc}(0) = 1$ .

**Exercise 4-4:** Why does  $\beta_N(t)$  look like a Gaussian function for  $N \geq 3$ ?

**Answer:** Because  $\beta_N(t)$  is  $\text{rect}(t)$  convolved with itself  $(N+1)$  times. By the central limit theorem of probability, convolving any square-integrable function with itself results in a function resembling a Gaussian.

**Exercise 4-5:** Show that the support of  $\beta_N(t)$  is  $-(N+1)/2 < t < (N+1)/2$ .

**Answer:** The support of  $\beta_N(t)$  is the interval outside of which  $\beta_N(t) = 0$ . The support of  $\text{rect}(t)$  is  $-1/2 < t < 1/2$ .  $\beta_N(t)$  is  $\text{rect}(t)$  convolved with itself  $N+1$  times, which has duration  $N+1$  centered at  $t = 0$ .

## 4-7.2 $N = 0$ : Nearest-Neighbor Interpolation

For  $N = 0$ , Eqs. (4.63) and (4.64) lead to

$$\begin{aligned} x(t) &= \sum_{m=-\infty}^{\infty} c[m] \beta_0\left(\frac{t}{\Delta} - m\right) \\ &= \sum_{m=-\infty}^{\infty} c[m] \text{rect}\left(\frac{t}{\Delta} - m\right) \\ &= \sum_{m=-\infty}^{\infty} c[m] \begin{cases} 1 & \text{for } m - \frac{1}{2} < \frac{t}{\Delta} < m + \frac{1}{2}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \quad (4.65)$$

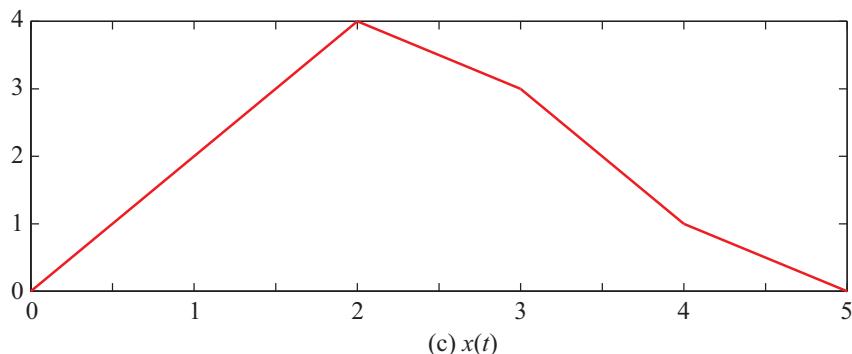
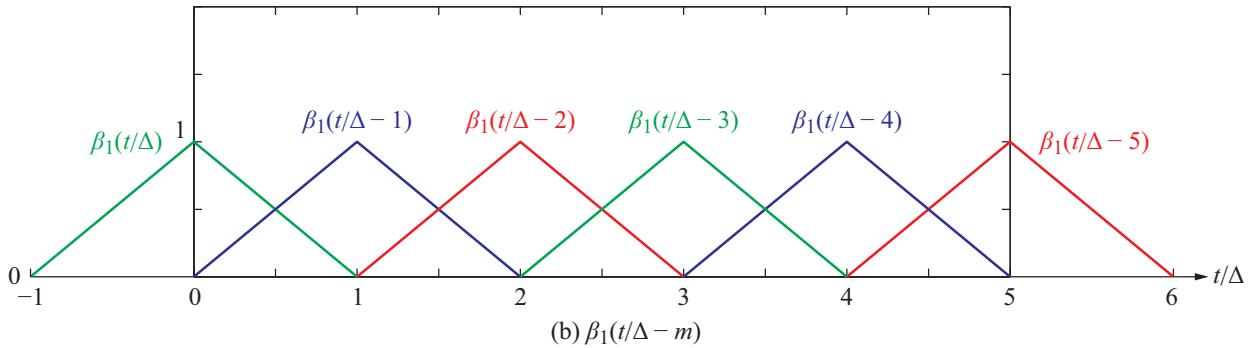
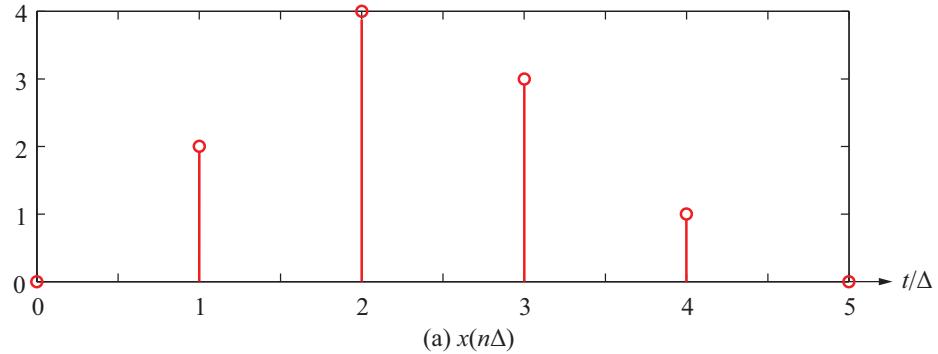
The expression given in Eq. (4.65) consists of a series of adjoining, but not overlapping, rectangle functions. For  $m = 0$ ,  $\text{rect}\left(\frac{t}{\Delta}\right)$  is centered at  $\frac{t}{\Delta} = 0$  and extends over the range  $-\frac{1}{2} < \frac{t}{\Delta} < \frac{1}{2}$ . Similarly, for  $m = 1$ ,  $\text{rect}\left(\frac{t}{\Delta} - 1\right)$  is centered at  $\frac{t}{\Delta} = 1$  and extends over  $\frac{1}{2} < \frac{t}{\Delta} < \frac{3}{2}$ . Hence, to satisfy the constraint that  $x(t) = x(n\Delta)$  at the sampled locations  $t = n\Delta$ , we need to set  $m = n$  and  $c[m] = x(n\Delta)$ :

$$x(t) = x(n\Delta) \text{rect}\left(\frac{t}{\Delta} - n\right), \quad n - \frac{1}{2} < \frac{t}{\Delta} < n + \frac{1}{2}. \quad (4.66)$$

The interpolated function  $x(t)$  is shown in **Fig. 4-9**, along with the 6 samples  $\{x(n\Delta)\}$ . The B-spline representation given by Eq. (4.66) is a **nearest-neighbor (NN)** interpolation:  $x(t)$  in the interval  $\{n\Delta \leq t \leq (n+1)\Delta\}$  is set to the closer (in time) of  $x(n\Delta)$  and  $x((n+1)\Delta)$ . The resulting  $x(t)$  is a piecewise constant, as shown in **Fig. 4-9**.

## 4-7.3 Linear Interpolation

For  $N = 1$ , the B-spline  $\beta_1\left(\frac{t}{\Delta} - m\right)$  assumes the shape of a triangle (**Fig. 4-7(b)**) centered at  $t/\Delta = m$ . **Figure 4-10(b)** displays the triangles centered at  $t/\Delta = m$  for  $m = 0$  through 5. Also displayed in the same figure are the values of  $x(n\Delta)$ . To



**Figure 4-10** B-spline linear interpolation ( $N = 1$ ).

satisfy Eq. (4.63) for  $N = 1$ , namely

$$x(t) = \sum_{m=-\infty}^{\infty} c[m] \beta_1 \left( \frac{t}{\Delta} - m \right), \quad (4.67)$$

as well as meet the condition that  $x(t) = x(n\Delta)$  at the 6 given points, we should set  $m = n$  and select

$$c[m] = c[n] = x(\Delta n) \quad (\text{N} = 1).$$

Consequently, for any integer  $n$ ,  $x(t)$  at time  $t$  between  $n\Delta$  and  $(n+1)\Delta$  is a weighted average given by

$$x(t) = x(n\Delta) \left( (n+1) - \frac{t}{\Delta} \right) + x((n+1)\Delta) \left( \frac{t}{\Delta} - n \right). \quad (4.68a)$$

The associated duration is

$$n\Delta \leq t \leq (n+1)\Delta. \quad (4.68b)$$

Application of the B-spline linear interpolation to the given samples  $x(n\Delta)$  leads to the continuous function  $x(t)$  shown in Fig. 4-10(c). The linear interpolation amounts to setting  $\{x(t), n\Delta \leq t \leq (n+1)\Delta\}$  to lie on the straight line connecting  $x(n\Delta)$  to  $x((n+1)\Delta)$ .

**Exercise 4-6:** Given the samples

$$\{x(0), x(\Delta), x(2\Delta), x(3\Delta)\} = \{7, 4, 3, 2\},$$

compute  $x(\Delta/3)$  by interpolation using: (a) nearest neighbor; (b) linear.

**Answer:** (a)  $\Delta/3$  is closer to 0 than to  $\Delta$ , so

$$x(\Delta/3) = x(0) = 7.$$

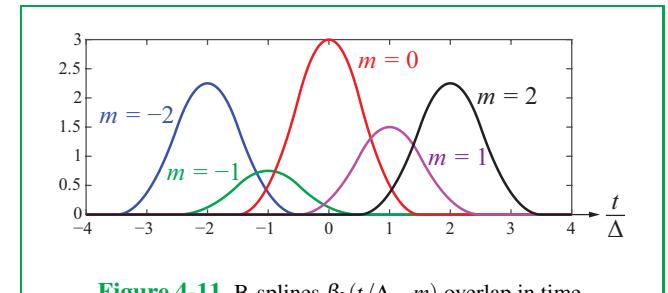
(b)

$$x(\Delta/3) = (2/3)x(0) + (1/3)x(\Delta) = (2/3)(7) + (1/3)(4) = 6.$$

#### 4-7.4 Quadratic Interpolation

For  $N \geq 2$ , interpolation using B-splines becomes more complicated than for  $N = 0$  and 1 because the supports of the basis functions  $\{\beta_N(\frac{t}{\Delta} - m)\}$  overlap in time for different values of  $m$ , as shown in Fig. 4-11 for  $N = 2$ . Unlike the cases  $N = 0$  and  $N = 1$ , wherein we set  $c[m] = x(m\Delta)$ , now  $c[m]$  is related to values of more than one of the discrete samples  $\{x(n\Delta)\}$ .

For  $N \geq 2$ , the relationships between coefficients  $\{c[m]\}$  and



**Figure 4-11** B-splines  $\beta_2(t/\Delta - m)$  overlap in time.

samples  $\{x(n\Delta)\}$  can be derived by starting with Eq. (4.63),

$$x(t) = \sum_{m=-\infty}^{\infty} c[m] \beta_N \left( \frac{t}{\Delta} - m \right), \quad (4.69)$$

and then setting  $t = n\Delta$ , which gives

$$x(n\Delta) = \sum_{m=-\infty}^{\infty} c[m] \beta_N(n - m) = c[n] * \beta_N(n), \quad (4.70)$$

where use was made of the discrete-time convolution relation given by Eq. (2.71a).

For  $N = 2$ , Eq. (4.61) indicates that  $\beta_2(n) \neq 0$  only for integers  $n = \{-1, 0, 1\}$ . Hence, the discrete-time convolution given by Eq. (4.70) simplifies to

$$\begin{aligned} x(n\Delta) &= c[n-1] \beta_2(1) + c[n] \beta_2(0) + c[n+1] \beta_2(-1) \\ &= \frac{1}{8} c[n-1] + \frac{3}{4} c[n] + \frac{1}{8} c[n+1] \quad (\text{N} = 2). \end{aligned} \quad (4.71)$$

In the second step, the constant coefficients were computed using Eq. (4.61) for  $\beta_2(t)$ . The sum truncates because  $\beta_2(t) = 0$  for  $|t| \geq 3/2$ , so only three basis functions overlap at any specific time  $t$ , as is evident in Fig. 4-11.

Similarly, for  $N = 3$ ,  $\beta_3(t) = 0$  for  $|t| \geq 2$ , which also leads to the sum of three terms:

$$\begin{aligned} x(n\Delta) &= c[n-1] \beta_3(1) + c[n] \beta_2(0) + c[n+1] \beta_3(-1) \\ &= \frac{1}{6} c[n-1] + \frac{4}{6} c[n] + \frac{1}{6} c[n+1] \quad (\text{N} = 3). \end{aligned} \quad (4.72)$$

If  $N$  is increased beyond 3, the number of terms increases, but the proposed method of solution to determine the values of  $c[n]$  remains the same. Specifically, we offer the following recipe:

**(1)** Delay  $\{x(n\Delta)\}$  to  $\{\tilde{x}(n\Delta)\}$  to make it causal. Compute the  $N_0$ th-order DFT  $\mathbf{X}[k]$  of  $\{\tilde{x}(n\Delta)\}$ , where  $N_0$  is the number of samples  $\{x(n\Delta)\}$ .

(2) Delay  $\{\beta_2(-1), \beta_2(0), \beta_2(1)\}$  by 1 to  $\{\tilde{\beta}_2(t)\}$  to make it causal. Compute the  $N_0$ th-order DFT  $\mathbf{B}[k]$  of

$$\{\beta_2(-1), \beta_2(0), \beta_2(1)\} = \left\{ \frac{1}{8}, \frac{3}{4}, \frac{1}{8} \right\}.$$

(3) Compute the  $N_0$ th-order inverse DFT to determine  $c[n]$ :

$$c[n] = \text{DFT}^{-1} \left( \frac{\mathbf{X}[k]}{\mathbf{B}[k]} \right). \quad (4.73)$$

#### Example 4-4: Quadratic Spline Interpolation

**Figure 4-12(a)** displays samples  $\{x(n\Delta)\}$  with  $\Delta = 1$ . Obtain an interpolated version  $x(t)$  using quadratic splines.

**Solution:** From **Fig. 4-12(a)**, we deduce that  $x(n)$  has 6 nonzero samples and is given by

$$\begin{aligned} x(n) &= \{x(-3), x(-2), x(-1), x(0), x(1), x(2)\} \\ &= \{3, 19, 11, \underline{17}, 26, 4\}. \end{aligned}$$

As noted earlier,

$$\beta_2(n) = \{\beta_2(-1), \beta_2(0), \beta_2(1)\} = \left\{ \frac{1}{8}, \frac{3}{4}, \frac{1}{8} \right\}.$$

Inserting  $x(n)$  and  $\beta(n)$  in Eq. (4.70) establishes the convolution problem

$$\{3, 19, 11, \underline{17}, 26, 4\} = \left\{ \frac{1}{8}, \frac{3}{4}, \frac{1}{8} \right\} * c[n].$$

Following the solution recipe outlined earlier—and demonstrated in Section 2-9—the deconvolution solution is

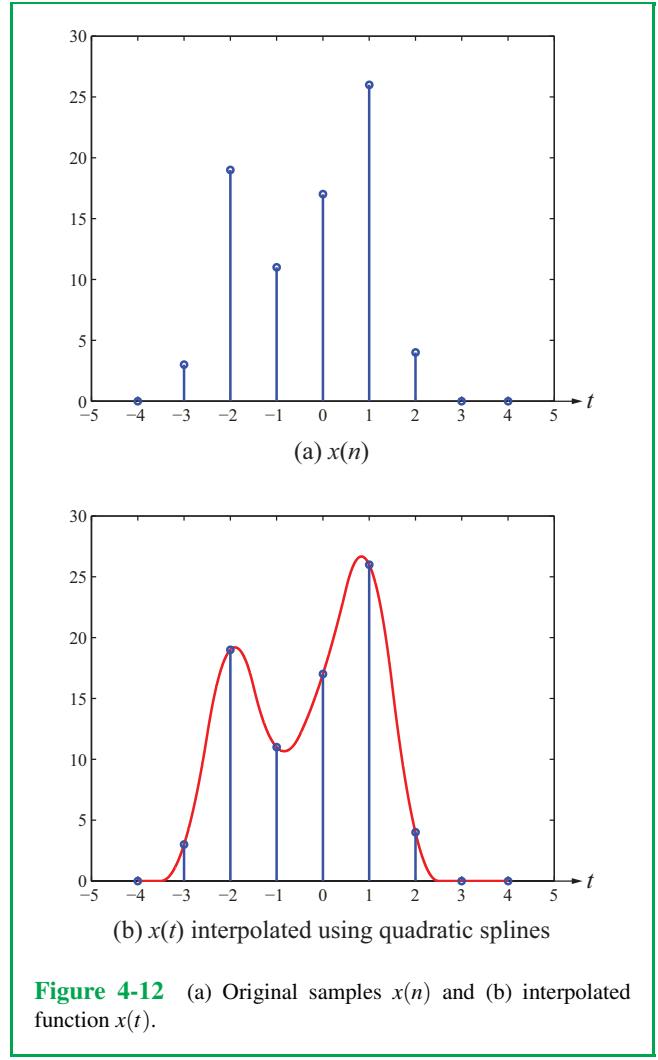
$$c[n] = \{24, 8, \underline{16}, 32\},$$

and, therefore, the interpolated continuous function is

$$x(t) = 24\beta_2(t+2) + 8\beta_2(t+1) + 16\beta_2(t) + 32\beta_2(t-1).$$

A plot of  $x(t)$  is shown in **Fig. 4-12(b)**. It is evident that  $x(t)$  has the same values as  $x(n)$  at  $t = \{-3, -2, -1, 0, 1, 2\}$ .

► Note: The MATLAB code for solving Example 4-4 is available on the book website. ◀



**Figure 4-12** (a) Original samples  $x(n)$  and (b) interpolated function  $x(t)$ .

**Concept Question 4-3:** What is the difference between zero-order-spline interpolation and nearest-neighbor interpolation?

**Concept Question 4-4:** Why use cubic interpolation, when quadratic interpolation produces smooth curves?

**Concept Question 4-5:** Why do quadratic and cubic interpolation require computation of coefficients, while linear interpolation does not?

**Exercise 4-7:** In Eq. (4.63), why isn't  $c[n] = x(n\Delta)$  for  $N \geq 2$ ?

**Answer:** Because three of the basis functions  $\beta_N(\frac{t}{\Delta} - m)$  overlap for any  $t$ .

## 4-8 2-D Spline Interpolation

1-D interpolation using splines generalizes directly to 2-D. The task now is to obtain a continuous function  $f(x, y)$  from samples  $\{f(n\Delta, m\Delta)\}$ .

The 2-D spline functions are separable products of the 1-D spline functions:

$$\beta_N(x, y) = \beta_N(x) \beta_N(y). \quad (4.74)$$

For example, for  $N = 1$  we have

$$\beta_1(x, y) = \begin{cases} (1 - |x|)(1 - |y|) & \text{for } 0 \leq |x|, |y| \leq 1 \\ 0 & \text{otherwise,} \end{cases} \quad (4.75)$$

which is pyramidal in shape. Interpolation using  $\beta_1(x, y)$  is called **bilinear** interpolation, which is a misnomer because  $\beta_1(x, y)$  includes a product term,  $|x||y|$ .

In 2-D, Eq. (4.63) becomes

$$f(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} c[n, m] \beta_N\left(\frac{x}{\Delta} - n\right) \beta_N\left(\frac{y}{\Delta} - m\right). \quad (4.76)$$

For  $N = 0$  and  $N = 1$ ,  $c[n, m] = f(n\Delta, m\Delta)$ , but for  $N \geq 2$ ,  $c[n, m]$  is computed from  $\{f(n\Delta, m\Delta)\}$  using a 2-D version of the DFT recipe outlined in Section 4-7.4.

### 4-8.1 Nearest-Neighbor (NN) image Interpolation

NN interpolation of images works in the same way as NN interpolation of 1-D signals. After locating the four samples surrounding a location  $(x_0, y_0)$ —thereby specifying the applicable values of  $n$  and  $m$ , the value assigned to  $f(x, y)$  at location  $(x_0, y_0)$  is the value of the nearest-location neighbor among those

four neighbors:

$$\{f(n\Delta, m\Delta), f((n+1)\Delta, m\Delta), f(n\Delta, (m+1)\Delta), f((n+1)\Delta, (m+1)\Delta)\}. \quad (4.77)$$

### 4-8.2 Bilinear Image Interpolation

Linear interpolation is performed as follows:

(1) Each image location  $(x_0, y_0)$  has four nearest sampled values given by Eq. (4.77), with a unique set of values for  $n$  and  $m$ .

(2) Compute:

$$\begin{aligned} f(x_0, m\Delta) &= f(n\Delta, m\Delta) \left( n + 1 - \frac{x_0}{\Delta} \right) \\ &\quad + f((n+1)\Delta, m\Delta) \left( \frac{x_0}{\Delta} - n \right), \end{aligned} \quad (4.78a)$$

$$\begin{aligned} f(x_0, (m+1)\Delta) &= f(n\Delta, (m+1)\Delta) \left( n + 1 - \frac{x_0}{\Delta} \right) \\ &\quad + f((n+1)\Delta, (m+1)\Delta) \left( \frac{x_0}{\Delta} - n \right), \end{aligned} \quad (4.78b)$$

and then combine them to find

$$\begin{aligned} f(x_0, y_0) &= f(x_0, m\Delta) \left( m + 1 - \frac{y_0}{\Delta} \right) \\ &\quad + f(x_0, (m+1)\Delta) \left( \frac{y_0}{\Delta} - m \right). \end{aligned} \quad (4.79)$$

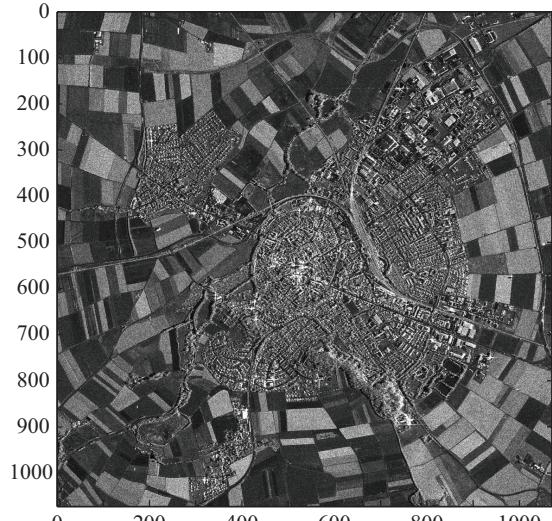
The preceding computation linearly interpolates in  $x$  for  $y = m\Delta$ , and again for  $y = (m+1)\Delta$ , and then linearly interpolates in  $y$  for  $x = x_0$ .

### 4-8.3 Cubic Spline Interpolation

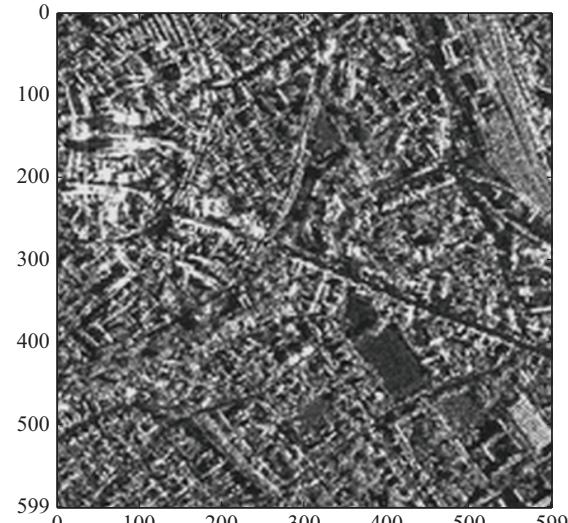
We describe the cubic spline interpolation procedure through an example. **Figure 4-13(a)** shows a synthetic-aperture radar (SAR) image of a metropolitan area, and part (b) of the figure shows a magnified version of the central part of the original image using cubic-spline interpolation. The magnification factor is 3 along each direction. The interpolation uses

$$x[n, m] = c[n, m] * * (\beta_3[n] \beta_3[m]) \quad (4.80)$$

to compute  $x[n, m]$  at  $x = n/3$  and  $y = m/3$  for integers  $n$  and  $m$ . The values of  $c[n, m]$  are determined using the DFT recipe outlined in Section 4-7.4 in combination with the product of cubic spline functions given by Eq. (4.62) evaluated at



(a) Original SAR image to be magnified



(b) SAR image magnified by 3 using cubic splines

**Figure 4-13** The image in (b) is the  $(200 \times 200)$  central part of the synthetic-aperture radar (SAR) image in (a) magnified by a factor of 3 along each direction using cubic-spline interpolation.

$(-1, 0, 1)$ :

$$\begin{aligned} & \begin{bmatrix} \beta_3(-1) \\ \beta_3(0) \\ \beta_3(1) \end{bmatrix} \begin{bmatrix} \beta_3(-1) & \beta_3(0) & \beta_3(1) \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{6} \\ \frac{4}{6} \\ \frac{1}{6} \end{bmatrix} \begin{bmatrix} \frac{1}{6} & \frac{4}{6} & \frac{1}{6} \end{bmatrix} = \frac{1}{36} \begin{bmatrix} 1 & 4 & 1 \\ 4 & 16 & 4 \\ 1 & 4 & 1 \end{bmatrix}. \end{aligned} \quad (4.81)$$

**Exercise 4-8:** The “image”  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$  is interpolated using bilinear interpolation. What is the interpolated value at the center of the image?

**Answer:**  $\frac{1}{4}(1+2+3+4)=2.5$

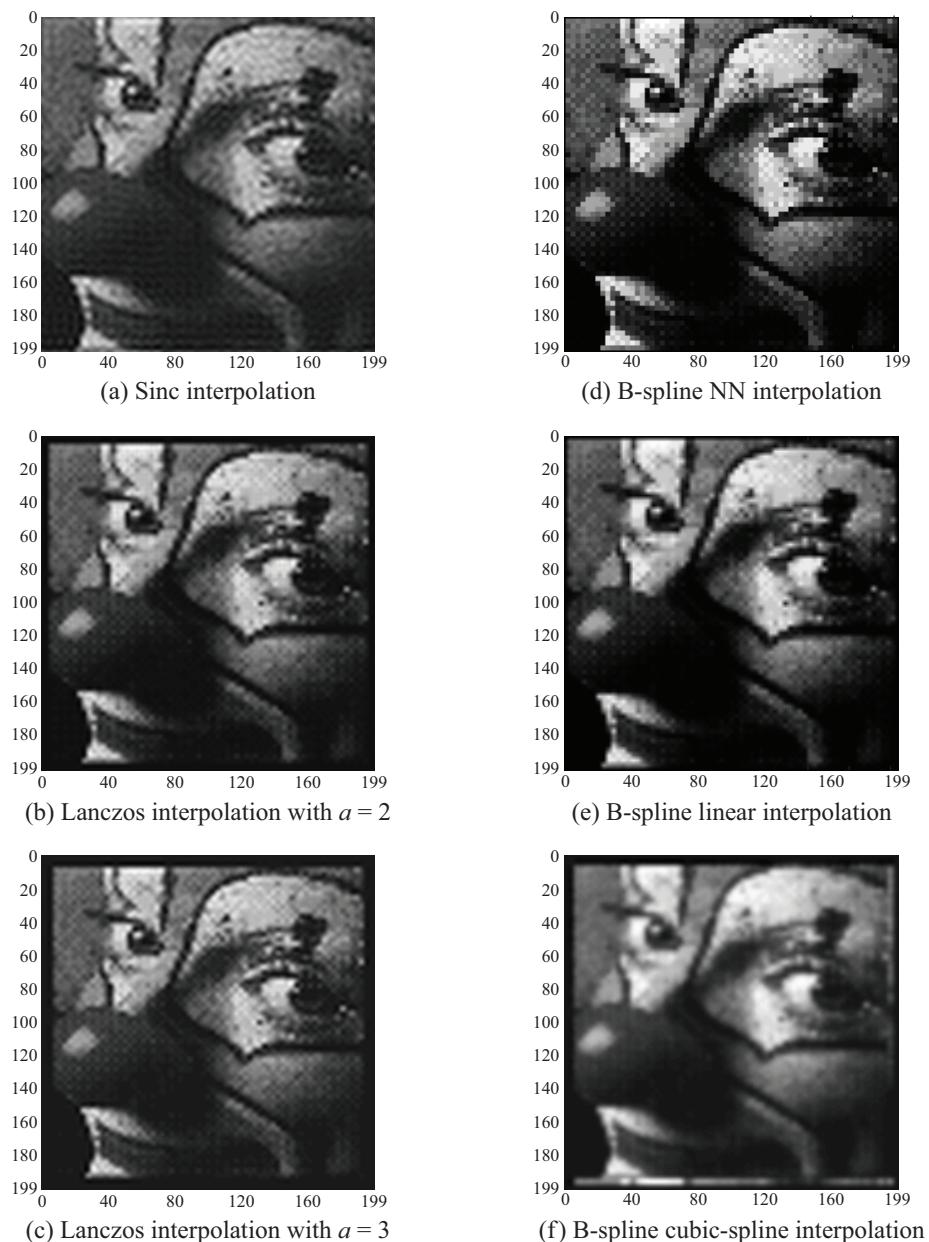
## 4-9 Comparison of 2-D Interpolation Methods

In this chapter, we have discussed three image interpolation methods: The sinc interpolation formula in Section 4-1.1, the Lanczos interpolation formula in Section 4-1.2, and the B-spline interpolation method in Section 4-8. To compare the effectiveness of the different methods, we chose the original clown image shown in Fig. 4-14(a) and then downsampled it by a factor of 9 by retaining only 1/9 of the original pixels (1/3 along each direction). The locations of the downsampled pixels and the resulting  $(67 \times 67)$  downsampled image are shown in parts (b) and (c) of Fig. 4-14, respectively.

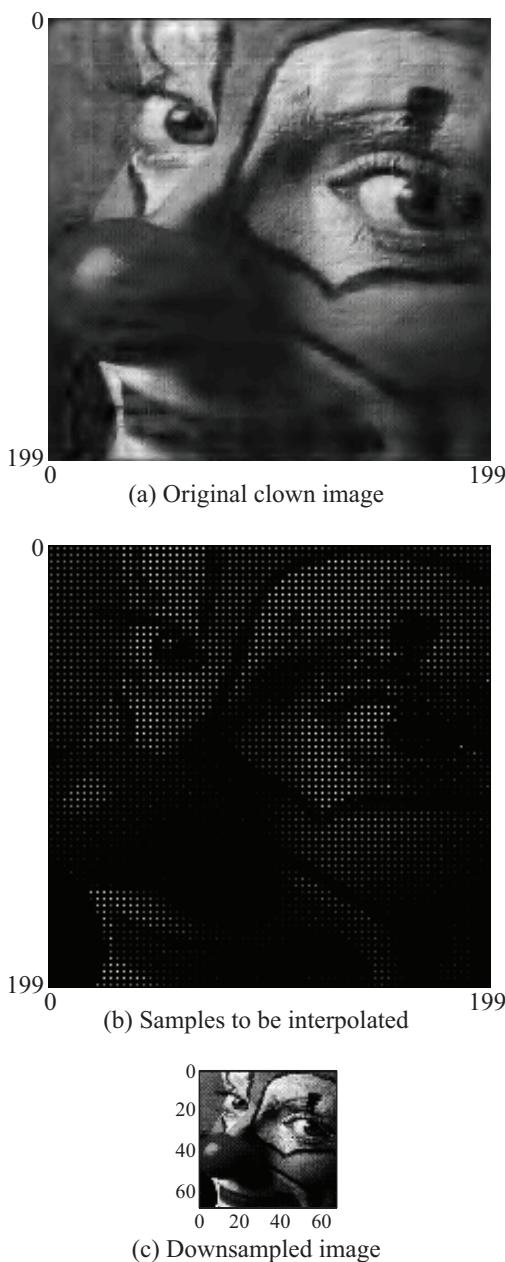
Next, we applied the various interpolation methods listed earlier to the downsampled clown image in Fig. 4-14(c) so as to generate an interpolated version of the original clown image. This is equivalent to magnifying the  $(67 \times 67)$  downsampled clown image in Fig. 4-14(c) by a factor of 3 in each dimension. The results, displayed in Fig. 4-15, deserve the following commentary:

(a) Relative to the original clown image, of the first three interpolated images, the Lanczos with  $a = 3$  is slightly better than that with  $a = 2$ , and both are better than the sinc-interpolated image.

(b) Among the B-spline images, significant improvement is realized in image quality as  $N$  is increased from  $N = 0$  (nearest neighbor) to  $N = 1$  (bilinear) and then to  $N = 3$  (cubic). MATLAB code for this example is available on the book website.



**Figure 4-15** Comparison of three interpolation methods: (a) sinc interpolation; (b) and (c) Lanczos interpolation with  $\alpha = 2$  and  $\alpha = 3$ , respectively; and (d) to (f) B-spline with  $N = 0$ ,  $N = 1$ , and  $N = 3$ , respectively.



**Figure 4-14** The original clown image in (a) was downsampled to the image in (c) by sampling only 1/9 of the pixels of the original image.

## 4-10 Examples of Image Interpolation Applications

### Example 4-5: Image Rotation

Recall from Eq. (3.12), that rotating an image  $f(x,y)$  by an angle  $\theta$  leads to a rotated image  $g(x,y)$  given by

$$g(x,y) = f(x \cos \theta + y \sin \theta, y \cos \theta - x \sin \theta). \quad (4.82)$$

Sampling  $g(x,y)$  at  $x = n\Delta$  and  $y = m\Delta$  gives

$$\begin{aligned} g(n\Delta, m\Delta) = \\ f(n\Delta \cos \theta + m\Delta \sin \theta, m\Delta \cos \theta - n\Delta \sin \theta), \end{aligned} \quad (4.83)$$

which clearly requires interpolation of  $f(x,y)$  at the required points from its given samples  $f(n\Delta, m\Delta)$ . In practice, nearest neighbor (NN) interpolation is usually sufficient to realize the necessary interpolation.

**Figure 4-16(a)** displays a zero-padded clown image, and part (b) displays the image after rotation by  $45^\circ$  using NN interpolation. The rotated image bears a very good resemblance to the rotated original. The MATLAB code for this figure is on the book website.

### Example 4-6: Exponential Image Warping

Image **warping** or **morphing** entails creating a new image  $g(x,y)$  from the original image  $f(x,y)$  where

$$g(x,y) = f(T_x(x), T_y(y)) \quad (4.84)$$

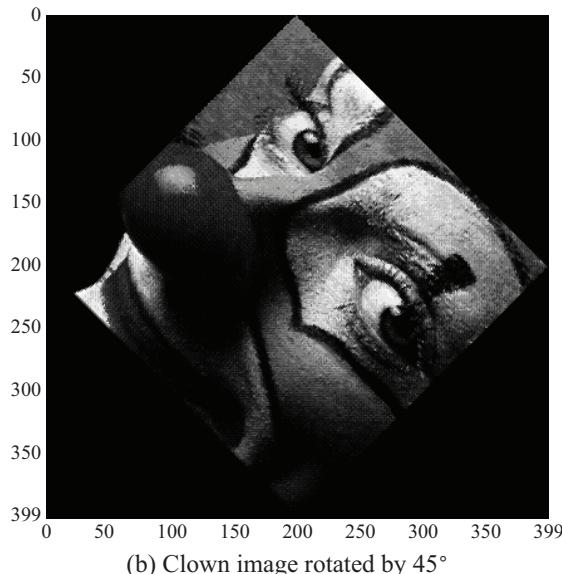
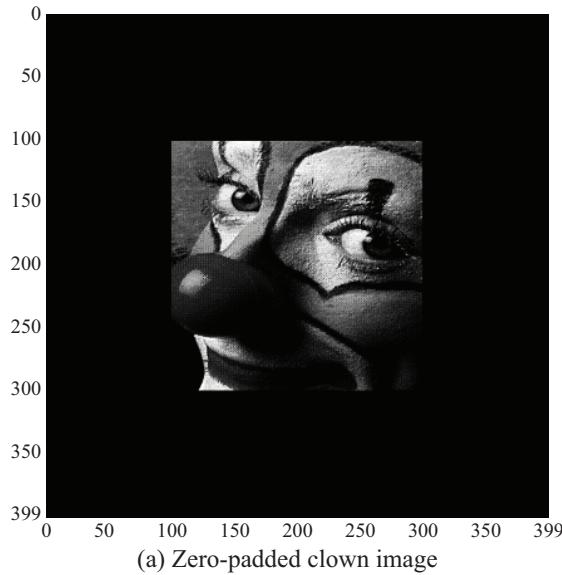
for some 1-D transformations  $T_x(x)$  and  $T_y(y)$ . **Image shifting** by  $(x_0, y_0)$  can be implemented using

$$T_x(x) = x - x_0 \quad \text{and} \quad T_y(y) = y - y_0.$$

Magnification by a factor of  $a$  can be implemented using

$$T_x(x) = \frac{x}{a} \quad \text{and} \quad T_y(y) = \frac{y}{a}.$$

More interesting warping of images can be performed using nonlinear transformations, as demonstrated by the following illustrations.



**Figure 4-16** Clown image before and after rotation by 45°.

### (a) Warped Clown Image

After shifting the clown image so that the origin  $[n, m] = [0, 0]$  is at the center of the image, the image was warped using  $T_n(n) = ne^{-|n|/300}$  and  $T_m(m) = me^{-|m|/300}$  and NN interpolation. The warped image is shown in [Fig. 4-17\(a\)](#). Repeating the process with the space constant 300 replaced with 200 leads to greater warping, as shown in [Fig. 4-17\(b\)](#).

### Example 4-7: Square-Root and Inverse Image Warping

Another form of image warping is realized by applying a square-root function of the form

$$T_n(n) = \frac{n\sqrt{|n|}}{25} \quad \text{and} \quad T_m(m) = \frac{m\sqrt{|m|}}{25}.$$

Repetition of the steps described in the previous example, but using the square-root transformation instead, leads to the images in [Fig. 4-18\(a\)](#). The MATLAB code for this figure is available on the book website.

Another transformation is the inverse function given by

$$T_n(n) = \frac{n|n|}{a} \quad \text{and} \quad T_m(m) = \frac{m|m|}{a}.$$

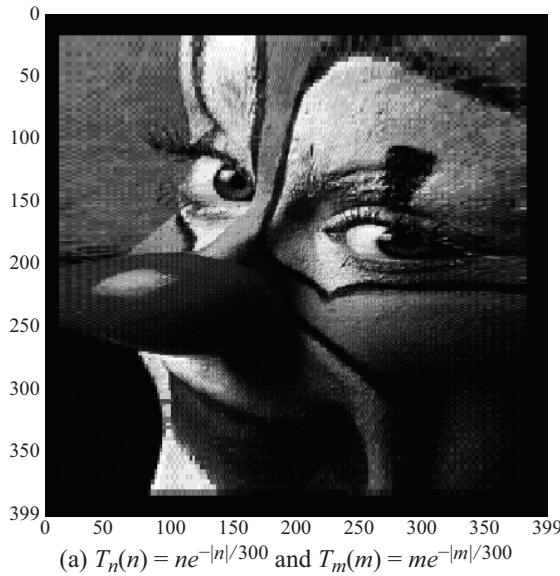
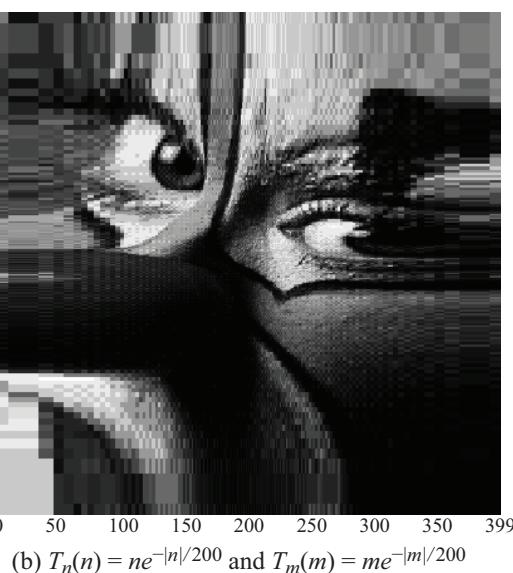
The result of warping the clown image with  $a = 300$  is shown in [Fig. 4-18\(b\)](#). The MATLAB code for this figure is available on the book website.

**Concept Question 4-6:** Provide four applications of interpolation.

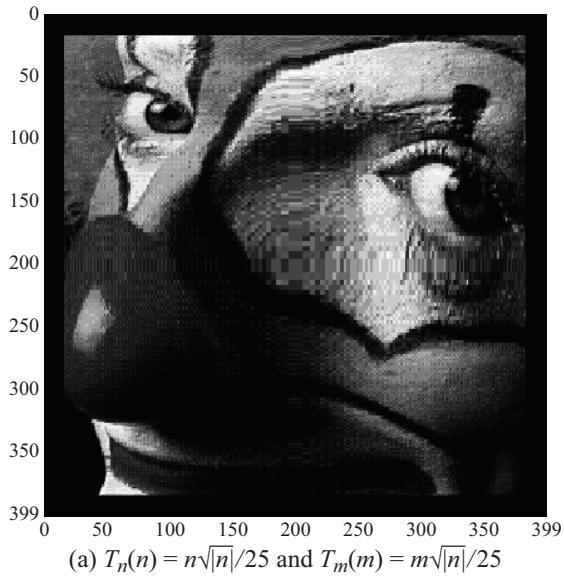
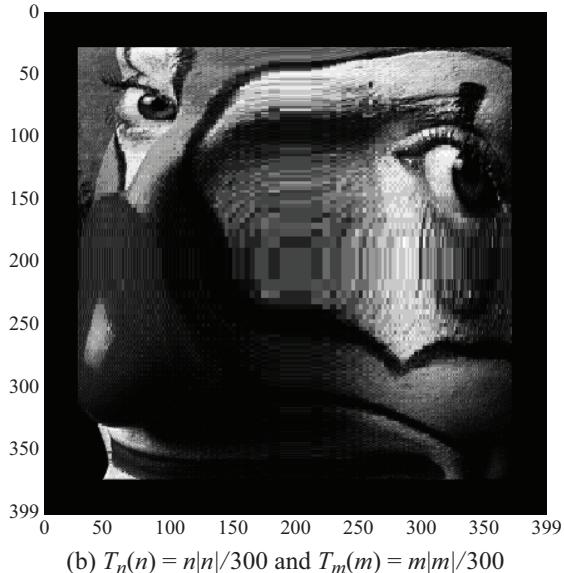
**Exercise 4-9:** The “image”  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$  is rotated counter-clockwise 90°. What is the result?

**Answer:**  $\begin{bmatrix} 2 & 4 \\ 1 & 3 \end{bmatrix}$

**Exercise 4-10:** The “image”  $\begin{bmatrix} 4 & 8 \\ 12 & 16 \end{bmatrix}$  is magnified by a factor of three. What is the result, using: (a) NN; (b) bilinear interpolation?

(a)  $T_n(n) = ne^{-|n|/300}$  and  $T_m(m) = me^{-|m|/300}$ (b)  $T_n(n) = ne^{-|n|/200}$  and  $T_m(m) = me^{-|m|/200}$ 

**Figure 4-17** Nonlinear image warping with space constants of (a) 300 and (b) 200.

(a)  $T_n(n) = n\sqrt{|n|}/25$  and  $T_m(m) = m\sqrt{|m|}/25$ (b)  $T_n(n) = n|n|/300$  and  $T_m(m) = m|m|/300$ 

**Figure 4-18** Clown image warped with (a) square-root transformation and (b) inverse transformation.

**Answer:**

(a)

$$\begin{bmatrix} 4 & 4 & 4 & 8 & 8 & 8 \\ 4 & 4 & 4 & 8 & 8 & 8 \\ 4 & 4 & 4 & 8 & 8 & 8 \\ 12 & 12 & 12 & 16 & 16 & 16 \\ 12 & 12 & 12 & 16 & 16 & 16 \\ 12 & 12 & 12 & 16 & 16 & 16 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 1 & 2 & 1 & 2 & 4 & 2 \\ 2 & 4 & 2 & 4 & 8 & 4 \\ 1 & 2 & 1 & 2 & 4 & 2 \\ 3 & 6 & 3 & 4 & 8 & 4 \\ 6 & 12 & 6 & 8 & 16 & 8 \\ 3 & 6 & 3 & 4 & 8 & 4 \end{bmatrix}$$

## Summary

### Concepts

- Interpolation is “connecting the dots” of 1-D samples, and “filling in the gaps” of 2-D samples.
- Interpolation can be used to rotate and to warp or “morph” images.
- Upsampling an image can be performed by inserting rows and columns of zeros in the 2-D DFT of the image. Care must be taken to preserve conjugate symmetry in the 2-D DFT.
- Downsampling an image can be performed by deleting rows and columns in the 2-D DFT of the image. Care

must be taken to preserve conjugate symmetry in the 2-D DFT. Deleting rows and columns performs lowpass filtering so that the downsampled image is not aliased.

- B-splines are piecewise polynomial functions that can be used to interpolate samples in 1-D and 2-D.
- For  $N \geq 2$ , computation of the coefficients  $\{c[m]\}$  from samples  $\{x(m\Delta)\}$  can be formulated as a deconvolution problem.
- 2-D interpolation using B-splines is a generalization of 1-D interpolation using B-splines.

### Mathematical Formulae

#### B-splines

$$\beta_N(t) = \underbrace{\text{rect}(t) * \dots * \text{rect}(t)}_{N+1 \text{ times}}$$

#### B-spline

$$\beta_0(t) = \text{rect}(t) = \begin{cases} 1 & \text{for } |t| < 1/2, \\ 0 & \text{for } |t| > 1/2 \end{cases}$$

#### B-spline

$$\beta_1(t) = \begin{cases} 1 - |t| & \text{for } |t| \leq 1, \\ 0 & \text{for } |t| \geq 1 \end{cases}$$

#### B-spline

$$\beta_2(t) = \begin{cases} 3/4 - t^2 & \text{for } 0 \leq |t| \leq 1/2, \\ (3/2 - |t|)^2/2 & \text{for } 1/2 \leq |t| \leq 3/2, \\ 0 & \text{for } |t| \geq 3/2 \end{cases}$$

#### B-spline

$$\beta_3(t) = \begin{cases} 2/3 - t^2 + |t|^3/2 & \text{for } |t| \leq 1, \\ (2 - |t|)^3/6 & \text{for } 1 \leq |t| \leq 2, \\ 0 & \text{for } |t| \geq 2 \end{cases}$$

#### B-spline 1-D interpolation

$$x(t) = \sum_{m=-\infty}^{\infty} c[m] \beta_N \left( \frac{t}{\Delta} - m \right)$$

#### Nearest-neighbor 1-D interpolation

$$x(t) = x(n\Delta) \text{ for } |t - n\Delta| < \Delta/2$$

#### Linear 1-D interpolation

$$x(t) = x(n\Delta) \left( (n+1) - \frac{t}{\Delta} \right) + x((n+1)\Delta) \left( \frac{t}{\Delta} - n \right) \quad \text{for } n\Delta \leq t \leq (n+1)\Delta$$

### Important Terms

Provide definitions or explain the meaning of the following terms:

B-spline    downsampling    interpolation    Lanczos function    nearest-neighbor    thumbnail image    upsampling

## PROBLEMS

### Section 4-3: Upsampling and Interpolation

**4.1** Write a MATLAB program that loads the  $50 \times 50$  image in `tinyclown.mat` and magnifies it by four using upsampling. Note that 50 is an even number.

**4.2** Write a MATLAB program that loads the  $50 \times 50$  image in `tinyclown.mat`, deletes the last row and column to make it  $49 \times 49$ , and magnifies it by four using upsampling. This is easier than Problem 4.1 since 49 is an odd number.

**4.3** Write a MATLAB program that loads the  $64 \times 64$  image in `tinyletters.mat` and magnifies it by four using upsam-

pling. Note that 64 is an even number.

**4.4** Write a MATLAB program that loads the  $64 \times 64$  image in `tinyletters.mat`, deletes the last row and column to make it  $63 \times 63$ , and magnifies it by four using upsampling. This is easier than Problem 4.3 since 63 is an odd number.

## Section 4-5: Downsampling

**4.5** Write a MATLAB program that loads the  $200 \times 200$  image in `clown.mat`, antialias lowpass filters it, and demagnifies it by four using downsampling. (This is how the image in `tinyclown.mat` was created.)

**4.6** Repeat Problem 4.5, but skip the antialias lowpass filter.

**4.7** Write a MATLAB program that loads the  $256 \times 256$  image in `letters.mat`, antialias lowpass filters it, and demagnifies it by four using downsampling. (This is how the image in `tinyletters.mat` was created.)

**4.8** Repeat Problem 4.7, but skip the antialias lowpass filter.

**4.9** Show that if the sinc interpolation formula is used to upsample an  $M \times M$  image  $f[n, m]$  to an  $N \times N$  image  $g[n, m]$  by an integer factor  $L$  (so that  $N = ML$ ), then  $g[nL, mL] = f[n, m]$ , so that the values of  $f[n, m]$  are preserved after upsampling.

## Section 4-8: 2-D Spline Interpolation

**4.10** Write a MATLAB program that loads the  $50 \times 50$  image in `tinyclown.mat` and magnifies it by four using nearest-neighbor interpolation.

**4.11** Write a MATLAB program that loads the  $64 \times 64$  image in `tinyletters.mat` and magnifies it by four using nearest-neighbor interpolation.

**4.12** NN interpolation is used to magnify

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

by three. What is the result?

**4.13** Linear interpolation is used to magnify

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

by two. What is the result?

**4.14** Another way to derive the formula for linear interpolation is as follows: The goal is to interpolate the four points  $\{f(0,0), f(1,0), f(0,1), f(1,1)\}$  using a formula  $f(x,y) = f_0 + f_1x + f_2y + f_3xy$ , where  $\{f_0, f_1, f_2, f_3\}$  are found from the given points. This extends to

$$\{f(n,m), f(n+1, ), f(n, m+1), f(n+1, m+1)\}$$

for any integers  $n, m$ .

(a) Set up a linear system of equations with unknowns  $\{f_0, f_1, f_2, f_3\}$  and knowns

$$\{f(0,0), f(1,0), f(0,1), f(1,1)\}.$$

(b) Solve the system to obtain a closed-form expression for  $f(x,y)$  as a function of  $\{f(0,0), f(0,1), f(1,0), f(1,1)\}$ .

**4.15** The image

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

is rotated  $90^\circ$  clockwise. What is the result?

**4.16** The image

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

is rotated  $45^\circ$  clockwise and magnified by  $\sqrt{2}$  using linear interpolation. What is the result?

**4.17** Recall from Eq. (3.12) that rotating an image  $f(x,y)$  by  $\theta$  to get  $g(x,y)$  is implemented by

$$g(x,y) = f(x',y') = f(x\cos\theta + y\sin\theta, y\cos\theta - x\sin\theta),$$

where from Eq. (3.11)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

This is point-by-point, and so it is very slow. A faster way to rotate an image by transforming it first in  $y$  and then in  $x$  is as follows: (1) Let  $h(x,y) = f(x, (y/\cos\theta - x\tan\theta))$ ; (2) then  $g(x,y) = h(\cos\theta + y\sin\theta, y)$ .

(a) Show that this computes  $g(x,y)$  from  $f(x,y)$ .

(b) Show that this amounts to the matrix factorization

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ * & * \end{bmatrix} \begin{bmatrix} * & * \\ 0 & 1 \end{bmatrix}$$

for some elements  $*$ .

**4.18** Write a MATLAB program that loads the  $50 \times 50$  image in `tinyclown.mat` and magnifies it by four using cubic spline interpolation.

**4.19** Write a MATLAB program that loads the  $50 \times 50$  image in `tinyletters.mat` and magnifies it by four using cubic spline interpolation.

**4.20** This problem is for readers who have some familiarity with 1-D DSP. Use of 1-D quadratic splines on  $N$  interpolation points  $\{x(n\Delta)\}$  requires solving

$$x(n\Delta) = c[n-1] \beta_2(1) + c[n] \beta_2(0) + c[n+1] \beta_2(1)$$

(Eq. (4.64)) for  $\{c[n]\}$  from  $\{x(n\Delta)\}$ . In Section 4-7.4 this was solved using the DFT, which requires  $(N/2)\log_2 N$  multiplications. This problem gives a faster method, requiring only  $2N < (N/2)\log_2 N$  multiplications.

Let

$$h[n] = \{\beta_2(-1), \underline{\beta_2(0)}, \beta_2(1)\} = \beta_2(1)\{1, r, 1\},$$

where  $r = \beta_2(0)/\beta_2(1)$ .

(a) Show that  $h[n]$  can be written as

$$h[n] = \beta_2(1)\{1, \underline{1/\rho}\} * \{1, \rho\}$$

for some constant  $\rho$ . Determine  $\rho$ .

(b) Show that  $h[n]$  can be implemented by the following two systems connected in series:

$$\begin{aligned} y_1[n] &= h_1[n] * x_1[n] \\ &= x_1[n] + \rho x_1[n-1]; y_2[n] \\ &= h_2[n] * x_2[n] \\ &= x_2[n+1] + \frac{1}{\rho} x_2[n]. \end{aligned}$$

(c) Show that for each of these systems,  $x_i[n]$  can be computed recursively and stably from  $y_i[n]$  using

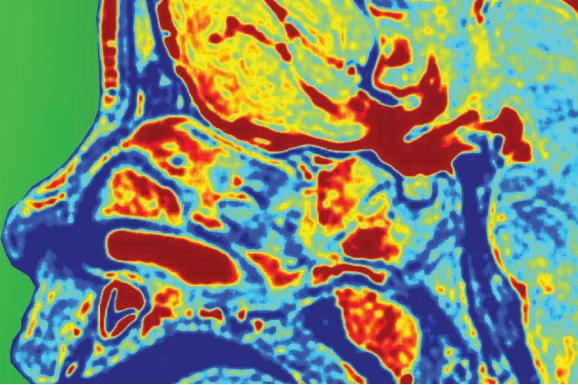
$$x_1[n] + \rho x_1[n-1] = y_1[n]; x_2[n] + \rho x_2[n+1] = \rho y_2[n].$$

The latter system must be run backwards in time  $n$ .

(d) Determine a recipe for computing  $\{c[n]\}$  from  $\{x(n\Delta)\}$  using these concepts.

**4.21** Repeat Problem 4.20 for cubic splines.

# CHAPTER 5



## 5

## Image Enhancement

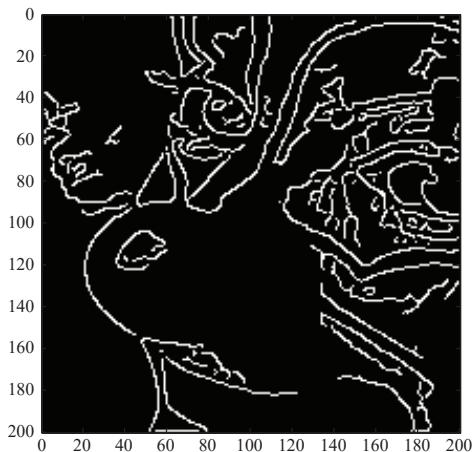
### Contents

- Overview, 160
- 5-1** Pixel Value Transformation, 160
- 5-2** Unsharp Masking, 163
- 5-3** Histogram Equalization, 167
- 5-4** Edge Detection, 171
- 5-5** Summary of Image Enhancement
  - Techniques, 176
  - Problems, 178

### Objectives

Learn to:

- Use linear or gamma transformation to alter pixel values to bring out image features.
- Use unsharp masking or the Laplacian to sharpen an image.
- Use histogram equalization to brighten an image.
- Use Sobel or Canny edge detection to produce an edge image of a given image.



This chapter covers various types of image enhancement, in which the goal is to deliberately alter the image to brighten it, increase its contrast, sharpen it, or enhance features such as edges. Unsharp masking sharpens an image using a high-pass filter, but this also makes the image noisier. Histogram equalization nonlinearly alters pixel values to spread them out more evenly over the display range of the image. Edge enhancement produces an edge image of just the edges of the image, which can be useful in image recognition in computer vision.

## Overview

**Image enhancement** is an operation that transforms an image  $f[n, m]$  to another image  $g[n, m]$  in which features of  $f[n, m]$ , such as edges or contrasts between different pixel values, are emphasized. It is not the same as **image restoration**, which includes **denoising** (removing noise from an image), **deblurring** (refocusing an out-of-focus image), and the more general case of **deconvolution** (undoing the effect of a PSF on an image). In these three types of operations, the goal is to recover the true image  $f[n, m]$  from its noisy or blurred version  $g[n, m]$ . Image restoration is covered in Chapter 6.

Image enhancement techniques covered in this chapter include: linear and nonlinear transformations of pixel values for displaying images more clearly; **unsharp masking**, a technique originally developed for sharpening images in film-based photography; **histogram equalization** for brightening images; and **edge detection** for identifying edges in images.

## 5-1 Pixel Value Transformation

The **image dynamic range**  $R_i$  of an image  $f[n, m]$  is defined as the range of pixel values contained in the image, extending between a minimum value  $f_{\min}$  and a maximum value  $f_{\max}$ :

$$R_i = f_{\max} - f_{\min}$$

For a display device, such as a printed page or a computer monitor, the **display dynamic range**  $R_d$  of the display intensity  $g[n, m]$  is the full range available for displaying an image, extending from a minimum of zero to a maximum  $g_{\max}$ :

$$R_d = g_{\max} - 0$$

Ideally, the display device should display an image such that the information content of the image is conveyed to the user most optimally. This is accomplished by applying a preprocessing transformation of pixel values. If  $R_i$  extends over a narrow range of  $R_d$ , a transformation can be used to expand  $R_i$  to take full advantage of the available extent of  $R_d$ . Conversely, if  $R_i$  extends over several orders of magnitude, displaying the image over the limited linear range  $R_d$  would lead to pixel-value truncation. To avoid the truncation issue, a nonlinear transformation is needed so as to convert the dynamic range  $R_i$  into a range that is more compatible with the dynamic range  $R_d$  of the display device. We now explore both types of transformations.

### 5-1.1 Linear Transformation of Pixel Values

In general, image  $f[n, m]$  may have both positive and negative pixel values. A **linear transformation** linearly transforms the individual pixel values from  $f[n, m]$  to  $g[n, m]$ , with 0 displayed as pure black and  $g_{\max}$  displayed as pure white. Functionally, the linear transformation is given by

$$g[n, m] = g_{\max} \left( \frac{f[n, m] - f_{\min}}{f_{\max} - f_{\min}} \right). \quad (5.1)$$

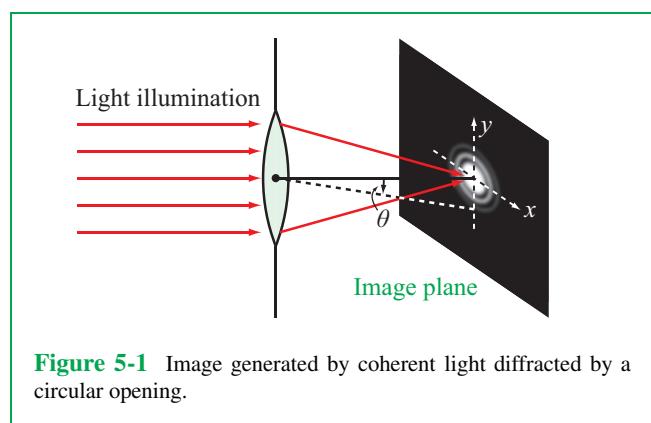
Usually,  $g[n, m]$  is normalized so that  $g_{\max} = 1$ .

Without the linear transformation, a display device would display all negative values of  $f[n, m]$  as black and would display all values larger than  $g_{\max}$  as white. In MATLAB, the command `imagesc(X)`, `colormap(gray)` applies the linear transformation given by Eq. (5.1), with  $g_{\min} = 0$  and  $g_{\max} = 1$ , prior to displaying an image, thereby ensuring that the full range of values of array  $X$  are displayed properly, including negative values.

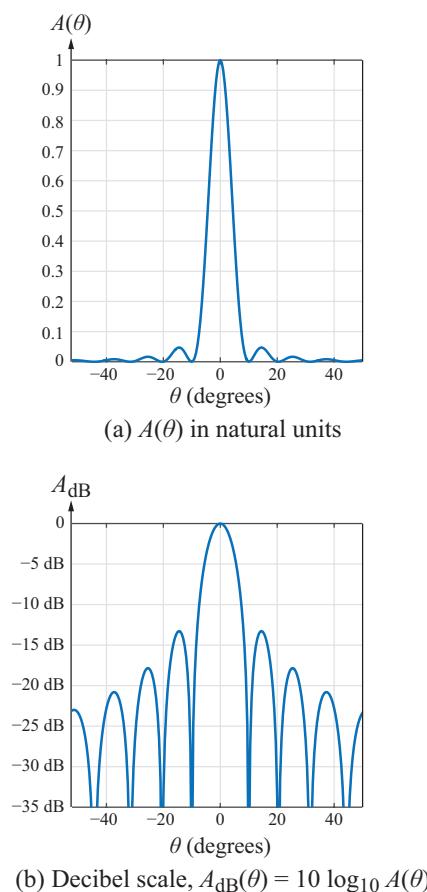
### 5-1.2 Logarithmic Transformation of Pixel Values

If coherent light is used to illuminate a circular opening, as depicted by the diagram in **Fig. 5-1**, the light diffracted by the opening generates an interference pattern in the image plane, consisting of a “ring-like” structure. The 1-D image intensity along any direction in the image plane is given by

$$I(\theta) = I_0 \operatorname{sinc}^2(a\theta), \quad (5.2)$$



**Figure 5-1** Image generated by coherent light diffracted by a circular opening.



**Figure 5-2** Plot of the normalized intensity as a function of angle  $\theta$ .

where  $I_0$  and  $a$  are constants related to the wavelength  $\lambda$  the diameter of the opening, and the overall imaging geometry.

A plot of the **normalized intensity**

$$A(\theta) = \frac{I(\theta)}{I_0} = \text{sinc}^2(a\theta) \quad (5.3)$$

with  $a = 0.1$  is displayed in **Fig. 5-2(a)** as a function of angle  $\theta$ , with  $\theta$  expressed in degrees. The peak value of  $A(\theta)$  is 1, and the  $\text{sinc}^2(a\theta)$  function exhibits sidelobes that decrease in intensity with increasing value of  $|\theta|$ . The plot provides a good

representation of the normalized intensity in the range between the first pair of nulls, but it is difficult to examine the plot quantitatively outside the range between the two nulls.

**Figure 5-2(b)** provides an alternative format for displaying the normalized intensity, namely as

$$A_{\text{dB}}(\theta) = 10 \log_{10} A(\theta) = 10 \log_{10} [\text{sinc}^2(a\theta)]. \quad (5.4)$$

The use of the logarithm in the decibel (dB) scale serves to compress the dynamic range; the range between 1 and  $10^{-3}$ , for example, gets converted to between 0 and  $-30$  dB. Consequently, very small values of  $A(\theta)$  become more prominent, while values close to 1 get compressed. This is evident in the plot shown in **Fig. 5-2(b)**, where it is now much easier to “read” the values of the second and third lobes, compared with doing so using the linear scale shown in **Fig. 5-2(a)**. We should note that at angles  $\theta$  where  $A(\theta) = 0$ , the corresponding value of  $A_{\text{dB}}(\theta)$  is  $-\infty$ . In plots like the one in **Fig. 5-2(b)**, the lower limit along the vertical axis is truncated to some finite value, in this case  $-35$  dB.

The decibel scale also is used in displaying image spectra, an example of which is shown in **Fig. 5-3**, as well as in multiple fields of science and engineering, including:

- Acoustics:

$$P_{\text{dB}} = 20 \log_{10} \left( \frac{\text{pressure}}{20 \mu\text{pascals}} \right), \quad (5.5)$$

where  $20 \mu\text{pascals}$  is the smallest acoustic pressure that can create an audible sound. On this  $P_{\text{dB}}$  scale, a whisper is about 30 dB and the sound intensity of a jet plane taking off is about 130 dB, making the latter 100 dB (or, equivalently, 100,000) times louder than a whisper.

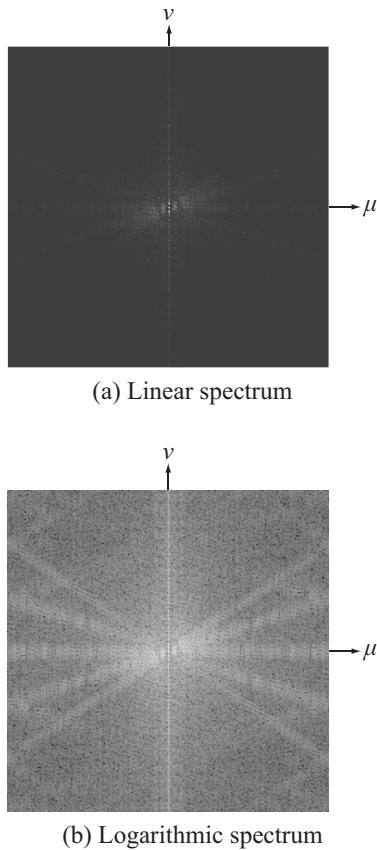
- Richter Scale for Earthquakes:

$$D_{\text{dB}} = \log_{10} \left( \frac{\text{displacement}}{1 \mu\text{m}} \right), \quad (5.6)$$

where “displacement” is defined as the horizontal ground displacement at a location 100 km from the earthquake’s epicenter. For an earthquake of Richter magnitude 6 the associated ground motion is 1 m. In contrast, the ground displacement associated with a Richter magnitude 3 earthquake is only 1 mm.

- Stellar Magnitude (of stars viewed from Earth):

$$S_{\text{dB}} = -2.512 \log_{10} \left( \frac{\text{star brightness}}{\text{brightness of Vega}} \right). \quad (5.7)$$



**Figure 5-3** Spectrum  $\mathbf{F}(\mu, v)$  in (a) linear scale and (b) logarithmic scale.

A first-magnitude star, such as Spica in the constellation Virgo, has a stellar magnitude of approximately 1. Stars of magnitude 6 are barely visible to the naked eye (depending on viewing conditions) and are 100 times less bright than a first-magnitude star. The factor of  $-2.512$  was chosen so that a first-magnitude star has a brightness equal to 40% of that of the star Vega, and Vega was chosen as the star with a reference brightness of zero magnitude.

The dB scale also is used in voltage and power ratios, and in defining signal-to-noise ratio.

When applied to images, the **logarithmic** transformation of

pixel values is given by the functional form

$$g[n, m] = a \log_{10}(f[n, m] + b), \quad (5.8)$$

where constant  $a$  is chosen so that the maximum value of  $g[n, m]$  is equal to  $g_{\max}$ , and constant  $b$  is chosen so that the smallest value of  $g[n, m]$  is zero, which corresponds to setting the smallest value of  $(f[n, m] + b)$  equal to 1. The logarithmic transformation of pixel values also is used when displaying image spectra  $\mathbf{F}(\Omega_1, \Omega_2)$ , because the range of values of  $|\mathbf{F}(\Omega_1, \Omega_2)|$  usually is very large.

### 5-1.3 Gamma Transformation of Pixel Values

For most display devices—including cathode-ray tubes, printers, and scanners, the intensity  $I$  of the displayed image is related to the signal voltage  $V$  by a power-law relation of the form

$$I[n, m] = aV^b[n, m], \quad (5.9)$$

where  $a$  and  $b$  are constants, and  $b$  having a value in the range  $0.4 \leq b \leq 0.55$ . Because  $b$  is on the order of 0.5, the dynamic range of the intensity  $I$  is much smaller than that of  $V$ . Hence, for a display device with a fixed dynamic range, the displayed intensity is a “compressed” version of what a display of  $V$  would have looked like. To correct for this compression, the image can be preprocessed prior to displaying it by applying a **gamma transformation** of pixel values given by

$$g[n, m] = g_{\max} \left( \frac{f[n, m] - f_{\min}}{f_{\max} - f_{\min}} \right)^{\gamma}, \quad (5.10)$$

where  $\gamma$  is an application-dependent constant with the same range as  $1/b$  of Eq. (5.9), namely

$$1.8 \leq \gamma \leq 2.5.$$

One role of the gamma transformation is to correct for the power-law relationship given by Eq. (5.9), thereby generating an image display of the true signal  $f[n, m]$ . Here,  $f[n, m]$  is the true pixel value,  $g[n, m]$  is the output of the preprocessing step, which makes it the input to the display device. That is,  $g[n, m] = V[n, m]$  and

$$\begin{aligned} I[n, m] &= aV^b[n, m] = ag^b[n, m] \\ &= a \left[ g_{\max} \left( \frac{f[n, m] - f_{\min}}{f_{\max} - f_{\min}} \right)^{\gamma} \right]^b. \end{aligned} \quad (5.11)$$

If  $\gamma$  of the preprocessor is chosen such that it is approximately equal to  $1/b$  of the display device, then Eq. (5.11) simplifies to

$$I[n, m] = a' \left( \frac{f[m, n] - f_{\min}}{f_{\max} - f_{\min}} \right), \quad (5.12)$$

where  $a' = ag_{\max}^b$ .

**Figure 5-4(a)** is an image of the planet Saturn, displayed with no preprocessing. By comparison, the image in part (b) of the figure had been subjected to a preprocessing step using a gamma transformation with  $\gamma = 3$  (the value that seemed to best enhance the image). The cloud bands are much more apparent in the transformed image than in the original.

**Concept Question 5-1:** Why do we not simply use pixel values directly as numerical measures of image intensities?

**Concept Question 5-2:** Why are logarithmic scales so useful in so many fields?

**Exercise 5-1:** A star of magnitude 6 is barely visible to the naked eye in a dark sky. How much fainter is a star of magnitude 6 than Vega, whose magnitude is 0?

**Answer:** From Eq. (5.7),  $10^{6/2.512} = 244.6$ , so a sixth magnitude star is 245 times fainter than Vega.

**Exercise 5-2:** What is another name for gamma transformation with  $\gamma = 1$ ?

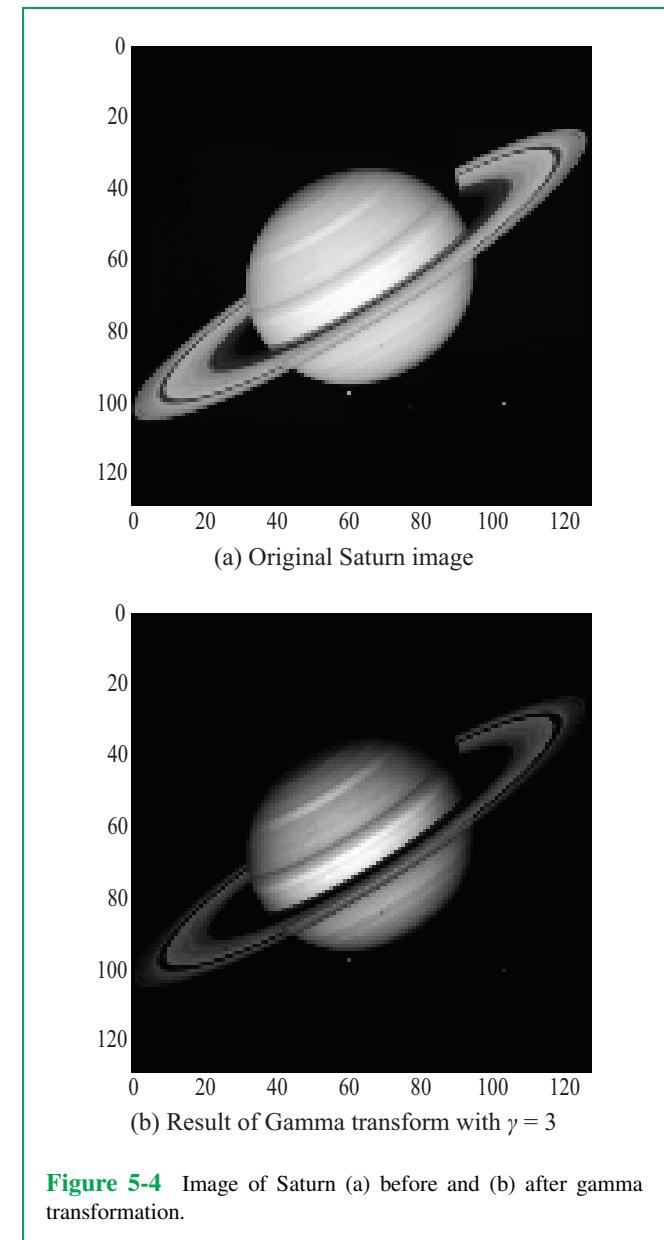
**Answer:** Linear transformation. Compare Eq. (5.10) and Eq. (5.1).

## 5-2 Unsharp Masking

### 5-2.1 Film Photography Version of Unsharp Masking

**Unsharp masking** is a contrast-enhancement procedure commonly used in both photographic and electronic displays. The procedure acts like a high-pass spatial filter that enhances high spatial-frequency (fast varying) image components.

► Thus, unsharp masking enhances edges and fast-varying parts of the image, and is a standard tool available in Adobe® Photoshop®. ◀



**Figure 5-4** Image of Saturn (a) before and (b) after gamma transformation.

The “unsharp” and “masking” parts of the name are associated with a technique in which a blurred (or unsharp) image is used to create a corrective “mask” for removing the blurriness from the image. The audio equivalent of unsharp masking is turning

up the treble.

In its original form, unsharp masking was developed for film photography to deal with blurring caused by the printing process, which involves the passage of the light image through a sheet of glass. If the original image is  $f(x,y)$  and the blurred printed version is  $f_{\text{blur}}(x,y)$ , the difference is called  $f_{\text{mask}}(x,y)$ ,

$$f_{\text{mask}}(x,y) = f(x,y) - f_{\text{blur}}(x,y). \quad (5.13)$$

Image  $f_{\text{mask}}(x,y)$  can be formed in a darkroom by adding a “negative” version of  $f_{\text{blur}}(x,y)$  to  $f(x,y)$ . The blurring process caused by the imperfect printing process is, in effect, a lowpass-filtering process. Hence,  $f_{\text{blur}}(x,y)$  represents a lowpass-filtered version of  $f(x,y)$ , and the “**mask**” image

$$f_{\text{mask}}(x,y) = f(x,y) - f_{\text{blur}}(x,y)$$

represents a high-pass filtered version of  $f(x,y)$ . A highpass spatial filter emphasizes the presence of edges; hence the name “mask.”

By photographically adding the mask image to the original image, we obtain a **sharpened image**  $f_{\text{sh}}(x,y)$  in which high spatial-frequency components of  $f(x,y)$  are *boosted* relative to low spatial-frequency components:

$$\begin{aligned} f_{\text{sh}}(x,y) &= f(x,y) + f_{\text{mask}}(x,y) \\ &= f(x,y) + [f(x,y) - f_{\text{blur}}(x,y)]. \end{aligned} \quad (5.14)$$

In digital image processing, high spatial-frequency components can also be boosted by applying the discrete form of the **Laplacian operator** to  $f(x,y)$ .

## 5-2.2 Laplacian Operator in Continuous Space

In continuous space, the **Laplacian**  $g(x,y)$  of a 2-D image  $f(x,y)$  is defined as

$$g(x,y) = \nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \quad (5.15)$$

The spatial frequency response of the Laplacian can be obtained by computing the 2-D CSFT of  $g(x,y)$ . Application of property #5 in **Table 2-4** leads to

$$\mathbf{G}(\mu,v) = -4\pi^2(\mu^2 + v^2) \mathbf{F}(\mu,v). \quad (5.16a)$$

Conversion to polar coordinates  $(\rho,\phi)$  in the spatial frequency domain leads to

$$\mathbf{G}(\rho,\phi) = -4\pi^2\rho^2 \mathbf{F}(\rho,\phi). \quad (5.16b)$$

In the spatial frequency domain  $\mathbf{G}(\mu,v)$  is the product of the ***spatial frequency response of the Laplacian operator***,  $\mathbf{H}_{\text{Laplace}}(\mu,v)$ , and the spectrum  $\mathbf{F}(\mu,v)$ :

$$\mathbf{G}(\mu,v) = \mathbf{H}_{\text{Laplace}}(\mu,v) \mathbf{F}(\mu,v). \quad (5.17)$$

Hence,

$$\mathbf{H}_{\text{Laplace}}(\mu,v) = -4\pi^2(\mu^2 + v^2). \quad (5.18a)$$

Similarly, in polar coordinates

$$\mathbf{H}_{\text{Laplace}}(\rho,\phi) = -4\pi\rho^2. \quad (5.18b)$$

It is evident from the definitions given by Eqs. (5.18a and b) that the Laplacian emphasizes high spatial-frequency components (proportional to  $\rho^2$ ) of the input image  $f(x,y)$ . It is equally evident that all frequency components of  $\mathbf{H}_{\text{Laplace}}(\mu,v)$  and  $\mathbf{H}_{\text{Laplace}}(\rho,\phi)$  have negative values.

## 5-2.3 Laplacian in Discrete Space

Derivatives in continuous space are approximated as differences in discrete space. Hence, in discrete space, the Laplacian  $g[n,m]$  of a 2-D image  $f[n,m]$  is defined as

$$\begin{aligned} g[n,m] &= f[n+1,m] + f[n-1,m] \\ &\quad + f[n,m+1] + f[n,m-1] - 4f[n,m]. \end{aligned} \quad (5.19)$$

This operation is equivalent to the convolution

$$g[n,m] = f[n,m] * * h_{\text{Laplace}}[n,m], \quad (5.20)$$

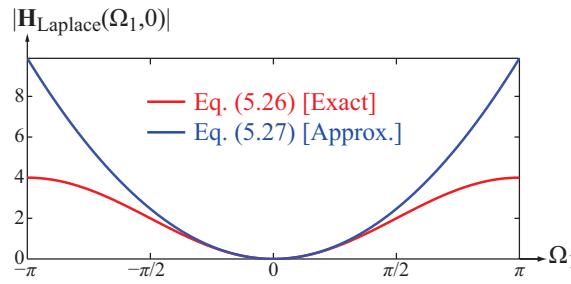
where  $h_{\text{Laplace}}[n,m]$  is the point-spread-function (PSF) of the Laplacian operator, and is given by

$$h_{\text{Laplace}}[n,m] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \quad (5.21)$$

The DSFT  $\mathbf{F}(\Omega_1, \Omega_2)$  of image  $f[n,m]$  was defined by Eq. (3.73a) as

$$\mathbf{F}(\Omega_1, \Omega_2) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n,m] e^{-j(\Omega_1 n + \Omega_2 m)}, \quad (5.22)$$

and the properties of the DSFT are direct 2-D generalizations of the properties of the 1-D DTFT given in **Table 2-7**. Of particular interest is the time-shift property in **Table 2-7**, which when



**Figure 5-5**  $|\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2)|$  versus  $\Omega_1$  at  $\Omega_2 = 0$  (in red) and the approximation for small values of  $\Omega_1$  and  $\Omega_2$  blue.

extended to 2-D, leads to

$$\mathbf{f}[n - n_0, m - m_0] \leftrightarrow \mathbf{F}(\Omega_1, \Omega_2) e^{-j(n_0\Omega_1 + m_0\Omega_2)}. \quad (5.23)$$

Application of Eq. (5.23) to Eq. (5.19) leads to

$$\begin{aligned} \mathbf{G}(\Omega_1, \Omega_2) &= \mathbf{F}(\Omega_1, \Omega_2) e^{-j\Omega_1} + \mathbf{F}(\Omega_1, \Omega_2) e^{j\Omega_1} \\ &\quad + \mathbf{F}(\Omega_1, \Omega_2) e^{-j\Omega_2} + \mathbf{F}(\Omega_1, \Omega_2) e^{j\Omega_2} \\ &\quad - 4\mathbf{F}(\Omega_1, \Omega_2) \\ &= \mathbf{F}(\Omega_1, \Omega_2) [(e^{-j\Omega_1} + e^{j\Omega_1}) + (e^{-j\Omega_2} + e^{j\Omega_2}) - 4] \\ &= \mathbf{F}(\Omega_1, \Omega_2) [2\cos\Omega_1 + 2\cos\Omega_2 - 4]. \end{aligned} \quad (5.24)$$

The spectrum  $\mathbf{G}(\Omega_1, \Omega_2)$  is the product of the spectrum of the original image,  $\mathbf{F}(\Omega_1, \Omega_2)$  and the **Laplacian's spatial frequency response**  $\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2)$ :

$$\mathbf{G}(\Omega_1, \Omega_2) = \mathbf{F}(\Omega_1, \Omega_2) \mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2). \quad (5.25)$$

Equating Eqs. (5.24) and (5.25) leads to

$$\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2) = 2[\cos\Omega_1 + \cos\Omega_2 - 2]. \quad (5.26)$$

In **Fig. 5-5**, we display a plot of  $|\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2)|$  as a function of  $\Omega_1$ , with (for simplicity)  $\Omega_2$  set equal to zero. The plot would look the same as a function of the **radial discrete-space frequency**

$$\mathcal{R} = [\Omega_1^2 + \Omega_2^2]^{1/2}.$$

The frequency response  $|\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2)|$  exhibits a shape similar to that of a high-frequency filter, with  $|\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2)| = 0$  at the origin and then increasing rapidly with increasing  $\mathcal{R}$ . In

fact, in close proximity to the origin, such that  $|\Omega_1|, |\Omega_2| \ll 1$ , expansion of the cosine functions in Eq. (5.26) in a Taylor series gives

$$\cos\Omega_1 \approx 1 - \frac{\Omega_1^2}{2} + \dots,$$

$$\cos\Omega_2 \approx 1 - \frac{\Omega_2^2}{2} + \dots,$$

which when used in Eq. (5.26), the latter simplifies to

$$\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2) \approx -\Omega_1^2 - \Omega_2^2 = -\mathcal{R}^2, \quad \text{for } \mathcal{R} \ll 1. \quad (5.27)$$

This frequency dependence of the discrete-space Laplacian is analogous to the response given by Eq. (5.18b) for the frequency response of the continuous-space Laplacian; both have negative signs and both vary as the square of the spatial frequency ( $\rho$  and  $\mathcal{R}$ ).

The blue plot in **Fig. 5-5** represents the approximate expression for  $|\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2)|$  given by Eq. (5.27). It confirms that the approximation is valid not only for  $|\Omega_1|, |\Omega_2| \ll 1$ , but also up to  $|\Omega_1|, |\Omega_2| \approx 1$ .

In **Fig. 5-5**, the plots for the exact and approximate expressions of  $|\mathbf{H}_{\text{Laplace}}(\Omega_1, 0)|$  are displayed over the range  $-\pi < \Omega_1 < \pi$ . They are in close agreement over approximately the central one-third of the spectral range, and they deviate significantly as  $|\Omega_1|$  exceeds 1 (or  $|\Omega_2|$  exceeds 1), or more generally, as the radial frequency  $\mathcal{R}$  exceeds 1. In most images, the bulk of the image “energy” is contained within this central region.

This last statement deserves further elaboration. To do so, we refer the reader to Eq. (2.64), which relates the frequency  $\Omega_0$  in discrete time to the frequency  $f_0$  in continuous time, namely

$$\Omega_0 = 2\pi f_0 \Delta, \quad (5.28)$$

where  $\Delta$  is the sampling interval in seconds. Extending the relationship to 2-D provides the connections

$$\Omega_1 = 2\pi \mu \Delta \quad (5.29a)$$

and

$$\Omega_2 = 2\pi v \Delta, \quad (5.29b)$$

where now  $\Omega_1$  and  $\Omega_2$  are continuous spatial frequencies associated with the discrete image  $f[n, m]$ ,  $\mu$  and  $v$  are spatial frequencies (in cycles/m) associated with the continuous image  $f(x, y)$ , and  $\Delta$  is the sampling length in meters/sample. Since  $2\pi$

represents the amount in radians per a single complete cycle, *the units of  $\Omega_1$  and  $\Omega_2$  are radians/sample*.

If an image  $f(x,y)$  is sampled at the Nyquist rate such that  $\Delta = 1/2B$ , where  $B$  is the maximum spatial frequency of the image spectrum  $\mathbf{F}(\mu, \nu)$ , then the maximum discrete-space frequency is

$$\Omega_1(\max) = 2\pi \times B \times \frac{1}{2B} = \pi, \quad (5.30)$$

and the same conclusion applies to  $\Omega_2$  and  $\mathcal{R}$ . This is why the plots in Fig. 5-5 extend over the range  $-\pi \leq \Omega_1 \leq \pi$ .

Most images are sampled at rates greater than the Nyquist rate. If an image is sampled at three times the Nyquist rate (i.e., at  $\Delta = 1/6B$ ), then  $\Omega_1(\max) = \pi/3 \approx 1$ . In such a case, the approximation given by Eq. (5.27) becomes valid over the complete relevant ranges of  $\Omega_1$ ,  $\Omega_2$ , and  $\mathcal{R}$ .

## 5-2.4 Image Sharpening

An image  $f[n,m]$  can be sharpened into image  $g_{\text{sharp}}[n,m]$  by subtracting  $g[n,m]$  of Eq. (5.20) from the original image:

$$\begin{aligned} g_{\text{sharp}}[n,m] &= f[n,m] - f[n,m] \ast \ast h_{\text{Laplace}}[n,m] \\ &= f[n,m] \ast \ast h_{\text{sharp}}[n,m], \end{aligned} \quad (5.31)$$

where  $h_{\text{sharp}}[n,m]$  is an *image sharpening filter* with PSF

$$h_{\text{sharp}}[n,m] = \delta[n] \delta[m] - h_{\text{Laplace}}[n,m]. \quad (5.32)$$

This operation is analogous to Eq. (5.14) for film photography, except that in the present case we used a minus sign (rather than a plus sign) in the first step of Eq. (5.31) because

$$f[n,m] \ast \ast h_{\text{Laplace}}[n,m] \rightarrow \mathbf{F}(\Omega_1, \Omega_2) \mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2), \quad (5.33)$$

and  $\mathbf{H}_{\text{Laplace}}(\Omega_1, \Omega_2)$  is always negative.

Use of Eq. (5.21) in Eq. (5.32) leads to

$$h_{\text{sharp}}[n,m] = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \quad (5.34)$$

Since  $h_{\text{sharp}}[n,m]$  is only  $3 \times 3$ , it is faster to compute the 2-D convolution given by Eq. (5.31) in the spatial  $[n,m]$  domain than by multiplying zero-padded 2-D DFTs.

In a later part of this section, we will compare sharpened images to their original versions, but we should note that:

► A common detractor of all image sharpening algorithms is that because they emphasize high spatial frequencies, they also tend to emphasize high spatial-frequency noise. So in general, sharpened images tend to be noisy. ◀

## 5-2.5 Valid Convolution

When convolving an image  $f[n,m]$  with a filter characterized by a PSF  $h[n,m]$ , it is important that edge effects are dealt with appropriately. If the image size is  $(M \times M)$  and the filter size is  $(L \times L)$ , the *convolved image*

$$y[n,m] = f[n,m] \ast \ast h[n,m] \quad (5.35)$$

is  $(N \times N)$ , with  $N = M + L - 1$ . However, some parts of  $y[n,m]$  are not “valid” because they include pixels that are a result of convolving  $h[n,m]$  with pixels outside of the boundaries of  $f[n,m]$ . The “valid” part of  $y[n,m]$ , which we call  $y_{\text{valid}}[n,m]$ , is given by

$$y_{\text{valid}}[n,m] = \{y[n,m], 0 \leq n, m \leq N_{\text{valid}}\}, \quad (5.36)$$

where  $N_{\text{valid}} = M - L + 1$ . To illustrate with an example, let us consider the image given by

$$f[n,m] = \begin{bmatrix} 4 & 8 & 12 \\ 16 & 20 & 24 \\ 28 & 32 & 36 \end{bmatrix}, \quad (5.37a)$$

and let us assume that we wish to perform local averaging by sliding a  $2 \times 2$  window across the image, both horizontally and vertically. Such a filter has a PSF given by

$$h[n,m] = \frac{1}{4} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}. \quad (5.37b)$$

Upon performing the convolution given by Eq. (5.35) onto the arrays given in Eqs. (5.37a and b), we obtain

$$y[n,m] = \begin{bmatrix} 1 & 3 & 5 & 3 \\ 5 & 12 & 16 & 9 \\ 11 & 24 & 28 & 15 \\ 7 & 15 & 17 & 9 \end{bmatrix}. \quad (5.38)$$

The border rows and columns are not the average values of 4 neighboring pixels, but of only 1 neighboring pixel and 3 zeros or 2 neighboring pixels and 2 zeros. These are invalid entries. The more realistic valid output is  $y_{\text{valid}}[n,m]$ , obtained via Eq. (5.33) or equivalently, by removing the top and bottom

rows and the columns at the far left and far right. Either approach leads to

$$y_{\text{valid}}[n, m] = \begin{bmatrix} 12 & 16 \\ 24 & 28 \end{bmatrix}. \quad (5.39)$$

Since  $f[n, m]$  is  $M \times M = 3 \times 3$  and  $h[n, m]$  is  $L \times L = 2 \times 2$ ,  $y_{\text{valid}}[n, m]$  is  $N_{\text{valid}} \times N_{\text{valid}}$  with

$$N_{\text{valid}} = M - L + 1 = 3 - 2 + 1 = 2.$$

We now use  $y_{\text{valid}}[n, m]$  as the filtered image in the following two examples:

(1) Image of an electronic circuit, before and after applying the sharpening algorithm: [Fig. 5-6](#).

(2) Image of US coins, before and after sharpening: [Fig. 5-7](#).

**Concept Question 5-3:** Why does image sharpening not work well on noisy images?

**Concept Question 5-4:** Why is the Laplacian a common choice for a high-pass filter?

**Exercise 5-3:** Show that the valid convolution of a Laplacian with a constant image  $f[n, m] = c$  is zero.

**Answer:** The valid convolution (defined in Section 5-2.5) is the usual convolution with edge effects omitted.

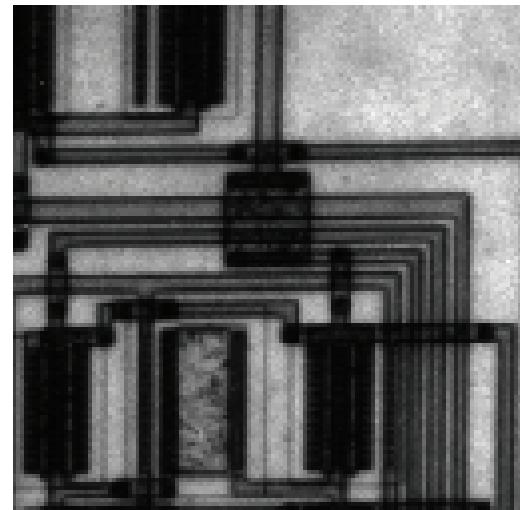
$$f[n, m] * * h_{\text{Laplace}}[n, m] = c \sum_{n=-1}^1 \sum_{m=-1}^1 \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = 0.$$

## 5-3 Histogram Equalization

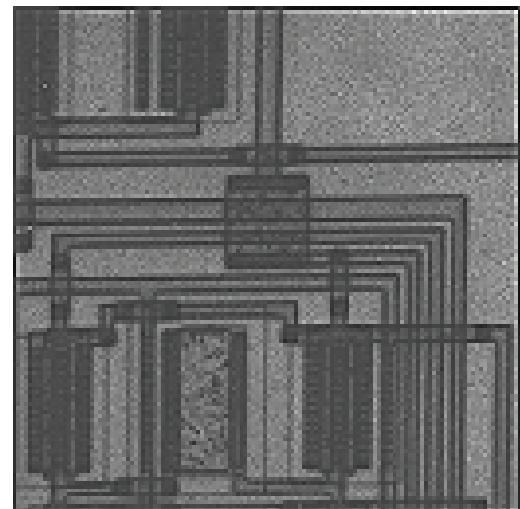
Consider the two clown images shown in parts (a) and (b) of [Fig. 5-8](#). The first one, labeled “original image,” is rather dark, making it difficult to discern some of the features in the clown’s face. In contrast, the image labeled “histogram-equalized” exhibits a broader range of intensities, thereby allowing the viewer to see details that are difficult to see in the original image. What is a **histogram-equalized image**? That is the topic of the present section.

Given an image  $f[n, m]$  with pixel values that extend over the input **dynamic range**  $R_i$ , with

$$R_i = \{ f_{\min} \leq f[n, m] \leq f_{\max} \}, \quad (5.40a)$$



(a) Original image

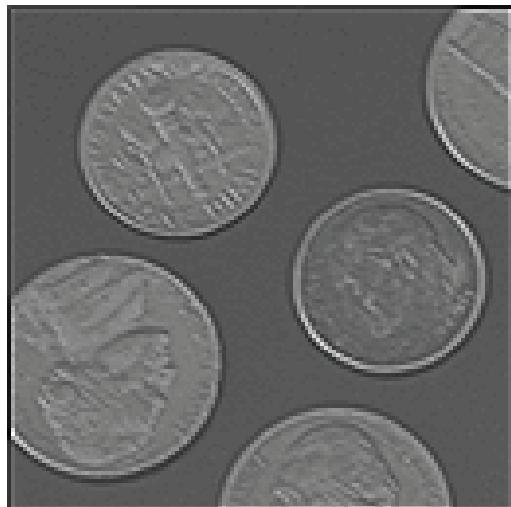


(b) Sharpened image

**Figure 5-6** Image of electronic circuit before and after application of the sharpening filter.



(a) Original image



(b) Sharpened image

**Figure 5-7** Coins image, before and after sharpening.

the objective of histogram equalization is to convert the pixel values  $f[n, m]$  into a new set  $g[n, m]$  so that they become more evenly distributed across the dynamic range of the display device  $R_d$ , with

$$R_d = \{ 0 \leq g[n, m] \leq g_{\max} \}. \quad (5.40b)$$

In Section 5-1, the conversion from  $f[n, m]$  to  $g[n, m]$  was accomplished by “stretching”  $R_i$  to fit  $R_d$ , but now we explore a different approach that relies on converting the histogram of  $f[n, m]$  into a new histogram associated with the converted image  $g[n, m]$ . The **histogram** of an image (or signal) is simply a bar graph of the number of times that each pixel value occurs in the image. The histograms associated with the original and histogram-equalized images shown in **Figs. 5-8(a)** and **(b)** are displayed in **Figs. 5-8(c)** and **(d)**. In both histograms, the horizontal axis represents pixel value and the vertical axis represents the number of times that a specific pixel value occurs in the image. For image  $f[n, m]$ , its continuous original  $f(x, y)$  had been sampled to  $200 \times 200$  discrete locations  $[n, m]$ , and its non-negative values had been quantized using 8-bit precision, which means that the **pixel value**  $f_0$  can take on any integer value between 0 and 255:

$$f[n, m] \in \{ 0, 1, 2, \dots, 255 \}, \quad \text{for } \{ 0 \leq n, m \leq 199 \}.$$

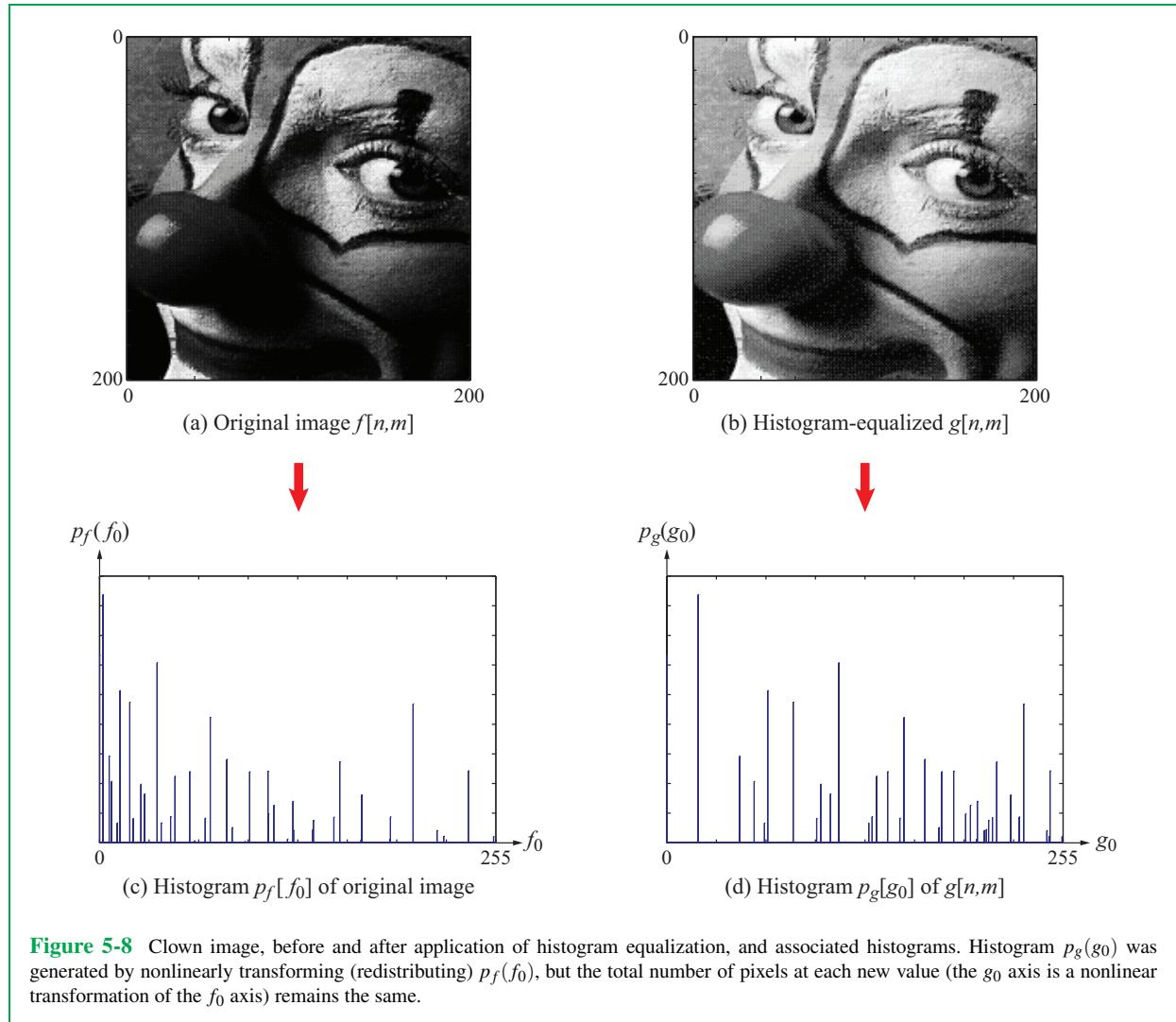
The horizontal axis in a histogram of  $f[n, m]$  represents the pixel value  $f_0$ , which may range between 0 and 255, and the vertical axis is  $p_f[f_0]$ , which is the number of times that pixel value  $f_0$  occurs in the image. We refer to  $p_f[f_0]$  as the **distribution function**, or histogram, of  $f[n, m]$ .

A related quantity is the **cumulative distribution function (CDF)**,  $P_f[f_0]$ , which is the cumulative sum of  $p_f[f_0]$ :

$$P_f[f_0] = \sum_{f'=0}^{f_0} p_f[f']. \quad (5.41)$$

The CDF  $P_f[f_0]$  is a non-decreasing function that jumps upward in value at the values of  $f_0$  for which  $p_f[f_0] \neq 0$ . **Figure 5-9(a)** displays the histogram of the original clown image,  $p_f[f_0]$ , and a plot of its associated CDF,  $P_f[f_0]$ . We observe that  $P_f[f_0]$  increases rapidly with  $f_0$  between  $f_0 = 0$  and  $f_0 = 100$  and at a lower rate at higher values of  $f_0$ . Thus, the majority of the image pixels are concentrated in the range corresponding to low image intensity, thereby giving the image its dark appearance.

The **pixel-value transformation** converts each pixel value  $f[n, m]$  into a new pixel value  $g[n, m]$ , based on the CDF  $P_f[f_0]$ ,



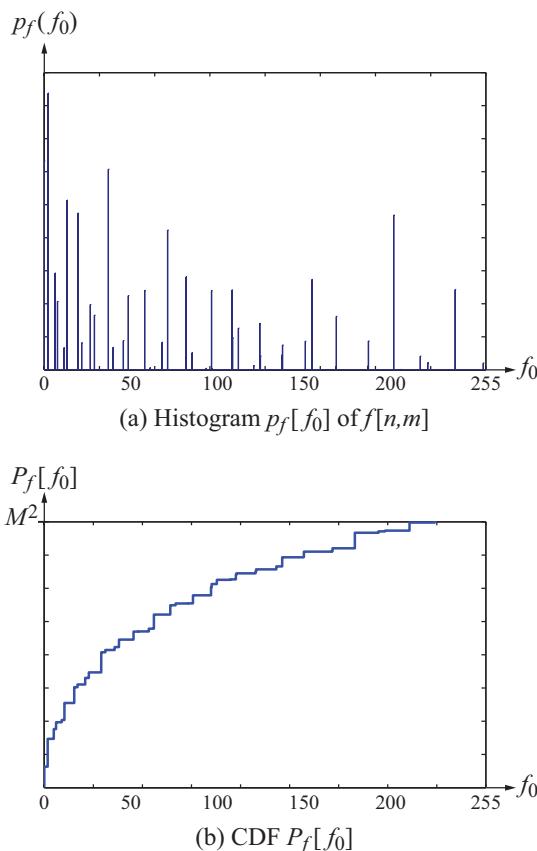
**Figure 5-8** Clown image, before and after application of histogram equalization, and associated histograms. Histogram  $p_g(g_0)$  was generated by nonlinearly transforming (redistributing)  $p_f(f_0)$ , but the total number of pixels at each new value (the  $g_0$  axis is a nonlinear transformation of the  $f_0$  axis) remains the same.

evaluated at  $f_0 = f[n,m]$ . That is,

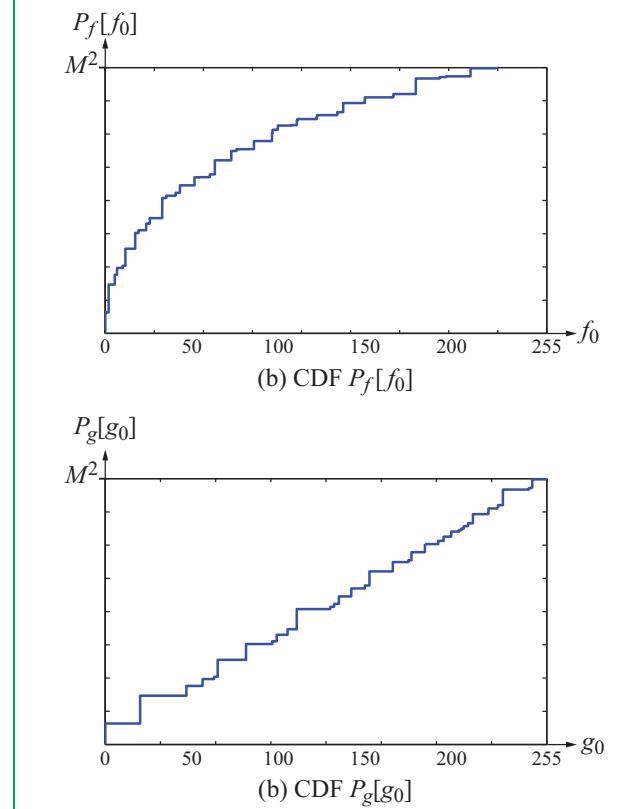
$$g[n,m] = P_f[f_0]|_{f_0=f[n,m]}. \quad (5.42)$$

Such a transformation leads to a histogram  $p_g(g_0)$  that is more

uniformly spread out over the range 0 to 255 than the histogram of the original image,  $p_f(f_0)$ . The associated CDF,  $P_g[g_0]$ , approximates a straight line that starts at coordinates  $(0,0)$  and concludes at  $(255, M^2)$ , where  $M^2$  is the total number of pixels. These attributes are evident in Fig. 5-10 for the histogram-equalized clown image.



**Figure 5-9** Histogram and associated CDF of original clown image  $f[n, m]$ . Image size is  $(M \times M)$ .



**Figure 5-10** CDF for an  $(M \times M)$  image before and after application of the histogram-equalization algorithm given by Eq. (5.42).

**Concept Question 5-5:** What is the purpose for using histogram equalization?

**Concept Question 5-6:** Why is histogram equalization often described as a nonlinear warping transformation of pixel values?

**Exercise 5-4:** Perform histogram equalization on the  $(2 \times 2)$  “image”  $\begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}$ .

**Answer:** The histogram of the image is: 0.2 occurring twice and 0.3 occurring twice. The CDF is

$$P_f(f_0) = \begin{cases} 0 & \text{for } 0 \leq f_0 < 0.2, \\ 2 & \text{for } 0.2 \leq f_0 < 0.3, \\ 4 & \text{for } 0.3 \leq f_0 < 1. \end{cases}$$

Pixel value 0.2 is mapped to  $P_f(0.2) = 2$  and pixel value 0.3 is mapped to  $P_f(0.3) = 4$ . The histogram-equalized “image” is  $\begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$ , which has a wider range of values than the original “image.”

## 5-4 Edge Detection

An **edge** in an image is a sharp boundary between two different regions of an image. Here “sharp” means a width of at most a few pixels, and a “boundary” means that significant differences exist in the pixel values between the two sides of the edge. “Significant” is not clearly defined; its definition depends on the characteristics of the image and the reason why edges are of interest. This is a nebulous definition, but there is no uniform definition for an edge.

The goal of **edge detection** is to determine the locations  $[n, m]$  of edges in an image  $f[n, m]$ . Edge detection is used to **segment** an image into different regions, or to determine the boundaries of a region of interest. For example, a medical image may consist of different human organs. Interpretation of the image is easier if (say) the region of the image corresponding to the pancreas is identified separately from the rest of the image. Identification of a face is easier if the eyes in an image of the face are identified as a region separate from the rest of the image. Ideally, an edge is a contour that encloses a region of the image whose values differ significantly from the values around it. Edge detection also is important in computer vision.

### 5-4.1 1-D Edge Detection

We start by examining a simple 1-D edge-detection method, as it forms the basis for a commonly used 2-D edge-detection algorithm.

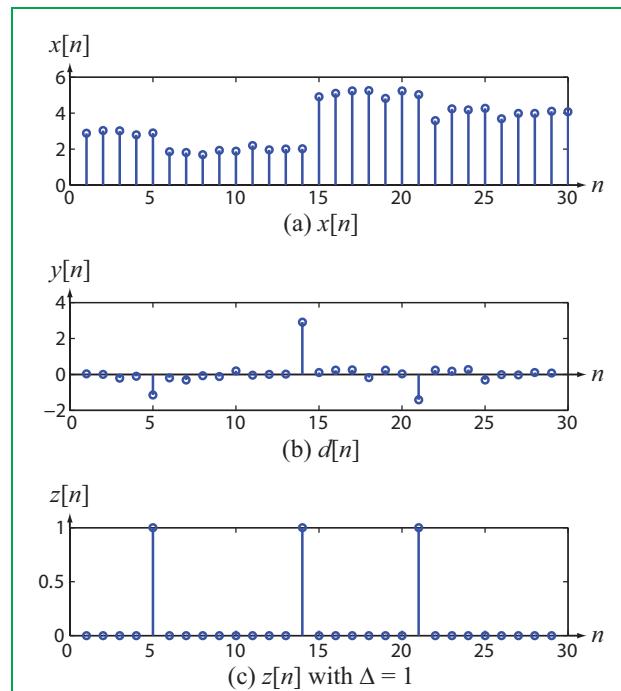
An obvious approach to detecting the locations of sharp changes in a 1-D signal  $x(t)$  is to compute its derivative  $x'(t) = dx/dt$ . Rapid changes of  $x(t)$  with  $t$  generate derivatives with large magnitudes, and slow changes generate derivatives with small magnitudes. The times  $t_0$  at which  $|x'(t_0)|$  is large represent **potential edges** of  $x(t)$ . The threshold for “large” has to be defined in the context of the signal  $x(t)$  itself.

For a 1-D discrete-time signal  $x[n]$ , the discrete-time counterpart to the derivative is the **difference operator**

$$d[n] = x[n+1] - x[n]. \quad (5.43)$$

The difference  $d[n]$  is large when  $x[n]$  changes rapidly with  $n$ , making it possible to easily pinpoint the time  $n_0$  of an edge. As simple as it is, computing the difference  $d[n]$  and thresholding  $|d[n]|$  is a very effective method for detecting 1-D edges. If the threshold is set at a value  $\Delta$ , the edge-detection algorithm can be cast as

$$z[n] = \begin{cases} 1 & \text{for } |d[n]| > \Delta, \\ 0 & \text{for } |d[n]| < \Delta. \end{cases} \quad (5.44)$$



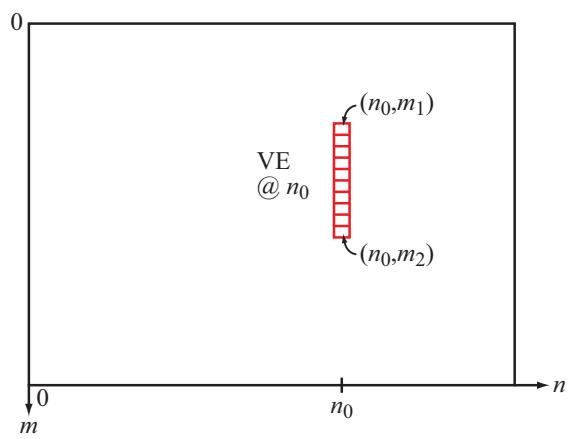
**Figure 5-11** Edge detection by thresholding absolute values of differences: (a) original signal  $x[n]$ , (b)  $d[n] = x[n+1] - x[n]$ , (c)  $z[n]$  with  $\Delta = 1$ .

The times  $n_i$  at which  $z[n_i] = 1$  denote the edges of  $x[n]$ . Specification of the threshold level  $\Delta$  depends on the **character** of  $x[n]$ . In practice, for a particular class of signals, the algorithm is tested for several values of  $\Delta$  so as to determine the value that provides the best results for the intended application.

For the signal  $x[n]$  displayed in Fig. 5-11(a), the difference operator  $d[n]$  was computed using Eq. (5.44) and then plotted in Fig. 5-11(b). It is evident that  $|d[n]|$  exhibits significant values at  $n = 5, 14$ , and  $21$ . Setting  $\Delta = 1$  would detect all three edges, as shown in part (c) of the figure, but had we chosen  $\Delta$  to be 2, for example, only the edge at  $n = 15$  would have been detected. The choice depends on the intended application.

### 5-4.2 2-D Edge Detection

The 1-D edge detection method can be extended to edge detection in 2-D images. Let us define a **vertical edge** (VE) as a vertical line at  $n = n_0$ , extending from  $m = m_1$  to  $m = m_2$ , as



**Figure 5-12** Vertical edge VE at  $n = n_0$  extends from  $m = m_1$  to  $m = m_2$ , and its length is  $(m_2 - m_1 + 1)$ .

shown in **Fig. 5-12**. That is,

$$\text{VE} = \{ [n, m] : n = n_0; m_1 \leq m \leq m_2 \}. \quad (5.45)$$

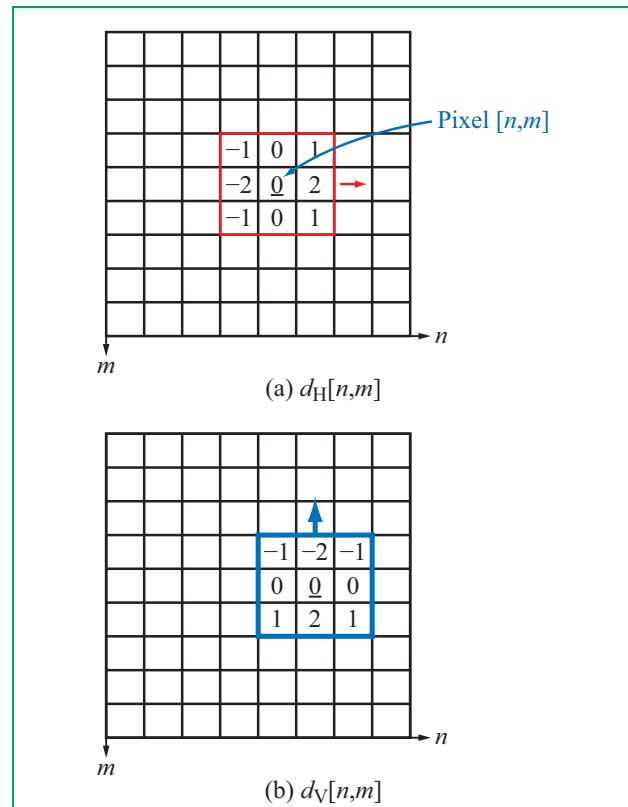
The total length of VE is  $(m_2 - m_1 + 1)$ .

One way to detect a vertical edge is to apply the difference operator given by Eq. (5.43) to each row of the image. In 2-D, the difference operator for row  $m$  is given by

$$d[n, m] = f[n+1, m] - f[n, m]. \quad (5.46)$$

If  $d[n, m]$  satisfies a specified threshold for a group of continuous pixels (all at  $n = n_0$ ) extending between  $m = m_1$  and  $m_2$ , then we call the group a vertical edge. In real images, we may encounter situations where  $d[n, m]$  may exhibit a large magnitude, but it is associated with a local variation in tone, not an edge. A vertical edge at  $n = n_0$  requires that not only  $d[n_0, m]$  at row  $m$  be large, but also that  $d[n_0, m+1]$  at the row above  $m$  and  $d[n_0, m-1]$  at the row below  $m$  be large as well. All three differences should be large and of the same polarity in order for the three pixels to qualify as a vertical edge.

This requirement suggests that a vertical edge detector should not only compute horizontal differences, but also vertical sums of the differences. The magnitude of a vertical sum becomes an indicator of the presence of a true vertical edge. A relatively simple edge operator is illustrated in **Fig. 5-13** for both a **horizontal-direction vertical-edge detector**  $d_H[n, m]$  and a **vertical-direction horizontal-edge detector**  $d_V[n, m]$ . Each



**Figure 5-13** Point spread functions  $d_H[n, m]$  (in red) and  $d_V[n, m]$  (in blue), displayed in center-of-image format.

detector consists of a  $3 \times 3$  window centered at the pixel of interest. Detector  $d_H[n, m]$  computes the difference between the values of pixel  $[n+1, m]$  and pixel  $[n-1, m]$ , whose positions are to the left and right of pixel  $[n, m]$ , respectively. Similar differences are performed for the row above and the row below row  $m$ . Then, the three differences are added up together, with the middle difference assigned twice the weight of the two other differences. The net result is

$$\begin{aligned} d_H[n, m] &= f[n+1, m+1] - f[n-1, m+1] \\ &\quad + 2f[n+1, m] - 2f[n-1, m] \\ &\quad + f[n+1, m-1] - f[n-1, m-1]. \end{aligned} \quad (5.47)$$

The coefficients of the six terms of Eq. (5.47) are the nonzero weights shown in **Fig. 5-13(a)**.

Computing  $d_H[n, m]$  for every pixel is equivalent to validly convolving (see Section 5-2.5) image  $f[n, m]$  with the window's point spread function  $h_H[n, m]$  along the horizontal direction. That is,

$$d_H[n, m] = f[n, m] * * h_H[n, m], \quad (5.48)$$

with

$$h_H[n, m] = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \quad (5.49)$$

To compute  $d_H[n, m]$  for all pixels  $[n, m]$  in the image, it is necessary to add an extra row above of and identical with the top row, and a similar add-on is needed at the bottom end of the image. The decision as to whether or not a given pixel is part of a vertical edge is made by comparing the magnitude of  $d_H[n, m]$  with a predefined **gradient threshold**  $\Delta$  whose value is selected heuristically (based on practical experience for the class of images under consideration).

Horizontal edges can be detected by a vertical-direction edge detector  $d_V[n, m]$  given by

$$d_V[n, m] = f[n, m] * * h_V[n, m], \quad (5.50)$$

where  $h_V[n, m]$  is the **point spread function (PSF)** for a pixel  $(n, m)$ . By exchanging the roles of the rows and column in Eq. (5.49), we have

$$h_V[n, m] = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \quad (5.51)$$

Of course, most edges are neither purely horizontal nor purely vertical, so an edge at an angle different from  $0^\circ$  or  $90^\circ$  (with  $0^\circ$  denoting the horizontal dimension of the image) should have edge components along both the horizontal and vertical directions. Hence, the following **edge-detection gradient** is often used:

$$g[n, m] = \sqrt{d_H^2[n, m] + d_V^2[n, m]}. \quad (5.52)$$

For each pixel  $[n, m]$ , we define the **edge indicator**  $z[n, m]$  as

$$z[n, m] = \begin{cases} 1 & \text{if } g[n, m] > \Delta, \\ 0 & \text{if } g[n, m] < \Delta, \end{cases} \quad (5.53)$$

where, again,  $\Delta$  is a prescribed **gradient threshold**. In the image, pixels for which  $z[n, m] = 1$  are shown in white, and those with  $z[n, m] = 0$  are shown in black. Usually, the value of  $\Delta$  is selected empirically by examining a histogram of  $g[n, m]$  or through repeated trials.

### 5-4.3 Sobel Edge Detector Examples

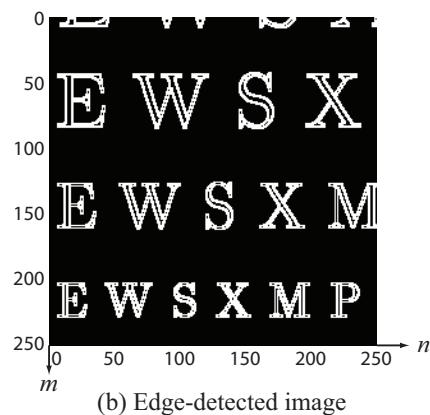
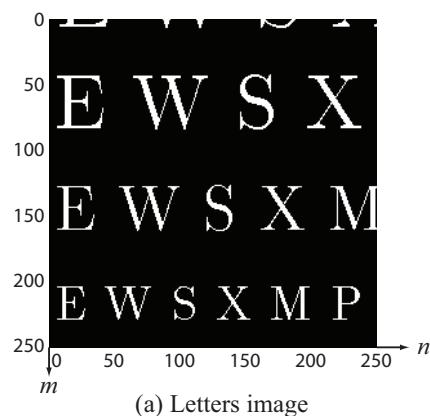
The gradient algorithm given by Eq. (5.53) is known as the **Sobel edge detector**, named after Irwin Sobel, who developed it in 1968, when computer-based image processing was in its infancy and only simple algorithms could be used. Application of the Sobel edge detector to the letters image in part (a) of Fig. 5-14 leads to the image in part (b). Through repeated applications using different values of  $\Delta$ , it was determined that  $\Delta = 200$  provided an image with clear edges, including diagonal and curved edges. The value specified for  $\Delta$  depends in part on the values assigned to black and white tones in the image.

The Sobel edge detector does not always capture all of the major edges contained in an image. When applied to the clown image of Fig. 5-15(a), the edge detector identified some parts of continuous edges, but failed to identify others, which suggests the need for a detector that can track edges and complete edge contours as needed. Such a capability is provided by the Canny edge detector, the subject of the next subsection.

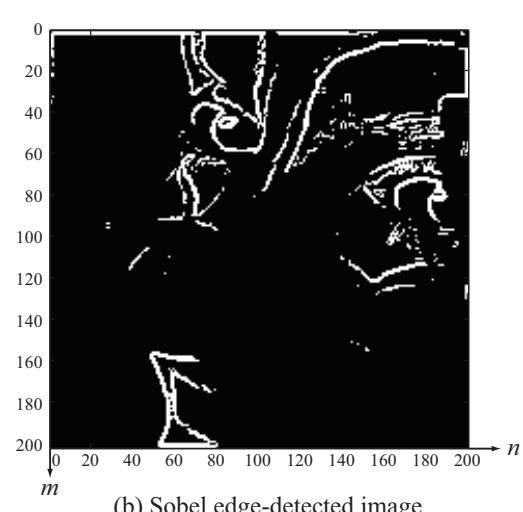
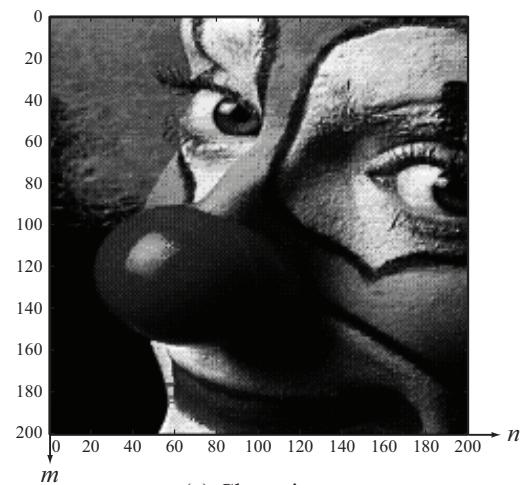
### 5-4.4 Canny Edge Detector

The **Canny edge detector** is a commonly used algorithm that extends the capabilities of the Sobel detector by applying preprocessing and postprocessing steps. The preprocessing step involves the use of a 2-D Gaussian PSF to reduce image noise and to filter out isolated image features that are not edges. After computing the Sobel operator given by Eq. (5.52), the Canny algorithm performs an **edge thinning** step, separating detected edges into different candidate categories, and then applies certain criteria to decide whether or not the candidate edges should be connected together. The five-step process of the Canny detection algorithm are:

**Step 1:** Image  $f[n, m]$  is **blurred** (filtered) by convolving it with a truncated Gaussian point spread function. An example of a practical function that can perform the desired operation is the



**Figure 5-14** Application of the Sobel edge detector to the image in (a) with  $\Delta = 200$  led to the image in (b).



$5 \times 5$  PSF given by

$$h_G[n, m] = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}. \quad (5.54)$$

The standard deviation of the truncated Gaussian function is 1.4.

Application of  $h_G[n, m]$  to image  $f[n, m]$  generates a filtered image  $f_1[n, m]$  given by

$$f_1[n, m] = h_G[n, m] * * f[n, m]. \quad (5.55)$$

**Figure 5-15** Application of the Sobel edge detector to the image in (a) captures some of the edges in the image, but also misses others.

**Step 2:** For image  $f_1[n, m]$ , compute the horizontal and vertical edge detectors given by Eqs. (5.48) and (5.50).

**Step 3:** Compute the gradient magnitude and orientation:

$$g[n, m] = \sqrt{d_H^2[n, m] + d_V^2[n, m]} \quad (5.56a)$$

and

$$\theta[n, m] = \tan^{-1} \left( \frac{d_V[n, m]}{d_H[n, m]} \right). \quad (5.56b)$$

For a vertical edge,  $d_V[n, m] = 0$ , and therefore  $\theta[n, m] = 0$ . Similarly, for a horizontal edge,  $d_H[n, m] = 0$  and  $\theta = 90^\circ$ .

**Step 4:** At each pixel  $[n, m]$ , round  $\theta[n, m]$  to the nearest of  $\{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$ . Next, determine whether to keep the value of  $g[n, m]$  of pixel  $[n, m]$  as is or to replace it with zero. The decision logic is as follows:

(a) For a pixel  $[n, m]$  with  $\theta[n, m] = 0^\circ$ , compare the value of  $g[n, m]$  to the values of  $g[n+1, m]$  and  $g[n-1, m]$ , corresponding to the pixels at the immediate right and left of pixel  $[n, m]$ . If  $g[n, m]$  is the largest of the three gradients, keep its value as is; otherwise, set it to zero.

(b) For a pixel  $[n, m]$  with  $\theta = 45^\circ$ , compare the value of  $g[n, m]$  to the values of  $g[n-1, m+1]$  and  $g[n+1, m-1]$ , corresponding to the pixel neighbors along the  $45^\circ$  diagonal. If  $g[n, m]$  is the largest of the three gradients, keep its value as is; otherwise, set it to zero.

(c) For a pixel  $[n, m]$  with  $\theta = 90^\circ$ , compare the value of  $g[n, m]$  to the values of  $g[n, m-1]$  and  $g[n, m+1]$ , corresponding to pixels immediately above and below pixel  $[n, m]$ . If  $g[n, m]$  is the largest of the three gradients, keep its value as is; otherwise, set it to zero.

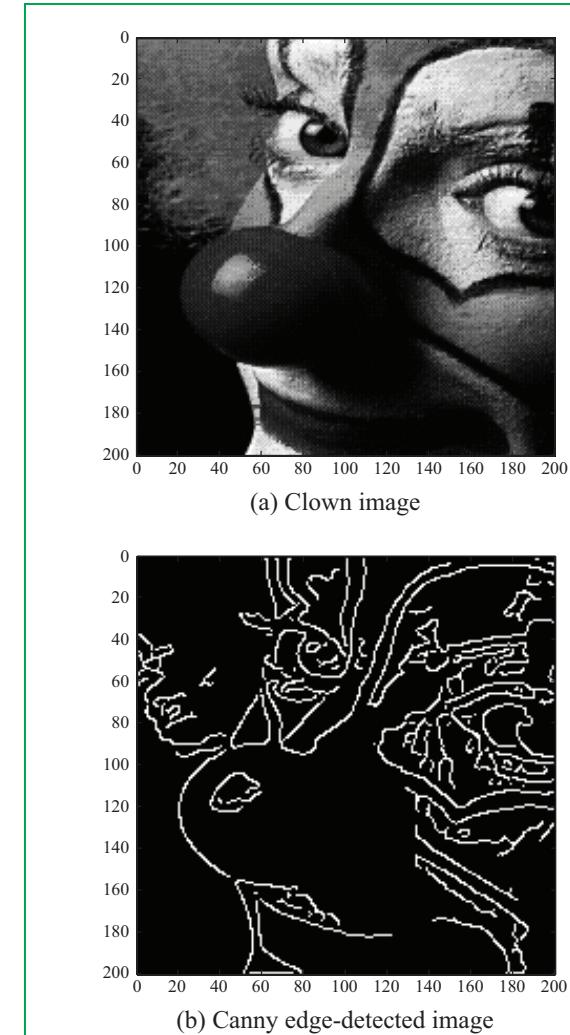
(d) For a pixel  $[n, m]$  with  $\theta = 135^\circ$ , compare the value of  $g[n, m]$  to the values of  $g[n-1, m-1]$  and  $g[n+1, m+1]$ . If  $g[n, m]$  is the largest of the three gradients, keep its value as is; otherwise, set it to zero.

The foregoing operation is called **edge thinning**, as it avoids making an edge wider than necessary in order to indicate its presence.

**Step 5:** Replace the edge indicator algorithm given by Eq. (5.53) with a double-threshold algorithm given by

$$z[n, m] = \begin{cases} 2 & \text{if } g[n, m] > \Delta_2, \\ 1 & \text{if } \Delta_1 < g[n, m] < \Delta_2, \\ 0 & \text{if } g[n, m] < \Delta_1. \end{cases} \quad (5.57)$$

The edge indicator  $z[n, m]$  may assume one of three values, indicating the **presence** of an edge ( $z[n, m] = 2$ ), the **possible presence** of an edge ( $z[n, m] = 1$ ), and the **absence** of an edge ( $z[n, m] = 0$ ). The middle category requires resolution into one



**Figure 5-16** The Canny edge detector provides better edge-detection performance than the Sobel detector in Fig. 5-15.

of the other two categories. This is accomplished by converting pixel  $[n, m]$  with  $z[n, m] = 1$  into an edge if any one of its nearest 8 neighbors is a confirmed edge. That is, pixel  $[n, m]$  is an edge location only if it adjoins another edge location.

The values assigned to thresholds  $\Delta_1$  and  $\Delta_2$  are selected through multiple trials. For example, the clown edge-image shown in Fig. 5-16 was obtained by applying the Canny algo-

rithm to the clown image with  $\Delta_1 = 0.05$  and  $\Delta_2 = 0.125$ . This particular combination provides an edge-image that successfully captures the contours that segment the clown image.

► Edge detection can be implemented in MATLAB's Image Processing Toolbox using the commands

`E=edge(X, 'sobel', T1)` for Sobel and  
`E=edge(X, 'canny', T1, T2)` for Canny.

The image is stored in array `X`, the edge image is stored in array `E` and `T1` and `T2` are the thresholds. MATLAB assigns default values to the thresholds, computed from the image, if they are not specified. ◀

**Concept Question 5-7:** Why does edge detection not work well on noisy images?

**Concept Question 5-8:** In edge detection, why do we not simply take differences, as in Eq. (5.46), instead of taking differences in one direction and sums in the other, as in Eq. (5.47)?

**Exercise 5-5:** Show that edge detection applied to a constant image  $f[n, m] = c$  gives no edges.

**Answer:** The valid convolution (defined in Section 5-2.5) is the usual convolution with edge effects omitted. The two valid convolutions ( $h_H[n, m]$  and  $h_V[n, m]$ ) are defined in Eq. (5.49) and Eq. (5.51))

$$f[n, m] \ast \ast h_H[n, m] = c \sum_{n=-1}^1 \sum_{m=-1}^1 \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = 0,$$

$$f[n, m] \ast \ast h_V[n, m] = c \sum_{n=-1}^1 \sum_{m=-1}^1 \begin{bmatrix} -1 & -1 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = 0,$$

Hence, the edge-direction gradient  $g[n, m]$  defined in Eq. (5.52) is zero.

## 5-5 Summary of Image Enhancement Techniques

- To increase contrast in an image by altering the range of pixel values, use a gamma transformation. Simply try different values of  $\gamma$  and choose the one that gives the best visual result.
- To compress the range of values; e.g., in a spectrum, use a log transformation.
- To sharpen an image, use unsharp masking, bearing in mind that this also makes the image noisier.
- To make an image brighter, use histogram equalization.
- To detect edges, or create an image of edges, use Canny edge detection (if available), otherwise use Sobel edge detection.

## Summary

### Concepts

- Image enhancement transforms a given image into another image in which image features such as edges or contrast has been enhanced to make them more apparent.
- Linear, logarithmic, and gamma transformations alter the range of pixel values so that they fit the range of the display.

- Unsharp masking and Laplacians sharpen an image, but increase noise.
- Histogram equalization nonlinearly alters pixel values to brighten images.
- Edge detection produces an image consisting entirely of edges of the image.

### Mathematical Formulae

#### Linear transformation

$$g[n, m] = g_{\max} \frac{f[n, m] - f_{\min}}{f_{\max} - f_{\min}}$$

#### Logarithmic transformation

$$g[n, m] = a \log_{10}(f[n, m] + b)$$

#### Gamma transformation

$$g[n, m] = g_{\max} \left( \frac{f[n, m] - f_{\min}}{f_{\max} - f_{\min}} \right)^\gamma$$

#### Unsharp masking

$$g(x, y) = f(x, y) + \underbrace{f(x, y) - f_{\text{blur}}(x, y)}_{f_{\text{mask}}(x, y)}$$

#### Unsharp masking

$$g[n, m] = f[n, m] - f[n, m] * * h_{\text{Laplace}}[n, m]$$

#### Laplacian

$$g(x, y) = \nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

#### Laplacian

$$\begin{aligned} g[n, m] = & f[n+1, m] + f[n-1, m] + f[n, m+1] \\ & + f[n, m-1] - 4f[n, m] \end{aligned}$$

#### Cumulative distribution

$$P_f[f_0] = \sum_{f'=1}^{f_0} p_f[f']$$

#### Histogram equalization

$$g[n, m] = P_f[f_0]_{f_0=f[n, m]}$$

#### Horizontal and vertical edge detectors

$$d_H[n, m] = f[n, m] * * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$d_V[n, m] = f[n, m] * * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

#### Sobel edge detector

$$z[n, m] = \begin{cases} 1 & \text{if } \sqrt{d_h[n, m]^2 + d_v[n, m]^2} > \Delta, \\ 0 & \text{if } \sqrt{d_h[n, m]^2 + d_v[n, m]^2} < \Delta \end{cases}$$

### Important Terms

Provide definitions or explain the meaning of the following terms:

Canny edge detector  
gamma transformation

histogram equalization  
Laplacian

logarithmic transformation  
Sobel edge detector

unsharp masking

## PROBLEMS

### Section 5-1: Pixel Value Transformations

**5.1** Explain why, in the gamma transformation (Eq. (5.10)),  $\gamma > 1$  tends to darken images, while  $\gamma < 1$  tends to lighten images.

**5.2** Use gamma transformation with  $\gamma = 3$  to darken the image in `coins1.mat`.

**5.3** Use gamma transformation with  $\gamma = 3$  to darken the image in `coins2.mat`.

### Section 5-2: Unsharp Masking

**5.4** Use the sharpening filter Eq. (5.32) to sharpen the two images in the files (a) `plane.mat` and (b) `coins2.mat`.

**5.5** Use the sharpening filter Eq. (5.32) to sharpen the two images in the files (a) `quarter.mat` and (b) `rice.mat`.

**5.6** Use sharpening filter Eq. (5.32) to sharpen the two images in the files (a) `moon.mat` and (b) `unsharp.mat`.

**5.7** Unsharp masking was originally based on Eq. (5.14), which in discrete space is

$$g[n, m] = f[n, m] + (f[n, m] - f_{\text{blur}}[n, m]).$$

$f_{\text{blur}}[n, m]$  is a lowpass version of  $f[n, m]$ . If  $f_{\text{blur}}[n, m]$  is the average of  $\{f[n+1, m], f[n-1, m], f[n, m+1], f[n, m-1]\}$ , show that

$$g[n, m] = f[n, m] - f[n, m] * \frac{1}{4} h_{\text{Laplacian}}[n, m],$$

similar to Eq. (5.32).

**5.8** Unsharp masking was originally based on Eq. (5.14), which is

$$g(x, y) = f(x, y) + (f(x, y) - f_{\text{blur}}(x, y)).$$

$f_{\text{blur}}(x, y)$  is a lowpass version of  $f(x, y)$ . Adobe® Photoshop® uses the following form of unsharp masking:

$$f_{\text{blur}}(x, y) = f(x, y) * f_g(\sqrt{x^2 + y^2}),$$

where

$$f_g(\sqrt{x^2 + y^2}) = \frac{e^{-(x^2+y^2)/(2\sigma^2)}}{2\pi\sigma^2}.$$

Plot a cross-section (with  $v = 0$ ) of the spatial frequency response of this form of unsharp masking with that of the continuous-space version of Eq. (5.32), which is

$$h_{\text{sharpen}}(x, y) = \delta(x) \delta(y) - \nabla^2 f(x, y).$$

*Hint:* Use the result of Problem 3.4, which is  $F_g(\rho) = e^{-2\sigma^2\pi^2\rho^2}$ . Use  $\sigma^2 = 2$ .

**5.9** Use unsharp masking as defined in Problem 5.8 to sharpen the two images in the files (a) `circuit.mat` and (b) `quarter.mat`. Use `unsharp.m`.

**5.10** Use unsharp masking as defined in Problem 5.8 to sharpen the two images in the files (a) `tire.mat` and (b) `coins2.mat`. Use `unsharp.m`.

### Section 5-3: Histogram Equalization

**5.11** This problem applies histogram equalization to a tiny ( $3 \times 3$ ) image. The goal is for the reader to work the problem entirely by hand, thereby aiding understanding. The  $(3 \times 3)$  image is

$$f[n, m] = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 9 \\ 3 & 2 & 9 \end{bmatrix}.$$

**(a)** Plot the histogram of the image.

**(b)** List its distribution and CDF in a table.

**(c)** List values of  $f[n, m]$  and values of the histogram-equalized image  $g[n, m]$  in a table.

**(d)** Depict  $g[n, m]$  as a  $3 \times 3$  matrix, similar to the depiction of  $f[n, m]$ .

**(e)** Depict  $f[n, m]$  and  $g[n, m]$  as images, and plot their respective histograms and CDF's.

**5.12** Use the program `hist.m` to apply histogram equalization to the image in `circuit.mat`. Print out the images, histograms, and CDFs of the original and equalized images.

**5.13** Use the program `hist.m` to apply histogram equalization to the image in `pout.mat`. Print out the images, histograms, and CDFs of the original and equalized images.

**5.14** Use the program `hist.m` to apply histogram equalization to the image in `tire.mat`. Print out the images, histograms, and CDFs of the original and equalized images.

**5.15** Use the program `hist.m` to apply histogram equalization to the image in `coins.mat`. Print out the images, histograms, and CDFs of the original and equalized images.

## Section 5-4: Edge Detection

**5.16** Sobel edge detection works by convolving the image  $f[n,m]$  with the two PSFs

$$h_H[n,m] = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

and

$$h_V[n,m] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

Compute the discrete-space frequency response  $\mathbf{H}(\Omega_1, \Omega_2)$  of each of these PSFs.

**5.17** Sobel edge detection works by convolving the image  $f[n,m]$  with the two PSFs

$$h_H[n,m] = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

and

$$h_V[n,m] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

Show how to implement these two convolutions using

- (a)  $16N^2$  additions and subtractions since doubling is two additions;
- (b)  $10N^2$  additions and subtractions since  $h_H[n,m]$  and  $h_V[n,m]$  are separable.

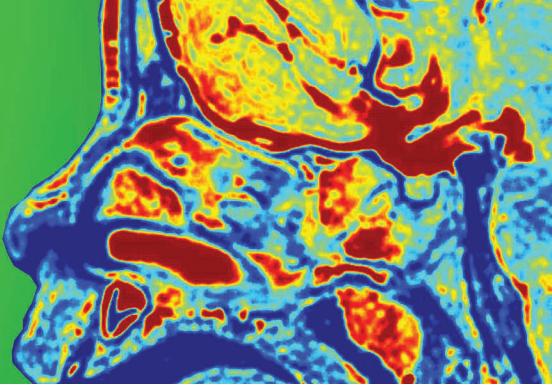
**5.18** Apply (a) Sobel edge detection and (b) Canny edge detection to the image in `plane.mat` using the programs `sobel.m` and `canny.m`. Compare results.

**5.19** Apply (a) Sobel edge detection and (b) Canny edge detection to the image in `quarter.mat` using the programs `sobel.m` and `canny.m`. Compare results.

**5.20** Apply (a) Sobel edge detection and (b) Canny edge detection to the image in `moon.mat` using the programs `sobel.m` and `canny.m`. Compare results.

**5.21** Apply (a) Sobel edge detection and (b) Canny edge detection to the image in `saturn.mat` using the programs `sobel.m` and `canny.m`. Compare results.

# CHAPTER 6



## 6 Deterministic Approach to Image Restoration

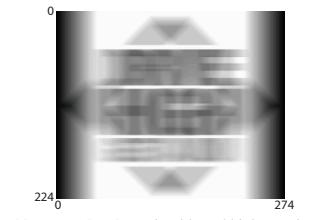
### Contents

- Overview, 181
- 6-1** Direct and Inverse Problems, 181
- 6-2** Denoising by Lowpass Filtering, 183
- 6-3** Notch Filtering, 188
- 6-4** Image Deconvolution, 191
- 6-5** Median Filtering, 194
- 6-6** Motion-Blur Deconvolution, 195
- Problems, 199

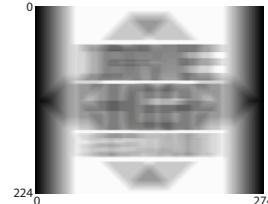
### Objectives

Learn to:

- Denoise a noisy image using the 2-D DFT and a Hamming-windowed lowpass filter.
- Notch-filter an image with sinusoidal interference added to it.
- Use median filtering to denoise an image with salt-and-pepper noise added to it.
- Use Tikhonov regularization in a deconvolution problem.
- Deconvolve an image blurred with a known point-spread function.
- Deblur a motion-blurred image.



(a) Image  $g[n,m]$ : motion-blurred highway sign



(b) Image  $g'[n,m]$ : motion-blurred highway sign with additive noise



(c) Reconstructed highway sign

This chapter covers image restoration from a noisy or blurred version of it, where the blur and noise were introduced by the imaging system. Denoising can be performed by lowpass-filtering the noisy image using the 2-D DFT (a Hamming-windowed filter works better than a brick-wall filter). Deblurring (deconvolution) can be performed using the 2-D DFT, although Tikhonov regularization is usually required. Motion-blur deblurring is a common application; refocusing an out-of-focus image is also common.

## Overview

The goal in **image restoration** is to recover the true image  $f[n,m]$ —or a close version of it—from its noisy or blurred version  $g[n,m]$ . Examples include:

(1) **Denoising**: removing noise that had been added by the imaging system.

(2) **Removing interference**: subtracting an unwanted image that had been added to  $f[n,m]$ .

(3) **Deblurring**: undoing the effect of the convolution of a system PSF  $h[n,m]$  with image  $f[n,m]$ , where  $h[n,m]$  is the system response due to motion blur or defocusing.

Image restoration methods are categorized into two groups: **deterministic** and **probabilistic**. Deterministic methods apply algorithms—such as lowpass filtering—that do not incorporate knowledge of probability and random processes associated with the image or the image formation process. The present chapter deals exclusively with deterministic restoration methods applied to discrete-space images. Image restoration methods using the probabilistic approach are treated in Chapter 9.

## 6-1 Direct and Inverse Problems

In image processing, we encounter two interrelated operations commonly called the **direct** and **inverse** problems, and our goal is to obtain solutions to both problems.

### 6-1.1 The Direct Problem

The solution to the direct problem consists of a mathematical model that correctly accounts for the two forms of distortions commonly encountered in the generation of an image: (1) **blurring** by the imaging system and (2) the introduction of **additive noise**. As noted in Chapter 1, an imaging sensor—be it our eye’s pupil, a camera, an MRI system, or any other 2-D image-forming configuration—has a non-zero “beam” described by a point spread function  $h(x,y)$ . The image formation process is equivalent to convolving a filter  $h(x,y)$  with the true image  $f(x,y)$ , which results in blurring. The second form of distortion involves the addition of random noise, which may be contributed entirely by the electronics of the imaging and recording systems, or it may also include a component due to peripheral sources in the imaged scene.

Mathematically, the direct problem is modeled as

$$g(x,y) = h(x,y) * * f(x,y) + v(x,y), \quad (6.1)$$

where

$$\begin{aligned} g(x,y) &= \text{recorded image}, \\ f(x,y) &= \text{true image of the scene}, \\ h(x,y) &= \text{PSF of the imaging system}, \\ v(x,y) &= \text{additive noise}. \end{aligned}$$

To illustrate the impact of each type of distortion separately, let us start with a noise-free imaging process by setting  $v(x,y) = 0$ . In [Fig. 6-1](#), we show a high-quality MRI image labeled  $f(x,y)$ . We will treat it as if it were a true image, and then through simulations, we will subject  $f(x,y)$  to convolution with  $h_i(x,y)$  to generate image  $g_i(x,y)$ :

$$g_i(x,y) = h_i(x,y) * * f(x,y), \quad (6.2)$$

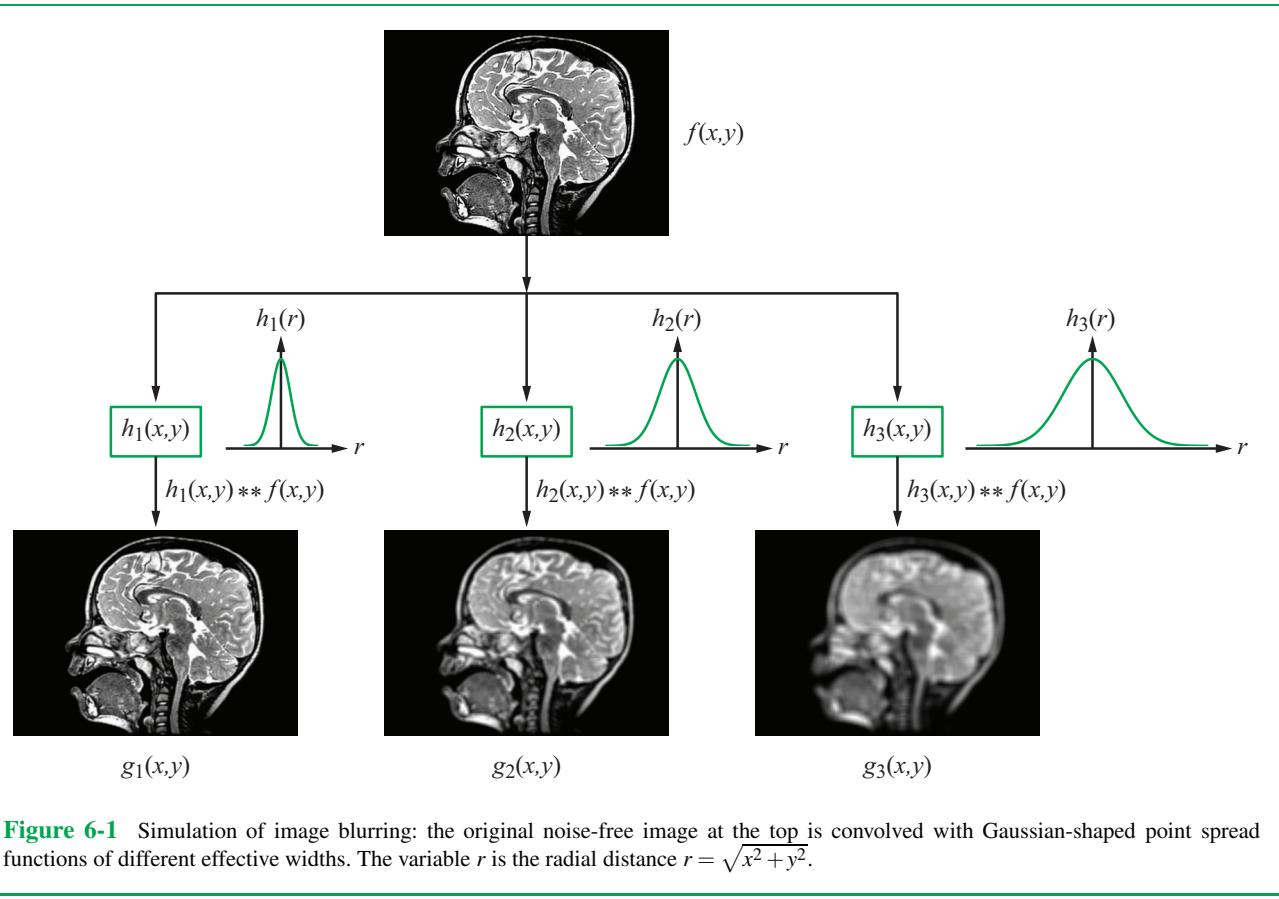
with index  $i = 1, 2$ , or  $3$  referring to Gaussian-shaped filters  $h_i(x,y)$  of different effective radii. Image  $g_1(x,y)$  is the result of convolving  $f(x,y)$  with a narrow PSF  $h_1(x,y)$ . Because  $h_1(x,y)$  is narrow, the blurring is visually unnoticeable. Image  $g_2(x,y)$  and  $g_3(x,y)$  are the result of convolving the same original image with a medium-wide PSF  $h_2(x,y)$  and wide PSF  $h_3(x,y)$ , respectively. Not surprisingly, increasing the filter’s effective width leads to more image blurring.

Next we simulate the impact that additive random noise imparts onto the appearance of an image. The noise is added to  $f(x,y)$  after convolving it with the narrow filter  $h_1(x,y)$ , so the applicable model is

$$g_{1j}(x,y) = h_1(x,y) * * f(x,y) + v_j(x,y), \quad (6.3)$$

where  $j = 1, 2$ , or  $3$  refers to three noise-addition simulations, characterized by different signal-to-noise ratios. All three output images in [Fig. 6-2](#) were generated by adding noise randomly to different segments of the convolved image, but in  $g_{11}(x,y)$ , the average power content of the added noise is much smaller than the average power of the noise-free image, whereas the signal (image) and noise powers are comparable to one another in  $g_{12}(x,y)$ , and the noise power is much larger than the signal power in  $g_{13}(x,y)$ . Noise distorts an image by changing its amplitude, whereas the PSF distorts it through spatial averaging. Most images include both types of distortions.

The solution to the direct problem entails computing  $g(x,y)$  from  $f(x,y)$  using Eq. (6.1). Doing so requires knowledge of the PSF of the imaging system,  $h(x,y)$ , and the statistical nature of the added noise  $v(x,y)$ . The PSF usually is determined through calibration tests in which the output  $g(x,y)$  is measured in response to a strong point-like target placed at the center of the scene, as illustrated in [Fig. 1-8](#). The strong target allows us to ignore the noise  $v(x,y)$ , and its point-like spatial extent makes it equivalent to a 2-D impulse  $\delta(x,y)$ . Setting  $v(x,y) = 0$  and



$f(x,y) = \delta(x,y)$  in Eq. (6.1) gives

$$g(x,y) = h(x,y) * * \delta(x,y) = h(x,y). \quad (6.4)$$

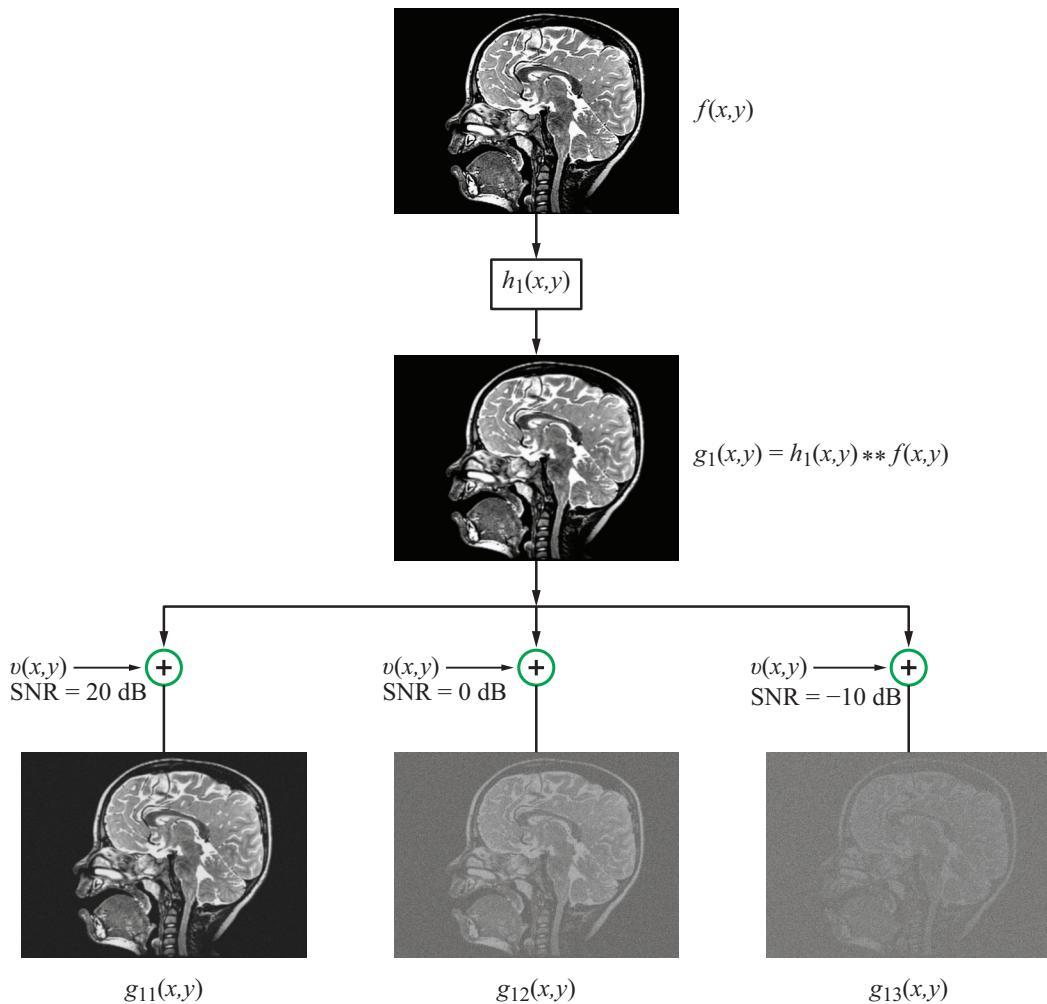
In most imaging systems, the noise  $v(x,y)$  is random in nature and usually modeled as a zero-mean Gaussian random variable (Chapter 8). Accordingly,  $v(x,y)$  is described by a **probability density function (pdf)** that contains a single parameter, the **noise variance**  $\sigma_v^2$ . The pdf and associated variance can be measured experimentally by recording the output  $g(x,y)$  for many locations  $(x,y)$ , while having no signal as input ( $f(x,y) = 0$ ). For a camera, this is equivalent to imaging a perfectly dark object. The recorded image in that case is the noise added by the camera.

Once  $h(x,y)$  has been characterized and  $v(x,y)$  has been modeled appropriately, image  $g(x,y)$  can be readily computed

using Eq. (6.1), thereby providing a possible solution of the direct problem. Because  $v(x,y)$  is random in nature, each simulation of Eq. (6.1) will result in a statistically different, but comparable image  $g(x,y)$ .

### 6-1.2 The Inverse Problem

Whereas the solution of the direct problem seeks to generate image  $g(x,y)$  from image  $f(x,y)$ , the solution of the inverse problem seeks to do the exact opposite, namely to extract the true image  $f(x,y)$ —or a close facsimile of  $f(x,y)$ —from the blurred and noisy image  $g(x,y)$ . The process involves (a) **denoising**  $g(x,y)$  by filtering out  $v(x,y)$ —or at least most of it—and (b) deconvolution of  $g(x,y)$  to generate a close approximation of the true image  $f(x,y)$ . The denoising and deconvolution steps of the inversion algorithm are performed using deterministic



**Figure 6-2** The image in the upper center,  $g_1(x,y)$ , had been convolved with the narrow filter before noise was added to it. SNR = 20 dB corresponds to average signal power/average noise power = 100, so  $g_{11}(x,y)$  is essentially noise-free. In contrast, SNR = 0 dB in  $g_{12}(x,y)$ , which means that the average signal and noise powers are equal, and in  $g_{13}(x,y)$  the noise power is 10× the signal power..

methods, as demonstrated in later sections of the present chapter, or they are performed using stochastic (probabilistic) methods, which we cover later in Chapter 9. As a “heads up”, we note that the stochastic approach usually outperforms the deterministic approach.

## 6-2 Denoising by Lowpass Filtering

In Section 3-7, we introduced and defined the 2-D **discrete-space Fourier transform (DSFT)**  $F(\Omega_1, \Omega_2)$  of discrete-space image  $f[n, m]$ . Here,  $\Omega_1$  and  $\Omega_2$  are *continuous* spatial frequen-

cies one period of which is over the range

$$-\pi \leq \Omega_1, \Omega_2 \leq \pi.$$

We will refer to this continuous-frequency domain as the **discrete-space spatial frequency (DSSF)** domain.

In the DSSF domain, most of the energy in the spectra of typical images is concentrated in a small central region surrounding the origin ( $\Omega_1 = 0, \Omega_2 = 0$ ). In contrast, additive noise may be distributed over a wide range of frequencies  $\Omega_1$  and  $\Omega_2$ . If we denote  $\mathbf{G}(\Omega_1, \Omega_2)$  as the spectrum of noisy image  $g[n, m]$ , the rationale behind lowpass filtering is to remove high-frequency noise from  $\mathbf{G}(\Omega_1, \Omega_2)$  while preserving (as much as possible) the spectrum of the original image  $\mathbf{F}(\Omega_1, \Omega_2)$ . The disadvantage of lowpass filtering is that the high-DSSF regions of an image may represent features of interest, such as edges.

- ▶ Straightforward lowpass filtering—the subject of the present section—eliminates high-frequency noise, but also eliminates edges and sharp variations in the image. If preserving edges is important, alternative methods should be used, such as the wavelet-denoising approach described in Chapter 7. ◀

## 6-2.1 Brickwall Lowpass Filtering

**Signal-to-noise (SNR)** is a measure of how significant (or insignificant) the presence of noise is and the degree to which it is likely to distort the image. If the **noisy image**  $g[n, m]$  is composed of the true image  $f[n, m]$  plus **additive noise**  $v[n, m]$ ,

$$g[n, m] = f[n, m] + v[n, m], \quad (6.5)$$

then the SNR in dB is defined as

$$\text{SNR} = 10 \log_{10} \left( \frac{\sum \sum f^2[n, m]}{\sum \sum v^2[n, m]} \right), \quad (6.6)$$

where the summations are performed over all image pixels.

A **brickwall lowpass filter** passes all frequency components below a specified cutoff frequency  $\Omega_c$  (along both  $\Omega_1$  and  $\Omega_2$ ) and removes all components at frequencies above  $\Omega_c$ . The DSSF response of the brickwall lowpass filter over 1 period of  $(\Omega_1, \Omega_2)$  is

$$\mathbf{H}_{\text{brick}}(\Omega_1, \Omega_2) = \begin{cases} 1 & \text{for } 0 \leq \Omega_1, \Omega_2 \leq \Omega_c, \\ 0 & \text{for } \Omega_c < \Omega_1, \Omega_2 \leq \pi. \end{cases} \quad (6.7)$$

Application of the lowpass filter to the spectrum  $\mathbf{G}(\Omega_1, \Omega_2)$  of noisy image  $g[n, m]$  generates spectrum

$$\mathbf{G}_{\text{brick}}(\Omega_1, \Omega_2) = \mathbf{H}_{\text{brick}}(\Omega_1, \Omega_2) \mathbf{G}(\Omega_1, \Omega_2). \quad (6.8)$$

The operation given by Eq. (6.8) can be performed in the  $(N \times N)$  2-D DFT domain (Section 3-8) by defining a 2-D DFT **cutoff index**  $K$  such that

$$K = \frac{\Omega_c N}{2\pi}, \quad (6.9)$$

and then setting to zero those elements of the 2-D DFT of  $\mathbf{G}[k_1, k_2]$  that fall in the range

$$K \leq k_1, k_2 \leq N + 2 - K. \quad (6.10)$$

We note that the operation preserves conjugate symmetry in the filtered spectrum  $\mathbf{G}_{\text{brick}}[k_1, k_2]$ . Consequently, the inverse 2-D DFT  $g_{\text{brick}}[n, m]$  of  $\mathbf{G}_{\text{brick}}[k_1, k_2]$  is real-valued.

To illustrate the trade-off between noise reduction and preservation of fast-varying (with position) image features, we consider a noisy letters image, which characteristically has many edges. Starting with a noise-free image  $f[n, m]$ , we synthesized a noisy image  $g[n, m]$  by adding random amounts of noise to the image pixels. The noisy image, consisting of  $(256 \times 256)$  pixels, is shown in Fig. 6-3(a). The values  $f[n, m]$  of the original image ranged between 0 (representing black) and 255 (representing white). Thus,

$$0 \leq f[n, m] \leq 255.$$

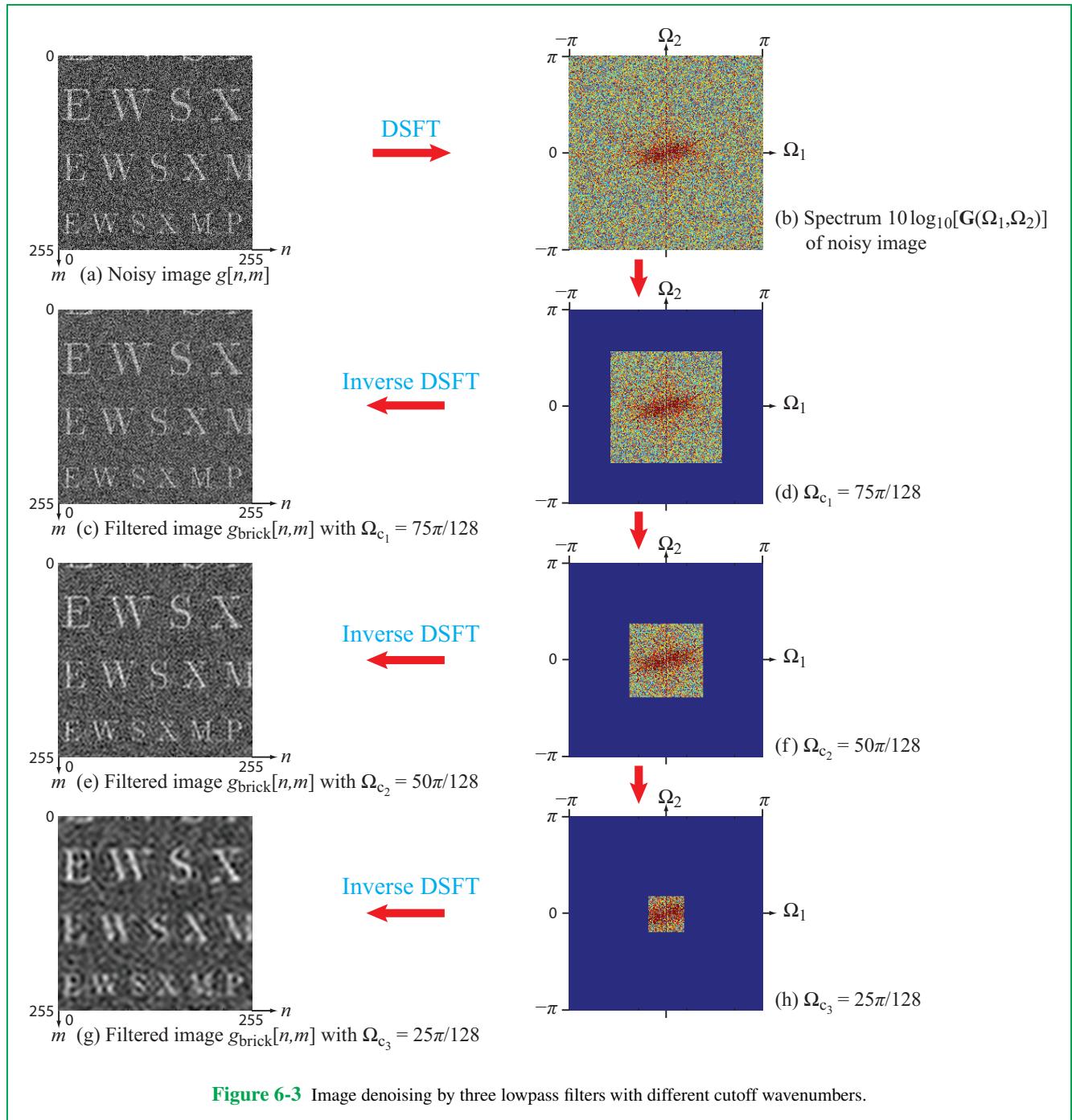
Intentionally, the amount of noise that was added to the original image was much greater in total energy than the energy of the original image itself, thereby producing a very noisy looking image. The associated SNR is  $-12.8$  dB, which means that the total energy of  $f[n, m]$  is only 5.25% of that of the total noise energy. To preserve non-negativity of the noisy image  $g[n, m]$ , the noise image  $v[n, m]$  was assigned the range 0 to 500:

$$0 \leq v[n, m] \leq 500.$$

Consequently, the range of pixel values in noisy image  $g[n, m]$  is

$$0 \leq g[n, m] \leq 755.$$

- **Figure 6-3(b)** displays spectrum  $10 \log_{10}[\mathbf{G}(\Omega_1, \Omega_2)]$  of  $g[n, m]$ , which extends between  $-\pi$  and  $\pi$  along both  $\Omega_1$  and  $\Omega_2$ .
- Multiplication of the spectrum in **Fig. 6-3(b)** by a brickwall



lowpass filter with  $\Omega_c = 75\pi/128$  leads to the spectrum in **Fig. 6-3(d)**. The two spectra are identical within the square defined by  $|\Omega_1|, |\Omega_2| \leq \Omega_{c1}$ . The fractional size of the square is  $(75/128)^2 = 34\%$  of the spectrum of the original image.

Inverse transforming the spectrum in **Fig. 6-3(d)** produces the lowpass-filtered image in **Fig. 6-3(c)**. We observe that the noise is reduced, albeit only slightly, but the letters are hardly distorted.

- Narrowing the filtered spectrum down to a box with  $\Omega_{c2} = 50\pi/128$  leads to the spectrum in **Fig. 6-3(f)**. The associated filtered image  $g_{\text{brick}}[n, m]$  is shown in **Fig. 6-3(e)**. In this case, only  $(50/128)^2 = 15\%$  of the spectrum of the original noisy image is retained. The filtered image contains less noise, but the letters are distorted slightly.
- Repeating the process, but limiting the spectrum to  $\Omega_{c3} = 25\pi/128$ —in which case, only  $(25/128)^2 = 4\%$  of the spectrum is retained—leads to the image in **Fig. 6-3(g)**. The noise is greatly reduced, but the edges of the letters are fuzzy.

► This example illustrates the trade-off inherent in Fourier-based lowpass filtering: noise can be reduced, but at the expense of distorting the high-frequency content of the image. As noted earlier, in Chapter 7 we show how to avoid this trade-off using wavelets instead of Fourier transforms. ◀

## 6-2.2 Tapered Lowpass Filtering

Even though it is not apparent in the filtered images of **Fig. 6-3**, at lower cutoff DSSFs, some of what appears to be noise is actually “ringing” caused by the abrupt “brickwall” filtering of the spectrum of the noisy image. Fortunately, the ringing effect can be reduced significantly by modifying the brickwall filter into a tapered filter. We will examine both the problem and the proposed solution for 2-D images, but before we do so, it will be instructive to consider the case of a periodic 1-D signal.

### A. 1-D brickwall lowpass-filtered signal

Signal  $x(t)$ , shown in **Fig. 6-4(a)** is a **square wave** with period  $T = 2\pi$  and amplitude  $A = 1$ . Its Fourier series expansion is given by

$$x(t) = \sum_{k=1}^{\infty} \frac{4}{k\pi} \sin(kt). \quad (6.11)$$

The **fundamental frequency** of  $x(t)$  is  $f_0 = 1/T = 1/2\pi$ . We can apply the equivalent of a brickwall lowpass filter with cutoff frequency  $k_c f_0$  by truncating the Fourier series at  $k = k_c$ . For example, if we select  $k_c = 21$ , we obtain a brickwall lowpass-filtered version of  $x(t)$  given by

$$y_{\text{brick}}(t) = \sum_{\substack{k=1 \\ k=\text{odd}}}^{21} \frac{4}{k\pi} \sin(kt). \quad (6.12)$$

The truncated summation contains 11 nonzero terms. The plot of  $y_{\text{brick}}(t)$  displayed in **Fig. 6-4(b)** resembles the original square wave, except that it also exhibits small oscillations; i.e., the ringing effect we referred to earlier.

### B. 1-D tapered lowpass-filtered signal

The ringing in the lowpass-filtered signal, which is associated with the sharp cutoff characteristic of the brickwall filter, can be reduced significantly by multiplying the terms in Eq. (6.12) by a decreasing sequence of weights, thereby tapering those terms gradually to zero. Several tapering formats are available, one of which is the **Hamming window** defined by Eq. (2.83). Adapting the expression for the Hamming window to the square wave leads to

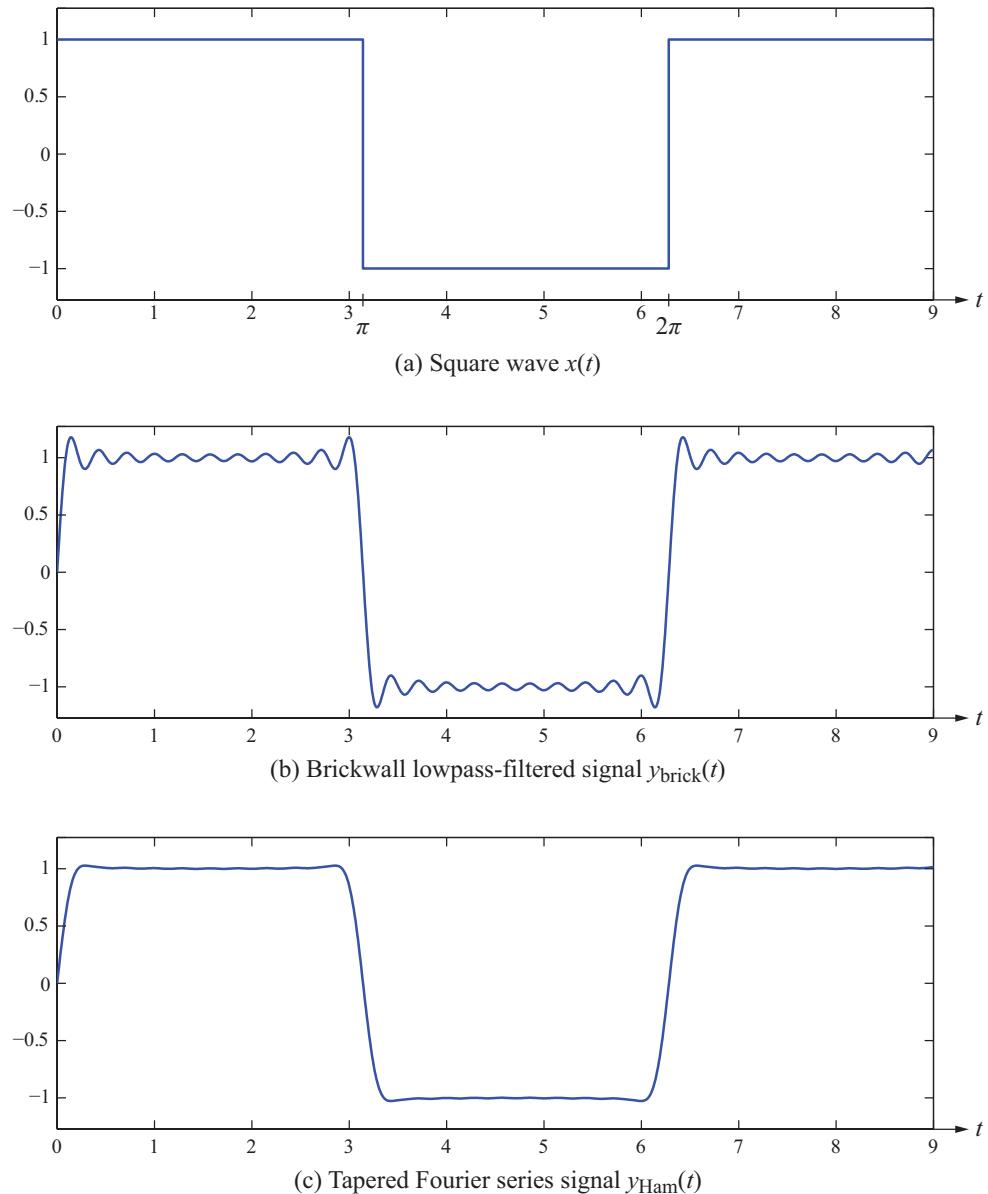
$$y_{\text{Ham}}(t) = \sum_{\substack{k=1 \\ k=\text{odd}}}^{21} \frac{4}{k\pi} \sin(kt) \left\{ 0.54 + 0.46 \cos \left[ \frac{\pi(k-1)}{20} \right] \right\}. \quad (6.13)$$

A plot of the tapered signal  $y_{\text{Ham}}(t)$  is shown in **Fig. 6-4(c)**. Even though the tapered signal includes the same number of frequency harmonics as before, the oscillations have disappeared and the transitions at  $t = \text{integer values of } T/2 = \pi$  are relatively smooth.

### C. 2-D brickwall lowpass-filtered image

The ringing observed in 1-D signals also manifests itself in 2-D images whenever the image spectrum is lowpass-filtered by a sharp filter. The DSSF response  $\mathbf{H}_{\text{brick}}(\Omega_1, \Omega_2)$  of a brickwall lowpass filter with cutoff frequency  $\Omega_c$  along both  $\Omega_1$  and  $\Omega_2$  is given by Eq. (6.7). The corresponding inverse DSFT is

$$\begin{aligned} h_{\text{brick}}[n, m] &= h_{\text{brick}}[n] h_{\text{brick}}[m] \\ &= \left( \frac{\Omega_c}{\pi} \right)^2 \operatorname{sinc} \left( \frac{\Omega_c n}{\pi} \right) \operatorname{sinc} \left( \frac{\Omega_c m}{\pi} \right). \end{aligned} \quad (6.14)$$



**Figure 6-4** (a) Square wave  $x(t)$ , (b) brickwall lowpass-filtered version, and (c) Hamming-windowed version.

Per the definition of the sinc function given by Eq. (2.35), namely  $\text{sinc}(x) = [\sin(\pi x)]/(\pi x)$ ,

$$\text{sinc}\left(\frac{\Omega_c n}{\pi}\right) = \frac{\sin\left(\frac{\pi\Omega_c n}{\pi}\right)}{\left(\frac{\pi\Omega_c n}{\pi}\right)} = \frac{\sin(\Omega_c n)}{\Omega_c n}, \quad (6.15)$$

and a similar definition applies to  $\text{sinc}(\Omega_c m/\pi)$ .

For an image  $g[n, m]$  with a corresponding DSFT  $\mathbf{G}(\Omega_1, \Omega_2)$ , the filtered image spectrum is

$$\mathbf{G}_{\text{brick}}(\Omega_1, \Omega_2) = \mathbf{H}_{\text{brick}}(\Omega_1, \Omega_2) \mathbf{G}(\Omega_1, \Omega_2), \quad (6.16a)$$

and the corresponding spatial-domain relationship is

$$g_{\text{brick}}[n, m] = h_{\text{brick}}[n, m] * * g[n, m]. \quad (6.16b)$$

The impact of the brickwall lowpass-filtering process on a 2-D image was illustrated earlier through Fig. 6-3.

## D. 2-D tapered lowpass-filtered image

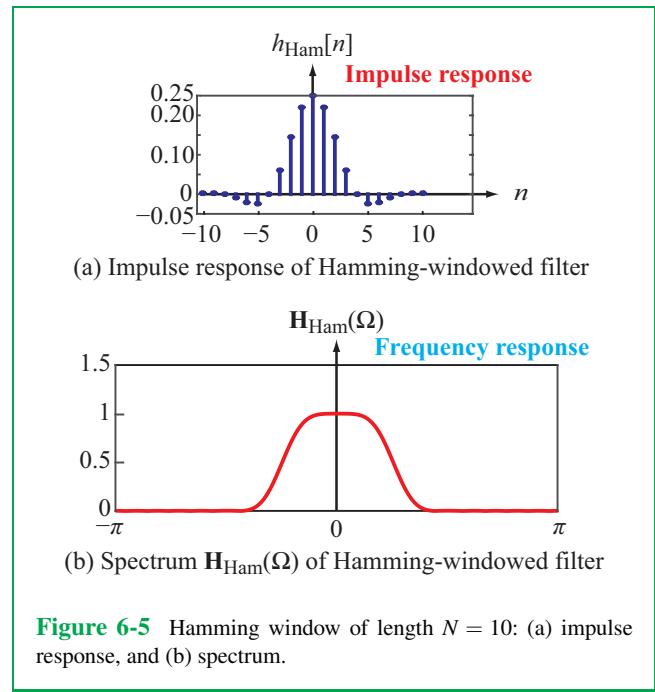
**Figure 6-5** displays the spatial and frequency domain responses of a Hamming window with  $N = 10$ , adapted from Fig. 2-13. For a 2-D image  $g[n, m]$ , lowpass filtering its spectrum with a **Hamming window** of length  $N$  is equivalent to performing the convolution

$$g_{\text{Ham}}[n, m] = h_{\text{Ham}}[n, m] * * g[n, m] \quad (6.17a)$$

with

$$h_{\text{Ham}}[n, m] = h_{\text{Ham}}[n] h_{\text{Ham}}[m] = \begin{cases} \left(\frac{\Omega_c}{\pi}\right)^2 \text{sinc}\left(\frac{\Omega_c n}{\pi}\right) [0.54 + 0.46 \cos\left(\frac{\pi n}{N}\right)] \\ \times \text{sinc}\left(\frac{\Omega_c m}{\pi}\right) [0.54 + 0.46 \cos\left(\frac{\pi m}{N}\right)] & \text{for } |n|, |m| \leq N, \\ 0 & \text{for } |n|, |m| > N. \end{cases} \quad (6.17b)$$

To illustrate the presence of “ringing” when a sharp-edged filter like a brickwall is used, and its absence when a Hamming windowed filter is used instead, we refer the reader to Fig. 6-6. In part (a), we show a noiseless letters image, and its corresponding spectrum is displayed in part (b). Application of a brickwall lowpass filter (with the impulse response given by Eq. (6.14)) leads to the image in Fig. 6-6(c). The “ringing” in the image is visible in the form of whorls that resemble a giant thumbprint.



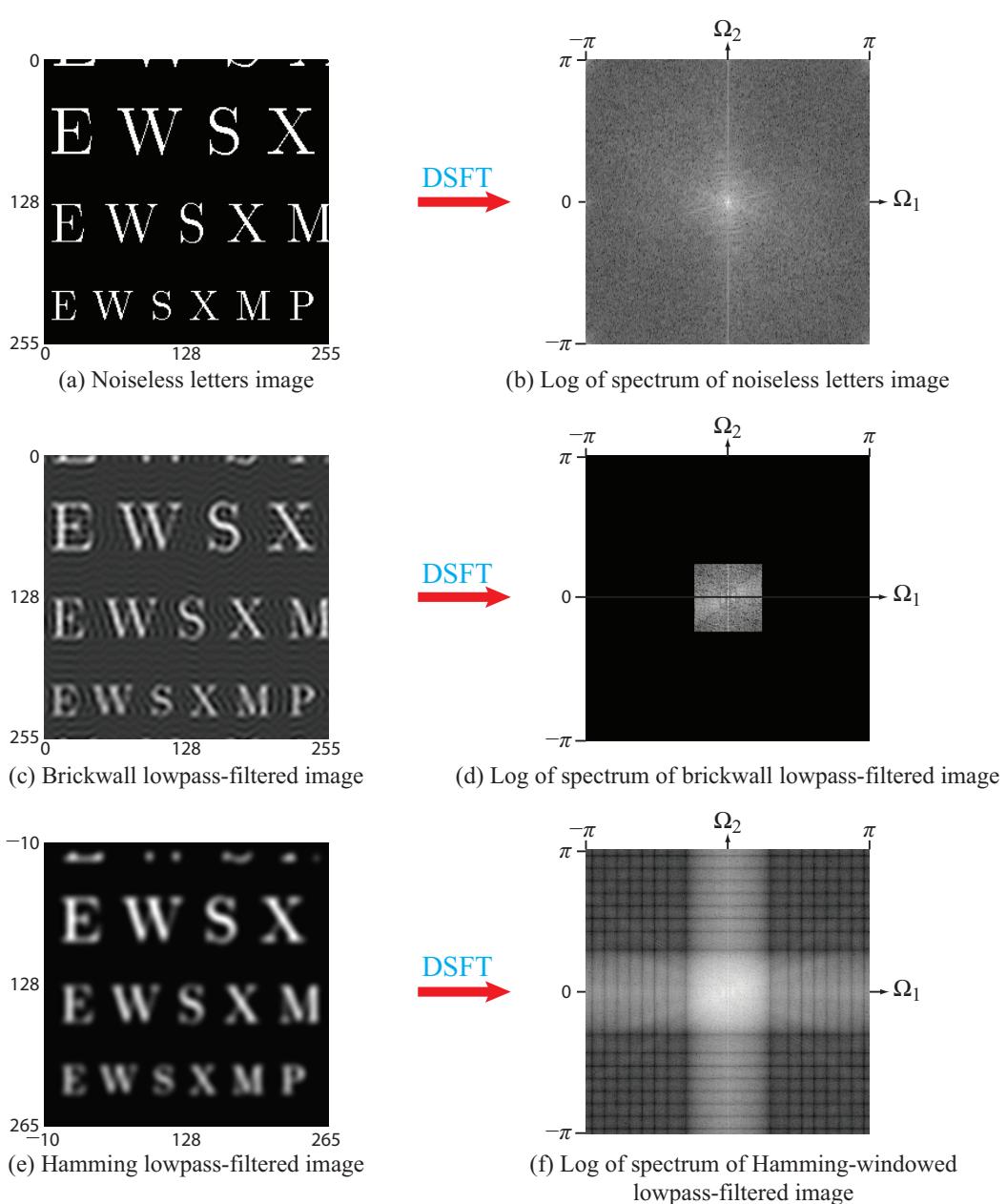
**Figure 6-5** Hamming window of length  $N = 10$ : (a) impulse response, and (b) spectrum.

In contrast, the image in Fig. 6-6(e)—which was generated by applying a Hamming windowed filter to the original image—exhibits no “ringing.” Note that the spectrum of the Hamming-windowed spectrum in Fig. 6-6(f) tapers gradually from the center outward. It is this tapering profile that eliminates the ringing effect.

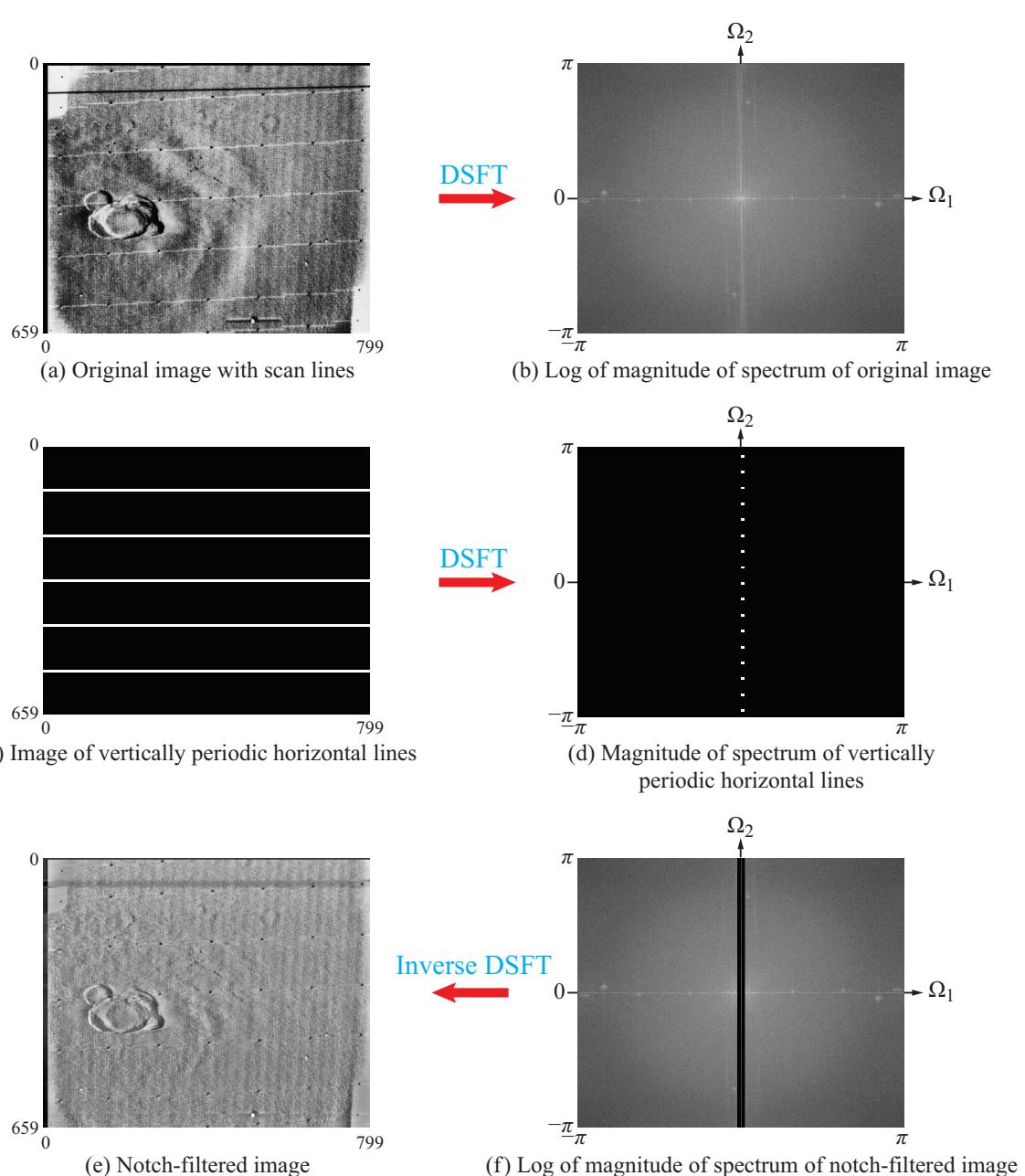
**Concept Question 6-1:** For lowpass filtering, why would we use a Hamming-windowed filter instead of a brick-wall filter?

## 6-3 Notch Filtering

Occasionally, an image may contain a 2-D **sinusoidal interference** contributed by an electromagnetic source, such as the ac power cable in a camera. In 1-D discrete-time signals, sinusoidal interference can be eliminated by subjecting the signal’s spectrum to a **notch filter**, which amounts to setting the spectrum at that specific frequency to zero. A similar process can be applied to a 2-D image. To illustrate, let us consider the example portrayed in Fig. 6-7. In part (a) of the figure, we have a  $660 \times 800$  image of the planet Mars recorded by a Mariner space



**Figure 6-6** Letters image and its spectrum in (a) and (b); brickwall lowpass-filtered version in (c) and (d), and brickwall lowpass-filtered version with a Hamming window in (e) and (f). The logarithmic scale enhances small values of the spectrum.



**Figure 6-7** Process for notch-filtering horizontal scan lines. Fig. 6-7(a) courtesy of NASA.

probe. In addition to the image of the planet, the image contains near-horizontal scan lines that we wish to eliminate by applying notch filtering.

Let us designate the image containing only the near-horizontal lines as  $f'(x, y)$ , which we display in **Fig. 6-7(c)**, along with its spectrum in **Fig. 6-7(d)**. Along the vertical direction, the image contains  $(2M + 1)$  lines, with one line passing through the center and  $M$  lines each above and below the center line. The separation between adjacent lines is  $\Delta$ . For the present, we will treat the near-horizontal lines as if they were perfectly horizontal (correction for skewness will be applied later). Accordingly,  $f'(x, y)$  is given by

$$f'(x, y) = \sum_{m=-M}^M \delta(y - m\Delta), \quad (6.18)$$

and (using entry #3 in **Table 2-4** and property #2 in **Table 2-5**) the associated spectrum of  $f'(x, y)$  is

$$\mathbf{F}'(\mu, v) = \sum_{m=-M}^M e^{-j2\pi v m \Delta} \delta(\mu). \quad (6.19)$$

The sum in Eq. (6.19) is similar in form to the sum in Eq. (2.87),

$$\sum_{m=-M}^M e^{-j\Omega m} = \frac{\sin((2M+1)\Omega/2)}{\sin(\Omega/2)}, \quad (6.20)$$

thereby allowing us to rewrite Eq. (6.19) as

$$\mathbf{F}'(\mu, v) = \frac{\sin((2M+1)\pi v \Delta)}{\sin(\pi v \Delta)} \delta(\mu). \quad (6.21)$$

The spectrum, which is a discrete sinc function along the  $v$  (vertical spatial frequency) direction and an impulse function along the  $\mu$  (horizontal spatial frequency) direction, is displayed in **Fig. 6-7(d)**. Only the peaks of the discrete sinc are visible in the image.

Recall from Section 3-4.2 that rotating an image causes its spectrum to rotate by the same angle. Hence, the interfering vertically periodic near-horizontal lines in **Fig. 6-7(a)** should appear in its spectrum (**Fig. 6-7(b)**) as a near-vertical line rotated slightly counterclockwise.

The spectrum shown in **Fig. 6-7(f)** is the spectrum of the original image after setting the spectrum associated with the interfering lines to zero. Application of the inverse transform leads to the filtered image shown in **Fig. 6-7(e)**. The interfering horizontal lines have been eliminated, with minor degradation to the rest of the image.

**Concept Question 6-2:** From where does a notch filter get its name?

**Concept Question 6-3:** What is notch filtering used for?

## 6-4 Image Deconvolution

When our eyes view a scene, they form an approximate image of the scene, because the optical imaging process performed by the eyes *distorts* the true scene, with the degree of distortion being dependent on the imaging properties of the eyes' lenses. The same is true when imaging with a camera, a medical imaging system, and an optical or radio telescope. Distortion can also be caused by the intervening medium between the imaged scene and the imaging sensor. Examples include the atmosphere when a telescope is used to image a distant object, or body tissue when a medical ultrasound sensor is used to image body organs. In all cases, the imaging process involves the convolution of a *true image scene*  $f[n, m]$  with a *point spread function*  $h[n, m]$  representing the imaging sensor (and possibly the intervening medium). The *recorded* (sensed) image  $g[n, m]$  is, therefore, given by

$$g[n, m] = h[n, m] * * f[n, m]. \quad (6.22)$$

The goal of *image deconvolution* is to *deconvolve* the recorded image so as to extract the true image  $f[n, m]$ , or a close approximation of it. Doing so requires knowledge of the PSF  $h[n, m]$ . In terms of size:

- $f[n, m]$  is the *unknown true image*, with size  $(M \times M)$ .
- $h[n, m]$  is the *known PSF*, with size  $(L \times L)$ .
- $g[n, m]$  is the *known recorded image*, with size  $(L + M - 1) \times (L + M - 1)$ .

As noted earlier in Chapter 1, the PSF of the imaging sensor can be established by imaging a small object representing a 2-D impulse.

Since convolution in the discrete-time domain translates into multiplication in the frequency domain, the DSFT-equivalent of Eq. (6.22) is given by

$$\mathbf{G}(\Omega_1, \Omega_2) = \mathbf{H}(\Omega_1, \Omega_2) \mathbf{F}(\Omega_1, \Omega_2). \quad (6.23)$$

Before we perform the frequency transformation of  $g[n, m]$ , we should round up  $(L + M - 1)$  to  $N$ , where  $N$  is the smallest power of 2 greater than  $(L + M - 1)$ . The rounding-up step allows us

to use the fast radix-2 2-D FFT to compute 2-D DFTs of order  $(N \times N)$ . Alternatively, the Cooley-Tukey FFT can be used, in which case  $N$  should be an integer with a large number of small factors.

Sampling the DSFT at  $\Omega_1 = 2\pi k_1/N$  and  $\Omega_2 = 2\pi k_2/N$  for  $k_1 = 0, 1, \dots, N-1$  and  $k_2 = 0, 1, \dots, N-1$  provides the DFT complex coefficients  $\mathbf{G}[k_1, k_2]$ .

A similar procedure can be applied to  $h[n, m]$  to obtain coefficients  $\mathbf{H}[k_1, k_2]$ , after zero-padding  $h[n, m]$  so that it also is of size  $(N \times N)$ . The DFT equivalent of Eq. (6.23) is then given by

$$\mathbf{G}[k_1, k_2] = \mathbf{H}[k_1, k_2] \mathbf{F}[k_1, k_2]. \quad (6.24)$$

The objective of deconvolution is to compute the DFT coefficients  $\mathbf{F}[k_1, k_2]$ , given the DFT coefficients  $\mathbf{G}[k_1, k_2]$  and  $\mathbf{H}[k_1, k_2]$ .

#### 6-4.1 Nonzero $\mathbf{H}[k_1, k_2]$ Coefficients

In the ideal case where none of the DFT coefficients  $\mathbf{H}[k_1, k_2]$  are zero, the DFT coefficients  $\mathbf{F}[k_1, k_2]$  of the unknown image can be obtained through simple division,

$$\mathbf{F}[k_1, k_2] = \frac{\mathbf{G}[k_1, k_2]}{\mathbf{H}[k_1, k_2]}. \quad (6.25)$$

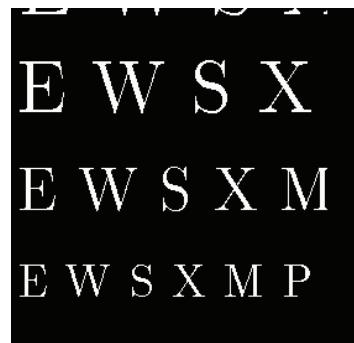
Exercising the process for all possible values of  $k_1$  and  $k_2$  leads to an  $(N \times N)$  2-D DFT for  $\mathbf{F}[k_1, k_2]$ , whereupon application of an inverse 2-D DFT process yields a zero-padded version of  $f[n, m]$ . Upon discarding the zeros, we obtain the true image  $f[n, m]$ . The deconvolution procedure is straightforward, but it hinges on a critical assumption, namely that none of the DFT coefficients of the imaging system's transfer function is zero. Otherwise, division by zero in Eq. (6.25) would lead to undeterminable values for  $\mathbf{F}[k_1, k_2]$ .

#### 6-4.2 Image Deconvolution Example

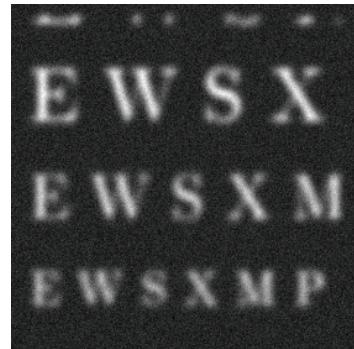
To demonstrate the performance of the deconvolution process, we used a noise-free version of the letters image, shown in Fig. 6-8(a), which we denote  $f[n, m]$ . Then, we convolved it with a truncated 2-D Gaussian PSF (a common test PSF) given by

$$h[n, m] = e^{-(m^2+n^2)/20}, \quad -10 \leq n, m \leq 10. \quad (6.26)$$

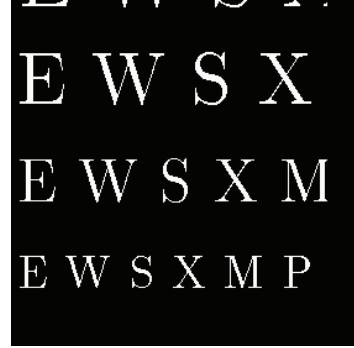
The PSF represents the imaging system. The convolution process generated the blurred image shown in Fig. 6-8(b), which



(a) Letters image  $f[n, m]$



(b) Blurred image  $g[n, m]$



(c) Deconvolved image  $f[n, m]$

**Figure 6-8** The blurred image in (b) was generated by convolving  $f[n, m]$  with a Gaussian PSF, and the image in (c) was recovered through deconvolution of  $g[n, m]$ .

we label  $g[n, m]$ . The image sizes are:

- Original letters image  $f[n, m]$ :  $256 \times 256$
- Gaussian PSF  $h[n, m]$ :  $21 \times 21$
- Blurred image  $g[n, m]$ :  $276 \times 276$ , where  $276 = 256 + 21 - 1$ .

After zero-padding all three images to  $280 \times 280$ , the 2-D FFT was applied to all three images. Then, Eq. (6.25) was applied to find coefficients  $\mathbf{F}[k_1, k_2]$ , which ultimately led to the deconvolved image  $f[n, m]$  displayed in [Fig. 6-8\(c\)](#). The deconvolved image matches the original image shown in [Fig. 6-8\(a\)](#). The process was successful because none of the  $\mathbf{H}[k_1, k_2]$  coefficients had zero values and  $\mathbf{G}[k_1, k_2]$  was noise-free. To avoid division by  $\mathbf{H}[k_1, k_2]$  when computing  $\mathbf{F}[k_1, k_2]$ , we use image regularization and Wiener filtering, as discussed in the next subsections.

### 6-4.3 Tikhonov Image Regularization

All electronic imaging systems generate some noise of their own. The same is true for the eye-brain system. Hence, Eq. (6.24) should be modified to

$$\mathbf{G}[k_1, k_2] = \mathbf{H}[k_1, k_2] \mathbf{F}[k_1, k_2] + \mathbf{V}[k_1, k_2], \quad (6.27)$$

where  $\mathbf{V}[k_1, k_2]$  represents the spectrum of the *additive noise* contributed by the imaging system. The known quantities are the measured image  $\mathbf{G}[k_1, k_2]$  and the PSF of the system,  $\mathbf{H}[k_1, k_2]$ , and the sought-out quantity is the true image  $\mathbf{F}[k_1, k_2]$ . Dividing both sides of Eq. (6.27) by  $\mathbf{H}[k_1, k_2]$  and solving for  $\mathbf{F}[k_1, k_2]$  gives

$$\mathbf{F}[k_1, k_2] = \frac{\mathbf{G}[k_1, k_2]}{\mathbf{H}[k_1, k_2]} - \frac{\mathbf{V}[k_1, k_2]}{\mathbf{H}[k_1, k_2]}. \quad (6.28)$$

In many practical applications,  $\mathbf{H}[k_1, k_2]$  may assume very small values for large values of  $[k_1, k_2]$ . Consequently, the second term in Eq. (6.28) may end up *amplifying* the noise component and may *drown out* the first term. To avoid the *noise-amplification problem*, the deconvolution can be converted into a *regularized* estimation process. Regularization involves the use of a *cost function* that trades off *estimation accuracy* (of  $f[n, m]$ ) against *measurement precision*. The process generates an estimate  $\hat{f}[n, m]$  of the true image  $f[n, m]$ . Accuracy refers to a bias associated with all pixel values of the reconstructed image  $\hat{f}[n, m]$  relative to  $f[n, m]$ . Precision refers to the  $\pm$  uncertainty associated with each individual pixel value due to noise.

A commonly used regularization model is the *Tikhonov*

*regularization*, which seeks to minimize the cost function

$$e = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} [(g[n, m] - h[n, m] * \hat{f}[n, m])^2 + (\lambda \hat{f}[n, m])^2], \quad (6.29)$$

where zero-padding to size  $N \times N$  has been implemented so that all quantities in Eq. (6.29), except for  $\lambda$ , are of the same order. The parameter  $\lambda$  is non-negative and it is called a *regularization parameter*. The second term on the right-hand side of Eq. (6.29) represents the bias error associated with  $\hat{f}[n, m]$  and the first term represents the variance. Setting  $\lambda = 0$  reduces Eq. (6.29) to the unregularized state we dealt with earlier in Section 6-4.1, wherein the measurement process was assumed to be noise-free. For realistic imaging processes,  $\lambda$  should be greater than zero, but there is no simple method for specifying its value, so usually its value is selected heuristically (by trial and error).

The estimation process may be performed iteratively in the discrete-time domain by selecting an initial estimate  $\hat{f}[n, m]$  and then recursively iterating the estimate until the error  $e$  approaches a minimum level. Alternatively, the process can be performed in the frequency domain using a Wiener filter, as discussed next.

### 6-4.4 Wiener Filter

Using Rayleigh's theorem (entry #9 in [Table 3-3](#)), the frequency domain DFT equivalent of the Tikhonov cost function given by Eq. (6.29) is

$$E = \frac{1}{N^2} \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} [|\mathbf{G}[k_1, k_2] - \mathbf{H}[k_1, k_2] \hat{\mathbf{F}}[k_1, k_2]|^2 + \lambda^2 |\mathbf{X}[k_1, k_2]|^2]. \quad (6.30)$$

The error can be minimized separately for each  $(k_1, k_2)$  combination. The process can be shown (see Problem 6-11) to lead to the solution

$$\hat{\mathbf{F}}[k_1, k_2] = \mathbf{G}[k_1, k_2] \frac{\mathbf{H}^*[k_1, k_2]}{|\mathbf{H}[k_1, k_2]|^2 + \lambda^2}, \quad (6.31)$$

where  $\mathbf{H}^*[k_1, k_2]$  is the complex conjugate of  $\mathbf{H}[k_1, k_2]$ . The quantity multiplying  $\mathbf{G}[k_1, k_2]$  is called a *Wiener filter*  $\mathbf{W}[k_1, k_2]$ . That is,

$$\hat{\mathbf{F}}[k_1, k_2] = \mathbf{G}[k_1, k_2] \mathbf{W}[k_1, k_2], \quad (6.32a)$$

with

$$\mathbf{W}[k_1, k_2] = \frac{\mathbf{H}^*[k_1, k_2]}{|\mathbf{H}[k_1, k_2]|^2 + \lambda^2}. \quad (6.32b)$$

The operation of the Wiener filter is summarized as follows:

(a) For values of  $(k_1, k_2)$  such that  $|\mathbf{H}[k_1, k_2]| \gg \lambda$ , the Wiener filter implementation leads to

$$\hat{\mathbf{F}}[k_1, k_2] \approx \mathbf{G}[k_1, k_2] \frac{\mathbf{H}^*[k_1, k_2]}{|\mathbf{H}[k_1, k_2]|^2} = \frac{\mathbf{G}[k_1, k_2]}{\mathbf{H}[k_1, k_2]}, \quad (6.33a)$$

which is the same as Eq. (6.25).

(b) For values of  $(k_1, k_2)$  such that  $|\mathbf{H}[k_1, k_2]| \ll \lambda$ , the Wiener filter implementation leads to

$$\hat{\mathbf{F}}[k_1, k_2] \approx \mathbf{G}[k_1, k_2] \frac{\mathbf{H}^*[k_1, k_2]}{\lambda^2}. \quad (6.33b)$$

In this case, the Wiener filter avoids the noise amplification problem that would have occurred with the use of the unregularized deconvolution given by Eq. (6.25).

## 6-4.5 Wiener Filter Deconvolution Example

To demonstrate the capabilities of the Wiener filter, we compare image deconvolution performed with and without regularization. The demonstration process involves images at various stages, namely:

- $f[n, m]$ : true letters image (**Fig. 6-9(a)**).
- $g[n, m] = h[n, m] * f[n, m] + v[n, m]$ : the imaging process not only distorts the image (through the PSF), but also adds random noise  $v[n, m]$ . The result, displayed in **Fig. 6-9(b)**, is an image with signal-to-noise ratio of 10.8 dB, which means that the random noise energy is only about 8% of that of the signal.
- $\hat{f}_1[n, m]$ : estimate of  $f[n, m]$  obtained without regularization (i.e., using Eq. (6.25)). Image  $\hat{f}_1[n, m]$ , displayed in **Fig. 6-9(c)**, does not show any of the letters present in the original image, despite the fact that the noise level is small relative to the signal.
- $\hat{f}_2[n, m]$ : estimate of  $f[n, m]$  obtained using the Wiener filter of Eq. (6.31) with  $\lambda^2 = 5$ . The deconvolved image (**Fig. 6-9(d)**) displays all of the letters contained in the original image, but some high wavenumber noise also is present.

**Concept Question 6-4:** What does the Wiener filter given by Eq. (6.31) reduce to when  $\lambda = 0$ ?

**Concept Question 6-5:** Why is Tikhonov regularization needed in deconvolution?

**Exercise 6-1:** Apply Tikhonov regularization with  $\lambda = 0.01$  to the 1-D deconvolution problem

$$\{x[0], x[1]\} * \{h[0], h[1], h[2]\} = \{2, -5, 4, -1\},$$

where  $h[0] = h[2] = 1$  and  $h[1] = -2$ .

**Answer:**  $\mathbf{H}(0) = 1 - 2 + 1 = 0$  so  $\mathbf{X}(\Omega) = \frac{\mathbf{Y}(\Omega)}{\mathbf{H}(\Omega)}$  will not work at  $\Omega = 0$ . But

$$\mathbf{X}(\Omega) = \frac{\mathbf{H}^*(\Omega)}{|\mathbf{H}(\Omega)|^2 + \lambda^2} \mathbf{Y}(\Omega)$$

does work. Using 4-point DFTs (computable by hand) gives  $x[n] = \{1.75, -1.25, -0.25, -0.25\}$ , which is close to the actual  $x[n] = \{2, -1\}$ . MATLAB code:

```
h=[1 -2 1]; x=[2 -1]; y=conv(x, h);
H=fft(h, 4); Y=fft(y);
Z=conj(H).*Y./ (abs(H).*abs(H)+0.0001);
z=real(ifft2(Z)) provides the estimated x[n].
```

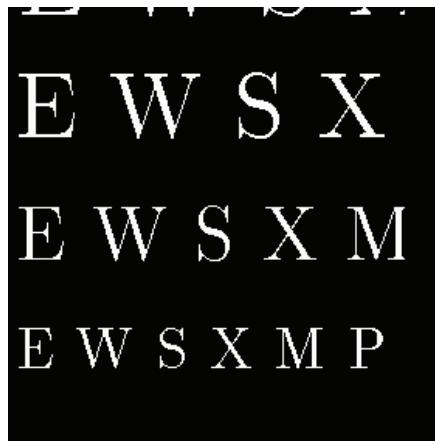
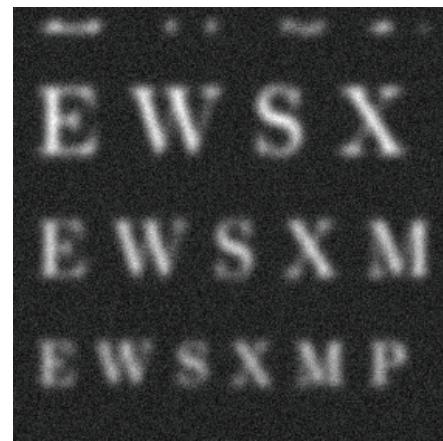
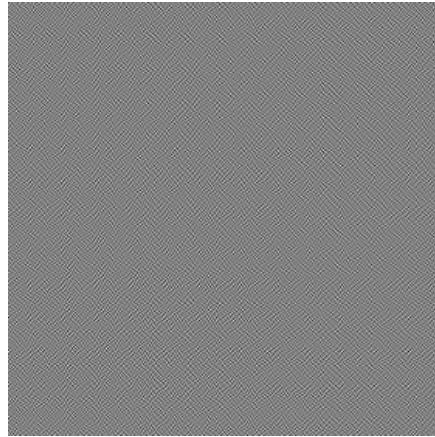
## 6-5 Median Filtering

**Median filtering** is used to remove **salt-and-pepper** noise, often due to bit errors or **shot noise** associated with electronic devices. The concept of median filtering is very straightforward:

► A median filter of order  $L$  replaces each pixel with the median value of the  $L^2$  pixels in the  $L \times L$  block centered on that pixel. ◀

For example, a median filter of order  $L = 3$  replaces each pixel  $[n, m]$  with the median value of the  $3 \times 3 = 9$  pixels centered at  $[n, m]$ . **Figure 6-10(a)** shows an image corrupted with salt-and-pepper noise, and part (b) of the same figure shows the image after the application of a median filter of order  $L = 5$ .

**Concept Question 6-6:** When is median filtering useful?

(a)  $f[n,m]$ (b)  $g[n,m] = h[n,m] \ast\ast f[n,m] + v[n,m]$ (c)  $\hat{f}_1[n,m]$  without regularization(d)  $\hat{f}_2[n,m]$  with Wiener filter

**Figure 6-9** (a) Original noise-free undistorted letters image  $f[n,m]$ , (b) blurred image due to imaging system PSF and addition of random noise  $v[n,m]$ , (c) deconvolution using Eq. (6.25), and (d) deconvolution using Eq. (6.31) with  $\lambda^2 = 5$ .

## 6-6 Motion-Blur Deconvolution

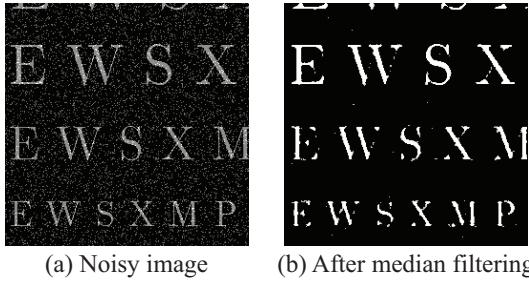
### 6-6.1 Continuous Space

If, during the recording time for generating a still image of an object or scene, the imaged object or scene is in motion relative to the imaging system, the recorded image will exhibit a streak-

ing pattern known as ***motion blur***. An example is the simulated photograph of the highway sign shown in [Fig. 6-11\(a\)](#), taken from a moving car. Often, the direction and duration of the blur can be discerned from the blurred image.

To describe motion blur mathematically, we start by making the following assumptions:

(1) The blurred image has been appropriately ***rotated*** so that



**Figure 6-10** Median filtering example: (a) letters image corrupted by salt-and-pepper noise, and (b) image after application of median filtering using a  $5 \times 5$  window.

the direction of motion is aligned along the  $x$  axis.

(2) **Image recording** is  $T$  (seconds) in duration, with instantaneous start and stop actions, thereby allowing us to represent the process in terms of a rectangle function.

(3) The motion of the imager, relative to the scene, is linear (along the  $x$  axis) and at a **constant speed**  $s$  (meters/s).

In terms of the unblurred image  $f(x,y)$ , corresponding to no-motion conditions, the blurred image  $g(x,y)$  consists of a superposition of copies of  $f(x,y)$  shifted along the  $x$  axis by distance  $x' = st$ :

$$g(x,y) = \int_0^T f(x-x', y) dt. \quad (6.34)$$

Upon replacing  $dt$  with  $dx'/s$  and changing the upper integration limit to  $D = sT$ , we have

$$g(x,y) = \frac{1}{s} \int_0^D f(x-x', y) dx'. \quad (6.35)$$

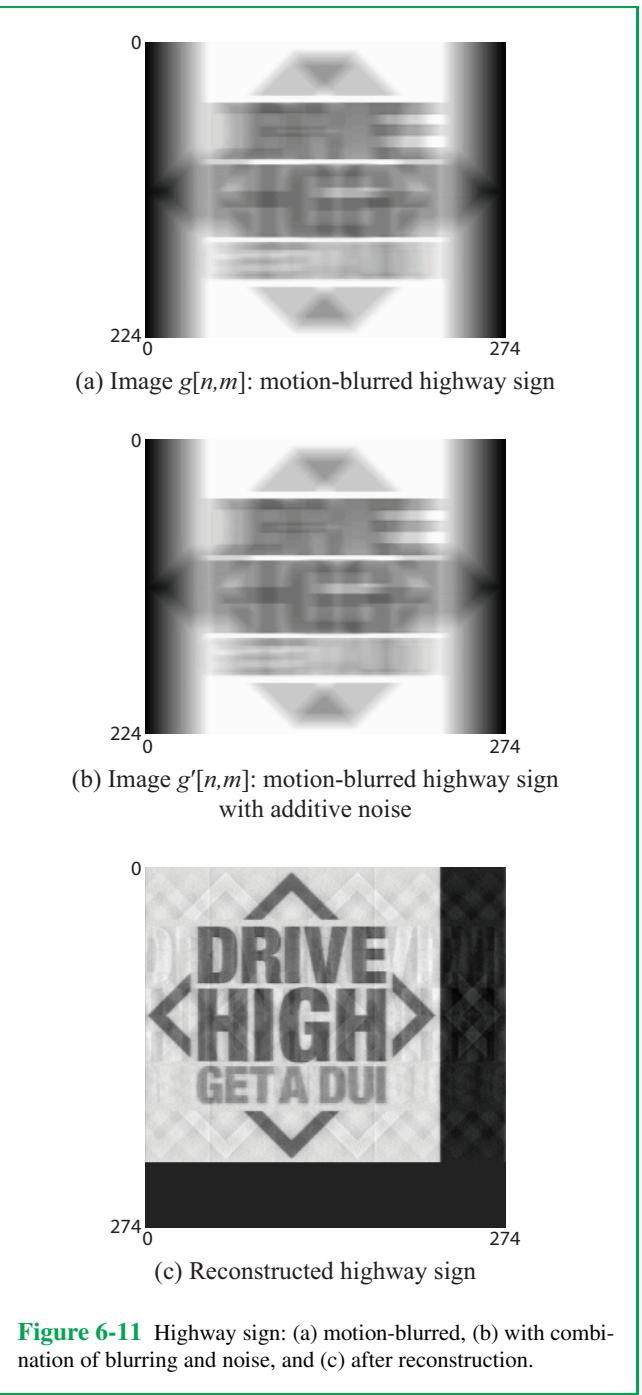
The spatial shift occurs along  $x$  only, and its length is  $D$ , which is equivalent to defining  $g(x,y)$  as the convolution of  $f(x,y)$  with a point spread function  $h(x,y)$  composed of a rectangle function of spatial duration  $D = sT$  and centered at  $D/2$ :

$$g(x,y) = f(x,y) * * h(x,y), \quad (6.36)$$

with

$$h(x,y) = \frac{1}{s} \text{rect}\left(\frac{x-D/2}{D}\right) \delta(y). \quad (6.37)$$

The spatial frequency response  $\mathbf{H}(\mu, v)$  is the 2-D Fourier transform of  $h(x,y)$ , which separates into two 1-D Fourier



**Figure 6-11** Highway sign: (a) motion-blurred, (b) with combination of blurring and noise, and (c) after reconstruction.

transforms. Using entries #1a and #4 in **Table 2-5** gives

$$\begin{aligned}\mathbf{H}(\mu, v) &= \mathcal{F}_{x \rightarrow \mu} \left\{ \text{rect} \left( \frac{x - D/2}{D} \right) \right\} \mathcal{F}_{y \rightarrow v} \left\{ \frac{\delta(y)}{s} \right\} \\ &= \frac{D}{s} \text{sinc}(\mu D) e^{-j\pi\mu D}.\end{aligned}\quad (6.38)$$

The convolution in the spatial domain given by Eq. (6.36) becomes a product in the spatial frequency domain:

$$\mathbf{G}(\mu, v) = \mathbf{F}(\mu, v) \mathbf{H}(\mu, v). \quad (6.39)$$

To recover the unblurred image  $f(x, y)$ , we need to:

(a) divide Eq. (6.39) by  $\mathbf{H}(\mu, v)$  to obtain

$$\mathbf{F}(\mu, v) = \left( \frac{1}{\mathbf{H}(\mu, v)} \right) \mathbf{G}(\mu, v), \quad (6.40)$$

where  $\mathbf{G}(\mu, v)$  is the spatial frequency spectrum of the blurred image, and then

(b) perform an inverse transform on  $\mathbf{F}(\mu, v)$ .

However, in view of the definition of the sinc function,

$$\text{sinc}(\mu D) = \frac{\sin(\pi\mu D)}{\pi\mu D}, \quad (6.41)$$

it follows that the spatial frequency response  $\mathbf{H}(\mu, v) = 0$  for integer values of  $\mu D$ . Consequently, the inverse filter  $1/\mathbf{H}(\mu, v)$  is undefined for nonzero integer values of  $\mu D$ , thereby requiring the use of regularization (Section 6-4.3).

## 6-6.2 Motion Blur after Sampling

To convert image representation from the continuous-space case of the previous subsection to the sampled-space case, we start by sampling unblurred (still) image  $f(x, y)$  and the motion-blurred image  $g(x, y)$  at  $x = n\Delta$  and  $y = m\Delta$ :

$$f[n, m] = f(x = n\Delta, y = m\Delta), \quad (6.42a)$$

$$g[n, m] = g(x = n\Delta, y = m\Delta). \quad (6.42b)$$

We also discretize time  $t$  as

$$t = i\Delta_t, \quad (6.43)$$

where  $\Delta_t$  is the time interval associated with the movement of the imager (relative to the scene) by a distance  $\Delta$ :

$$\Delta_t = \frac{\Delta}{s}, \quad (6.44)$$

where  $s$  is the relative speed of the imager (along the  $x$  axis). The total number of time shifts  $N$  that occur during the total recording time  $T$  is

$$N = \frac{T}{\Delta_t}. \quad (6.45)$$

In terms of these new quantities, the discrete-case analogues to Eqs. (6.36) and (6.37) are

$$g[n, m] = \sum_{i=0}^N f[n - i, m] \Delta_t = f[n, m] * * h[n, m], \quad (6.46)$$

where the discrete-space PSF  $h[n, m]$  is

$$h[n, m] = \text{rect} \left[ \frac{n - N/2}{N/2} \right] \delta[m] \Delta_t. \quad (6.47)$$

The rectangle function is of duration  $(N + 1)$ , extending from  $n = 0$  to  $n = N$ , and centered at  $N/2$ . We assume that  $N$  is an even integer. As with the continuous-space case, the deblurring operation (to retrieve  $f[n, m]$  from  $g[n, m]$ ) is performed in the spatial frequency domain, wherein the assumption that  $N$  is an even integer is not relevant, so the assumption is mathematically convenient, but not critical.

The spatial frequency domain analogue of Eq. (6.46) is

$$\mathbf{G}(\Omega_1, \Omega_2) = \mathbf{F}(\Omega_1, \Omega_2) \mathbf{H}(\Omega_1, \Omega_2). \quad (6.48)$$

Here,  $\mathbf{G}(\Omega_1, \Omega_2)$  is the 2-D spectrum of the recorded blurred image, and  $\mathbf{H}(\Omega_1, \Omega_2)$  is the discrete-space spatial frequency response function (DSSF) response of  $h[n, m]$ . From entry #7 in **Table 2-8**, and noting that  $\Delta_t = T/N$ , the DSSF response function corresponding to Eq. (6.47) is given by

$$\begin{aligned}\mathbf{H}(\Omega_1, \Omega_2) &= \text{DSFT}\{h[n, m]\} \\ &= \text{DTFT}_{n \rightarrow \Omega_1} \left\{ \text{rect} \left[ \frac{n - N/2}{N/2} \right] \right\} \\ &\quad \times \text{DTFT}_{m \rightarrow \Omega_2} \{ \delta[m] \} \frac{T}{N} \\ &= \frac{T}{N} \frac{\sin[\Omega_1 \left( \frac{N+1}{2} \right)]}{\sin(\Omega_1/2)} e^{-j\Omega_1 N/2}.\end{aligned}\quad (6.49)$$

The sinc function dictates that  $\mathbf{H}(\Omega_1, \Omega_2) = 0$  for  $\Omega_1(N + 1)/2 = k\pi$  for nonzero integer values of  $k$ . Consequently, the inverse filter  $1/\mathbf{H}(\Omega_1, \Omega_2)$  is undefined at these values of  $\Omega_1$ , thereby requiring regularization.

Regularization can be accomplished using the Wiener filter

(Section 6-4.4), which leads to

$$\begin{aligned} \mathbf{F}(\Omega_1, \Omega_2) &= \frac{\mathbf{G}(\Omega_1, \Omega_2) \mathbf{H}^*(\Omega_1, \Omega_2)}{|\mathbf{H}(\Omega_1, \Omega_2)|^2 + \lambda^2} \\ &= \frac{\mathbf{G}(\Omega_1, \Omega_2) \frac{T}{N} \frac{\sin[\Omega_1(\frac{N+1}{2})]}{\sin(\Omega_1/2)} e^{j\Omega_1 N/2}}{\left(\frac{T}{N}\right)^2 \left| \frac{\sin[\Omega_1(\frac{N+1}{2})]}{\sin(\Omega_1/2)} \right|^2 + \lambda^2}. \quad (6.50) \end{aligned}$$

The image deblurring process is illustrated by the three images shown in [Fig. 6-11](#).

**(a)** A (still)  $(225 \times 225)$  image  $f[n, m]$  has been motion-blurred into a  $(225 \times 275)$  image  $g[n, m]$ . The image is a simulated highway sign taken from a moving vehicle. The length  $(N+1)$  of the blur is 51.

**(b)** To further distort the blurred image, noise was added to image  $g[n, m]$  to produce

$$g'[n, m] = g[n, m] + v[n, m], \quad (6.51)$$

with  $v[n, m]$  being a zero-mean random variable with a variance of 100. The consequent signal-to-noise ratio is 5.35 dB.

**(c)** Application of the Wiener filter recipe given by Eq. (6.50) and then inverting to the spatial domain to obtain  $f[n, m]$  leads to the reconstructed image shown in [Fig. 6-11\(c\)](#). The implementation involved the use of 2-D DFTs of order  $(275 \times 275)$ . Reconstructed images were generated for different values of  $\lambda$  (in Eq. (6.50)); the value of  $\lambda = 1$  provided the best result visually.

**Concept Question 6-7:** How does one determine the length  $N+1$  of the PSF for motion blur from the spectrum of the blurred image?

**Concept Question 6-8:** How does one determine the value of  $\lambda$  to use in Tikhonov regularization?

**Exercise 6-2:** For motion blur with a PSF of length  $N+1 = 60$  in the direction of motion, at what spatial frequencies  $\Omega$  will the spatial frequency response be zero?

**Answer:** From Eq. (6.50), the numerator of the spatial frequency response is zero when  $\Omega = \pm k\pi/30$  for any nonzero integer  $k$ .

## Concepts

- Image restoration is about reconstructing an image from its blurred, noisy, or interference-corrupted version.
- Lowpass filtering reduces noise, but it blurs edges and fine-scale image features. Wavelet-based denoising (in Chapter 7) reduces noise while preserving edges.
- A Hamming-windowed PSF reduces “ringing” in the

- filtered image.
- Notch filtering reduces sinusoidal interference caused by AC interference.
- Motion blur deconvolution undoes the blur caused by camera motion.

## Summary

## Mathematical Formulae

### 1-D Hamming-windowed lowpass filter

$$h_{\text{FIR}}[n] = \begin{cases} \frac{\Omega_c}{\pi} \text{sinc}\left(\frac{\Omega_c}{\pi}n\right) \left[0.54 + 0.46 \cos\left(\frac{\pi n}{N}\right)\right] & |n| \leq N, \\ 0 & |n| > N \end{cases}$$

### 2-D Hamming-windowed lowpass filter

$$h_{\text{FIR}}[n, m] = h_{\text{FIR}}[n] h_{\text{FIR}}[m]$$

### Deconvolution formulation

$$g[n, m] = h[n, m] * * f[n, m] + v[n, m]$$

### Deconvolution implementation by 2-D DFT

$$\mathbf{F}[k_1, k_2] = \frac{\mathbf{G}[k_1, k_2]}{\mathbf{H}[k_1, k_2]}$$

### Tikhonov regularization criterion

$$e = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} [(g[n, m] - h[n, m] * * \hat{f}[n, m])^2 + \lambda^2 \hat{f}[n, m]^2]$$

### Wiener filter

$$\hat{\mathbf{F}}[k_1, k_2] = \mathbf{G}[k_1, k_2] \frac{\mathbf{H}^*[k_1, k_2]}{|\mathbf{H}[k_1, k_2]|^2 + \lambda^2}$$

### Motion blur PSF

$$h(x, y) = \frac{1}{s} \text{rect}\left(\frac{x - D/2}{D}\right) \delta(y)$$

### Motion blur PSF

$$h[n, m] = \text{rect}\left[\frac{n - N/2}{N/2}\right] \delta[m] \Delta_t$$

## Important Terms

Provide definitions or explain the meaning of the following terms:

deconvolution

Hamming window

motion blur

notch filter

Tikhonov criterion

Wiener filter

## PROBLEMS

### Section 6-2: Denoising by Lowpass Filtering

**6.1** The usual procedure for lowpass filtering an  $(N \times N)$  image  $f[n, m]$  is to set its 2-D DFT  $\mathbf{F}[k_1, k_2] = 0$  for  $K \leq k_1, k_2 \leq N + 2 - K$  for some index  $K$ . Specify two problems with using this brick-wall filtering approach.

**6.2** Explain how a Hamming-windowed filter solves the problems in Problem 6.1.

**6.3** This problem investigates denoising images by 2-D brick-wall lowpass filtering. The program adds noise to the “clown” image, then 2-D brick-wall lowpass filters it:

```
load clown.mat;
Y=X+0.2*randn(200,200);FY=fft2(Y);
FZ=FY;L=??;FZ(L:202-L,L:202-L)=0;
Z=real(ifft2(FZ));imagesc(Z),colormap(gray)
```

- (a) Run this for  $L = 101, 20$ , and  $10$ . Display the filtered images.
- (b) Discuss the trade-offs involved in varying the cutoff frequency.

**6.4** This problem denoises by 2-D lowpass filtering with a separable 2-D lowpass filter  $h[m, n] = h[m] h[n]$ , where  $h[n]$  is an FIR lowpass filter designed by windowing the impulse response of a brick-wall lowpass filter, which suppresses “ringing.”

- (a) Design a 1-D lowpass filter  $h[n]$  of duration 31 by using a Hamming window on the impulse response of a brick-wall lowpass filter with cutoff frequency  $\Omega_0 = \frac{\pi}{3}$ .
- (b) Filter the “letters” image by 2-D convolution with  $h[m] h[n]$ . Display the result.
- (c) Try varying the filter duration and cutoff frequency. See if this improves the result.

### Section 6-3: Notch Filtering

**6.5** We derive a system for performing notch filtering of the 2-D sinusoidal signal  $f[n, m] = \cos((\Omega_1)_0 n) \cos((\Omega_2)_0 m)$ , which is to be eliminated from an image. Let the PSF

$$h[n, m] = \begin{bmatrix} 1 & 0 & 1 \\ 0 & h[0, 0] & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

where  $h[0, 0] = -4 \cos((\Omega_1)_0) \cos((\Omega_2)_0)$ .

- (a) Compute the spatial frequency response  $\mathbf{H}(\Omega_1, \Omega_2)$  of this system.  
 (b) Show that it does indeed eliminate  $f[n, m]$ .  
 (c) Specify a problem with using this LSI system as a notch filter.

**6.6** Download file P66.mat. Use notch filtering to eliminate stripes in the clown image. Print out the striped image, its spectrum, and the notch-filtered image and its spectrum. Hint: Let  $x[n]$  have length  $N$  and its  $N$ -point DFT  $\mathbf{X}[k]$  have a peak at  $k = k_0$ . From Eq. (2.88), the peak represents a sinusoid with frequency  $\Omega_0 = 2\pi k_0/N$ . The sinusoid will repeat about  $\Omega/(2\pi)N = k_0$  times over the length  $N$  of  $x[n]$ .

**6.7** Download file P67.mat. Use notch filtering to eliminate stripes in the head. Print out the striped image, its spectrum, and the notch-filtered image and its spectrum. Hint: Let  $x[n]$  have length  $N$  and its  $N$ -point DFT  $\mathbf{X}[k]$  have a peak at  $k = k_0$ . From Eq. (2.88), the peak represents a sinusoid with frequency  $\Omega_0 = 2\pi k_0/N$ . The sinusoid will repeat about  $\Omega/(2\pi)N = k_0$  times over the length  $N$  of  $x[n]$ .

**6.8** Download file P68.mat. Use notch filtering to eliminate two sets of lines. Note that there are horizontal lines on top of the image, and vertical lines in the image. Use the procedure used in Section 6-2, but in both horizontal and vertical directions. Print out the original image, its spectrum, and the notch-filtered image and its spectrum.

**6.9** Download file P69.mat. Use notch filtering to eliminate two sets of lines. Note that there are horizontal lines on top of the image, and vertical lines in the image. Use the procedure used in Section 6-2, but in both horizontal and vertical directions. Print out the original image, its spectrum, and the notch-filtered image and its spectrum.

## Section 6-4: Image Deconvolution

**6.10** Derive the Wiener filter by showing that the  $\hat{f}[n, m]$  minimizing the Tikhonov functional

$$\mathcal{T} = \sum_n \sum_m [(g[n, m] - h[n, m] * \hat{f}[n, m])^2 + \lambda^2 (\hat{f}[n, m])^2]$$

has 2-D DFT

$$\hat{\mathbf{F}}[k_1, k_2] = \mathbf{G}[k_1, k_2] \frac{\mathbf{H}[k_1, k_2]^*}{|\mathbf{H}[k_1, k_2]|^2 + \lambda^2}.$$

Hints: Use Parseval's theorem and

$$|a + b|^2 = aa^* + ab^* + ba^* + bb^*.$$

Add and subtract  $|\mathbf{GH}^*|^2$ , divide by  $(\mathbf{HH}^* + \lambda^2)$ , and complete the square.

**6.11** This is an introductory image deconvolution problem using a Wiener filter. A crude lowpass filter is equivalent to convolution with PSF

$$h[n, m] = \begin{cases} 1/L^2 & \text{for } 0 \leq n, m \leq L-1, \\ 0 & \text{otherwise.} \end{cases}$$

This problem undoes this crude lowpass filter using a Wiener filter with  $\lambda = 0.01$ .

- (a) Blur the clown image with  $h[n, m]$  for  $L = 11$  using:

```
clear; load clown;
H=ones(11,11)/121; Y=conv2(X,H);
```

- (b) Deblur the blurred image using a Wiener filter using:

```
imagesc(Y), colormap(gray);
FY=fft2(Y); FH=fft2(H, 210, 210);
FZ=FY.*conj(FH)./(abs(FH).*abs(FH)+.0001);
Z=real(ifft2(FZ));
figure, imagesc(Z), colormap(gray)
```

**6.12** This is an introductory image deconvolution problem using a Wiener filter. A crude lowpass filter is equivalent to convolution with PSF

$$h[n, m] = \begin{cases} 1/L^2 & \text{for } 0 \leq n, m \leq L-1, \\ 0 & \text{otherwise.} \end{cases}$$

This problem undoes this crude lowpass filter using a Wiener filter with  $\lambda = 0.01$ .

- (a) Blur the letters image with  $h[n, m]$  for  $L = 15$  using:

```
clear; load letters; H=ones(15,15)/225;
Y=conv2(X,H);
```

- (b) Deblur the blurred image using a Wiener filter using:

```
imagesc(Y), colormap(gray); FY=fft2(Y);
FH=fft2(H, 270, 270);
FZ=FY.*conj(FH)./(abs(FH).*abs(FH)+.0001);
Z=real(ifft2(FZ));
figure, imagesc(Z), colormap(gray)
```

**6.13** Deblurring due to an out-of-focus camera can be modelled crudely as a 2-D convolution with a disk-shaped point-spread function

$$h[n, m] = \begin{cases} 1 & \text{for } n^2 + m^2 < R^2, \\ 0 & \text{for } n^2 + m^2 > R^2. \end{cases}$$

This problem deblurs an out-of-focus image in the (unrealistic) absence of noise.

- (a) Blur the letter image with an (approximate) disk PSF using
 

```
H(25,25)=0; for I=1:25; for J=1:25;
if((I-13)*(I-13)+(J-13)*(J-13)<145);
H(I,J)=1;end;end;end;
load letters;Y=conv2(X,H);
subplot(221),imagesc(Y),colormap(gray)
```
- (b) Deblur this out-of-focus image using the command
 

```
Z=real(ifft2(fft2(Y)./fft2(H,280,280)));
subplot(222),imagesc(Z),colormap(gray)
```

 Note that the size of the blurred image is  $256 + 25 - 1 = 280$ .
- (c) Explain why this approach will not work in the real world (i.e., in the presence of noise).

**6.14** Repeat Problem 6.13, only now add noise to the blurred image:

- (a) Add noise to the blurred image using
 

```
Y=Y+100*randn(280,280);
```
- (b) Deblur the image as in Problem 6.13. You should get noise!
- (c) Deblur the image using a Wiener filter, using
 

```
FH=fft2(H,280,280);
W=real(ifft2(fft(Y).*conj(FH)./
(abs(FH).*abs(FH)+10)));
subplot(221),imagesc(Z),colormap(gray)
subplot(222),imagesc(W),colormap(gray)
```

**6.15** Repeat Problem 6.13 using the clown image. Note that the size of the blurred image is now  $200 + 25 - 1 = 224$ .

**6.16** Repeat Problem 6.14 using the clown image. Note that the size of the blurred image is now  $200 + 25 - 1 = 224$ . Add noise using  $Y=Y+randn(224,224)$ ; and use  $\lambda^2 = 100$ , since the clown pixel values have a maximum value of only 1, while the letters pixel values have a maximum value of 255.

## Section 6-6: Motion Blur Deconvolution

**6.17** The motion sensor in a Steadicam camera records its motion as being in increasing  $x$  and  $y$  at an angle  $\theta$  from the horizontal ( $x$ ) at speed  $r$  cm/s for  $T$  s for a diagonal distance of  $D = rT$  cm. Compute the spatial frequency response of the camera motion.

**6.18** The motion sensor in a Steadicam camera records its motion as being:

- (a) horizontal, in increasing  $x$ , at speed  $r_x$  cm/s for  $T_x$  s,

- (b) followed by, for an additional  $T_y - T_x$  s (for  $T_x < t < T_y$ ),
- (c) vertical, in increasing  $y$ , at speed  $r_y$  cm/s for  $T_y - T_x$  s.

Compute the spatial frequency response of the camera motion.

**6.19** Download file `P619.mat`. The goal is to deconvolve the motion blur.

- (a) Compute and display the spectrum of the blurred image. What causes the vertical bands of zeros (VBZ)? Hint: See the numerator of Eq. (6.45).
- (b) From the spacing between the VBZ, compute  $N$  in Eq. (6.45).
- (c) Deconvolve the image using the Wiener filter Eq. (6.46). Let  $T = 1$  and  $\lambda = 0.01$ .

**6.20** Download file `P620.mat`. The goal is to deconvolve the motion blur.

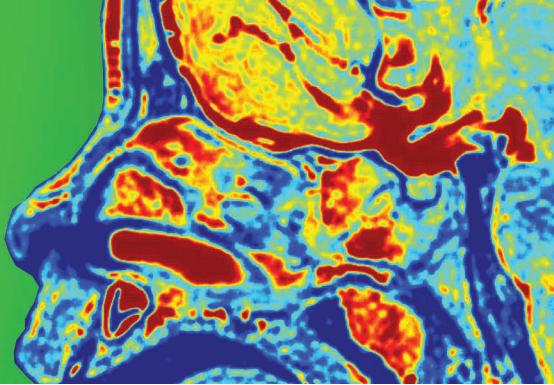
- (a) Compute and display the spectrum of the blurred image. What causes the vertical bands of zeros (VBZ)? Hint: See the numerator of Eq. (6.45).
- (b) From the spacing between the VBZ, compute  $N$  in Eq. (6.45).
- (c) Deconvolve the image using the Wiener filter Eq. (6.46). Let  $T = 1$  and  $\lambda = 0.01$ .

**6.21** Download file `P621.mat`. The goal is to deconvolve the motion blur. The blurred image in this problem is the SAR image from Chapter 4.

- (a) Compute and display the spectrum of the blurred image. What causes the vertical bands of zeros (VBZ)? Hint: See the numerator of Eq. (6.45).
- (b) From the spacing between the VBZ, compute  $N$  in Eq. (6.45).
- (c) Deconvolve the image using the Wiener filter Eq. (6.46). Let  $T = 1$  and  $\lambda = 0.01$ .

# CHAPTER

# 7



7

# Wavelets and Compressed Sensing

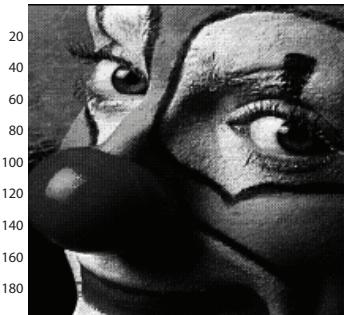
## Contents

- Overview, 203
  - 7-1** Tree-Structured Filter Banks, 203
  - 7-2** Expansion of Signals in Orthogonal Basis Functions, 206
  - 7-3** Cyclic Convolution, 209
  - 7-4** Haar Wavelet Transform, 213
  - 7-5** Discrete-Time Wavelet Transforms, 218
  - 7-6** Sparsification Using Wavelets of Piecewise-Polynomial Signals, 223
  - 7-7** 2-D Wavelet Transform, 228
  - 7-8** Denoising by Thresholding and Shrinking, 232
  - 7-9** Compressed Sensing, 236
  - 7-10** Computing Solutions to Underdetermined Equations, 238
  - 7-11** Landweber Algorithm, 241
  - 7-12** Compressed Sensing Examples, 242
- Problems, 251

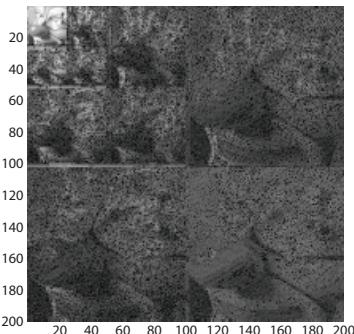
## Objectives

Learn to:

- Compute wavelet transforms of 1-D signals and 2-D images.
- Design discrete-time Daubechies wavelets of various orders.
- Use wavelet transforms to compress signals and images.
- Use thresholding and shrinkage of its wavelet transform to denoise an image.
- Use compressed sensing to reconstruct an image from a reduced set of measurements.



(a)  $200 \times 200$  clown image



(b) 2-D D3 Daubechies wavelet transform of the clown image

The wavelet transform is an important tool. Its applications in image processing include: denoising while preserving edges, compression, and compressed sensing, which is reconstruction of an image from a reduced set of linear measurements of it. After a review of the 1-D discrete-time Haar and Daubechies wavelet transforms, we present applications of them to denoising, compression, and compressed sensing, including image inpainting and X-ray tomography (CAT).

## Overview

The **wavelet transform** is an important signal processing tool for representing signals or images consisting mostly of slowly varying regions, but containing a few fast-varying regions. Like the discrete Fourier transform (DFT), it represents signals or images as a linear combination of **basis functions**.

One characteristic of the DFT is that images, even with only a few fast-varying segments such as edges, require high spatial frequency complex exponentials to represent them. Hence, most or all of the DFT values  $X[k_1, k_2]$  are nonzero. In contrast, the basis functions used in wavelet transforms are localized in time and frequency. This means that a signal that is mostly slowly-varying but has a few localized fast-varying regions requires only a few low-resolution basis functions to represent the slowly-varying regions, and a few high-resolution basis functions to represent just the localized fast-varying regions. Many of the wavelet transform values are thus zero (or near zero). This feature leads to the following three major applications of wavelet transforms:

- **Compression** of signals and images: they are represented in the wavelet transform domain by many fewer numbers than in the original signal or image. The JPEG-2000 image compression standard uses the wavelet transform.
- **Compressed sensing** of signals and images: since the signal or image in the wavelet transform domain requires many fewer numbers to represent it, it can be reconstructed from many fewer observations than would be required to reconstruct the original signal or image. An introduction to compressed sensing is presented later in this chapter.
- **Filtering of signals and images**: since the signal or image in the wavelet transform domain requires many fewer numbers to represent it, thresholding small values of the wavelet transform of a noisy signal or image to zero reduces the noise in the original signal or image. We will show that the combination of **thresholding** and **shrinkage** gives results far superior to using the 2-D DFT for noise reduction.

After this Overview section, we present the **Haar** wavelet transform, which is the simplest wavelet transform, and yet illustrates many features of the family of wavelet transforms. We then present **quadrature-mirror filters (QMFs)** and derive the **Smith-Barnwell condition** for perfect reconstruction of the original signal from its wavelet transform. We conclude our treatment of wavelets by deriving the **Daubechies wavelet function**, which is the most commonly used wavelet function because it sparsifies many real-world signals. Finally, examples of image compression, denoising, and compressed sensing are provided.

## 7-1 Tree-Structured Filter Banks

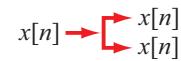
Wavelet transforms can be viewed as a generalization of **tree-structured filter banks (TSFBs)** and **subband coding**. A filter bank is a set of bandpass filters connected in parallel; each bandpass filter passes a different range of frequencies. So the signal input into the filter bank is separated into different components, each of which consists of a different part of the spectrum of the input signal.

Filter banks are used in audio signal processing. In human hearing, some frequencies cannot be heard as well as others. Also a large component at one frequency can mask a component at another frequency. So it makes sense to keep only the frequency bands that humans can hear. The basic idea behind coding of signals is to omit the frequency bands that contribute little to the perception of the signal, and the process is called **subband coding**. The mp3 coding of music uses this idea (and many others).

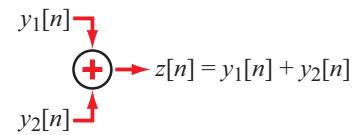
An efficient tree-like filter structure for separating a 1-D sampled signal into different frequency bands (**subband decomposition**) is shown in Fig. 7-1, in which  $g[n]$  is a lowpass filter with cutoff frequency  $\Omega_c = \pi/2$  and  $h[n]$  is a highpass filter with the same cutoff frequency. The concept is easily extendable to 2-D images.

Filter-bank diagrams may involve five types of operations:

- (1) Duplication:



- (2) Addition:



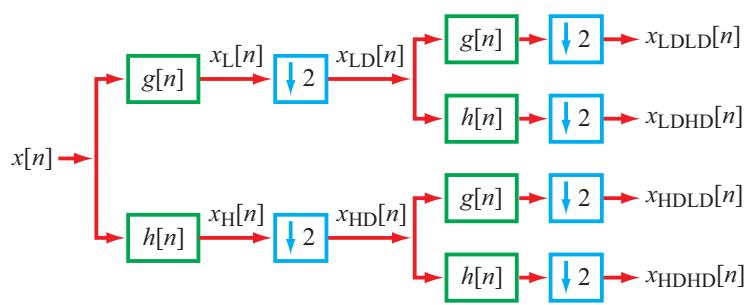
- (3) Downsampling (decimation):



Discarding every other sample in  $x[n]$ .

- (4) Upsampling (zero-stuffing):

$$x[n] \rightarrow \boxed{\uparrow 2} \rightarrow y_u[n] = \begin{cases} x[n/2] & \text{for } n \text{ even} \\ 0 & \text{for } n \text{ odd} \end{cases}$$



**Figure 7-1** Tree-like filter structure for subband decomposition. The green boxes denote lowpass and highpass frequency filters, realized through cyclic convolution.

Inserting a zero between successive values of  $x[n]$ .

(5) Convolution:

$$x[n] \rightarrow \boxed{h[n]} \rightarrow y[n] = h[n] * x[n]$$

In Fig. 7-1, the input signal  $x[n]$  with spectrum (DTFT)  $\mathbf{X}(\Omega)$  is separated into a low-frequency-band signal  $x_L[n]$  whose spectrum is roughly

$$\mathbf{X}_L(\Omega) = \begin{cases} \mathbf{X}(\Omega) & \text{for } 0 \leq |\Omega| < \pi/2, \\ 0 & \text{for } \pi/2 < |\Omega| \leq \pi, \end{cases} \quad (7.1)$$

and a high-frequency-band signal  $x_H[n]$  whose spectrum is roughly

$$\mathbf{X}_H(\Omega) = \begin{cases} 0 & \text{for } 0 \leq |\Omega| < \pi/2, \\ \mathbf{X}(\Omega) & \text{for } \pi/2 < |\Omega| \leq \pi. \end{cases} \quad (7.2)$$

Each signal can be downsampled by 2 without aliasing, resulting in  $x_{LD}[n] = x_L[2n]$  and  $x_{HD}[n] = x_H[2n]$ . There are now two different signals, each of which is sampled only half as often as  $x[n]$ , so the total number of samples is unaltered, and each represents a different frequency band of the original signal.

This same decomposition can then be applied to each of the two downsampled signals  $x_{LD}$  and  $x_{HD}$ , which results in four signals, each of which is sampled only one fourth as often as  $x[n]$ , so the total number of samples is the same, and each represents a different frequency band of bandwidth  $\pi/4$ . Repeating this decomposition  $N$  times,  $x[n]$  can be decomposed

into  $2^N$  signals, each of which represents a different frequency band of bandwidth  $\pi/2^N$  and is sampled only  $1/2^N$  as often as  $x[n]$ . Use of  $N = 5$ , resulting in  $2^5 = 32$  subbands, is a common choice.

In Fig. 7-1:

- $x_{LD}[n]$  is the lowpass part,  $0 \leq |\Omega| \leq \frac{\pi}{2}$ , of  $x[n]$ .
- $x_{HD}[n]$  is the highpass part,  $\frac{\pi}{2} \leq |\Omega| \leq \pi$ , of  $x[n]$ .

The signals at the second stage of the filter bank have spectra that are roughly as follows:

- $x_{LLDL}[n]$  is the lowpass part,  $0 \leq |\Omega| \leq \frac{\pi}{2}$ , of  $x_{LD}[n]$ , which is equivalent to the lowpass part,  $0 \leq |\Omega| \leq \frac{\pi}{4}$ , of  $x[n]$ .
- $x_{LDHD}[n]$  is the highpass part,  $\frac{\pi}{2} \leq |\Omega| \leq \pi$ , of  $x_{LD}[n]$ , which is equivalent to the bandpass part  $\frac{\pi}{4} \leq |\Omega| \leq \frac{\pi}{2}$  of  $x[n]$ .

If we were to extend the filter bank in Fig. 7-1 to another stage, the signals at the third stage would have spectra that are roughly:

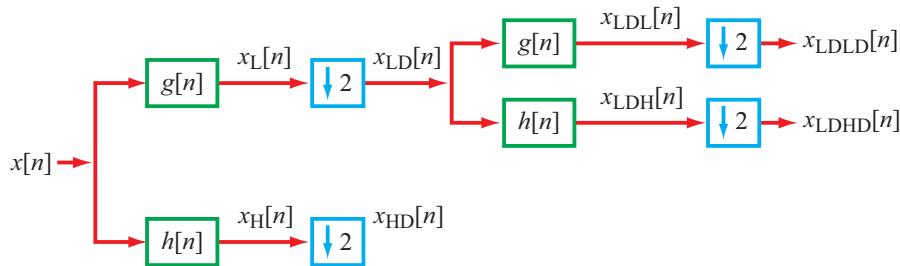
$x_{LLDLHD}[n]$  is the lowpass part  $0 \leq |\Omega| \leq \frac{\pi}{2}$ , of  $x_{LLDL}[n]$ , which is equivalent to the lowpass part,  $0 \leq |\Omega| \leq \frac{\pi}{8}$ , of  $x[n]$ .

$x_{LDLDHD}[n]$  is the highpass part,  $\frac{\pi}{2} \leq |\Omega| \leq \pi$ , of  $x_{LLDL}[n]$ , which is equivalent to the bandpass part  $\frac{\pi}{8} \leq |\Omega| \leq \frac{\pi}{4}$  of  $x[n]$ .

At each stage, decimation (halving the sampling rate) expands the spectrum of each signal to the full range  $0 \leq |\Omega| < \pi$  (see Section 7-1.1).

### 7-1.1 Octave-Based Filter Banks

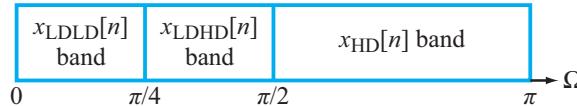
The tree structure in Fig. 7-1 can be replaced with the simpler structure shown in Fig. 7-2. As we show later in Section 7-5,



**Figure 7-2** Octave-based filter bank structure for subband decomposition. Note that only the upper half (lowpass) of each stage is decomposed further.

the wavelet transform is implemented using this simpler tree structure.

This form of filter bank decomposes the spectrum  $\mathbf{X}(\Omega)$  of  $x[n]$  into **octaves**. An octave is a frequency band  $f_{\min} < f < f_{\max}$  in which  $f_{\max} = 2f_{\min}$ , using either continuous-time frequency  $f$  in Hz or discrete-time frequency  $\Omega$ . We use  $\Omega$  in the sequel, since the signals are all in discrete time. The octave  $\pi/2^K \leq \Omega \leq \pi/(2^{K-1})$  is represented by a single signal. Since the width of this band is  $\pi/2^K$ , the sampling rate can be reduced by a factor of  $2^K$  using decimation by 2 at each of  $K$  stages. The lowest band  $0 \leq \Omega \leq \pi/2^K$  is represented by a single signal, as shown in **Fig. 7-3** for  $K = 2$ .



**Figure 7-3** Partitioning of signal spectrum by octave-based filter bank for  $K = 2$ .

## 7-1.2 Reconstruction Using Octave-Based Filter Banks

The original signal  $x[n]$  can be recovered from its octave-based frequency decomposition using the reconstruction filter bank shown in **Fig. 7-4**. The inverse wavelet transform in Section 7-5 is implemented using this same structure. Each filter bank stage consists of **zero-stuffing** (inserting zeros between samples),

denoted as

$$x[n] \rightarrow \begin{array}{|c|} \hline \uparrow 2 \\ \hline \end{array} \rightarrow y_u[n] = \begin{cases} x[n/2] & \text{for } n \text{ even,} \\ 0 & \text{for } n \text{ odd,} \end{cases} \quad (7.3)$$

which gives

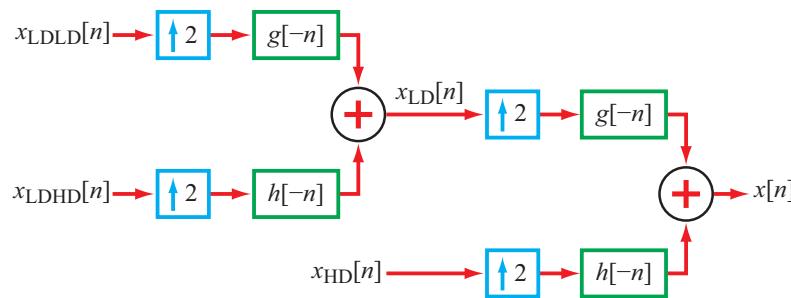
$$y_u[n] = \{ \dots, \underline{x[0]}, 0, x[1], 0, x[2], 0, x[3], \dots \},$$

followed by **interpolation** using a filter  $g[-n]$  or  $h[-n]$  to replace the zeros introduced by zero-stuffing with the correct values of the signal input into that filter-bank stage. For octave-based filter banks, the time reversals are unnecessary, but time reversals are necessary when using this same structure for the inverse wavelet transform introduced later in Section 7-5.

Zero-stuffing is presented in more detail in Subsection 7-4.3.

## 7-1.3 Significance of Wavelets

The wavelet transform differs from this subband coding in that  $x[n]$  is not recursively decomposed explicitly into lower and higher frequency bands, although the decomposition is still roughly into lower and higher frequency bands. Instead, the lower frequency bands are replaced with signals that represent slowly-varying parts of  $x[n]$ , and the higher frequency bands are replaced with signals that represent fast-varying parts of  $x[n]$ . The latter signals are mostly zero-valued if  $x[n]$  is slowly-varying most of the time.



**Figure 7-4** Octave-based filter bank structure for subband reconstruction.

► Real-world signals and images do tend to consist of mostly slowly-varying regions, containing a few localized regions in which they are fast-varying. Wavelets are good at representing such signals with wavelet transforms that are mostly zero-valued. ▶

To see why representing a signal using only a few wavelet transform components is useful, consider periodic signals. A periodic signal  $x(t)$  with period  $T_0$  and maximum frequency  $B$  (in Hz) can be represented in the frequency domain using only  $BT_0$  frequencies, since its spectrum consists of harmonics at frequencies  $k/T_0$  Hz for integers  $k$ . The maximum frequency  $B$  must equal  $N/T_0$  for some integer  $N$ , or equivalently,  $N = BT_0$  frequencies. Instead of storing  $x(t)$ , we can generate it using  $BT_0$  sinusoidal generators, each of which requires only an amplitude and phase. So  $x(t)$  can be **compressed** into  $2BT_0$  (plus a dc term, if present) numbers.

If noise had been added to  $x(t)$ , most of the noise can be eliminated because any part of the spectrum of the noisy  $x(t)$  that is not at a harmonic  $k/T_0$  in Hz is noise and can be filtered out. We will perform similar actions on signals and images that are not periodic, but which have wavelet transforms that are mostly zero-valued. This includes many real-world signals and images.

**Concept Question 7-1:** Why is it useful to represent a signal or image using a few wavelet coefficients?

**Exercise 7-1:** (a) An input signal of duration  $N$  is fed into a tree-based filter bank of five stages. What is the combined total duration of the output signals? Repeat for (b) an octave-based filter bank.

**Answer:** (a)  $2^5N = 32N$ . (b)  $N$ , because lower-wavenumber bands can be sampled less often.

## 7-2 Expansion of Signals in Orthogonal Basis Functions

### 7-2.1 Signal Expansion

The general form of the expansion of a signal  $x[n]$  into a linear combination of **basis functions**  $\{\phi_k[n]\}$  with coefficients  $\{\mathbf{x}_k\}$  (often called an **orthogonal expansion** of  $x[n]$ ) is

$$x[n] = \sum_{k=1}^{\infty} \mathbf{x}_k \phi_k[n]. \quad (7.4)$$

The coefficients  $\{\mathbf{x}_k\}$  constitute the **transform** of  $x[n]$  using the basis functions  $\{\phi_k[n]\}$ . For wavelet transforms, the basis functions  $\{\phi_k[n]\}$  and coefficients  $\{\mathbf{x}_k\}$  are real-valued, but for many other orthogonal expansions, such as the DFT and Fourier series, they are complex-valued. We assume here that the signal  $x[n]$  to be expanded is real-valued.

Basis functions  $\{\phi_k[n]\}$  are chosen to be **orthogonal**, which means that, for some constant  $C$ ,

$$\sum_{n=-\infty}^{\infty} \phi_{k_1}[n] \phi_{k_2}^*[n] = C \delta[k_1 - k_2]. \quad (7.5)$$

An important consequence of the orthogonality property is that coefficients  $\mathbf{x}_k$  can be computed from  $x[n]$  using

$$\mathbf{x}_k = \frac{1}{C} \sum_{n=-\infty}^{\infty} x[n] \Phi_k^*[n]. \quad (7.6)$$

The other part of the significance is Rayleigh's theorem (Section 7-2.3).

## 7-2.2 Expansion Coefficients

Equation (7.6) can be derived as follows:

1. In Eq. (7.4), change index  $k$  to index  $k_2$ , giving

$$x[n] = \sum_{k_2=1}^{\infty} \mathbf{x}_{k_2} \Phi_{k_2}[n]. \quad (7.7)$$

2. Upon multiplying both sides of Eq. (7.7) by  $\Phi_{k_1}^*[n]$  and summing over index  $n$ , we have

$$\sum_{n=-\infty}^{\infty} x[n] \Phi_{k_1}^*[n] = \sum_{n=-\infty}^{\infty} \sum_{k_2=1}^{\infty} \mathbf{x}_{k_2} \Phi_{k_2}[n] \Phi_{k_1}^*[n]. \quad (7.8)$$

3. Interchanging the order of summations in the right side of Eq. (7.8) leads to

$$\sum_{n=-\infty}^{\infty} x[n] \Phi_{k_1}^*[n] = \sum_{k_2=1}^{\infty} \mathbf{x}_{k_2} \sum_{n=-\infty}^{\infty} \Phi_{k_2}[n] \Phi_{k_1}^*[n]. \quad (7.9)$$

4. Using the orthogonality property given by Eq. (7.5) gives

$$\sum_{n=-\infty}^{\infty} x[n] \Phi_{k_1}^*[n] = \sum_{k_2=1}^{\infty} \mathbf{x}_{k_2} C \delta[k_2 - k_1] = C \mathbf{x}_{k_1}. \quad (7.10)$$

Dividing by  $C$  and replacing  $k_1$  with  $k$  gives Eq. (7.6).

A good example of an orthogonal expansion is the 1-D DFT and its inverse defined in Eq. (2.89) for a finite-length signal  $x[n]$ :

$$\mathbf{X}[k] = \sum_{n=0}^{M-1} x[n] e^{-j2\pi nk/N}, \quad k = 0, \dots, N-1, \quad (7.11a)$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{X}[k] e^{j2\pi nk/N}, \quad n = 0, \dots, M-1. \quad (7.11b)$$

The ranges in the summations in Eqs. (7.11) are particular to the DFT and differ from those in the generic definition of the basis

function given by Eq. (7.4) and its inverse in Eq. (7.6). **Table 7-1** compares attributes of the generic orthogonal expansion basis function with those of the DFT and the continuous-time Fourier series.

► If  $C = 1$  in Eq. (7.6), the orthogonal basis functions are said to be **orthonormal**. The wavelet transform (Section 7-5) uses orthonormal basis functions. ◀

## 7-2.3 Parseval's Theorem and its Significance

For two **continuous-time signals**  $x(t)$  and  $y(t)$  and their associated Fourier transforms  $\mathbf{X}(f)$  and  $\mathbf{Y}(f)$ , Parseval's theorem is stated in Eq. (2.27) as

$$\int_{-\infty}^{\infty} x(t) y^*(t) dt = \int_{-\infty}^{\infty} \mathbf{X}(f) \mathbf{Y}^*(f) df. \quad (7.12a)$$

The special case wherein  $x(t) = y(t)$  is known as Rayleigh's theorem:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |\mathbf{X}(f)|^2 df, \quad (7.12b)$$

which states that the energies of  $x(t)$  and  $\mathbf{X}(f)$  are equal.

Similarly, Rayleigh's theorem for a discrete-time signal  $x[n]$  is given by Eq. (2.80) as

$$\sum_{n=-\infty}^{\infty} |x[n]|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |\mathbf{X}(\Omega)|^2 d\Omega, \quad (7.13)$$

where  $\mathbf{X}(\Omega)$  is the DTFT of  $x[n]$ .

The statements given by Eqs. (7.12) and (7.13) can be generalized to the generic orthogonal basis function expressed in Eq. (7.4):

$$\sum_{n=-\infty}^{\infty} |x[n]|^2 = C \sum_{k=-\infty}^{\infty} |\mathbf{x}_k|^2 \quad (7.14a)$$

**(Rayleigh's theorem),**

$$\sum_{n=-\infty}^{\infty} x[n] y[n]^* = C \sum_{k=-\infty}^{\infty} \mathbf{x}_k \mathbf{y}_k^* \quad (7.14b)$$

**(Parseval's theorem),**

where  $x[n]$  and  $y[n]$  are any two discrete-time functions. Our interest in this book is in real-valued 2-D images, so the complex conjugation on  $x[n]$  and  $y[n]$  in Eq. (7.14a) is irrelevant, but we have decided to retain it for the sake of completeness.

**Table 7-1** 1-D DFT and Fourier series compared with generic orthogonal expansion function.

	Generic Orthogonal Function	DFT	Fourier Series
<b>Basis function</b>	$\phi_k[n]$	$e^{j2\pi kn/N}$	$e^{jn\omega_0 t}$
<b>Expansion</b>	$x[n] = \sum_{k=1}^{\infty} \mathbf{x}_k \phi_k[n]$	$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{X}[k] e^{j2\pi nk/N},$ $n = 0, \dots, M-1$	$x(t) = \sum_{n=-\infty}^{\infty} \mathbf{x}_n e^{jn\omega_0 t}$
<b>Coefficients</b>	$\mathbf{x}_k = \frac{1}{C} \sum_{n=-\infty}^{\infty} x[n] \phi_k^*[n]$	$\mathbf{X}[k] = \sum_{n=0}^{M-1} x[n] e^{-j2\pi nk/N},$ $k = 0, \dots, N-1$	$\mathbf{x}_n = \frac{1}{T_0} \int_0^{T_0} x(t) e^{-jn\omega_0 t} dt$
<b>Orthogonality property</b>	$\sum_{n=-\infty}^{\infty} \phi_{k_1}[n] \phi_{k_2}^*[n] = C\delta[k_1 - k_2]$	$\frac{1}{N} \sum_{k=0}^{N-1} e^{j2\pi(m-n)k/N} = \delta[m-n]$	$\int_0^{T_0} e^{jm\omega_0 t} e^{-jn\omega_0 t} dt = T_0 \delta[m-n]$

For an orthonormal set of basis functions with  $C = 1$ , Rayleigh's theorem states that the energy of  $x[n]$ , summed over all  $n$ , is equal to the energy of coefficients  $\{\mathbf{x}_k\}$ , summed over all  $k$ . The statement is equally applicable to small perturbations in total energy. Consider, for example, a small **perturbation**  $\varepsilon[n]$  from signal  $x[n]$ . The **perturbed** signal is

$$y[n] = x[n] + \varepsilon[n]. \quad (7.15)$$

Signals  $x[n]$  and  $y[n]$  can each be expanded as:

$$x[n] = \sum_{k=1}^{\infty} \mathbf{x}_k \phi_k[n], \quad (7.16a)$$

$$y[n] = \sum_{k=1}^{\infty} \mathbf{y}_k \phi_k[n]. \quad (7.16b)$$

Use of Eqs. (7.16a) and (7.16b) in Eq. (7.15) leads to

$$\varepsilon[n] = y[n] - x[n] = \sum_{k=1}^{\infty} (\mathbf{y}_k - \mathbf{x}_k) \phi_k[n] = \sum_{k=1}^{\infty} \boldsymbol{\varepsilon}_k \phi_k[n], \quad (7.17)$$

where

$$\boldsymbol{\varepsilon}_k = \mathbf{y}_k - \mathbf{x}_k.$$

Application of Rayleigh's theorem, as stated by Eq. (7.14b), to sampled signal  $\varepsilon[n]$  leads to

$$\sum_{n=-\infty}^{\infty} |\varepsilon[n]|^2 = \sum_{k=-\infty}^{\infty} |\boldsymbol{\varepsilon}_k|^2, \quad (7.18)$$

which confirms that the energy of the perturbation  $\varepsilon[n]$  of  $x[n]$  is

equal to the energy of the transform coefficients  $\{\boldsymbol{\varepsilon}_k\}$ .

► An orthonormal transformation does not amplify perturbations, a property that will prove highly significant to the application of wavelets for denoising images. ◀

**Concept Question 7-2:** What is the significance of Rayleigh's theorem?

**Exercise 7-2:** A square wave  $x(t)$  has the Fourier series expansion  $x(t) = \sum_{k=1}^{\infty} \frac{1}{k} \sin(2\pi kt)$ . If  $x(t)$  is passed through a brick-wall lowpass filter with a cutoff frequency of 2.5 Hz. What is the ratio of the average power of the output signal to the average power of  $x(t)$ ? Hint:

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}.$$

**Answer:** By Rayleigh's theorem, the average power of  $x(t)$  is

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6},$$

and the average power of the output signal is

$$\sum_{k=1}^2 \frac{1}{k^2} = 1.25.$$

This is because the lowpass filter sets the Fourier series coefficients for  $k \geq 3$  to zero.

$$\frac{1.25}{\pi^2/6} = 0.76.$$

## 7-3 Cyclic Convolution

### 7-3.1 Why Use Cyclic Convolutions?

In Section 2-7.2, we introduced the concept of cyclic convolution  $x_1[n] \odot x_2[n]$  between two signals  $x_1[n]$  and  $x_2[n]$ , and we showed how it can be computed from the traditional linear convolution  $x_1[n] * x_2[n]$ , as demonstrated in Example 2-6, or by applying the DFT method.

The wavelet transform—the prime topic of this chapter—employs convolution, decimation, and zero-stuffing. If we use linear convolutions in computing the wavelet transform, the total length of the decimated signal will be longer than that of the original signal. At each stage in [Fig. 7-1](#) or [Fig. 7-2](#), for example, the linear convolution with  $h[n]$  or  $g[n]$  would generate a new signal longer than that of the input signal by the length of  $h[n]$  or  $g[n]$ , respectively. *The advantage of cyclic convolution is that the new signal remains at the same length as that of the input signal.* This property limits the computational storage to the same storage required for the original signal.

As noted, the cyclic convolution can be computed directly from the linear convolution, or indirectly by applying the DFT method. In preparation for the material presented in forthcoming sections, we present reviews of both computational approaches.

### 7-3.2 Computing Linear Convolution

Suppose we are given a causal signal  $x[n]$  and a causal filter (or another signal)  $h[n]$  defined as

$$x[n] = \{ \underline{x[0]}, x[1], \dots, x[N_1] \}, \quad (7.19a)$$

and

$$h[n] = \{ \underline{h[0]}, h[1], \dots, h[N_2] \}. \quad (7.19b)$$

Their linear convolution is

$$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} h[i] x[n-i]. \quad (7.20)$$

If  $x[n]$  has

**support:**  $N_{x\ell} \leq n \leq N_{xu}$ , and

**duration:**  $N_x = N_{xu} - N_{x\ell} + 1$ ,

where second subscripts  $\ell$  and  $u$  refer to the lower and upper values of  $N_x$ , and if  $h[n]$  has

**support:**  $N_{h\ell} \leq n \leq N_{hu}$ , and

**duration:**  $N_h = N_{hu} - N_{h\ell} + 1$ ,

then their linear convolution  $y[n]$  has

**support:**  $N_{y\ell} \leq n \leq N_{yu}$ , and

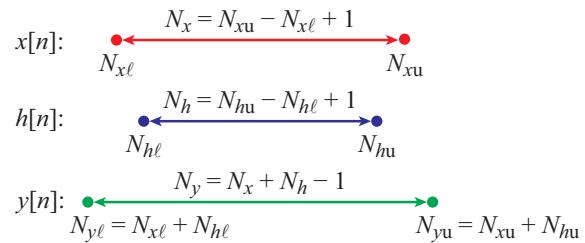
**duration:**  $N_{yu} - N_{y\ell} + 1 = N_x + N_h - 1$ ,

where

$$N_{y\ell} = N_{x\ell} + N_{h\ell}, \quad (7.21e)$$

$$N_{yu} = N_{xu} + N_{hu}. \quad (7.21f)$$

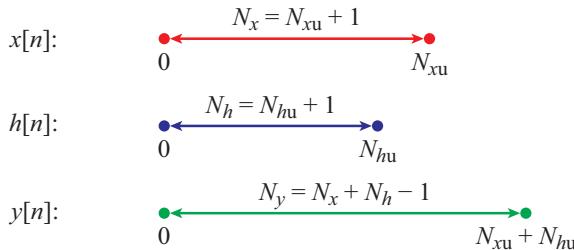
Graphically, these supports and associated durations are:



#### A. Causal \* Causal

If  $x[n]$  and  $h[n]$  are both causal signals, with  $N_{x\ell} = 0$  and  $N_{h\ell} = 0$ , then Eq. (7.20) simplifies to

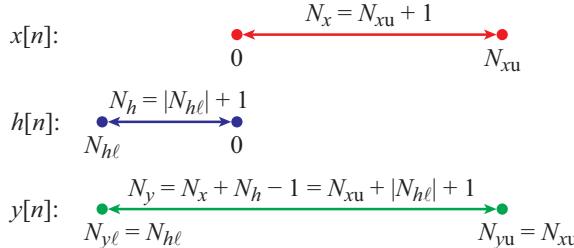
$$y[n] = \sum_{i=0}^n h[i] x[n-i], \quad 0 \leq n \leq N_{yu}. \quad (7.22)$$



### B. Anticausal \* Causal

If  $x[n]$  is causal with  $N_{x\ell} = 0$  and  $h[n]$  is anticausal with its upper limit  $N_{hu} = 0$ , then Eq. (7.20) simplifies to

$$y[n] = \sum_{i=N_{h\ell}}^n h[i] x[n-i], \quad N_{h\ell} \leq n \leq N_{xu}. \quad (7.23)$$



An easy way to compute the convolution given by Eq. (7.23) is to use the time-shift property of convolution (# 5 in **Table 2-6**):

$$\begin{aligned} h[n] * x[n] &= y[n] \\ \downarrow \\ h[n-n_1] * x[n-n_2] &= h[n-n_1-n_2], \end{aligned} \quad (7.24)$$

for any two integers  $n_1$  and  $n_2$ . The procedure involves the following steps:

- (1) Keeping in mind that  $N_{h\ell}$  is a negative integer, delay the anticausal signal  $h[n]$  by  $|N_{h\ell}|$  to obtain the causal signal  $h[n - |N_{h\ell}|]$ .
- (2) Compute  $h[n - |N_{h\ell}|] * x[n]$  using Eq. (7.22), the convolution expression for two causal signals.
- (3) Advance  $h[n - |N_{h\ell}|] * x[n]$  by  $|N_{h\ell}|$ .

### 7-3.3 Computing Cyclic Convolution

The *cyclic* convolution  $y_c[n]$  of order  $N \geq N_x, N_h$  of signals  $x[n]$  and  $h[n]$  as specified by Eq. (7.19), was defined in Eqs. (2.104),

(2.106) and (2.107) as

$$\begin{aligned} y_c[n] &= h[n] \odot x[n] = y[n] + y[(n)_N], \\ n &= 0, 1, \dots, N-1, \end{aligned} \quad (7.25)$$

where  $y[n]$  is the linear convolution of  $x[n]$  and  $h[n]$ , and  $y[(n)_N]$  is  $y[n]$  at values of  $n$  outside the range  $n = 0, 1, \dots, N-1$ , with those values of  $n$  reduced mod( $N$ ); i.e., the remainders after reducing  $n$  by the largest multiple of  $N$  without becoming negative.

### A. Causal \* Causal

If both  $h[n]$  and  $x[n]$  are causal, so that

$$N_{xd} = N_{hd} = 0,$$

then Eq. (7.25) simplifies to

$$y_c[n] = y[n] + y[n+N], \quad n = 0, 1, \dots, N-1, \quad (7.26)$$

where, by definition,  $y[n+N] = 0$  for  $n > N_{yu} - N$ .

The values  $\{y[N], y[N+1], \dots, y[N_{yu}]\}$  of  $y[n]$  for  $n \geq N$  get added point-by-point to the given values  $\{y[0], y[1], \dots, y[N_{yu}-N]\}$ . The process is illustrated graphically in **Fig. 7-5(a)**.

If  $N > N_{yu}$ , the cyclic convolution equals the linear convolution.

### B. Anticausal \* Causal

If  $x[n]$  is causal and  $h[n]$  is anticausal, so that  $N_{x\ell} = 0$  and  $N_{hu} = 0$ , then Eq. (7.25) simplifies to

$$y_c[n] = y[n] + y[n-N], \quad n = 0, 1, \dots, N-1, \quad (7.27)$$

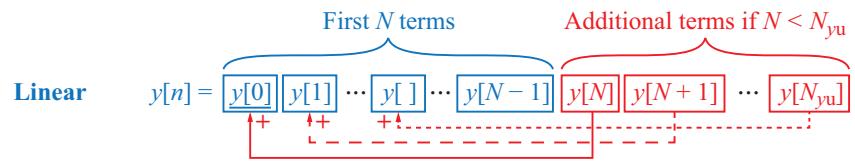
where we set  $y[n-N] = 0$  for  $n < N_{y\ell} + N$  and we assume  $N \geq N_{y\ell}$ . If  $N < N_{y\ell}$ , we must also add the values of  $\{y[n], N \leq n \leq N_{y\ell}\}$  to  $y[n]$  as described in the causal\*causal subsection. This situation seldom happens.

The anticausal values  $\{y[N_{y\ell}], \dots, y[-1]\}$  of  $y[n]$  for  $n < 0$  get added point-by-point to the given values  $\{y[N - |N_{y\ell}|], \dots, y[N-1]\}$ . The process is illustrated graphically in **Fig. 7-5(b)**.

#### Example 7-1: Cyclic Convolution

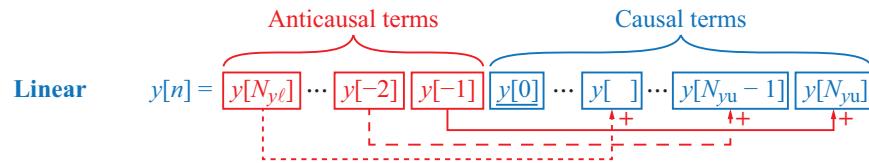
Given

$$x[n] = \{3, 4, 5, 6, 7, 8\},$$



**Cyclic**     $y_c[n] = \boxed{y[0] + y[N]} \boxed{y[1] + y[N+1]} \dots \boxed{y[N-1]}$

(a) Causal \* causal



**Causal**     $y_c[n] = \boxed{y[0]} \boxed{y[1]} \dots \boxed{y[N_{yu}-1] + y[-2]} \boxed{y[N_{yu}] + y[-1]}$

(b) Anticausal \* causal

**Figure 7-5** Graphical representation of obtaining cyclic convolution of order  $N$  from linear convolution.

$$\begin{aligned} h_1[n] &= \{1, 2, 3\}, \\ h_2[n] &= h_1[-n] = \{3, 2, 1\}, \end{aligned}$$

and  $N = 6$ , compute (a)  $y_{c_1} = h_1[n] \odot x[n]$  and (b)  $y_{c_2} = h_2[n] \odot x[n]$ .

$$\begin{aligned} y_{c_1}[n] &= h_1[n] \odot x[n] \\ &= y_1[n] + y_1[n+N] \\ &= \{3 + 37, 10 + 24, 22, 28, 34, 40\} \\ &= \{40, 34, 22, 28, 34, 40\}. \end{aligned}$$

The same result can be obtained by computing the cyclic convolution with the DFT (see Eq. (2.104)):

**Solution: (a)**

$$\begin{aligned} y_1[n] &= h_1[n] * x[n] \\ &= \begin{cases} 1 \times 3 = 3 & \text{for } n = 0, \\ 1 \times 4 + 2 \times 3 = 10 & \text{for } n = 1, \\ 1 \times 5 + 2 \times 4 + 3 \times 3 = 22 & \text{for } n = 2 \\ \vdots & \\ = \{3, 10, 22, 28, 34, 40, 37, 24\}, & \end{cases} \end{aligned}$$

`ifft(fft([3, 4, 5, 6, 7, 8]) .* fft([1, 2, 3], 6))`

(b)

$$\begin{aligned} y_2[n] &= h_2[n] * x[n] \\ &= \begin{cases} 3 \times 3 = 9 & \text{for } n = -2, \\ 3 \times 4 + 2 \times 3 = 18 & \text{for } n = -1, \\ 3 \times 5 + 2 \times 4 + 1 \times 3 = 26 & \text{for } n = 0, \\ \vdots & \\ = 9, 18, \underline{26}, 32, 38, 44, 23, 8. \end{cases} \end{aligned}$$

This is  $\{3, 2, 1\} * \{3, 4, 5, 6, 7, 8\}$  advanced in time by 2.

$$\begin{aligned} y_{c_2}[n] &= h_2[n] \circledcirc x[n] \\ &= y_1[n] + y_1[n - N] \\ &= \{\underline{26}, 32, 38, 44, 23 + 9, 8 + 18\} \\ &= \{\underline{26}, 32, 38, 44, 32, 26\}. \end{aligned}$$

Again, the same result can be obtained by computing the cyclic convolution with the DFT method:

```
ifft(fft([3, 4, 5, 6, 7, 8]) .* fft([1, 0, 0, 0, 3, 2]))
```

### 7-3.4 Decimating and Zero-Stuffing

Decimating 1-D signals by 2, and zero-stuffing 1-D signals by 2, are essential parts of computing discrete-time wavelet transforms. Hence, we present this quick review of these two concepts.

**Decimating** a signal  $x[n]$  by two means deleting every other value of  $x[n]$ :

$$x[n] \rightarrow \boxed{\downarrow 2} \rightarrow y_d[n] = x[2n] = \{ \dots, \underline{x[0]}, x[2], x[4], \dots \}. \quad (7.28)$$

**Zero-stuffing** a signal  $x[n]$  by two means inserting zeros between successive values of  $x[n]$ :

$$\begin{aligned} x[n] \rightarrow \boxed{\uparrow 2} \rightarrow y_u[n] &= \begin{cases} x[n/2] & \text{for } n \text{ even,} \\ 0 & \text{for } n \text{ odd,} \end{cases} \\ &= \{ \dots, \underline{x[0]}, 0, x[1], 0, x[2], 0, x[3], \dots \}. \end{aligned} \quad (7.29)$$

Decimating by 2, followed by zero-stuffing by 2, replaces  $x[n]$  with zeros for odd times  $n$ :

$$x[n] \rightarrow \boxed{\downarrow 2} \rightarrow \boxed{\uparrow 2} \rightarrow \{ \dots, \underline{x[0]}, 0, x[2], 0, x[4], 0, x[6], \dots \},$$

which is equivalent to multiplying  $x[n]$  by  $\frac{1}{2}(1 + (-1)^n)$ :

$$x[n] \rightarrow \boxed{\downarrow 2} \rightarrow \boxed{\uparrow 2} \rightarrow \frac{1}{2}(1 + (-1)^n)x[n]. \quad (7.30)$$

### 7-3.5 A Useful Convolution Relation

Recall from Eq. (7.20) that the discrete-time linear convolution is defined as

$$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} h[i] x[n-i]. \quad (7.31)$$

Let us consider the **modified functions**:

$$h'[n] = (-1)^n h[n], \quad (7.32a)$$

$$x'[n] = (-1)^n x[n]. \quad (7.32b)$$

The linear convolution of the modified functions is

$$\begin{aligned} y'[n] &= h'[n] * x'[n] = \sum_{i=-\infty}^{\infty} h'[i] x'[n-i] \\ &= \sum_{i=-\infty}^{\infty} (-1)^i h[i] (-1)^{n-i} x[n-i] \\ &= \sum_{i=-\infty}^{\infty} (-1)^n h[i] x[n-i] \\ &= (-1)^n \sum_{i=-\infty}^{\infty} h[i] x[n-i] \\ &= (-1)^n y[n]. \end{aligned} \quad (7.33)$$

Combining the result given by Eq. (7.33) with the definition of convolution given in Eq. (2.71a) leads to the conclusion:

$$(-1)^n \{ h[n] * x[n] \} = \{ (-1)^n h[n] \} * \{ (-1)^n x[n] \}. \quad (7.34)$$

### 7-3.6 Wavelet Applications

For applications involving wavelets, the following conditions apply:

- Batch processing is used almost exclusively. This is because the entire original signal is known before processing begins, so the use of non-causal filters is not a problem.
- The order  $N$  of the cyclic convolution is the same as the duration of the signal  $x[n]$ , which usually is very large.

- **Filtering**  $x[n]$  with a filter  $h[n]$  will henceforth mean computing the cyclic convolution  $h[n] \odot x[n]$ . The duration  $L$  of filter  $h[n]$  is much smaller than  $N$  ( $L \ll N$ ), so  $h[n]$  gets zero-padded (see Section 2-7.3) with  $(N - L)$  zeros. The result of the cyclic convolution is the same as  $h[n] * x[n]$ , except for the first  $(L - 1)$  values, which are aliased, and the final  $(L - 1)$  values, which are no longer present, but added to the first  $(L - 1)$  values.
- Filtering  $x[n]$  with the non-causal filter  $h[-n]$  gives the same result as the linear convolution  $h[-n] * x[n]$ , except that the non-causal part of the latter will alias the final  $(L - 1)$  places of the cyclic convolution.
- Zero-padding does not increase the computation, since multiplication by zero is known to give zero, so it need not be computed.
- For two filters  $g[n]$  and  $h[n]$ , both of length  $L$ ,  $g[n] \odot h[n]$  consists of  $g[n] * h[n]$  followed by  $N - (2L - 1)$  zeros.
- As long as the final result has length  $N$ , linear convolutions may be replaced with cyclic convolutions and the final result will be the same.

**Exercise 7-3:** Compute the cyclic convolution of  $\{1, 2\}$  and  $\{3, 4\}$  for  $N = 2$ .

**Answer:**  $\{1, 2\} * \{3, 4\} = \{3, 10, 8\}$ . Aliasing the output gives  $\{8 + 3, 10\} = \{11, 10\}$ .

**Exercise 7-4:**  $x[n] \rightarrow \boxed{\uparrow 2} \rightarrow \boxed{\downarrow 2} \rightarrow y[n]$ . Express  $y[n]$  in terms of  $x[n]$ .

**Answer:**  $y[n] = x[n]$ . Zero stuffing inserts zeros between consecutive values of  $x[n]$  and decimation removes those zeros, thereby restoring the original  $x[n]$ .

**Exercise 7-5:** If  $y[n] = h[n] * x[n]$ , what is

$$h[n](-1)^n * x[n](-1)^n$$

in terms of  $y[n]$ ?

**Answer:**  $y[n](-1)^n$ . See Eq. (7.34).

## 7-4 Haar Wavelet Transform

The Haar transform is by far the simplest wavelet transform, and yet it illustrates many of the concepts of how the wavelet transform works.

### 7-4.1 Single-Stage Decomposition

Consider the finite-duration signal  $x[n]$

$$x[n] = \{\underline{a}, b, c, d, e, f, g, h\}. \quad (7.35)$$

Define the lowpass and highpass filters with impulse responses  $g_{\text{haar}}[n]$  and  $h_{\text{haar}}[n]$ , respectively, as

$$g_{\text{haar}}[n] = \frac{1}{\sqrt{2}} \{\underline{1}, 1\}, \quad (7.36a)$$

$$h_{\text{haar}}[n] = \frac{1}{\sqrt{2}} \{\underline{1}, -1\}. \quad (7.36b)$$

The frequency responses of these filters are the DTFTs given by

$$\mathbf{G}_{\text{haar}}(\Omega) = \frac{1}{\sqrt{2}} (1 + e^{-j\Omega}) = \sqrt{2} \cos\left(\frac{\Omega}{2}\right) e^{-j\Omega/2}, \quad (7.37a)$$

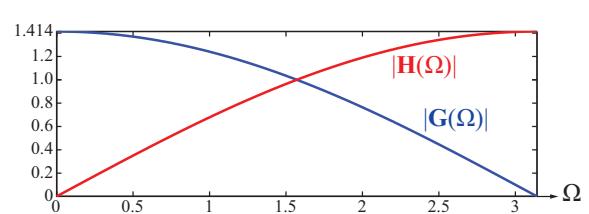
$$\mathbf{H}_{\text{haar}}(\Omega) = \frac{1}{\sqrt{2}} (1 - e^{-j\Omega}) = \sqrt{2} \sin\left(\frac{\Omega}{2}\right) j e^{-j\Omega/2}, \quad (7.37b)$$

which have lowpass and highpass frequency responses, respectively (Fig. 7-6).

Define the **average** (lowpass) signal  $x_L[n]$  as

$$x_L[n] = x[n] \odot g_{\text{haar}}[n] \quad (7.38a)$$

$$= \frac{1}{\sqrt{2}} \{\underline{a+h}, b+a, c+b, d+c, e+d, \dots\}$$



**Figure 7-6**  $|\mathbf{G}(\Omega)|$  (in blue) and  $|\mathbf{H}(\Omega)|$  (in red) for the Haar wavelet transform. This (quadrature-mirror filters) QMF pair has symmetry about the  $\Omega = \pi/2$  axis.

and the **detail** (highpass) signal  $x_H[n]$

$$\begin{aligned} x_H[n] &= x[n] \odot h_{\text{haar}}[n] \\ &= \frac{1}{\sqrt{2}} \{a-h, b-a, c-b, d-c, e-d, \dots\}. \end{aligned} \quad (7.38b)$$

Next, define the **downsampled average signal**  $x_{LD}[n]$  as

$$x_{LD}[n] = x_L[2n] = \frac{1}{\sqrt{2}} \{a+h, c+b, e+d, g+f\} \quad (7.39a)$$

and the **downsampled detail signal**  $x_{HD}[n]$  as

$$x_{HD}[n] = x_H[2n] = \frac{1}{\sqrt{2}} \{a-h, c-b, e-d, g-f\}. \quad (7.39b)$$

The signal  $x[n]$  of duration 8 has been replaced by the two signals  $x_{LD}[n]$  and  $x_{HD}[n]$ , each of durations 4, so no information about  $x[n]$  has been lost. We use cyclic convolutions instead of linear convolutions so that the cumulative length of the downsampled signals equals the length of the original signal. Using linear convolutions, each convolution with  $g_{\text{haar}}[n]$  or  $h_{\text{haar}}[n]$  would lengthen the signal unnecessarily. As we shall see, using cyclic convolutions instead of linear convolutions is sufficient to recover the original signal from its Haar wavelet transform.

## 7-4.2 Single-Stage Reconstruction

The single-stage Haar decomposition and reconstruction processes are depicted in Fig. 7-7. The signal  $x[n]$  can be reconstructed from  $x_{LD}[n]$  and  $x_{HD}[n]$  as follows:

**1.** Define the upsampled (zero-stuffed) signal  $x_{LDU}[n]$  as

$$\begin{aligned} x_{LDU}[n] &= \begin{cases} x_{LD}[n/2] & \text{for } n \text{ even} \\ 0 & \text{for } n \text{ odd} \end{cases} \\ &= \frac{1}{\sqrt{2}} \{a+h, 0, c+b, 0, e+d, 0, g+f, 0\}, \end{aligned} \quad (7.40a)$$

and the upsampled (zero-stuffed) signal  $x_{HDU}[n]$  as

$$\begin{aligned} x_{HDU}[n] &= \begin{cases} x_{HD}[n/2] & \text{for } n \text{ even} \\ 0 & \text{for } n \text{ odd} \end{cases} \\ &= \frac{1}{\sqrt{2}} \{a-h, 0, c-b, 0, e-d, 0, g-f, 0\}. \end{aligned} \quad (7.40b)$$

Note that downsampling by 2 followed by upsampling by 2 replaces values of  $x[n]$  with zeros for odd times  $n$ .

**2.** Next, filter  $x_{LDU}[n]$  and  $x_{HDU}[n]$  with filters  $g_{\text{haar}}[-n]$  and  $h_{\text{haar}}[-n]$ , respectively.

As noted earlier in Section 7-1.3, the term “filter” in the context of the wavelet transform means “cyclic convolution.” Filters  $g_{\text{haar}}[n]$  and  $h_{\text{haar}}[n]$  are called **analysis filters**, because they are used to compute the Haar wavelet transform. Their time reversals  $g_{\text{haar}}[-n]$  and  $h_{\text{haar}}[-n]$  are called **synthesis filters**, because they are used to compute the inverse Haar wavelet transform (that is, to reconstruct the signal from its Haar wavelet transform). The reason for using time reversals here is explained below.

The cyclic convolutions of  $x_{LDU}[n]$  with  $g_{\text{haar}}[-n]$  and  $x_{HDU}[n]$  with  $h_{\text{haar}}[-n]$  yield

$$x_{LDU}[n] \odot g_{\text{haar}}[-n] = \frac{1}{2} \{a+h, c+b, c+b, \dots, a+h\}, \quad (7.41a)$$

$$x_{HDU}[n] \odot h_{\text{haar}}[-n] = \frac{1}{2} \{a-h, b-c, c-b, \dots, h-a\}. \quad (7.41b)$$

**3.** Adding the outcomes of the two cyclic convolutions gives  $x[n]$ :

$$x[n] = x_{LDU}[n] \odot g_{\text{haar}}[-n] + x_{HDU}[n] \odot h_{\text{haar}}[-n]. \quad (7.42)$$

It is still not evident why this is worth doing. The following example provides a partial answer.

Consider the finite-duration ( $N = 16$ ) signal  $x[n]$ :

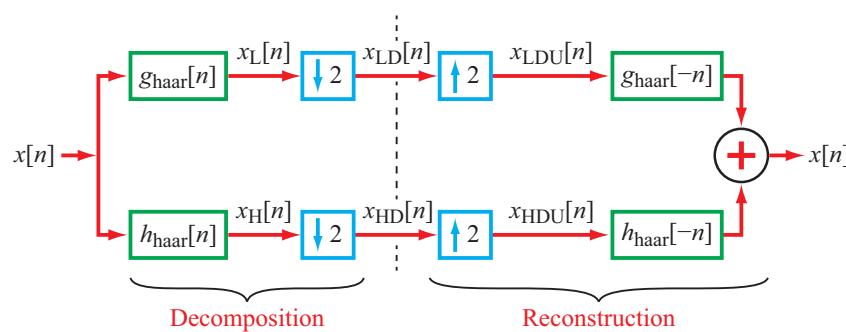
$$x[n] = \begin{cases} 4 & \text{for } 0 \leq n \leq 4 \\ 1 & \text{for } 5 \leq n \leq 9 \\ 3 & \text{for } 10 \leq n \leq 14 \\ 4 & \text{for } n = 15. \end{cases} \quad (7.43)$$

The Haar-transformed signals are

$$\begin{aligned} x_{LD}[n] &= x_L[2n] = \frac{1}{\sqrt{2}} \{8, 8, 8, 2, 2, 4, 6, 6\}, \\ x_{HD}[n] &= x_H[2n] = \frac{1}{\sqrt{2}} \{0, 0, 0, 0, 0, 2, 0, 0\}. \end{aligned} \quad (7.44)$$

These can be derived as follows. We have

$$x[n] = \{4, 4, 4, 4, 4, 1, 1, 1, 1, 3, 3, 3, 3, 3, 4\},$$



**Figure 7-7** Single-stage Haar decomposition and reconstruction of  $x[n]$ .

$$\begin{aligned}x_L[n] &= x[n] \odot \frac{1}{\sqrt{2}} \{1, 1\} \\&= \frac{1}{\sqrt{2}} \{8, 8, 8, 8, 8, 5, 2, 2, 2, 2, 4, 6, 6, 6, 6, 7\},\end{aligned}$$

$$\begin{aligned}x_H[n] &= x[n] \odot \frac{1}{\sqrt{2}} \{1, -1\} \\&= \frac{1}{\sqrt{2}} \{0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1\},\end{aligned}$$

$$x_{LD}[n] = x_L[2n] = \frac{1}{\sqrt{2}} \{8, 8, 8, 2, 2, 4, 6, 6\},$$

$$x_{HD}[n] = x_H[2n] = \frac{1}{\sqrt{2}} \{0, 0, 0, 0, 0, 2, 0, 0\}.$$

The original signal  $x[n]$  can be recovered from  $x_{LD}[n]$  and  $x_{HD}[n]$  by

$$x_{LDU}[n] = \frac{1}{\sqrt{2}} \{8, 0, 8, 0, 8, 0, 2, 0, 2, 0, 4, 0, 6, 0, 6, 0\},$$

$$x_{HDU}[n] = \frac{1}{\sqrt{2}} \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0\},$$

$$\begin{aligned}x_{LDU}[n] \odot \frac{1}{\sqrt{2}} \{1, 1\} \\&= \{4, 4, 4, 4, 4, 1, 1, 1, 1, 2, 2, 3, 3, 3, 3, 4\},\end{aligned}$$

$$\begin{aligned}x_{HDU}[n] \odot \frac{1}{\sqrt{2}} \{-1, 1\} \\&= \{0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, 0, 0\},\end{aligned}$$

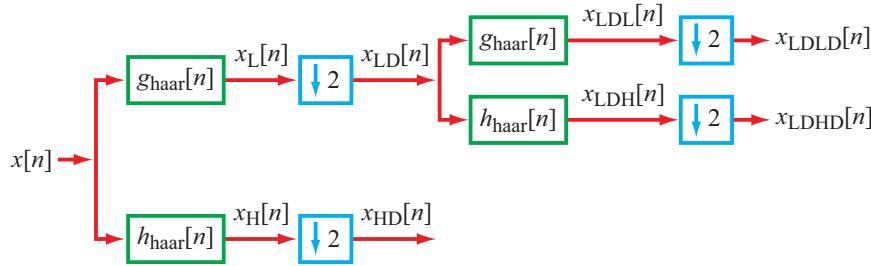
$$\begin{aligned}x_{LDU}[n] \odot \frac{1}{\sqrt{2}} \{1, 1\} + x_{HDU}[n] \odot \frac{1}{\sqrt{2}} \{-1, 1\} \\&= \{4, 4, 4, 4, 4, 1, 1, 1, 1, 3, 3, 3, 3, 3, 4\} = x[n].\end{aligned}$$

We observe that the outcome of the second cyclic convolution is **sparse** (mostly zero-valued). The Haar transform allows  $x[n]$ , which has duration 16, to be represented using the eight values of  $x_{LD}[n]$  and the single nonzero value (and its location  $n = 5$ ) of  $x_{HD}[n]$ . This saves almost half of the storage required for  $x[n]$ . Hence,  $x[n]$  has been **compressed** by 43%.

Even though  $x[n]$  is not sparse, it was transformed, using the Haar transform, into a sparse representation with the same number of samples, meaning that most of the values of the Haar-transformed signal are zero-valued. This reduces the amount of memory required to store  $x[n]$ , because only the times at which nonzero values occur (as well as the values themselves) need be stored. The few bits (0 or 1) required to store locations of nonzero values are considered to be negligible in number compared with the many bits required to store the actual nonzero values. Since the Haar transform is orthogonal,  $x[n]$  can be recovered perfectly from its Haar-transformed values.

### 7-4.3 Multistage Decomposition and Reconstruction

In the simple example used in the preceding subsection, only 1 element of the Haar-transformed signal  $x_{HD}[n]$  is nonzero, but all 8 elements of  $x_{LD}[n]$  are nonzero. We can reduce the number of nonzero elements of  $x_{LD}[n]$  by applying a second Haar transform stage to it. That is,  $x_{LD}[n]$  can be transformed into the two signals  $x_{LDLD}[n]$  and  $x_{LDHD}[n]$  by applying the steps outlined in **Fig. 7-8**.



**Figure 7-8** Two-stage Haar analysis filter bank. Note that only the upper half of the first stage is decomposed further.

Thus,

$$\begin{aligned} x_{LD}[n] &= \frac{1}{\sqrt{2}} \{8, 8, 8, 2, 2, 4, 6, 6\}, \\ x_{LDL}[n] &= x_{LD}[n] \odot \frac{1}{\sqrt{2}} \{1, 1\} \\ &= \{7, 8, 8, 5, 2, 3, 5, 6\}, \\ x_{LDH}[n] &= x_{LD}[n] \odot \frac{1}{\sqrt{2}} \{1, -1\} \\ &= \{1, 0, 0, -3, 0, 1, 1, 0\}, \\ x_{LDLD}[n] &= x_{LDL}[2n] = \{7, 8, 2, 5\}, \\ x_{LDHD}[n] &= x_{LDH}[2n] = \{1, 0, 0, 1\}. \end{aligned} \quad (7.45)$$

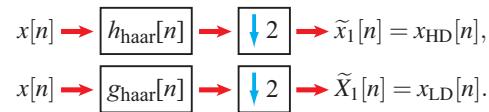
Signal  $x_{LDHD}[n]$  is again sparse: only two of its four values are nonzero. So  $x[n]$  can now be represented by the four values of  $x_{LDLD}[n]$ , the two nonzero values of  $x_{LDHD}[n]$ , and the one nonzero value of  $x_{HD}[n]$ . This reduces the storage required for  $x[n]$  by 57%.

The average signal  $x_{LDLD}[n]$  can in turn be decomposed even further. The result is an **analysis filter bank** that computes the Haar wavelet transform of  $x[n]$ . This analysis filter bank consists of a series of sections like the left half of Fig. 7-7, connected as in Fig. 7-8, except that each average signal is decomposed further. The signals computed at the right end of this analysis filter bank constitute the Haar wavelet transform of  $x[n]$ . Reconstruction of  $x[n]$  is shown in Fig. 7-9.

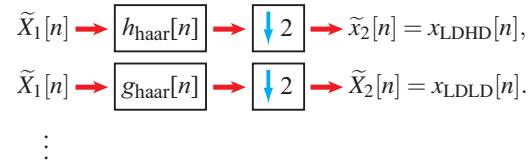
## Decomposition

A signal  $x[n]$  of duration  $N = 2^K$  (with  $K$  an integer) can be represented by the Haar wavelet transform through a  $K$ -stage decomposition process involving cyclic convolutions with filters  $g_{haar}[n]$  and  $h_{haar}[n]$ , as defined by Eq. (7.36). The signal  $x[n]$  can be zero-padded so that its length is a power of 2, if that is not already the case, just as is done for the FFT. The sequential process is:

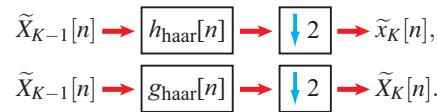
### Stage 1:



### Stage 2:



### Stage K:



## 7-4.4 Haar Wavelet Transform Filter Banks

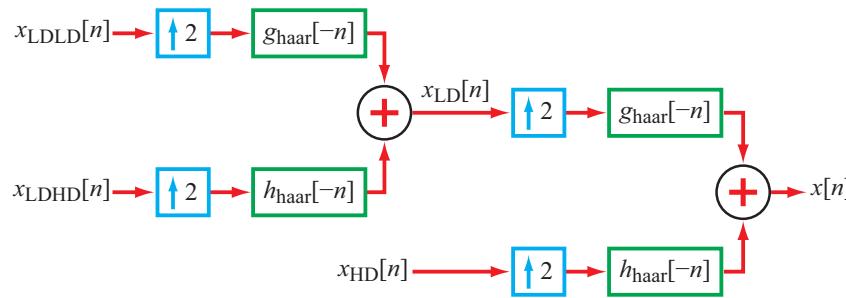


Figure 7-9 Reconstruction by a two-stage Haar synthesis filter bank.

The Haar transform of  $x[n]$  consists of the combination of  $K + 1$  signals:

$$\text{Duration } \rightarrow \underbrace{\{\tilde{x}_1[n], \tilde{x}_2[n], \tilde{x}_3[n], \dots, \tilde{x}_K[n], \tilde{X}_K[n]\}}_{N/2 \ N/4 \ N/8 \ N/2^K \ N/2^K}. \quad (7.46)$$

To represent  $x[n]$ , we need to retain the “high-frequency” outputs of all  $K$  stages (i.e.,  $\{\tilde{x}_1[n], \tilde{x}_2[n], \dots, \tilde{x}_K[n]\}$ ), but only the final output of the “low-frequency” sequence, namely  $\tilde{X}_K[n]$ . The total duration of all of the  $(K + 1)$  Haar transform signals is

$$\frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \dots + \frac{N}{2^K} + \frac{N}{2^K} = N, \quad (7.47)$$

which equals the duration  $N$  of  $x[n]$ . We use cyclic convolutions instead of linear convolutions so that the total lengths of the downsampled signals equals the length of the original signal. Were we to use linear convolutions, each convolution with  $g_{\text{haar}}[n]$  or  $h_{\text{haar}}[n]$  would lengthen the signal unnecessarily.

- The  $\tilde{x}_k[n]$  for  $k = 1, 2, \dots, K$  are called **detail** signals.
- The  $\tilde{X}_K[n]$  is called the **average** signal.

## Reconstruction

The inverse Haar wavelet transform can be computed in reverse order, starting with  $\{\tilde{X}_K[n], \tilde{x}_K[n], \tilde{x}_{K-1}[n], \dots\}$ .

### Stage 1:

$$\begin{aligned} \tilde{X}_K[n] &\xrightarrow{\text{upsample}} \tilde{x}_K[n] \xrightarrow{g_{\text{haar}}[-n]} A_{K-1}[n], \\ \tilde{x}_K[n] &\xrightarrow{\text{upsample}} \tilde{x}_K[n] \xrightarrow{h_{\text{haar}}[-n]} B_{K-1}[n], \\ \tilde{X}_{K-1}[n] &= A_{K-1}[n] + B_{K-1}[n]. \end{aligned}$$

### Stage 2:

$$\begin{aligned} \tilde{X}_{K-1}[n] &\xrightarrow{\text{upsample}} \tilde{x}_{K-1}[n] \xrightarrow{g_{\text{haar}}[-n]} A_{K-2}[n], \\ \tilde{x}_{K-1}[n] &\xrightarrow{\text{upsample}} \tilde{x}_{K-1}[n] \xrightarrow{h_{\text{haar}}[-n]} B_{K-2}[n], \\ \tilde{X}_{K-2}[n] &= A_{K-2}[n] + B_{K-2}[n]. \end{aligned}$$

⋮

### Stage K:

$$\begin{aligned} \tilde{X}_1[n] &\xrightarrow{\text{upsample}} \tilde{x}_1[n] \xrightarrow{g_{\text{haar}}[-n]} A_0[n], \\ \tilde{x}_1[n] &\xrightarrow{\text{upsample}} \tilde{x}_1[n] \xrightarrow{h_{\text{haar}}[-n]} B_0[n], \\ x[n] &= A_0[n] + B_0[n]. \end{aligned}$$

### 7-4.5 Haar Wavelet Transform in the Frequency Domain

As noted in Section 7-1, the original objective of subband coding was to decompose a signal into different frequency bands. Wavelets are more general than simple subbanding in that decomposition of the spectrum of the input into different bands is not the explicit purpose of the wavelet transform. Nevertheless, the decomposition does approximately allocate the signal into different frequency bands, so it is helpful to track and understand the approximate decomposition.

Recall from Eq. (7.37) that  $\mathbf{G}_{\text{haar}}(\Omega)$  is approximately a lowpass filter and  $\mathbf{H}_{\text{haar}}(\Omega)$  is approximately a highpass filter. Hence, at the output of the first stage of the Haar wavelet transform:

- $\tilde{X}_1[n]$  is the lowpass-frequency part of  $x[n]$ , covering the approximate range  $0 \leq |\Omega| \leq \pi/2$ , and
- $\tilde{x}_1[n]$  is the highpass-frequency part of  $x[n]$ , covering the approximate range  $\pi/2 \leq |\Omega| \leq \pi$ .

At each stage, downsampling expands the spectrum of each signal to the full range  $0 \leq |\Omega| \leq \pi$ . Hence, at the output of the second stage,

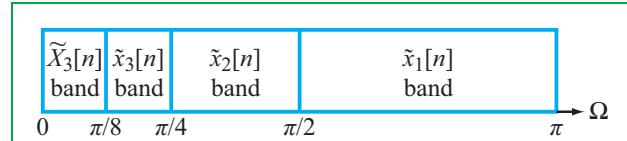
- $\tilde{X}_2[n]$  covers the frequency range  $0 \leq |\Omega| \leq \pi/2$  of  $\tilde{X}_1[n]$ , which corresponds to the range  $0 \leq |\Omega| \leq \pi/4$  of the spectrum of  $x[n]$ .
- $\tilde{x}_2[n]$  covers the frequency range  $\pi/2 \leq |\Omega| \leq \pi$  of  $\tilde{X}_1[n]$ , which corresponds to the range  $\pi/4 \leq |\Omega| \leq \pi/2$  of the spectrum of  $x[n]$ .

The Haar wavelet transform decomposes the spectrum of  $\mathbf{X}(\Omega)$  into **octaves**, with

$$\text{Octave } \frac{\pi}{2^k} \leq |\Omega| \leq \frac{\pi}{2^{k-1}} \text{ represented by } \tilde{x}_k[n].$$

Since the width of this band is  $\pi/2^k$ , the sampling rate can be reduced by a factor of  $2^k$  using downsampling.

For a signal of duration  $N = 2^K$ , the Haar wavelet transform decomposes  $x[n]$  into  $(K+1)$  components, namely  $\tilde{X}_K[n]$  and  $\{\tilde{x}_1[n], \tilde{x}_2[n], \dots, \tilde{x}_K[n]\}$ , with each component representing a frequency octave. The spectrum decomposition is illustrated in Fig. 7-10 for  $K = 3$ . Because the different components cover different octaves, they can be sampled at different rates. Furthermore, if the signal or image consists of slowly varying segments with occasional fast-varying features, as many real-world signals and images do, then  $\tilde{x}_k[n]$  for small  $k$  will be sparse, requiring few samples to represent it.



**Figure 7-10** Approximate frequency-band coverage by components of the Haar wavelet transform for  $K = 3$ .

**Concept Question 7-3:** Why do we use cyclic convolutions instead of linear convolutions?

**Exercise 7-6:** Compute the one-stage Haar transform of  $x[n] = \sqrt{2}\{4, 3, 3, 4\}$ .

**Answer:**

$$x[n] \odot \frac{1}{\sqrt{2}}[1, 1] = \{8, 7, 6, 7\}. \text{ Hence, } x_{LD}[n] = \{8, 6\}. \\ x[n] \odot \frac{1}{\sqrt{2}}[1, -1] = \{0, -1, 0, 1\}. \text{ Hence, } x_{HD}[n] = \{0, 0\}.$$

### 7-5 Discrete-Time Wavelet Transforms

The two Haar wavelet filters  $h_{\text{haar}}[n]$  and  $g_{\text{haar}}[n]$  introduced in the preceding section and defined by Eq. (7.36) are very simple in structure and short in duration, and yet they provide a powerful tool for decomposing signals (so they may be stored or transmitted more efficiently) and reconstructing them later. We now extend our presentation to the general case of wavelet transform filters  $h[n]$  and  $g[n]$ , which also use the octave-based filterbank **decomposition structure** shown in Fig. 7-2 and the octave-based filter-bank **reconstruction structure** shown in Fig. 7-4, but the main difference is that  $g[n]$  and  $h[n]$  are no longer explicitly lowpass and highpass filters, although their frequency responses are still *approximately* lowpass and highpass. Now  $h[n]$  is designed to sparsify (make mostly zero-valued) all of the octave-based filter bank outputs except the lowest frequency band output. Filters  $g[n]$  and  $h[n]$  are now called the **scaling** and **wavelet** functions, respectively.

In this and future sections of this chapter, we examine the structure, the properties, and some of the applications of the wavelet transform. We start by addressing the following two questions:

- (1) What conditions must  $g[n]$  and  $h[n]$  satisfy so that the output of the octave-based reconstruction filter bank is the same as the input to the octave-based decomposition filter

bank?

- (2) How should  $h[n]$  be chosen so that the outputs of all of the octave-based decomposition filter banks are sparse, except for the output of the lowest frequency band?

### 7-5.1 Conditions for Filter Bank Perfect Reconstruction

We now derive more general conditions on  $g[n]$  and  $h[n]$  so that the output of the octave-based reconstruction filter bank is identical to the input of the octave-based decomposition filter bank. If this occurs at each stage, then it will occur at all stages. Accordingly we consider the single-stage decomposition and reconstruction shown in [Fig. 7-11](#).

To determine the necessary conditions that  $g[n]$  and  $h[n]$  should satisfy in order for the input/output relationship in [Fig. 7-11](#) to be true, we replicate [Fig. 7-11](#) in equation form, but we also replace the combined downsampling/upsampling operations in the figure with the equivalent functional form given by Eq. (7.30):

$$\begin{aligned} & \left( (x[n] * g[n]) \left( \frac{1 + (-1)^n}{2} \right) \right) * g[-n] \\ & + \left( (x[n] * h[n]) \left( \frac{1 + (-1)^n}{2} \right) \right) * h[-n] = x[n]. \end{aligned} \quad (7.48)$$

Expanding Eq. (7.48) and using the convolution relation given by Eq. (7.34) gives

$$\begin{aligned} & x[n] * g[n] * g[-n] \\ & + \{(-1)^n x[n]\} * \{(-1)^n g[n]\} * g[-n] \\ & + x[n] * h[n] * h[-n] \\ & + \{(-1)^n x[n]\} * \{(-1)^n h[n]\} * h[-n] = 2x[n]. \end{aligned} \quad (7.49)$$

Collecting terms convolved with  $x[n]$  and  $(-1)^n x[n]$  separately leads to

$$\begin{aligned} & x[n] * \{g[n] * g[-n] + h[n] * h[-n]\} \\ & + (-1)^n x[n] * \{(-1)^n g[n] * g[-n]\} \\ & + (-1)^n h[n] * \{h[n] * h[-n]\} = 2x[n]. \end{aligned} \quad (7.50)$$

The expression given by Eq. (7.50) is satisfied for any input  $x[n]$  if and only if both of the following **perfect-reconstruction conditions** are satisfied:

(1)

$$g[n] * g[-n] + h[n] * h[-n] = 2\delta[n] \quad (7.51a)$$

and

(2)

$$(-1)^n g[n] * g[-n] + (-1)^n h[n] * h[-n] = 0. \quad (7.51b)$$

Next, we examine how to relate  $g[n]$  to  $h[n]$  so as to satisfy the two parts of Eq. (7.51).

### 7-5.2 Quadrature Mirror Filters

There are many possible solutions that satisfy Eq. (7.51). A sensible approach is to find a pair of filters  $g[n]$  and  $h[n]$  that automatically satisfy Eq. (7.51b), and then use them in Eq. (7.51a). One such pair is the **quadrature mirror filter (QMF)**, which is based on the **QMF relation** (where  $L$  is an odd integer):

$$g[n] = -(-1)^n h[L-n] = (-1)^{n-L} h[L-n]. \quad (7.52a)$$

Replacing  $n$  with  $-n$  implies:

$$g[-n] = -(-1)^{-n} h[L+n] = (-1)^{-(n+L)} h[L+n], \quad (7.52b)$$

where  $h[L-n]$  is  $h[-n]$  shifted in time by  $L$ , *with L chosen to be an odd integer and its length is such as to make g[n] causal*. Thus, if  $h[n]$  and  $g[n]$  are each of length  $N$ ,  $L$  should be equal to or greater than  $(N-1)$ .

Using Eq. (7.52), the first term in Eq. (7.51b) becomes

$$(-1)^n g[n] * g[-n] = (-1)^{2n-L} h[L-n] * (-1)^{-(n+L)} h[L+n].$$

In view of the time-shift property (#5 in [Table 2-6](#))

$$x_1[n-L] * x_2[n+L] = x_1[n] * x_2[n], \quad (7.53)$$

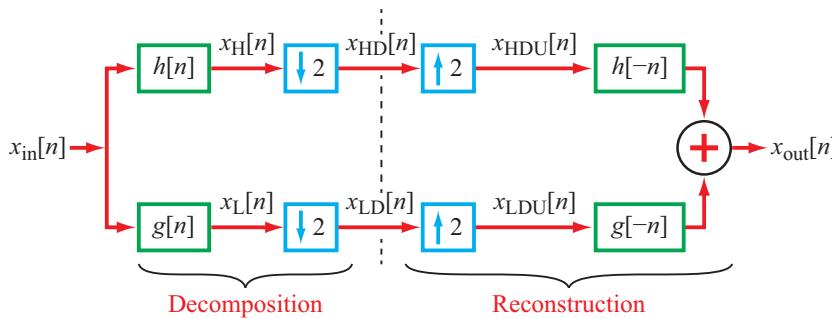
it follows that

$$(-1)^n g[n] * g[-n] = -(-1)^n h[n] * h[-n],$$

which satisfies the condition stated by Eq. (7.51b).

► We conclude that the QMF relations between  $g[n]$  and  $h[n]$ , as specified in Eq. (7.52), do indeed satisfy one of the two conditions required for perfect reconstruction. ◀

The significance of the odd-valued time shift  $L$  is best illustrated by a simple example.



**Figure 7-11** Single-stage decomposition of  $x[n]$  to  $x_{LD}[n]$  and  $x_{HD}[n]$ , and reconstruction of  $x[n]$  from  $x_{LD}[n]$  and  $x_{HD}[n]$ .

### Example 7-2: QMF

Show that for  $h[n] = \{1, 2, 3, 4\}$ , the QMF relations given by Eq. (7.52) satisfy the reconstruction condition given by Eq. (7.51b).

**Solution:** Since  $h[n]$  is of length 4, we select the time shift  $L$  to be 3. The second term in Eq. (7.51b) is

$$(-1)^n h[n] * h[-n] = \{1, -2, 3, -4\} * \{4, 3, 2, 1\} \quad (7.54a)$$

$$= \{4, -5, 8, -10, -8, -5, -4\}. \quad (7.54b)$$

For  $L = 3$  and the QMF relation defined by Eq. (7.52a),  $g[n]$  is

$$g[n] = -(-1)^n h[3-n],$$

which yields

$$g[0] = -h[3] = -4,$$

$$g[1] = h[2] = 3,$$

$$g[2] = -h[1] = -2,$$

$$g[3] = h[0] = 1.$$

Hence,

$$g[n] = \{-4, 3, -2, 1\}.$$

The first term of Eq. (7.51b) is then given by

$$(-1)^n g[n] * g[-n] = \{-4, -3, -2, -1\} * \{1, -2, 3, -4\} \quad (7.54c)$$

$$= \{4, -5, 8, -10, -8, -5, -4\}. \quad (7.54d)$$

The two sequences in Eqs. (7.54b) and (7.54d) are identical in value, but opposite in sign. Hence, their sum adds up to zero.

### 7-5.3 Smith-Barnwell Condition

We now turn our attention to the first of the two perfect-reconstruction conditions, namely the condition defined by Eq. (7.51a), and we do so in concert with the QMF relations given by Eq. (7.52).

Using the QMF relations, the first term in Eq. (7.51a) becomes

$$\begin{aligned} g[n] * g[-n] &= \{(-1)^{n-L} h[-(n-L)]\} * \{(-1)^{-(n+L)} h[n+L]\} \\ &= \{(-1)^n h[-n]\} * \{(-1)^n h[n]\}, \end{aligned} \quad (7.55)$$

where we used the time-shift property given by Eq. (7.53). Replacing the first term in Eq. (7.51a) with Eq. (7.55) and employing the commutativity property of convolution yields the **Smith-Barnwell condition for perfect reconstruction** using QMF filters:

$$\{(-1)^n h[n]\} * \{(-1)^n h[-n]\} + h[n] * h[-n] = 2\delta[n]. \quad (7.56)$$

Finally, upon taking advantage of the “useful convolution relation” encapsulated by Eq. (7.34), we obtain the result

$$(h[n] * h[-n])(-1)^n + h[n] * h[-n] = 2\delta[n]. \quad (7.57)$$

The usual form of the Smith-Barnwell condition uses the DTFT. Recall that the DTFT maps convolutions to products (entry #6 of **Table 2-7**) and modulation (entry #3 of **Table 2-7**) to frequency

shift:

$$e^{j\Omega_0 n} h[n] \leftrightarrow \mathbf{H}(\Omega - \Omega_0). \quad (7.58)$$

Setting  $\Omega_0 = \pi$  and recognizing that  $e^{j\pi n} = (-1)^n$  gives

$$(-1)^n h[n] \leftrightarrow \mathbf{H}(\Omega - \pi). \quad (7.59)$$

Using the properties of the DTFT, the **DTFT of the Smith-Barnwell condition** given by Eq. (7.57) is

$$|\mathbf{H}(\Omega)|^2 + |\mathbf{H}(\Omega - \pi)|^2 = 2. \quad (7.60)$$

Recognizing that

$$1 + (-1)^n = \begin{cases} 0 & \text{for odd } n, \\ 2 & \text{for even } n, \end{cases} \quad (7.61)$$

Eq. (7.57) holds for odd  $n$  for any  $h[n]$ , and for  $n$  even, it simplifies to

$$h[n] * h[-n] = \delta[n], \quad \text{for } n \text{ even.} \quad (7.62)$$

Writing out Eq. (7.62) for  $n$  even gives

$$n = 0 \quad \sum_{i=0}^L h^2[i] = 1, \quad (7.63a)$$

$$n = 2 \quad \sum_{i=2}^L h[i] h[i-2] = 0, \quad (7.63b)$$

$$n = 4 \quad \sum_{i=4}^L h[i] h[i-4] = 0, \quad (7.63c)$$

$\vdots$

$$n = L-1 \quad \sum_{i=L-1}^L h[i] h[i-(L-1)] = 0. \quad (7.63d)$$

Recall that since  $L$  is odd,  $L-1$  is even.

► The Smith-Barnwell condition is equivalent to stating that the autocorrelation  $r_h[n] = h[n] * h[-n]$  of  $h[n]$  is zero for even, nonzero,  $n$  and 1 for  $n = 0$ . This means that  $h[n]$  is orthonormal to even-valued translations of itself. ◀

#### 7-5.4 Wavelets as Basis Functions

It is easy to show that if  $h[n]$  satisfies the Smith-Barnwell condition given by Eq. (7.57), then  $g[n]$ , as defined by the QMF

relation in Eq. (7.52a), does also. Consequently,  $g[n]$  also is orthonormal to even translations of itself. Using Eq. (7.52a), it can be shown that  $g[n]$  and  $h[n]$  are also orthogonal to even translations of each other.

So the wavelet transform constitutes an **orthonormal expansion** of  $x[n]$  as

$$x[n] = \sum_{i=-\infty}^{\infty} g[2i-n] \tilde{X}_1[i] + \sum_{i=-\infty}^{\infty} h[2i-n] \tilde{x}_1[i], \quad (7.64)$$

where **average** signal  $\tilde{X}_1[n]$  and **detail** signal  $\tilde{x}_1[n]$  are defined as

$$\tilde{X}_1[n] = \sum_{i=-\infty}^{\infty} g[2n-i] x[i] \quad (7.65a)$$

and

$$\tilde{x}_1[n] = \sum_{i=-\infty}^{\infty} h[2n-i] x[i]. \quad (7.65b)$$

Computation of  $\tilde{X}_1[n]$  and  $\tilde{x}_1[n]$  in Eq. (7.65) is implemented using the wavelet analysis filter bank shown in Fig. 7-12(a). Computation of  $x[n]$  from  $\tilde{X}_1[n]$  and  $\tilde{x}_1[n]$  in Eq. (7.64) is implemented using the wavelet synthesis filter bank shown in Fig. 7-12(b).

The decimation in Fig. 7-12(a) manifests itself in Eq. (7.65) as the  $2n$  in  $g[2n-i]$  and  $h[2n-i]$ . The zero-stuffing and time reversals in Fig. 7-12(b) manifest themselves in Eq. (7.64) as the  $2i$  in  $g[2i-n]$  and  $h[2i-n]$ .

The average signal  $\tilde{X}_1[n]$  can in turn be decomposed similarly, as in the wavelet analysis filter bank shown in Fig. 7-12(a). So the first term in Eq. (7.64) can be decomposed further, resulting in the **K-stage decomposition**

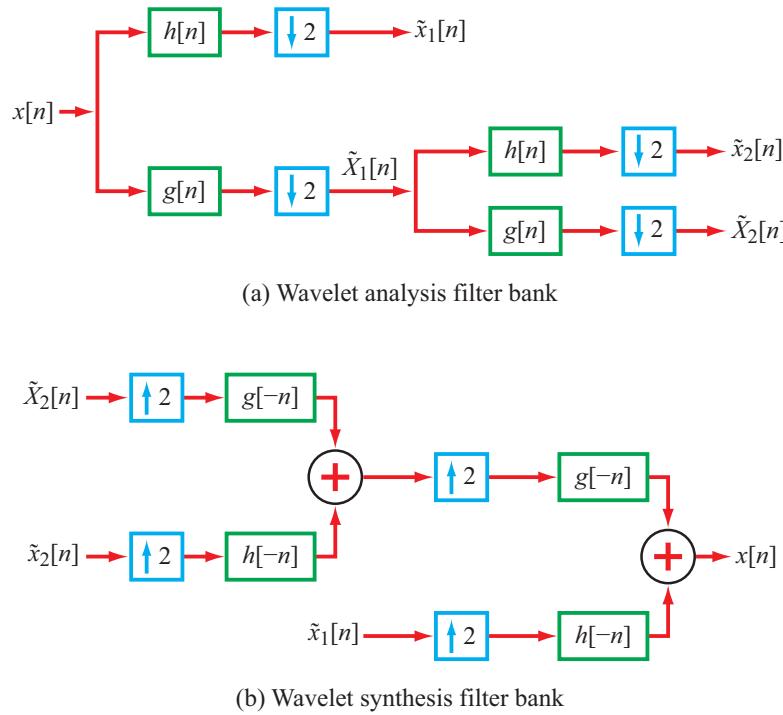
$$x[n] = \sum_{i=-\infty}^{\infty} \tilde{X}_K[i] g^{(K)}[2^K i - n] + \sum_{k=1}^K \sum_{i=-\infty}^{\infty} \tilde{x}_k[i] h^{(k)}[2^k i - n], \quad (7.66)$$

where average signal  $\tilde{X}_K[n]$  and detail signal  $\{\tilde{x}_k[n], k = 1, \dots, K\}$  are computed using

$$\tilde{X}_K[n] = \sum_{i=-\infty}^{\infty} g^{(K)}[2^K n - i] x[i]$$

and

$$\tilde{x}_k[n] = \sum_{i=-\infty}^{\infty} h^{(k)}[2^k n - i] x[i], \quad (7.67)$$



**Figure 7-12** (a) Wavelet analysis filter bank and (b) wavelet synthesis filter bank.

and the basis functions  $g^{(k)}[n]$  and  $h^{(k)}[n]$  can be computed recursively offline. We will not bother with explicit formulae for these, since the decomposition and reconstruction are performed much more easily using filter banks. The wavelet analysis and synthesis filter banks shown respectively in **Fig. 7-12(a)** and **(b)** are identical in form to the octave-band filter banks shown in **Figs. 7-2** and **7-4**, except for the following nomenclature changes:

$$\begin{aligned}\tilde{x}_1[n] &= x_{\text{HD}}[n], \\ \tilde{x}_2[n] &= x_{\text{LDHD}}[n], \\ \tilde{X}_1[n] &= x_{\text{LD}}[n], \\ \tilde{X}_2[n] &= x_{\text{LDLD}}[n].\end{aligned}\quad (7.68)$$

If  $x[n]$  has duration  $N = 2^K$ , the components of the wavelet

transform of  $x[n]$  have durations

$$\underbrace{\{\tilde{x}_1[n], \tilde{x}_2[n], \tilde{x}_3[n], \dots, \tilde{x}_K[n]\}}_{N/2} \underbrace{\tilde{X}_K[n]}_{N/2^K}. \quad (7.69)$$

### 7-5.5 Amount of Computation

The total duration of all of the wavelet transform signals together is

$$\frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \dots + \frac{N}{2^K} + \frac{N}{2^K} = N, \quad (7.70)$$

which equals the duration  $N$  of  $x[n]$ .

Again, we use cyclic convolutions instead of linear convolutions so that the total lengths of the decimated signals equals the length of the original signal.

The total amount of computation required to compute the wavelet transform of a signal  $x[n]$  of duration  $N$  can be computed as follows. Let the durations of  $g[n]$  and  $h[n]$  be the even integer

$(L+1)$  where  $L$  is the odd integer in Eq. (7.52). Convolving  $x[n]$  with both  $g[n]$  and  $h[n]$  requires  $2(2(L+1))N = 4(L+1)N$  multiplications-and-additions (MADs). But since the results will be decimated by two, only half of the convolution outputs must be computed, halving this to  $2(L+1)N$ .

At each successive decomposition,  $g[n]$  and  $h[n]$  are convolved with the average signal from the previous stage, and the result is decimated by two. So  $g[n]$  and  $h[n]$  are each convolved with the signals

$$\underbrace{\{x[n], \tilde{X}_1[n], \tilde{X}_2[n], \dots, \tilde{X}_K[n]\}}_{N \quad N/2 \quad N/4 \quad N/2^K}.$$

The total number of MADs required is thus

$$2(L+1) \left( N + \frac{N}{2} + \frac{N}{4} + \dots + \frac{N}{2^K} \right) < 4(L+1)N. \quad (7.71)$$

The additional computation for computing more decompositions (i.e., increasing  $K$ ) is thus minimal.

Since  $L$  is small (usually  $1 \leq L \leq 5$ ), this is comparable to the amount of computation  $\frac{N}{2} \log_2(N)$  required to compute the DFT, using the FFT, of a signal  $x[n]$  of duration  $N$ . But the DFT requires complex-valued multiplications and additions, while the wavelet transform uses only real-valued multiplications and additions.

**Concept Question 7-4:** What is the Smith-Barnwell condition?

**Exercise 7-7:** If  $h[n] = \frac{1}{\sqrt{2}}\{6, 2, h[2], 3\}$ , find  $h[2]$  so that  $h[n]$  satisfies the Smith-Barnwell condition.

**Answer:** According to Eq. (7.63), the Smith-Barnwell condition requires the autocorrelation of  $h[n]$  to be 1 for  $n = 0$  and to be 0 for even  $n \neq 0$ . For  $h[n] = \{a, b, c, d\}$ , these conditions give  $a^2 + b^2 + c^2 + d^2 = 1$ ,  $ac + bd = 0$ , and  $h[2] = -1$ .

## 7-6 Sparsification Using Wavelets of Piecewise-Polynomial Signals

The wavelet filters  $g[n]$  and  $h[n]$  are respectively called **scaling** and **wavelet** functions. In this section, we show how to design these functions such that they satisfy the Smith-Barnwell condition for perfect reconstruction given by Eq. (7.56), form a QMF pair, and have the property that the detail signals  $\tilde{x}_k[n]$  defined

in Eq. (7.67) and computed using the analysis filter bank shown in Fig. 7-12(a) are sparse (mostly zero-valued). In particular, we design  $g[n]$  and  $h[n]$  so that the wavelet transform detail signals are sparse when the input signal  $x[n]$  is **piecewise polynomial**, which we define next. The  $g[n]$  and  $h[n]$  filters are then used in the Daubechies wavelet transform.

### 7-6.1 Definition of Piecewise-Polynomial Signals

A signal  $x[n]$  is defined to be **piecewise-Mth-degree polynomial** if it has the form

$$x[n] = \begin{cases} \sum_{k=0}^M a_{0,k} n^k & \text{for } -\infty < n \leq N_0, \\ \sum_{k=0}^M a_{1,k} n^k & \text{for } N_0 < n \leq N_1, \\ \sum_{k=0}^M a_{2,k} n^k & \text{for } N_1 < n \leq N_2, \\ \vdots \end{cases} \quad (7.72)$$

This  $x[n]$  can be segmented into intervals, and in each interval  $x[n]$  is a polynomial in time  $n$  of degree  $M$ . The times  $N_i$  at which the coefficients  $\{a_{i,k}\}$  change values are sparse, meaning that they are scattered over time  $n$ . In continuous time, such a signal would be a spline (see Section 4-7), except that in the case of a spline, the derivatives of the signal must match at the knots (the times where coefficients  $\{a_{i,k}\}$  change values). The idea here is that the coefficients  $\{a_{i,k}\}$  can change completely at the times  $N_i$ ; there is no “smoothness” requirement. Indeed, these times  $N_i$  constitute the edges of  $x[n]$ .

#### A. Piecewise-Constant Signals

First, let  $x[n]$  be piecewise constant ( $M = 0$  in Eq. (7.72)), so that  $x[n]$  is of the form

$$x[n] = \begin{cases} a_0 & \text{for } -\infty < n \leq N_0, \\ a_1 & \text{for } N_0 < n \leq N_1, \\ a_2 & \text{for } N_1 < n \leq N_2, \\ \vdots \end{cases} \quad (7.73)$$

The value of  $x[n]$  changes only at a few scattered times. The amount by which  $x[n]$  changes at  $n = N_i$  is the jump  $a_{i+1} - a_i$ :

$$x[n+1] - x[n] = \begin{cases} 0 & \text{for } n \neq N_i, \\ a_{i+1} - a_i & \text{for } n = N_i. \end{cases} \quad (7.74)$$

This can be restated as

$$x[n+1] - x[n] = \sum_i (a_{i+1} - a_i) \delta[n - N_i]. \quad (7.75)$$

Taking differences sparsifies a piecewise-constant signal, as illustrated in **Fig. 7-13**.

Now let the wavelet function  $h[n]$  have the form, for some signal  $q[n]$  that is yet to be determined,

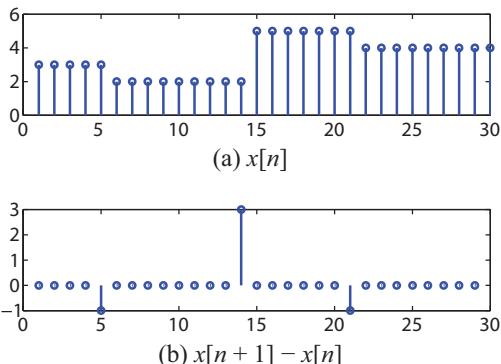
$$h[n] = q[n] * \{1, -1\}. \quad (7.76)$$

Since from **Fig. 2-3** the overall impulse response of two systems connected in series is the convolution of their impulse responses,  $h[n]$  can be implemented by two systems connected in series:



Convolution with  $h[n]$  sparsifies a piecewise-constant input  $x[n]$ , since (using the time-shift property of convolution)

$$\begin{aligned} x[n] * h[n] &= x[n] * \{1, -1\} * q[n] \\ &= (x[n+1] - x[n]) * q[n] \\ &= \sum_i (a_{i+1} - a_i) \delta[n - N_i] * q[n] \\ &= \sum_i (a_{i+1} - a_i) q[n - N_i]. \end{aligned} \quad (7.77)$$



**Figure 7-13** A piecewise-constant signal is compressed by taking differences. (a) a piecewise-constant signal, (b) differences of the signal.

In practice,  $q[n] = q[0] \delta[n]$  has duration = 1, so  $x[n] * h[n]$  is still mostly zero-valued.

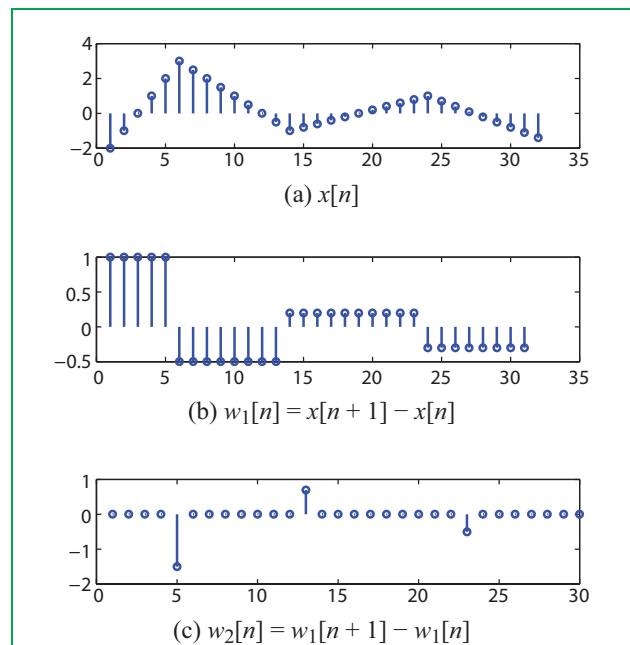
## B. Piecewise-Linear Signals

Next, let  $x[n]$  be piecewise linear ( $M = 1$  in Eq. (7.72)), so that  $x[n]$  is of the form

$$x[n] = \begin{cases} a_{0,1} n + a_{0,0}, & -\infty < n \leq N_0, \\ a_{1,1} n + a_{1,0}, & N_0 < n \leq N_1, \\ a_{2,1} n + a_{2,0}, & N_1 < n \leq N_2, \\ \vdots \end{cases} \quad (7.78)$$

Proceeding as in the case of piecewise-constant  $x[n]$ , taking differences, and then taking differences of the differences, will sparsify a piecewise-linear signal. The process is illustrated in **Fig. 7-14**.

The bottom signal in **Fig. 7-14** is in turn convolved with  $q[n]$ ,



**Figure 7-14** A piecewise-linear signal is compressed by taking successive differences. (a) A piecewise-linear signal, (b) differences of the top signal, (c) differences of the middle signal.

resulting in a series of scaled and delayed versions of  $q[n]$ . In practice,  $q[n]$  has length two, so  $x[n] * h[n]$  is still mostly zero-valued.

Now let the wavelet function  $h[n]$  have the form

$$h[n] = q[n] * \{1, -1\} * \{1, -1\}. \quad (7.79)$$

Convolution with  $h[n]$  sparsifies a piecewise-linear input  $x[n]$ , since

$$\begin{aligned} x[n] * h[n] &= x[n] * \{1, -1\} * \{1, -1\} * q[n] \\ &= (x[n+1] - x[n]) * \{1, -1\} * q[n] \\ &= (x[n+2] - 2x[n+1] + x[n]) * q[n]. \end{aligned} \quad (7.80)$$

The output consists of a set of scaled and delayed versions of  $q[n]$ . In practice,  $q[n]$  is very short (length two), so  $x[n] * h[n]$  is still mostly zero-valued.

## C. Piecewise-Polynomial Signals

Finally, let  $x[n]$  be piecewise polynomial, meaning that

$$x[n] = \begin{cases} \sum_{k=0}^M a_{0,k} n^k, & -\infty < n \leq N_0, \\ \sum_{k=0}^M a_{1,k} n^k, & N_0 < n \leq N_1, \\ \sum_{k=0}^M a_{2,k} n^k, & N_1 < n \leq N_2, \\ \vdots & \end{cases} \quad (7.81)$$

Taking  $(M+1)$  sets of successive differences will sparsify  $x[n]$ , since each difference will reduce the degrees of the polynomial segments by one. The continuous-time analogue of this is

$$y(t) = \frac{d^{M+1}x}{dt^{M+1}} = 0 \quad (7.82)$$

for any  $M$ th-degree polynomial  $x(t)$ .

Now let the wavelet function  $h[n]$  have the form

$$h[n] = q[n] * \underbrace{\{1, -1\} * \{1, -1\} \dots \{1, -1\}}_{M+1 \text{ differences}}. \quad (7.83)$$

The output consists of a set of scaled and delayed versions of  $q[n]$ . In practice,  $q[n]$  is still relatively short (length  $M+1$ ), so  $x[n] * h[n]$  is still mostly zero-valued.

**Table 7-2** Daubechies wavelet notation.

$K$	1	2	3
dbK	1	2	3
D(2K)	2	4	6
Number of differences	1	2	3
Duration = $2K$	2	4	6
Degree sparsified = $M$	0	1	2

## 7-6.2 Design of Wavelet Functions $h[n]$ of Desired Form

We now show how to design wavelet functions of the form given by Eq. (7.83). These wavelet functions compress piecewise- $M$ th-degree polynomials to sparse signals  $\tilde{x}[n]$ . Since many real-world signals can be so modeled, their wavelet transforms will always be sparse.

The functions  $h[n]$  given by Eq. (7.83) are called **Daubechies wavelet functions**. The order of Daubechies wavelet function has several different definitions. The Daubechies wavelet function that compresses piecewise- $(K-1)$ th-degree polynomials to sparse signals is termed “dbK Daubechies wavelet functions,” where “K” is a reference to the number of differences that need to be taken, “db” is short for Daubechies, and  $K = M + 1$ . Daubechies wavelet functions having duration  $N$  are termed “DN Daubechies wavelet functions.” We shall see that dbK Daubechies wavelet functions have duration  $2K$ , so a dbK Daubechies wavelet function is also a D( $2K$ ) Daubechies wavelet function. The DN notation is inefficient because  $N$  is always even. For that reason, we will use “dbK” to refer to the order of a Daubechies wavelet function. The Haar functions introduced in Section 7-4 are db1 and D2 wavelets. To confuse matters further, DN was used in the past to mean both DN and db( $N/2$ ). The notation is summarized in **Table 7-2**.

To make  $h[n]$  causal, we delay each of the differences in Eq. (7.83) to get

$$h[n] = q[n] * \underbrace{\{1, -1\} * \{1, -1\} * \dots * \{1, -1\}}_{M+1 \text{ differences}}. \quad (7.84)$$

A degrees-of-freedom analysis shows that  $q[n]$  must have duration  $M+1$ , so  $h[n]$  has duration  $(2M+2) = 2K$ . Function  $q[n]$  is determined by inserting the form Eq. (7.84) into the Smith-Barnwell condition given by Eq. (7.56). The scaling function  $g[n]$  is then determined by the QMF formula in Eq. (7.52a).

## A. Piecewise-Constant Signals

To compress piecewise-constant signals, we set  $M = 0$ ; a constant is a polynomial of degree zero. We wish to design a db1 = D2 order Daubechies wavelet, which has only 1 difference because  $M + 1 = 0 + 1 = 1$ , and a duration =  $2(1) = 2$ , so  $L = 1$ . The expression in Eq. (7.84) becomes

$$h[n] = (\underbrace{q[0] \delta[n]}_{\text{duration}=1}) * \underbrace{\{1, -1\}}_{\text{1 difference}} = \underbrace{\{q[0], -q[0]\}}_{\text{duration}=2}. \quad (7.85)$$

Inserting this result into the Smith-Barnwell condition of Eq. (7.63a) gives

$$q[0]^2 + q[0]^2 = 1. \quad (7.86)$$

The requirement given in Eq. (7.63b) is satisfied automatically, since  $h[n]$  has duration = 2. Hence,  $q[0] = \frac{1}{\sqrt{2}}$  and

$$h[n] = \frac{1}{\sqrt{2}} \{1, -1\}, \quad (7.87)$$

and the scaling function  $g[n]$  is, from Eq. (7.52a), given by

$$g[n] = -(-1)^n h[1-n] = \frac{1}{\sqrt{2}} \{1, 1\}. \quad (7.88)$$

We recognize this as the **Haar transform**. It compresses piecewise constant signals. Making the alternate choice in the solution of Eq. (7.86), namely  $q[0] = \frac{-1}{\sqrt{2}}$ , simply changes the signs of  $g[n]$  and  $h[n]$ .

## B. Piecewise-Linear Signals

To compress piecewise-linear signals, we set  $M = 1$ . We wish to design a db2 = D4 order Daubechies wavelet, which has  $M + 1 = 2$  differences and a duration =  $2(2) = 4$ . Function  $h[n]$  in Eq. (7.84) becomes a filter of duration 4:

$$\begin{aligned} h[n] &= \underbrace{\{q[0], q[1]\}}_{\text{duration}=2} * \underbrace{\{1, -1\}}_{M+1=2 \text{ differences}} * \{1, -1\} \\ &= \{q[0], q[1]\} * \{1, -2, 1\} \\ &= \{q[0], q[1] - 2q[0], q[0] - 2q[1], q[1]\}. \end{aligned} \quad (7.89)$$

The Smith-Barnwell condition for  $n = 0$  and  $n = 2$ , as stated in Eqs. (7.63b) and (7.63a), gives two equations:

$$0 = \sum_{n=2}^3 h[n] h[n-2], \quad (7.90a)$$

$$1 = \sum_{n=0}^3 h[n]^2. \quad (7.90b)$$

The first equation states that  $h[n]$  must be orthogonal to its even-valued translations, and the second equation simply states that the energy of  $h[n]$  must be one, which is a normalization requirement that should be imposed after  $q[0]$  and  $q[1]$  have been determined.

Substituting the elements of  $h[n]$  given in Eq. (7.89) into Eq. (7.90a) leads to

$$\begin{aligned} 0 &= (q[0] - 2q[1]) q[0] + (q[1] - 2q[0]) q[1] \\ &= q[0]^2 - 4q[0] q[1] + q[1]^2. \end{aligned} \quad (7.91)$$

Since the scale factor will be set by  $\sum h[n]^2 = 1$ , we can, without loss of generality, set  $q[0]=1$ . This gives

$$q[1]^2 - 4q[1] + 1 = 0, \quad (7.92)$$

which has the two roots  $q[1] = \{2 \pm \sqrt{3}\}$ . Using the smaller root,  $2 - \sqrt{3}$ , in Eq. (7.89) gives

$$h_1[n] = q_1[0] \{1, -\sqrt{3}, 2\sqrt{3} - 3, 2 - \sqrt{3}\}. \quad (7.93)$$

Finally, the energy normalization  $\sum h_1[n]^2 = 1$  requires  $q_1[0] = \pm 0.4830$ . Hence,

$$h_1[n] = \pm \{0.4830, -0.8365, 0.2241, 0.1294\}. \quad (7.94)$$

The length of  $h[n]$  is  $N = 2(M + 1) = 2K = 4$ . To satisfy Eq. (7.52), the time shift  $L$  should be an odd integer and equal to or greater than  $(N - 1)$ . Since  $N - 1 = 4 - 1 = 3$ , we set  $L = 3$ . Accordingly, the scaling function  $g_1[n]$  is, from Eq. (7.52a), given by

$$\begin{aligned} g_1[n] &= -(-1)^n h[3-n] \\ &= \pm \{0.1294, -0.2241, -0.8365, -0.4830\}. \end{aligned} \quad (7.95)$$

Using the larger root of Eq. (7.92), namely  $(2 + \sqrt{3})$ , in Eq. (7.89) gives

$$h_2[n] = q_2[0] \{1, \sqrt{3}, -2\sqrt{3} - 3, 2 + \sqrt{3}\}, \quad (7.96)$$

and the energy normalization  $\sum h[n]^2 = 1$  leads to

$$q_2[0] = \pm 0.1294.$$

Hence,

$$h_2[n] = \pm\{0.1294, 0.2241, -0.8365, 0.4830\}, \quad (7.97)$$

and the scaling function  $g[n]$  is then

$$\begin{aligned} g_2[n] &= -(-1)^n h[3-n] \\ &= \pm\{0.4830, 0.8365, 0.2241, -0.1294\}. \end{aligned} \quad (7.98)$$

The two choices of  $h[n]$  are time reversals of each other, and so are the choices of  $g[n]$ . The signs of  $g[n]$  and  $h[n]$  can be changed due to the (plus/minus) signs in Eqs. (7.94), (7.95), (7.97), and (7.98).

These  $g[n]$  and  $h[n]$  functions are called the Daubechies db2 or D4 scaling and wavelet functions. The Daubechies db3 or D6 wavelet transform sparsifies piecewise-quadratic signals, as there are three differences ( $M = 2$  and, hence,  $K = 3$ ) in  $h[n]$ .

- The Daubechies scaling functions  $g[n]$ , with  $g[0] > 0$ , are listed in **Table 7-3** for various values of  $K$ . ◀

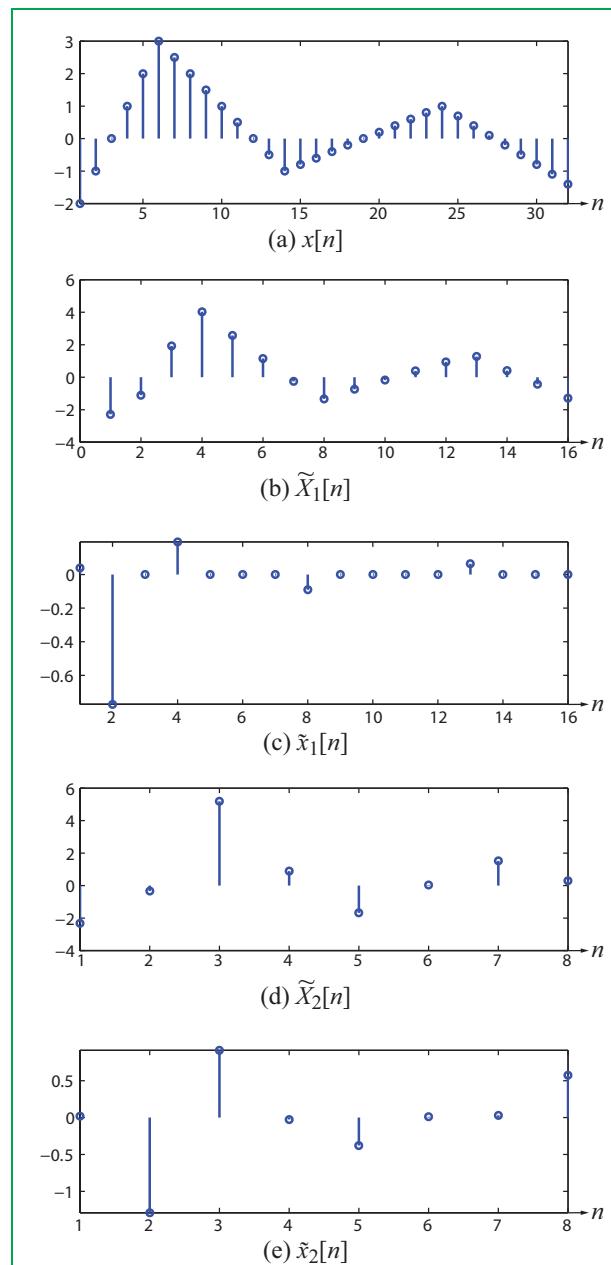
**Table 7-3** Daubechies scaling functions.

	$K = 1$	$K = 2$	$K = 3$	$K = 4$
$g[n]$	db1	db2	db3	db4
$g[0]$	.7071	.4830	.3327	.2304
$g[1]$	.7071	.8365	.8069	.7148
$g[2]$	0	.2241	.4599	.6309
$g[3]$	0	-.1294	-.1350	-.0280
$g[4]$	0	0	-.0854	-.1870
$g[5]$	0	0	.0352	.0308
$g[6]$	0	0	0	.0329
$g[7]$	0	0	0	-.0106

### Example 7-3: db2 Wavelet Transform

Compute the db2 (or equivalently, D4) Daubechies wavelet transform of the piecewise-linear signal  $x[n]$  shown in **Fig. 7-15(a)**.

**Solution:** The two-stage procedure involves the computation of average signals  $\tilde{X}_1[n]$  and  $\tilde{X}_2[n]$ , and detail signals  $\tilde{x}_1[n]$  and



**Figure 7-15** (a) Piecewise linear signal  $x[n]$ , (b) stage-1 average signal  $\tilde{X}_1[n]$ , (c) stage-1 detail signal  $\tilde{x}_1[n]$ , (d) stage-2 average signal  $\tilde{X}_2[n]$ , and (e) stage-2 detail signal  $\tilde{x}_2[n]$ .

$\tilde{x}_2[n]$ , using Eqs. (7.67), (7.94), and (7.95). The results are displayed in parts (b) to (e) of **Fig. 7-15**. We note that

- Average signals  $\tilde{X}_1[n]$  and  $\tilde{X}_2[n]$  are low-resolution versions of  $x[n]$ .
- Detail signals  $\tilde{x}_1[n]$  and  $\tilde{x}_2[n]$  are sparse (mostly zero), and their nonzero values are small in magnitude.
- The db2 wavelet transform of the given  $x[n]$  consists of  $\tilde{x}_1[n]$ ,  $\tilde{x}_2[n]$ , and  $\tilde{X}_2[n]$ .

These patterns explain the terms “average” and “detail.”

**Concept Question 7-5:** How is it possible that the wavelet transform requires less computation than the FFT?

**Exercise 7-8:** What are the finest-detail signals of the db3 wavelet transform of  $x[n] = 3n^2$ ?

**Answer:** Zero, because db3 wavelet basis function  $h[n]$  eliminates quadratic signals by construction.

**Exercise 7-9:** Show by direct computation that the db2 scaling function listed in **Table 7-3** satisfies the Smith-Barnwell condition.

**Answer:** The Smith-Barnwell condition requires the autocorrelation of  $g[n]$  to be 1 for  $n = 0$  and to be 0 for even  $n \neq 0$ . For  $g[n] = \{a, b, c, d\}$ , these conditions give  $a^2 + b^2 + c^2 + d^2 = 1$  and  $ac + bd = 0$ . The db2  $g[n]$  listed in **Table 7-3** is  $g[n] = \{.4830, .8365, .2241, -.1294\}$ . It is easily verified that the sum of the squares of these numbers is 1, and that  $(.4830) \times (.2241) + (.8365) \times (-.1294) = 0$ .

## 7-7 2-D Wavelet Transform

The real power of the wavelet transform becomes apparent when it is applied to 2-D images. A  $512 \times 512$  image has more than a quarter-million pixel values. Storing a sparse representation of an image, rather than the image itself, saves a huge amount of memory. Compressed sensing (covered later in Section 7-9) becomes very powerful when applied to images.

### 7-7.1 Decimation and Zero-Stuffing of Images

The concepts of downsampling (decimation) and upsampling (zero-stuffing) and interpolation generalize directly from 1-D

signals to 2-D images. Downsampling in 2-D involves downsampling in both directions. For example:

$$x[n, m] \rightarrow \boxed{\downarrow(3,2)} \rightarrow x[3n, 2m].$$

The downsampling factor in this case is 3 along the horizontal direction and 2 along the vertical direction. To illustrate the process, we apply it to a  $5 \times 7$  image:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 & 20 & 21 \\ 22 & 23 & 24 & 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 & 33 & 34 & 35 \end{bmatrix} \rightarrow \boxed{\downarrow(3,2)} \rightarrow \begin{bmatrix} 1 & 4 & 7 \\ 15 & 18 & 21 \\ 29 & 32 & 35 \end{bmatrix}.$$

Upsampling in 2-D involves upsampling in both directions. Upsampling a signal  $x[n, m]$  by a factor 3 along the horizontal and by a factor 2 along the vertical, for example, is denoted symbolically as

$$x[n, m] \rightarrow \boxed{\uparrow(3,2)} \rightarrow \begin{cases} x\left[\frac{n}{3}, \frac{m}{2}\right] & \text{for } n = \text{integer multiple of 3} \\ 0 & \text{and } m = \text{integer multiple of 2,} \\ & \text{otherwise.} \end{cases}$$

Applying this upsampling operation to a simple  $2 \times 2$  image yields

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \boxed{\uparrow(3,2)} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Downsampling and upsampling are used extensively in image processing operations.

### 7-7.2 Image Analysis Filter Bank

The generalization of the wavelet transform from signals to images is straightforward if **separable** scaling and wavelet functions are used. In this book we restrict our attention to separable functions; i.e., the 2-D filter  $g_2[n, m] = g[n] g[m]$ . With  $g[n]$  and  $h[n]$  denoting scaling and wavelet functions, such as the Haar or Daubechies functions, the image analysis filter bank performs the following operations:

### (1) Stage-1 decomposition

$$x[n, m] \rightarrow \begin{array}{|c|} \hline g[n] g[m] \\ \hline \end{array} \rightarrow \downarrow(2,2) \rightarrow \tilde{x}_{LL}^{(1)}[n, m] \quad (7.99a)$$

$$x[n, m] \rightarrow \begin{array}{|c|} \hline g[n] h[m] \\ \hline \end{array} \rightarrow \downarrow(2,2) \rightarrow \tilde{x}_{LH}^{(1)}[n, m] \quad (7.99b)$$

$$x[n, m] \rightarrow \begin{array}{|c|} \hline h[n] g[m] \\ \hline \end{array} \rightarrow \downarrow(2,2) \rightarrow \tilde{x}_{HL}^{(1)}[n, m] \quad (7.99c)$$

$$x[n, m] \rightarrow \begin{array}{|c|} \hline h[n] h[m] \\ \hline \end{array} \rightarrow \downarrow(2,2) \rightarrow \tilde{x}_{HH}^{(1)}[n, m] \quad (7.99d)$$

### (2) Stage-2 to stage- $K$ decomposition

$$\tilde{x}_{LL}^{(1)}[n, m] \rightarrow \begin{array}{|c|} \hline g[n] g[m] \\ \hline \end{array} \rightarrow \downarrow(2,2) \rightarrow \tilde{x}_{LL}^{(2)}[n, m]$$

(7.100a)

$$\tilde{x}_{LL}^{(1)}[n, m] \rightarrow \begin{array}{|c|} \hline g[n] h[m] \\ \hline \end{array} \rightarrow \downarrow(2,2) \rightarrow \tilde{x}_{LH}^{(2)}[n, m]$$

(7.100b)

$$\tilde{x}_{LL}^{(1)}[n, m] \rightarrow \begin{array}{|c|} \hline h[n] g[m] \\ \hline \end{array} \rightarrow \downarrow(2,2) \rightarrow \tilde{x}_{HL}^{(2)}[n, m]$$

(7.100c)

$$\tilde{x}_{LL}^{(1)}[n, m] \rightarrow \begin{array}{|c|} \hline h[n] h[m] \\ \hline \end{array} \rightarrow \downarrow(2,2) \rightarrow \tilde{x}_{HH}^{(2)}[n, m]$$

(7.100d)

The decomposition process is continued as above, until output  $\tilde{x}_{LL}^K[n, m]$  is reached, where  $K$  is the total number of stages.

### (3) Final wavelet transform

At the conclusion of the decomposition process,  $x[n, m]$  consists of:

(a) The coarsest **average image**  $\tilde{x}_{LL}^{(K)}[n, m]$ .

(b) Three **detail images** at each stage:

$$\{\tilde{x}_{LH}^{(K-1)}[n, m], \tilde{x}_{HL}^{(K-1)}[n, m], \tilde{x}_{HH}^{(K-1)}[n, m]\}$$

$$\{\tilde{x}_{LH}^{(K-2)}[n, m], \tilde{x}_{HL}^{(K-2)}[n, m], \tilde{x}_{HH}^{(K-2)}[n, m]\},$$

up to the largest (in size) three detail images:

$$\{\tilde{x}_{LH}^{(1)}[n, m], \tilde{x}_{HL}^{(1)}[n, m], \tilde{x}_{HH}^{(1)}[n, m]\}.$$

The average images  $\tilde{x}_{LL}^{(k)}[n, m]$  are analogous to the average signals  $\tilde{X}_k[m]$  of a signal, except that they are low-resolution versions of a 2-D image  $x[n, m]$  instead of a 1-D signal  $x[m]$ . In 2-D, there are now three detail images, while in 1-D there is only one detail signal.

In 1-D, the detail signals are zero except near edges, representing abrupt changes in the signal or in its slope. In 2-D, the three detail images play the following roles:

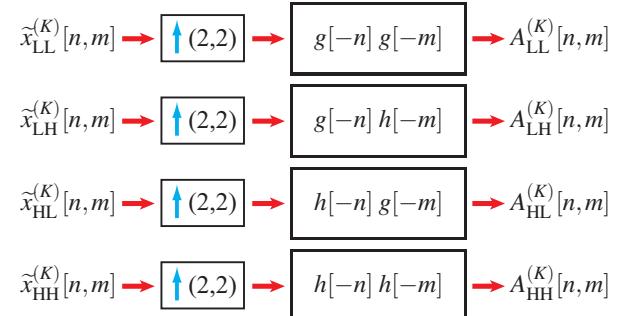
(a)  $\tilde{x}_{LH}^{(k)}[n, m]$  picks up vertical edges,

(b)  $\tilde{x}_{HL}^{(k)}[n, m]$  picks up horizontal edges,

(c)  $\tilde{x}_{HH}^{(k)}[n, m]$  picks up diagonal edges.

### 7-7.3 Image Synthesis Filter Bank

The image synthesis filter bank combines all of the detail images, and the coarsest average image,  $\tilde{x}_{LL}^{(K)}[n, m]$ , into the original image  $x[n, m]$ , as follows:



Average signal  $\tilde{x}_{LL}^{(K-1)}[n, m]$  is the sum of the above four outputs:

$$\begin{aligned} \tilde{x}_{LL}^{(K-1)}[n, m] = & A_{LL}^{(K)}[n, m] + A_{LH}^{(K)}[n, m] \\ & + A_{HL}^{(K)}[n, m] + A_{HH}^{(K)}[n, m]. \end{aligned} \quad (7.101)$$

Here  $\{A_{LL}^{(K)}[n, m], A_{LH}^{(K)}[n, m], A_{HL}^{(K)}[n, m], A_{HH}^{(K)}[n, m]\}$  are just four temporary quantities to be added. The analogy to signal analysis and synthesis filter banks is evident, except that at each stage there are three detail images instead of one detail signal.

Repetition of the reconstruction step represented by Eq. (7.101) through an additional  $K - 1$  stages leads to

$$x[n, m] = \tilde{x}_{LL}^{(0)}[n, m].$$

The condition for perfect reconstruction is the 2-D version of the Smith-Barnwell condition defined by Eq. (7.60), namely

$$\begin{aligned} & |\mathbf{H}_2(\Omega_1, \Omega_2)|^2 + |\mathbf{H}_2(\Omega_1 - \pi, \Omega_2 - \pi)|^2 \\ & + |\mathbf{H}_2(\Omega_1 - \pi, \Omega_2)|^2 + |\mathbf{H}_2(\Omega_1, \Omega_2 - \pi)|^2 = 4, \end{aligned} \quad (7.102)$$

where  $\mathbf{H}_2(\Omega_1, \Omega_2)$  is the DSFT of the 2-D wavelet function  $h_2[n, m] = h[n] h[m]$ . Since the 2-D wavelet function  $h_2[n, m]$  is separable, its DSFT is also separable:

$$\mathbf{H}_2(\Omega_1, \Omega_2) = \mathbf{H}(\Omega_1) \mathbf{H}(\Omega_2), \quad (7.103)$$

where  $\mathbf{H}(\Omega_1)$  is the DTFT of  $h[n]$ . The 2-D Smith-Barnwell condition is satisfied if the 1-D Smith-Barnwell condition given by Eq. (7.60) is satisfied.

### 7-7.4 2-D Haar Wavelet Transform of Shepp-Logan Phantom

The **Shepp-Logan phantom** is a piecewise-constant image that has been a test image for tomography algorithms since the 1970s. A  $256 \times 256$  image of the Shepp-Logan phantom is displayed in Fig. 7-16(a). To illustrate what the coarse and detail images generated by the application of the 2-D wavelet transform look like, we applied a 4-stage 2-D Haar wavelet transform to the image in Fig. 7-16(a). The results are displayed per the pattern in Fig. 7-16(b), with:

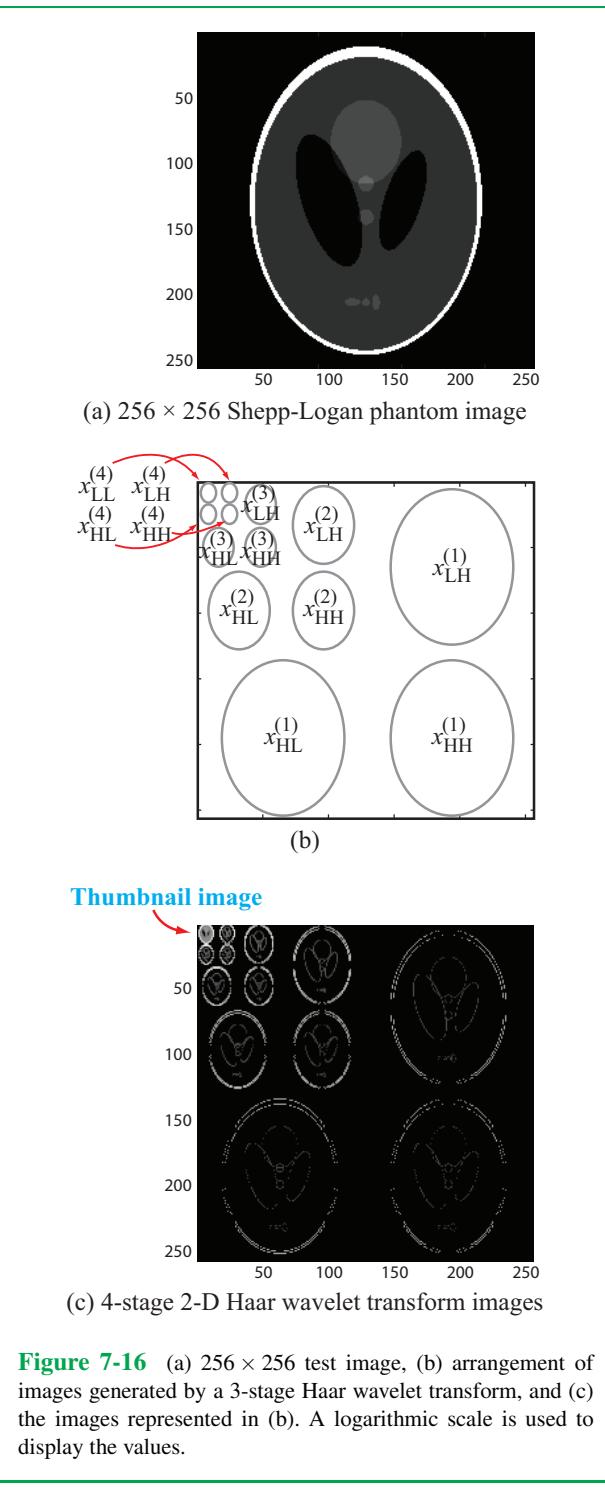
#### (1) Stage-4 images: $16 \times 16$

The coarsest average image,  $x_{LL}^{(4)}[n, m]$ , is used as a thumbnail image, and placed at the upper left-hand corner in Fig. 7-16(c). The three stage-4 detail images are arranged clockwise around image  $x_{LL}^{(4)}[n, m]$ .

#### (2) Stage-3 images: $32 \times 32$

The three stage-3 detail images are four times as large, and are arranged clockwise around the stage-4 images.

#### (3) Stage-2 images: $64 \times 64$



**Figure 7-16** (a)  $256 \times 256$  test image, (b) arrangement of images generated by a 3-stage Haar wavelet transform, and (c) the images represented in (b). A logarithmic scale is used to display the values.

#### (4) Stage-1 images: $128 \times 128$

The largest images in **Fig. 7-16(c)** are the three stage-1 images. The number of pixels in the 2-D Haar transform can be computed similarly to the 1-D Haar transform durations in Eq. (7.46):

**Stage 4 images:** The 4 ( $16 \times 16$ ) images (1 average and 3 detail) contain a total of  $4(16)^2 = 1024$  pixels.

**Stage 3 images:** The 3 ( $32 \times 32$ ) detail images contain a total of  $3(32)^2 = 3072$  pixels. The fourth  $32 \times 32$  image is the average image, which is decomposed into the stage 4 images.

**Stage 2 images:** The 3 ( $64 \times 64$ ) detail images contain a total of  $3(64)^2 = 12288$  pixels. The fourth  $64 \times 64$  image is the average image, which is decomposed into the stage 3 images.

**Stage 1 images:** The 3 ( $128 \times 128$ ) detail images contain a total of  $3(128)^2 = 49152$  pixels. The fourth  $128 \times 128$  image is the average image, which is decomposed into the stage 2 images.

The total number of pixels in the wavelet transform of the Shepp-Logan phantom is then:

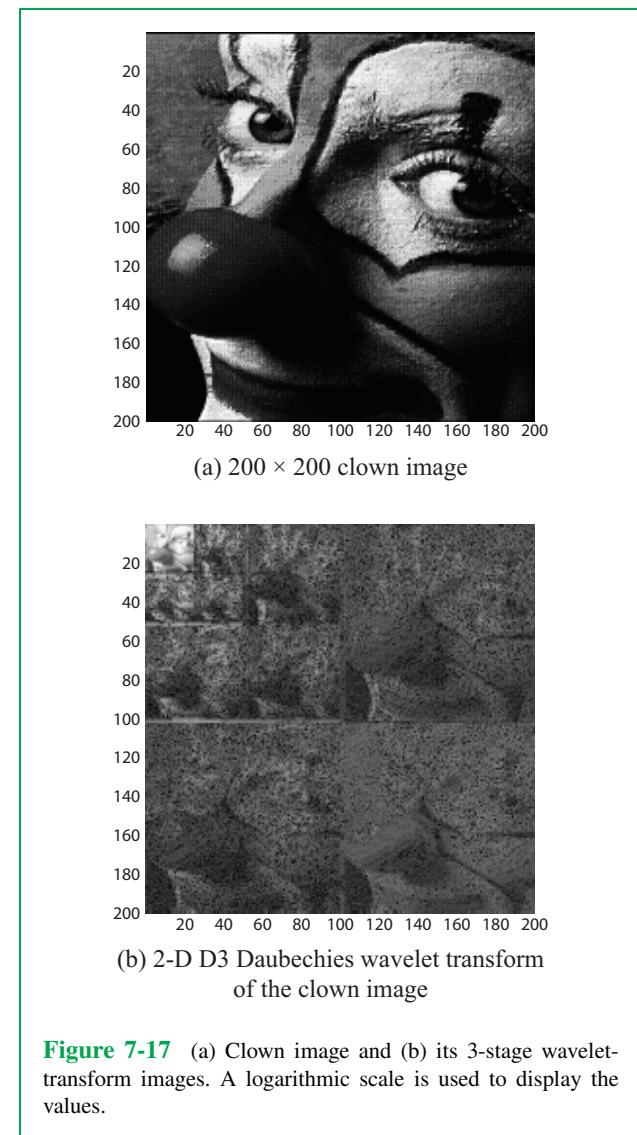
$$1024 + 3072 + 12288 + 49152 = 65536.$$

This equals the number of pixels in the Shepp-Logan phantom, which is  $256^2 = 65536$ .

Even though the coarsest average image is only  $16 \times 16$ , it contains almost all of the large numbers in the 2-D wavelet transform of the original image, and captures most of its primary features. That is why it is used as a **thumbnail image**. The pixel values of the stage-4 detail images are almost entirely zeros. The 3-stage composition process preserves the information content of the original image—composed of  $256^2 = 65536$  pixels—while compressing it down to 4 images containing only 3619 nonzero pixels. This is a 94.5% reduction in the number of nonzero pixels.

#### 7-7.5 2-D db3 Daubechies Wavelet Transform of Clown Image

For a second example, we repeated the steps outlined in the preceding subsection, but this time we used a 2-D db3 Daubechies wavelet transform on the  $200 \times 200$  clown image shown in **Fig. 7-17(a)**. The images generated by the 3-stage decomposition process are displayed in part (b) of the figure, using the same arrangement as shown earlier in **Fig. 7-17(b)**.



**Figure 7-17** (a) Clown image and (b) its 3-stage wavelet-transform images. A logarithmic scale is used to display the values.

#### 7-7.6 Image Compression by Thresholding Its Wavelet Transform

Image compression is an important feature of the wavelet transform. Not only is the original image represented by fewer pixels, but also many of the pixels of the wavelet-transform images are zero-valued. The compression ratio can be improved further by thresholding the output images of the final stage of

the wavelet-transform decomposition process. Thresholding a pixel means replacing it with zero if its absolute value is below a given **threshold level**  $\lambda$ . As noted earlier in connection with 1-D signals and 2-D images, most of the wavelet-transform detail signals and images have very small values, so little information is lost by setting their values to zero, which means that they no longer need to be stored, thereby reducing the storage capacity needed to store the wavelet transform of the image. Furthermore, since the wavelet transform is composed of orthogonal basis functions, a small change in the wavelet transform of an image will produce only a small change in the image reconstructed from these values (see Section 7-2.3).

To illustrate with an example, we compare in **Fig. 7-18** two images:

- (a) In part (a), we show the original  $200 \times 200$  clown image, and
- (b) in part (b) we show a reconstructed clown image, generated from the db3 Daubechies wavelet-transform images after thresholding the images with  $\lambda = 0.11$ .

The reconstructed image looks almost identical to the original image, even though only 6% of the pixels in the wavelet-transform images are nonzero.

**Concept Question 7-6:** Why is the 2-D wavelet transform useful for generating thumbnail images?

## 7-8 Denoising by Thresholding and Shrinking

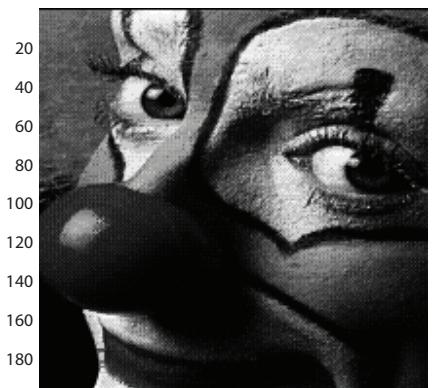
A noisy image is given by

$$y[n, m] = x[n, m] + v[n, m], \quad (7.104)$$

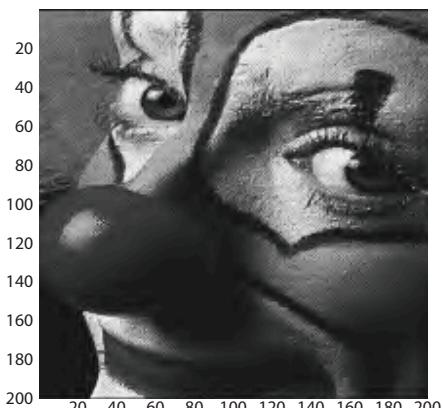
where  $x[n, m]$  is the desired image and  $v[n, m]$  is the noise that had been added to it. The goal of denoising is to recover the original image  $x[n, m]$ , or a close approximation thereof, from the noisy image  $y[n, m]$ . We now show that the obvious approach of simply thresholding the wavelet transform of the image does not work well. Then we show that the combination of thresholding and **shrinking** the wavelet transform of the image does work well.

### 7-8.1 Denoising by Thresholding Alone

One approach to denoising is to threshold the wavelet transform of  $y[n, m]$ . For small wavelet transform values, the signal-to-



(a)  $200 \times 200$  clown image

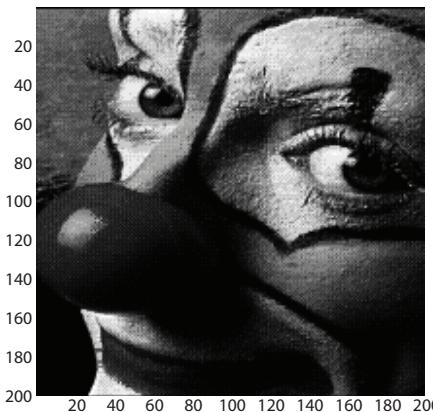


(b) Reconstructed clown image

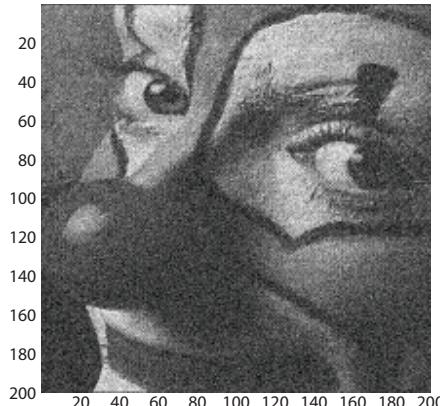
**Figure 7-18** (a) Original clown image, and (b) image reconstructed from thresholded db3 Daubechies wavelet transform images, requiring only 6% as much storage capacity as the original image.

noise ratio is low, so little of value is lost by thresholding these small values to zero. For large wavelet transform values, the signal-to-noise ratio is large, so these large values should be kept. This approach works poorly on wavelet transforms of noisy images, as the following example shows.

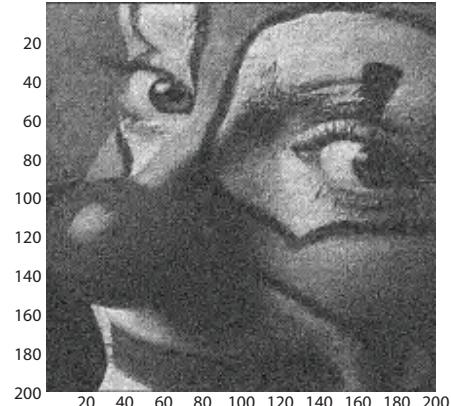
Zero-mean 2-D white Gaussian noise with standard deviation  $\sigma = 0.1$  was added to the clown image of **Fig. 7-19(a)**. The noisy



(a) Original noise-free image



(b) Noisy image



(c) Image denoised by thresholding its wavelet transform

**Figure 7-19** (a) Noise-free clown image, (b) noisy image with  $\text{SNR} = 11.5$ , and (c) image reconstructed from thresholded wavelet transform. Thresholding without shrinkage does not reduce noise.

image, shown in **Fig. 7-19(b)**, has a signal-to-noise ratio (SNR) of 11.5, which means that the noise level is, on average, only about 8.7% of that of the signal.

The db3 Daubechies wavelet transform was computed for the noisy image, then thresholded with  $\lambda = 0.11$ , which appeared to provide the best results. Finally, the image was reconstructed from the thresholded wavelet transform, and it now appears in **Fig. 7-19(c)**. Upon comparing the images in parts (b) and (c) of the figure, we conclude that the thresholding operation failed to

reduce the noise by any appreciable amount.

### 7-8.2 Denoising by Thresholding and Shrinkage

We now show that a combination of **thresholding** small wavelet-transform values to zero and **shrinking** other wavelet transform values by a small number  $\lambda$  performs much better in denoising images. First we show that shrinkage comes from minimizing a cost functional, just as Wiener filtering given by Eq. (6.31)

came from minimizing the Tikhonov cost functional given by Eq. (6.29).

In the material that follows, we limit our treatment to 1-D signals. The results are readily extendable to 2-D images.

Suppose we are given noisy observations  $y[n]$  of a signal  $x[n]$  that is known to be sparse (mostly zero),

$$y[n] = x[n] + v[n]. \quad (7.105)$$

All three signals have durations  $N$  and the noise  $v[n]$  is known to have zero-mean. The goal is to estimate  $x[n]$  from the noisy observations  $y[n]$ ; i.e., to denoise  $y[n]$  with the additional information that  $x[n]$  is mostly zero-valued.

The prior knowledge that  $x[n]$  is sparse can be incorporated by estimating  $x[n]$  from  $y[n]$ , not by simply using  $y[n]$  as an estimator for  $x[n]$ , but by minimizing over  $x[n]$  the **LASSO cost functional** (discussed in more detail in Section 7-9.2).

$$\Lambda = \underbrace{\frac{1}{2} \sum_{n=0}^{N-1} (y[n] - x[n])^2}_{\text{fidelity to data } y[n]} + \lambda \underbrace{\sum_{n=0}^{N-1} |x[n]|}_{\text{sparsity}}. \quad (7.106)$$

Readers familiar with basic estimation theory will note that  $\Lambda$  is the negative log-likelihood function for zero-mean white Gaussian noise  $v[n]$  with independent Laplacian *a priori* distributions for each  $x[n]$ . The coefficient  $\lambda$  is a **trade-off parameter** between fidelity to the data  $y[n]$  and imposition of sparsity. If  $\lambda = 0$ , then the estimator  $\hat{x}[n]$  of  $x[n]$  is just  $\hat{x}[n] = y[n]$ . Nonzero  $\lambda$  emphasizes sparsity, while allowing some difference between  $\hat{x}[n]$  and  $y[n]$ , which takes into account the noise  $v[n]$ .

**LASSO** is an acronym for **least absolute shrinkage and selection operator**.

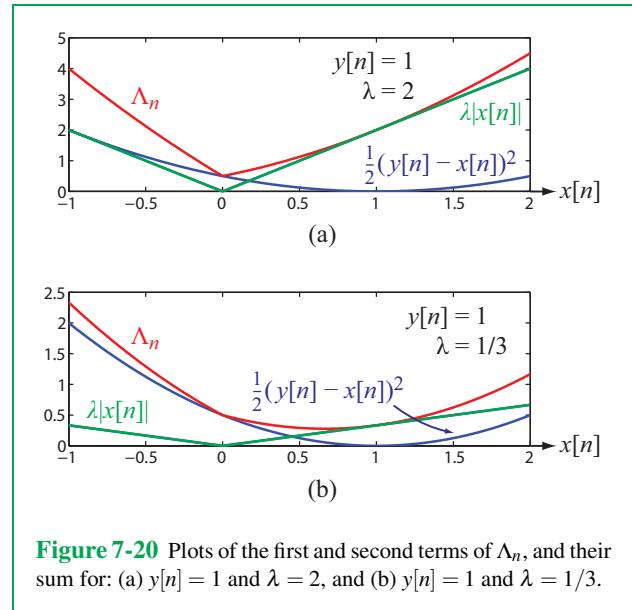
### 7-8.3 Minimization of LASSO Cost Functional

The minimization of  $\Lambda$  decouples in time  $n$ , so each term can be minimized separately. The goal then is to find an **estimate** of  $x[n]$ , which we denote  $\hat{x}[n]$ , that minimizes over  $x[n]$  the  $n$ th term  $\Lambda_n$  of Eq. (7.106), namely

$$\Lambda_n = \frac{1}{2}(y[n] - x[n])^2 + \lambda |x[n]|. \quad (7.107)$$

The expression given by Eq. (7.107) is the sum of two terms. The value of  $\lambda$  is selected to suit the specific application; if fidelity to the measured observations  $y[n]$  is highly prized, then  $\lambda$  is assigned a small value, but if sparsity is an important attribute, then  $\lambda$  may be assigned a large value.

Keeping in mind that  $\lambda \geq 0$ , the minimization problem



**Figure 7-20** Plots of the first and second terms of  $\Lambda_n$ , and their sum for: (a)  $y[n] = 1$  and  $\lambda = 2$ , and (b)  $y[n] = 1$  and  $\lambda = 1/3$ .

consists of four possible scenarios:

- (1)  $y[n] \geq 0$  and  $y[n] \leq \lambda$
- (2)  $y[n] \geq 0$  and  $y[n] \geq \lambda$
- (3)  $y[n] \leq 0$  and  $|y[n]| \leq \lambda$
- (4)  $y[n] \leq 0$  and  $|y[n]| \geq \lambda$

#### Case 1: $y[n] \geq 0$ and $y[n] \leq \lambda$

Let us consider the following example for a particular value of  $n$ :

Measurement  $y[n] = 1$   
Trade-off parameter  $\lambda = 2$   
Signal  $x[n]$ : unknown

The estimated value  $\hat{x}[n]$  of  $x[n]$  is found by minimizing  $\Lambda_n$ .

**Figure 7-20(a)** displays three plots, corresponding to the first and second terms in Eq. (7.107), and their sum. It is evident from the plot of  $\Lambda_n$  that  $\Lambda_n$  is minimized at  $x[n] = 0$ . Hence,

$$\hat{x}[n] = 0 \quad \text{for } y[n] \geq 0 \text{ and } y[n] \leq \lambda. \quad (7.108a)$$

#### Case 2: $y[n] \geq 0$ and $y[n] \geq \lambda$

Repetition of the scenario described by case 1, but with  $\lambda$  changed from 2 to 1/3, leads to the plots shown in **Fig. 7-20(b)**.

In this case,  $\Lambda_n$  is parabolic-like in shape, and its minimum occurs at a positive value of  $x[n]$  at which the slope of  $\Lambda_n$  is zero. That is,

$$\frac{d\Lambda_n}{dx[n]} = \frac{d}{dx[n]} \left( \frac{1}{2} (y[n] - x[n])^2 + \lambda x[n] \right) = 0,$$

which leads to

$$\hat{x}[n] = y[n] - \lambda, \quad \text{for } y[n] \geq 0 \text{ and } y[n] \geq \lambda. \quad (7.108b)$$

### Case 3: $y[n] \leq 0$ and $|y[n]| \leq \lambda$

This case, which is identical to case 1 except that now  $y[n]$  is negative, leads to the same result, namely

$$\hat{x}[n] = 0, \quad \text{for } y[n] \leq 0 \text{ and } |y[n]| \leq \lambda. \quad (7.108c)$$

### Case 4: $y[n] \leq 0$ and $|y[n]| \geq \lambda$

Repetition of the analysis of case 2, but with  $y[n]$  negative, leads to

$$\hat{x}[n] = y[n] + \lambda, \quad \text{for } y[n] \leq 0 \text{ and } |y[n]| \geq \lambda. \quad (7.108d)$$

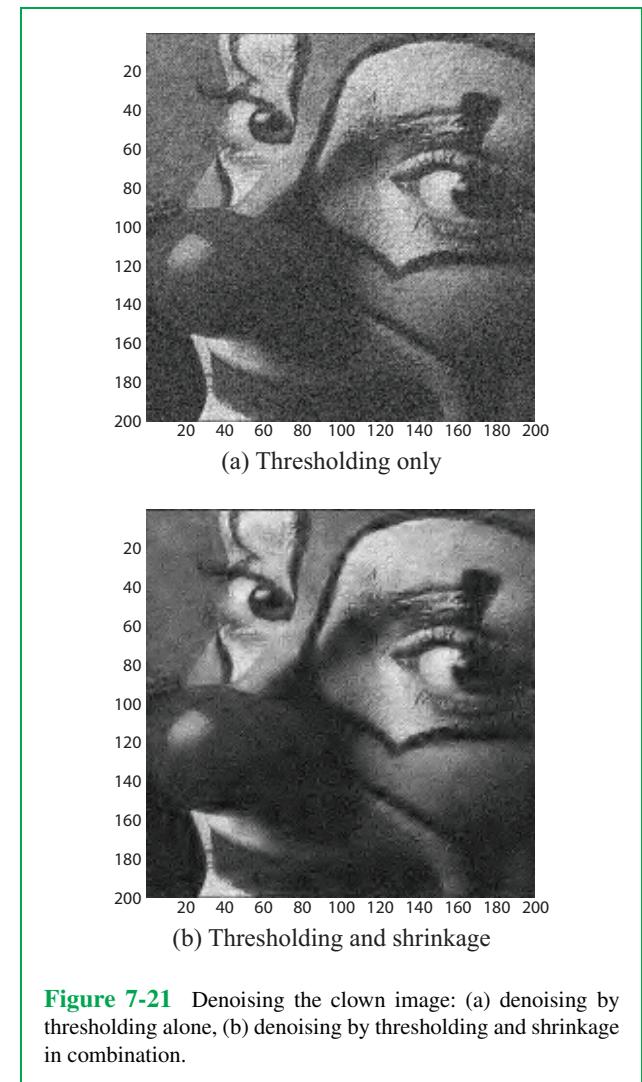
The four cases can be combined into

$$\hat{x}[n] = \begin{cases} y[n] - \lambda & \text{for } y[n] > +\lambda, \\ y[n] + \lambda & \text{for } y[n] < -\lambda, \\ 0 & \text{for } |y[n]| < \lambda. \end{cases} \quad (7.109)$$

Values of  $y[n]$  smaller in absolute value than the threshold  $\lambda$  are **thresholded** (set to zero). Values of  $y[n]$  larger in absolute value than the threshold  $\lambda$  are **shrunk** by  $\lambda$ , making their absolute values smaller. So  $\hat{x}[n]$  is computed by **thresholding and shrinking**  $y[n]$ . This is usually called (ungrammatically) “thresholding and shrinkage,” or “soft thresholding.”

The next example shows that denoising images works much better with thresholding and shrinkage than with thresholding alone.

When we applied thresholding alone to the noisy image of Fig. 7-19(b), we obtained the image shown in Fig. 7-19(c), which we repeat here in Fig. 7-21(a). Application of thresholding and shrinkage in combination, with  $\lambda = 0.11$ , leads to the image in Fig. 7-21(b), which provides superior rendition of the clown image by filtering much more of the noise, while



**Figure 7-21** Denoising the clown image: (a) denoising by thresholding alone, (b) denoising by thresholding and shrinkage in combination.

preserving the real features of the image.

**Concept Question 7-7:** Why is the combination of shrinkage and thresholding needed for noise reduction?

**Concept Question 7-8:** Why does wavelet-based denoising work so much better than lowpass filtering?

## 7-9 Compressed Sensing

The solution of an **inverse problem** in signal and image processing is the reconstruction of an unknown signal or image from measurements (known linear combinations) of the values of the signal or image. Such inverse problems arise in medical imaging, radar imaging, optics, and many other fields. For example, in tomography and **magnetic resonance imaging (MRI)**, the inverse problem is to reconstruct an image from measurements of some (but not all) of its 2-D Fourier transform values.

If the number of measurements equals or exceeds the size (duration in 1-D, number of pixels in 2-D) of the unknown signal or image, solution of the inverse problem in the absence of noise becomes a solution of a linear system of equations. In practice, there is always noise in the measurements, so some sort of regularization is required. In Section 6-4.3, the deconvolution problem required Tikhonov regularization to produce a recognizable solution when noise was added to the data. Furthermore, often the number of observations is less than the size of the unknown signal or image. For example, in tomography, the 2-D Fourier transform values of the image at very high wavenumbers are usually unknown. In this case, the inverse problem is **under-determined**; consequently, even in the absence of noise there is an infinite number of possible solutions. Hence, regularization is needed, not only to deal with the underdetermined formulation, but also to manage the presence of noise in the measurements.

We have seen that many real-world signals and images can be compressed, using the wavelet transform, into a sparse representation in which most of the values are zero. This suggests that the number of measurements needed to reconstruct the signal or image can be less than the size of the signal or image, because in the wavelet-transform domain, most of the values to be reconstructed are known to be zero. Had the locations of the nonzero values been known, the problem would have been reduced to a solution of a linear system of equations smaller in size than that of the original linear system of equations. In practice, however, neither the locations of the nonzero values nor their values are known.

**Compressed sensing** refers to a set of signal processing techniques used for reconstructing wavelet-compressible signals and images from measurements that are much fewer in number than the size of the signal or image, but much larger than the number of nonzero values in the wavelet transform of the signal or image. The general formulation of the problem is introduced in the next subsection. There are many advantages to reducing the number of measurements needed to reconstruct the signal or image. In tomography, for example, the acquisition of a fewer number of measurements reduces patient exposure to radiation.

In MRI, this reduces acquisition time inside the MRI machine, and in smartphone cameras, it reduces the exposure time and energy required to acquire an image.

This section presents the basic concepts behind compressed sensing and applies these concepts to a few signal and image inverse problems. Compressed sensing is an active area of research and development, and will experience significant growth in applications in the future.

### 7-9.1 Problem Formulation

To cast the compressed sensing problem into an appropriate form, we define the following quantities:

(a)  $\{x[n], n = 0 \dots N - 1\}$  is an **unknown signal** of length  $N$

(b) The corresponding (unknown) wavelet transform of  $x[n]$  is

$$\{\tilde{x}_1[n], \tilde{x}_2[n], \dots, \tilde{x}_L[n], \tilde{X}_L[n]\},$$

and the wavelet transform of  $x[n]$  is **sparse**: only  $K$  values of all of the  $\{\tilde{x}_k[n]\}$  are nonzero, with  $K \ll N$ .

(c)  $\{y[n], n = 0, 1, \dots, M - 1\}$  are  $M$  **known measurements**

$$y[0] = a_{0,0} x[0] + a_{0,1} x[1] + \dots + a_{0,N-1} x[N-1],$$

$$y[1] = a_{1,0} x[0] + a_{1,1} x[1] + \dots + a_{1,N-1} x[N-1],$$

⋮

$$y[M-1] = a_{M-1,0} x[0] + a_{M-1,1} x[1] + \dots \\ + a_{M-1,N-1} x[N-1],$$

where  $\{a_{n,i}, n = 0, 1, \dots, M - 1 \text{ and } i = 0, 1, \dots, N - 1\}$  are known.

(d)  $K$  is unknown, but we know that  $K \ll M < N$ .

The goal of compressed sensing is to compute signal  $\{x[n], n = 0, 1, \dots, N - 1\}$  from the  $M$  known measurements  $\{y[n], n = 0, 1, \dots, M - 1\}$ .

The compressed sensing problem can be divided into two components, a **direct problem** and an **inverse problem**. In the direct problem, the independent variable (input) is  $x[n]$  and the dependent variable (output) is the measurement  $y[n]$ . The roles are reversed in the inverse problem: the measurements become the independent variables (input) and the unknown signal  $x[n]$  becomes the output. The relationships between  $x[n]$  and  $y[n]$  involve vectors and matrices:

## A. Signal vector

$$\mathbf{x} = [x[0], x[1], \dots, x[N-1]]^T, \quad (7.110)$$

where  $T$  denotes the transpose operator, which converts a row vector into a column vector.

$$\mathbf{y} = [y[0], y[1], \dots, y[M-1]]^T. \quad (7.111)$$

## B. Wavelet transform vector

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} \tilde{x}_1[n] \\ \tilde{x}_2[n] \\ \vdots \\ \tilde{x}_L[n] \\ \tilde{X}_L[n] \end{bmatrix}, \quad (7.112)$$

$\mathbf{z}$  is of length  $N$ , and  $\tilde{x}_1[n]$  to  $\tilde{x}_L[n]$  are the detail signals of the wavelet transform and  $\tilde{X}_L[n]$  is the coarse signal.

## C. Wavelet transform matrix

$$\mathbf{z} = \mathbf{W} \mathbf{x}, \quad (7.113)$$

where  $\mathbf{W}$  is a known  $N \times N$  **wavelet transform matrix** that implements the wavelet transform of  $x[n]$  to obtain  $\mathbf{z}$ .

## D. Direct-problem formulation

$$\mathbf{y} = \mathbf{A} \mathbf{x}, \quad (7.114)$$

where  $\mathbf{A}$  is an  $M \times N$  matrix. Usually,  $\mathbf{A}$  is a known matrix based on a physical model or direct measurement of  $\mathbf{y}$  for a known  $\mathbf{x}$ . Combining Eqs. (7.113) and (7.114) gives

$$\mathbf{y} = \mathbf{A} \mathbf{W}^{-1} \mathbf{z} = \mathbf{A}_w \mathbf{z}, \quad (7.115a)$$

where

$$\mathbf{A}_w = \mathbf{A} \mathbf{W}^{-1}. \quad (7.115b)$$

Since  $\mathbf{A}$  and  $\mathbf{W}$  are both known matrices,  $\mathbf{A}_w$  also is known.

## E. Inverse-problem formulation

If, somehow,  $\mathbf{z}$  can be determined from the measurement vector  $\mathbf{y}$ , then  $\mathbf{x}$  can be computed by inverting Eq. (7.113):

$$\mathbf{x} = \mathbf{W}^{-1} \mathbf{z}. \quad (7.116)$$

For the orthogonal wavelet transforms, such as the Haar and Daubechies transforms covered in Sections 7-4 and 7-6,  $\mathbf{W}^{-1} = \mathbf{W}^T$ , so the inverse wavelet transform can be computed as easily as the wavelet transform. In practice, both are computed using analysis and synthesis filter banks, as discussed in earlier sections.

The crux of the compressed sensing problem reduces to finding  $\mathbf{z}$ , given  $\mathbf{y}$ . An additional factor to keep in mind is that only  $K$  values of the elements of  $\mathbf{z}$  are nonzero, with  $K \ll M$ . Algorithms for computing  $\mathbf{z}$  from  $\mathbf{y}$  rely on iterative approaches, as discussed in future sections.

Because  $\mathbf{z}$  is of length  $N$ ,  $\mathbf{y}$  of length  $M$ , and  $M < N$  (fewer measurements than unknowns), Eq. (7.115) represents an **underdetermined system** of linear equations, whose solution is commonly called an **ill-posed problem**.

## 7-9.2 Inducing Sparsity into Solutions

In seismic signal processing, explosions are set off on the Earth's surface, and echoes of the seismic waves created by the explosion are measured by seismometers. In the 1960s, sedimentary media (such as the bottom of the Gulf of Mexico) were modeled as a stack of layers, so the seismometers would record occasional sharp pulses reflected off of the interfaces between the layers. The amplitudes and times of the pulses would allow the layered medium to be reconstructed. However, the occasional pulses had to be deconvolved from the source pulse created by the explosions. The deconvolution problem was modeled as an underdetermined linear system of equations.

A common approach to finding a sparse solution to a system of equations is to choose the solution that minimizes the sum of absolute values of the solution. This is known as the **minimum  $\ell_1$  norm** solution. The  $\ell_1$  norm is denoted by the symbol  $\|\mathbf{z}\|_1$  and defined as

$$\|\mathbf{z}\|_1 = \sum_{i=1}^N |z_i|. \quad (7.117a)$$

The goal is to find the solution to the system of equations that minimizes  $\|\mathbf{z}\|_1$ .

A second approach called the **squared  $\ell_2$  norm**, which does not provide a sparse solution, finds the solution that minimizes the sum of squares of the solution. The squared  $\ell_2$  norm is denoted by the symbol  $\|\mathbf{z}\|_2^2$  and is defined as

$$\|\mathbf{z}\|_2^2 = \sum_{i=1}^N |z_i|^2. \quad (7.117b)$$

**Concept Question 7-9:** What is compressed sensing?

## 7-10 Computing Solutions to Underdetermined Equations

### 7-10.1 Basis Pursuit

As noted earlier, the unknown signal vector  $\mathbf{x}$  can be determined from Eq. (7.116), provided we have a solution for  $\mathbf{z}$  (because  $\mathbf{W}$  is a known matrix). One possible approach to finding  $\mathbf{z}$  is to implement the minimum  $\ell_1$  norm given by Eq. (7.117a), subject to the constraint given by Eq. (7.115a). That is, the goal is to find vector  $\mathbf{z}$  from measurement vector  $\mathbf{y}$  such that

$$\sum_{i=1}^N |z_i| \text{ is minimum,}$$

and

$$\mathbf{y} = \mathbf{A}_w \mathbf{z}.$$

The solution method is known as **basis pursuit**, and it can be formulated as a **linear programming** problem by defining the positive  $\mathbf{z}^+$  and negative  $\mathbf{z}^-$  parts of  $\mathbf{z}$  as:

$$z_i^+ = \begin{cases} +z_i & \text{if } z_i \geq 0, \\ 0 & \text{if } z_i < 0, \end{cases} \quad (7.118a)$$

$$z_i^- = \begin{cases} -z_i & \text{if } z_i \leq 0, \\ 0 & \text{if } z_i > 0. \end{cases} \quad (7.118b)$$

Vector  $\mathbf{z}$  is then given by

$$\mathbf{z} = \mathbf{z}^+ - \mathbf{z}^-, \quad (7.119)$$

and its  $\ell_1$  norm is

$$\|\mathbf{z}\|_1 = \sum_{i=1}^N (z_i^+ + z_i^-). \quad (7.120)$$

In terms of  $\mathbf{z}^+$  and  $\mathbf{z}^-$ , the basis pursuit problem becomes:

$$\begin{aligned} & \text{Minimize } \sum_{i=1}^N (z_i^+ + z_i^-) \\ & \text{subject to } \mathbf{y} = \mathbf{A}_w \mathbf{z}^+ - \mathbf{A}_w \mathbf{z}^-, \quad (z_i^+ \geq 0, z_i^- \geq 0), \end{aligned} \quad (7.121)$$

which is a straightforward linear programming problem that can be solved using `linprog` in MATLAB's Optimization Toolbox. The basis pursuit method is limited to noise-free signal and image problems, so in the general case, more sophisticated approaches are called for.

### 7-10.2 LASSO Cost Functional

A serious shortcoming of the basis pursuit solution method is that it does not account for the presence of noise in the observations  $\mathbf{y}$ . As noted earlier in Section 7-8.2, the **least absolute shrinkage and selection operator (LASSO)** provides an effective approach for estimating the true signal from noisy measurements. In the present context, the LASSO functional is defined as

$$\Lambda = \frac{1}{2} \underbrace{\|\mathbf{y} - \mathbf{A}_w \mathbf{z}\|_2^2}_{\text{fidelity}} + \lambda \underbrace{\|\mathbf{z}\|_1}_{\text{sparsity}}, \quad (7.122)$$

where  $\lambda$  is a trade-off parameter between sparsity of  $\mathbf{z}$  and fidelity to the measurement  $\mathbf{y}$ . Choosing the solution  $\mathbf{z}$  that minimizes Eq. (7.122), for a specified value of  $\lambda$ , is called **basis pursuit denoising**. The solution requires the use of an iterative algorithm. Two such algorithms are presented in later subsections, preceded by a short review of **pseudo inverses**.

### 7-10.3 Review of Pseudo-Inverses

#### (a) Overdetermined system

Consider the linear system of equations given by

$$\mathbf{y} = \mathbf{A}_w \mathbf{z}, \quad (7.123)$$

with  $\mathbf{z}$  of length  $N$ ,  $\mathbf{y}$  of length  $M$ , and  $\mathbf{A}_w$  of size  $M \times N$  with full rank. If  $M > N$ , then the system is **overdetermined** (more measurements than unknowns) and, in general, it has no solution. The vector  $\hat{\mathbf{z}}$  that minimizes  $\|\mathbf{y} - \mathbf{A}_w \mathbf{z}\|_2^2$  is called the **pseudo-inverse solution**, and is given by the **estimate**

$$\hat{\mathbf{z}} = (\mathbf{A}_w^T \mathbf{A}_w)^{-1} \mathbf{A}_w^T \mathbf{y}. \quad (7.124a)$$

Note that  $\mathbf{A}_w^T \mathbf{A}_w$  is an  $N \times N$  matrix with full rank.

To avoid matrix-inversion problems,  $\hat{\mathbf{z}}$  should be computed not by inverting  $\mathbf{A}_w^T \mathbf{A}_w$ , but by solving the linear system of equations

$$(\mathbf{A}_w^T \mathbf{A}_w) \hat{\mathbf{z}} = \mathbf{A}_w^T \mathbf{y} \quad (7.124b)$$

using the ***LU decomposition*** method or similar techniques. Here, LU stands for *lower upper*, in reference to the lower triangular submatrix and the upper triangular submatrix multiplying the unknown vector  $\hat{\mathbf{z}}$ .

### (b) Underdetermined system

Now consider the underdetermined system characterized by  $M < N$  (fewer measurements than unknowns). In this case, there is an infinite number of possible solutions. The vector  $\hat{\mathbf{z}}$  that minimizes  $\|\mathbf{z}\|_2^2$  among this infinite number of solutions also is called the pseudo-inverse solution, and is given by the estimate

$$\hat{\mathbf{z}} = \mathbf{A}_w^T (\mathbf{A}_w \mathbf{A}_w^T)^{-1} \mathbf{y}. \quad (7.125)$$

In the present case,  $\mathbf{A}_w \mathbf{A}_w^T$  is an  $M \times M$  matrix with full rank. Solution  $\hat{\mathbf{z}}$  should be computed not by inverting  $\mathbf{A}_w \mathbf{A}_w^T$ , but by initially solving the linear system

$$(\mathbf{A}_w \mathbf{A}_w^T) \hat{\mathbf{r}} = \mathbf{y}, \quad (7.126a)$$

to compute an intermediate estimate  $\hat{\mathbf{r}}$ , and then computing  $\hat{\mathbf{z}}$  by applying

$$\hat{\mathbf{z}} = \mathbf{A}_w^T \hat{\mathbf{r}}. \quad (7.126b)$$

### 7-10.4 Iterative Reweighted Least Squares (IRLS) Algorithm

According to Eq. (7.115a), measurement vector  $\mathbf{y}$  and wavelet transform vector  $\mathbf{z}$  are related by

$$\mathbf{y} = \mathbf{A}_w \mathbf{z}. \quad (7.127)$$

The ***iterative reweighted least squares (IRLS)*** algorithm uses the Tikhonov regularization functional given by Eq. (6.29), together with a diagonal weighting matrix  $\mathbf{D}$  to minimize the cost function

$$\mathcal{T} = \underbrace{\frac{1}{2} \|\mathbf{y} - \mathbf{A}_w \mathbf{z}\|_2^2}_{\text{fidelity}} + \lambda \underbrace{\|\mathbf{D} \mathbf{z}\|_2^2}_{\text{size}} \quad (7.128)$$

where  $\lambda$  is the trade-off parameter between the size of  $\mathbf{z}$  and the fidelity to the data  $\mathbf{y}$ . The goal is to trade off small differences between  $\mathbf{y}$  and  $\mathbf{A}_w \mathbf{z}$  so as to keep  $\mathbf{z}$  small. To compute  $\mathbf{z}$ , we first

rewrite Eq. (7.128) in the expanded form

$$\mathcal{T} =$$

$$\frac{1}{2} \left\| \begin{bmatrix} \mathbf{y}[0] \\ \mathbf{y}[1] \\ \vdots \\ \mathbf{y}[M-1] \end{bmatrix} - \begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,N-1} \\ \vdots & \vdots & & \vdots \\ a_{M-1,0} & a_{M-1,1} & \dots & a_{M-1,N-1} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{bmatrix} \right\|_2^2$$

$M \times 1 \qquad M \times N \qquad N \times 1$

$$+ \lambda \left\| \begin{bmatrix} D_{11} & & & & \\ & \ddots & & & \\ & & \ddots & & \\ 0 & & & \ddots & 0 \\ & & & & D_{NN} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{bmatrix} \right\|_2^2$$

$N \times N \qquad N \times 1$

(7.129)

where  $a_{i,j}$  is the  $(i,j)$ th element of  $\mathbf{A}_w$ , not of  $\mathbf{A}$ .

Both the unknown vector  $\mathbf{x}$  and its wavelet vector  $\mathbf{z}$  are of size  $N \times 1$ . This is in contrast with the much shorter measurement vector  $\mathbf{y}$ , which is of size  $M \times 1$ , with  $M < N$ .

We now introduce vector  $\mathbf{y}'$  and matrix  $\mathbf{B}_w$  as

$$[\mathbf{y}'] = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} \begin{array}{l} \} M \times 1 \\ \} N \times 1 \end{array}_{(M+N) \times 1}, \quad (7.130a)$$

$$[\mathbf{B}_w] = \begin{bmatrix} \mathbf{A}_w \\ \sqrt{2\lambda} \mathbf{D} \end{bmatrix} \begin{array}{l} \} M \times N \\ \} N \times N \end{array}_{(M+N) \times N}. \quad (7.130b)$$

Vector  $\mathbf{y}'$  is vector  $\mathbf{y}$  of length  $M$  stacked on top of vector  $\mathbf{0}$ , which is a column vector of zeros of length  $N$ . Similarly, matrix  $\mathbf{B}_w$  is matrix  $\mathbf{A}_w$  (of size  $M \times N$ ) stacked on top of matrix  $\mathbf{D}$  (of size  $N \times N$ ), multiplied by the scalar factor  $\sqrt{2\lambda}$ .

Next, we introduce the new cost function  $\mathcal{T}_1$  as

$$\begin{aligned}\mathcal{T}_1 &= \frac{1}{2} \|\mathbf{y}' - \mathbf{B}_w \mathbf{z}\|_2^2 \\ &= \frac{1}{2} \left\| \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{A}_w \\ \sqrt{2\lambda} \mathbf{D} \end{bmatrix} \mathbf{z} \right\|_2^2 \\ &= \frac{1}{2} \|\mathbf{y} - \mathbf{A}_w \mathbf{z}\|_2^2 + \lambda \|\mathbf{0} - \mathbf{D} \mathbf{z}\|_2^2 \\ &= \frac{1}{2} \|\mathbf{y} - \mathbf{A}_w \mathbf{z}\|_2^2 + \lambda \|\mathbf{D} \mathbf{z}\|_2^2 = \mathcal{T}. \quad (7.131)\end{aligned}$$

Hence, Eq. (7.128) can be rewritten in the form

$$\mathcal{T} = \frac{1}{2} \|\mathbf{y} - \mathbf{B}_w \mathbf{z}\|_2^2. \quad (7.132)$$

The vector  $\mathbf{z}$  minimizing  $\mathcal{T}$  is the pseudo-inverse given by

$$\begin{aligned}\hat{\mathbf{z}} &= (\mathbf{B}_w^T \mathbf{B}_w)^{-1} \mathbf{B}_w^T \mathbf{y}' \\ &= \left( \begin{bmatrix} \mathbf{A}_w^T & \sqrt{2\lambda} \mathbf{D}^T \end{bmatrix} \begin{bmatrix} \mathbf{A}_w \\ \sqrt{2\lambda} \mathbf{D} \end{bmatrix} \right)^{-1} \begin{bmatrix} \mathbf{A}_w^T & \sqrt{2\lambda} \mathbf{D}^T \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} \\ &= (\mathbf{A}_w^T \mathbf{A}_w + 2\lambda \mathbf{D}^T \mathbf{D})^{-1} \mathbf{A}_w^T \mathbf{y}. \quad (7.133)\end{aligned}$$

As always, instead of performing matrix inversion (which is susceptible to noise amplification), vector  $\hat{\mathbf{z}}$  should be computed by solving

$$(\mathbf{A}_w^T \mathbf{A}_w + 2\lambda \mathbf{D}^T \mathbf{D}) \hat{\mathbf{z}} = \mathbf{A}_w^T \mathbf{y}. \quad (7.134)$$

Once  $\hat{\mathbf{z}}$  has been determined, the unknown vector  $\mathbf{x}$  can be computed by solving Eq. (7.113).

To solve Eq. (7.134) for  $\hat{\mathbf{z}}$ , however, we need to know  $\mathbf{A}_w$ ,  $\lambda$ ,  $\mathbf{D}$ , and  $\mathbf{y}$ . From Eq. (7.115b),  $\mathbf{A}_w = \mathbf{A} \mathbf{W}^{-1}$ , where  $\mathbf{A}$  is a known matrix based on a physical model or calibration data, and  $\mathbf{W}$  is a known wavelet transform matrix. The parameter  $\lambda$  is specified by the user to adjust the intended balance between data fidelity and storage size (as noted in connection with Eq. (7.128)), and  $\mathbf{y}$  is the measurement vector. The only remaining quantity is the diagonal matrix  $\mathbf{D}$ , whose function is to assign weights to  $z_1$  through  $z_N$  so as to minimize storage size by having many elements of  $\mathbf{z} \rightarrow 0$ . Initially,  $\mathbf{D}$  is unknown, but it is possible to propose an initial function for  $\mathbf{D}$  and then iterate to obtain a solution for  $\mathbf{z}$  that minimizes the number of nonzero elements, while still satisfying Eq. (7.134).

The Tikhonov function given by Eq. (7.128) reduces to the

LASSO functional given by Eq. (7.122) if

$$\mathbf{D} = \text{diag} \left[ \frac{1}{\sqrt{|z_n|}} \right]. \quad (7.135)$$

This is because the second terms in the two equations become identical:

$$\|\mathbf{D} \mathbf{z}\|_2^2 = \sum_{n=1}^N \frac{z_n^2}{|z_n|} = \sum_{n=1}^N |z_n| = \|\mathbf{z}\|_1. \quad (7.136)$$

Given this correspondence between the two cost functionals, the IRLS algorithm uses the following iterative procedure to find  $\mathbf{z}$ :

(a) **Initial solution:** Set  $\mathbf{D} = \mathbf{I}$  and then compute  $\mathbf{z}^{(1)}$ , the initial iteration of  $\mathbf{z}$ , by solving Eq. (7.134).

(b) **Initial  $\mathbf{D}$ :** Use  $\mathbf{z}^{(1)}$  to compute  $\mathbf{D}^{(1)}$ , the initial iteration of  $\mathbf{D}$ :

$$\mathbf{D}^{(1)} = \text{diag} \left[ \frac{1}{\sqrt{|z_n^{(1)}| + \varepsilon}} \right]. \quad (7.137)$$

(c) **Second iteration:** Use  $\mathbf{D}^{(1)}$  to compute  $\mathbf{z}^{(2)}$  by solving Eq. (7.134) again.

(d) **Recursion:** Continue to iterate by computing  $\mathbf{D}^{(k)}$  from  $\mathbf{z}^{(k)}$  using

$$\mathbf{D}^{(k)} = \text{diag} \left[ \frac{1}{\sqrt{|z_n^{(k)}| + \varepsilon}} \right] \quad (7.138)$$

for a small deviation  $\varepsilon$  inserted in the expression to keep  $\mathbf{D}^{(k)}$  finite when elements of  $\mathbf{z}^{(k)} \rightarrow 0$ .

The iterative process ends when no significant change occurs between successive iterations. The algorithm, also called **focal underdetermined system solver (FOCUSS)**, is guaranteed to converge under mild assumptions. However, because the method requires a solution of a large system of equations at each iteration, the algorithm is considered unsuitable for most signal and image processing applications. Superior-performance algorithms are introduced in succeeding sections.

**Concept Question 7-10:** How can we get a useful solution to an underdetermined system of equations?

## 7-11 Landweber Algorithm

The **Landweber algorithm** is a recursive algorithm for solving linear systems of equations  $\mathbf{y} = \mathbf{Ax}$ . The **iterative shrinkage and thresholding algorithm (ISTA)** consists of the Landweber algorithm, with thresholding and shrinkage applied at each recursion. Thresholding and shrinkage were used in Section 7-7 to minimize the LASSO functional.

### 7-11.1 Underdetermined System

For an underdetermined system  $\mathbf{y} = \mathbf{Ax}$  with  $M < N$ , the solution  $\hat{\mathbf{x}}$  that minimizes the sum of squares of the elements of  $\mathbf{x}$  is, by analogy with Eq. (7.125), given by

$$\hat{\mathbf{x}} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{y}. \quad (7.139)$$

A useful relationship in matrix algebra states that if all of the eigenvalues  $\lambda_i$  of  $(\mathbf{A}\mathbf{A}^T)$  lie in the interval  $0 < \lambda_i < 2$ , then the coefficient of  $\mathbf{y}$  in Eq. (7.139) can be written as

$$\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1} = \sum_{k=0}^{\infty} (\mathbf{I} - \mathbf{A}^T\mathbf{A})^k \mathbf{A}^T. \quad (7.140)$$

The symbol  $\lambda_i$  for eigenvalue is unrelated to the trade-off parameter  $\lambda$  in Eq. (7.128).

Using Eq. (7.140) in Eq. (7.139) leads to

$$\hat{\mathbf{x}} = \sum_{k=0}^{\infty} (\mathbf{I} - \mathbf{A}^T\mathbf{A})^k \mathbf{A}^T \mathbf{y}. \quad (7.141)$$

A recursive implementation of Eq. (7.141) assumes the form

$$\mathbf{x}^{(K+1)} = \sum_{k=0}^K (\mathbf{I} - \mathbf{A}^T\mathbf{A})^k \mathbf{A}^T \mathbf{y}, \quad (7.142)$$

where the upper limit in the summation is now  $K$  (instead of  $\infty$ ). For  $K = 0$  and  $K = 1$ , we obtain the expressions

$$\mathbf{x}^{(1)} = \mathbf{A}^T \mathbf{y}, \quad (7.143a)$$

$$\mathbf{x}^{(2)} = \mathbf{A}^T \mathbf{y} + (\mathbf{I} - \mathbf{A}^T\mathbf{A})\mathbf{A}^T \mathbf{y} = (\mathbf{I} - \mathbf{A}^T\mathbf{A}) \mathbf{x}^{(1)} + \mathbf{A}^T \mathbf{y}. \quad (7.143b)$$

Extending the process to  $K = 2$  gives

$$\begin{aligned} \mathbf{x}^{(3)} &= \underbrace{\mathbf{A}^T \mathbf{y}}_{k=0} + \underbrace{(\mathbf{I} - \mathbf{A}^T\mathbf{A})\mathbf{A}^T \mathbf{y}}_{k=1} + \underbrace{(\mathbf{I} - \mathbf{A}^T\mathbf{A})^2 \mathbf{A}^T \mathbf{y}}_{k=2} \\ &= \mathbf{x}^{(2)} + \mathbf{A}^T(\mathbf{y} - \mathbf{A} \mathbf{x}^{(2)}). \end{aligned} \quad (7.143c)$$

Continuing the pattern leads to

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{A}^T(\mathbf{y} - \mathbf{A} \mathbf{x}^{(k)}). \quad (7.144)$$

The process can be initialized by  $\mathbf{x}^{(0)} = \mathbf{0}$ , which makes  $\mathbf{x}^{(1)} = \mathbf{A}^T \mathbf{y}$ , as it should.

The recursion process is called the **Landweber iteration**, which in optics is known as the **van Cittert iteration**. It is guaranteed to converge to the solution of  $\mathbf{y} = \mathbf{Ax}$  that minimizes the sum of squares of the elements of  $\mathbf{x}$ , provided that the eigenvalues  $\lambda_i$  of  $\mathbf{A}\mathbf{A}^T$  are within the range  $0 < \lambda_i < 2$ . If this condition is not satisfied, the formulation may be scaled to

$$\frac{\mathbf{y}}{c} = \frac{\mathbf{A}}{c} \mathbf{x}$$

or, equivalently,

$$\mathbf{u} = \mathbf{Bx}, \quad (7.145)$$

where  $\mathbf{u} = \mathbf{y}/c$  and  $\mathbf{B} = \mathbf{A}/c$ . The constant  $c$  is chosen so that the eigenvalue condition is satisfied. For example, if  $c$  is chosen to be equal to the sum of the squares of the magnitudes of all of the elements of  $\mathbf{A}$ , then the eigenvalues  $\lambda'_i$  of  $\mathbf{B}\mathbf{B}^T$  will be in the range  $0 < \lambda'_i < 1$ .

### 7-11.2 Overdetermined System

In analogy with Eq. (7.124a), the solution for an overdetermined system  $\mathbf{y} = \mathbf{Ax}$  is given by

$$\hat{\mathbf{x}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \mathbf{y}. \quad (7.146)$$

Using the equality

$$(\mathbf{A}^T\mathbf{A})^{-1} = \sum_{k=0}^{\infty} (\mathbf{I} - \mathbf{A}^T\mathbf{A})^k, \quad (7.147)$$

we can rewrite Eq. (7.146) in the same form as Eq. (7.141), namely

$$\hat{\mathbf{x}} = \sum_{k=0}^{\infty} (\mathbf{I} - \mathbf{A}^T\mathbf{A})^k \mathbf{A}^T \mathbf{y}. \quad (7.148)$$

Hence, the Landweber algorithm is equally applicable to solving overdetermined systems of linear equations.

### 7-11.3 Iterative Shrinkage and Thresholding Algorithm (ISTA)

For a linear system given by

$$\mathbf{y} = \mathbf{A} \mathbf{x}, \quad (7.149)$$

where  $\mathbf{x}$  is the unknown signal of length  $N$  and  $\mathbf{y}$  is the (possibly noisy) observation of length  $M$ , the LASSO cost functional is

$$\Lambda = \frac{1}{2} \sum_{n=0}^{N-1} (y[n] - (\mathbf{A}\mathbf{x})[n])^2 + \lambda \sum_{n=0}^{N-1} |x[n]|, \quad (7.150)$$

where matrix  $\mathbf{A}$  is  $M \times N$  and  $(\mathbf{A}\mathbf{x})[n]$  is the  $n$ th element of  $\mathbf{A}\mathbf{x}$ .

► In the system described by Eq. (7.149),  $\mathbf{x}$  and  $\mathbf{y}$  are generic input and output vectors. The Landweber algorithm provides a good estimate of  $\mathbf{x}$ , given  $\mathbf{y}$ . The estimation algorithm is equally applicable to any other linear system, including the system  $\mathbf{y} = \mathbf{A}_w \mathbf{z}$ , where  $\mathbf{A}_w$  is the matrix given by Eq. (7.115b) and  $\mathbf{z}$  is the wavelet transform vector. ◀

The ISTA algorithm combines the Landweber algorithm with the thresholding and shrinkage operation outlined earlier in Section 7-8.3, and summarized by Eq. (7.109). After each iteration, elements  $x^{(k)}[n]$ , of vector  $\mathbf{x}^{(k)}$ , whose absolute values are smaller than the trade-off parameter  $\lambda$  are thresholded to zero, and those whose absolute values are larger than  $\lambda$  are shrunk by  $\lambda$ . Hence, the ISTA algorithm combines Eq. (7.144) with Eq. (7.109):

$$\mathbf{x}^{(0)} = \mathbf{0}, \quad (7.151a)$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{A}^T(\mathbf{y} - \mathbf{A}\mathbf{x}^{(k)}), \quad (7.151b)$$

with

$$x_i^{(k+1)} = \begin{cases} x_i^{(k+1)} - \lambda & \text{if } x_i^{(k+1)} > \lambda, \\ x_i^{(k+1)} + \lambda & \text{if } x_i^{(k+1)} < -\lambda, \\ 0 & \text{if } |x_i^{(k+1)}| < \lambda, \end{cases} \quad (7.151c)$$

where  $x_i^{(k+1)}$  is the  $i$ th component of  $\mathbf{x}^{(k+1)}$ .

The ISTA algorithm converges to the value of  $\mathbf{x}$  that minimizes the LASSO functional given by Eq. (7.150), provided all of the eigenvalues  $\lambda_i$  of  $\mathbf{A}\mathbf{A}^T$  obey  $|\lambda_i| < 1$ .

The combination of thresholding and shrinking is often called **soft thresholding**, while thresholding small values to zero without shrinking is called **hard thresholding**. There are many varia-

tions on ISTA, with names like **SPARSA** (*sparse reconstruction by separable approximation*), **FISTA** (*fast iterative shrinkage and thresholding algorithm*), and **TWISTA** (*two-step iterative shrinkage and thresholding algorithm*).

**Concept Question 7-11:** Why do we need an iterative algorithm to find the LASSO-minimizing solution?

### 7-12 Compressed Sensing Examples

To illustrate the utility of the ISTA described in the preceding section, we present four examples of compressed sensing:

- **Reconstruction** of an image from *some*, but not all, of its 2-D DFT values.
- **Image inpainting**, which entails filling in holes (missing pixel values) in an image.
- **Valid deconvolution** of an image from only part of its convolution with a known point spread function (PSF).
- **Tomography**, which involves reconstruction of a 3-D image from slices of its 2-D DFT.

These are only a few of many more possible types of applications of compressed sensing.

The ISTA was used in all four cases, and the maximum number of possible iterations was set at 1000, or fewer if the algorithm converges to where no apparent change is observed in the reconstructed images. The LASSO functional parameter was set at  $\lambda = 0.01$ , as this value seemed to provide the best results.

#### 7-12.1 Image Reconstruction from Subset of DFT Values

Suppose that after computing the 2-D DFT of an image  $x[n, m]$ , some of the DFT values  $\mathbf{X}[k_1, k_2]$  were lost or no longer available. The goal is to reconstruct image  $x[n, m]$  from the partial subset of its DFTs. Since the available DFT values are fewer than those of the unknown signal, the system is underdetermined and the application is a good illustration of compressed sensing.

For a 1-D signal  $\{x[n], n = 0, 1, \dots, N-1\}$ , its DFT can be implemented by the matrix-vector product

$$\mathbf{y} = \mathbf{A} \mathbf{x}, \quad (7.152)$$

where the  $(k, n)$ th element of  $\mathbf{A}$  is  $A_{k,n} = e^{-j2\pi nk/N}$ , the  $n$ th element of vector  $\mathbf{x}$  is  $x[n]$ , and the  $k$ th element of vector  $\mathbf{y}$  is

$\mathbf{X}[k]$ . Multiplication of both sides by  $\mathbf{A}^H$  implements an inverse 1-D DFT within a factor  $1/N$ .

The 2-D DFT of an  $N \times N$  image can be implemented by multiplication by an  $N^2 \times N^2$  block matrix  $\mathbf{B}$  whose  $(k_2, n_2)$  block is the  $N \times N$  matrix  $\mathbf{A}$  multiplied by the scalar  $e^{-j2\pi n_2 k_2 / N}$ . So the element  $B_{k_1+Nk_2, n_1+Nn_2}$  of  $\mathbf{B}$  is  $e^{-j2\pi n_1 k_1 / N} e^{-j2\pi n_2 k_2 / N}$ , where  $0 \leq n_1, n_2, k_1, k_2 \leq N - 1$ .

The absence of some of the DFT values is equivalent to deleting some of the rows of  $\mathbf{B}$  and  $\mathbf{y}$ , thereby establishing an underdetermined linear system of equations. To illustrate the reconstruction process for this underdetermined system, we consider two different scenarios applied to the same set of data.

- ▶ For convenience, we call the values  $\mathbf{X}[k_1, k_2]$  of the 2-D DFT the *pixels of the DFT image*. ◀

### (a) Least-squares reconstruction

Starting with the  $(256 \times 256)$  image shown in Fig. 7-22(a), we compute its 2-D DFT and then we randomly select a subset of the pixels in the DFT image and label them as *unknown*. The complete 2-D DFT image consists of  $256^2 = 65536$  pixel values of  $\mathbf{X}[k_1, k_2]$ . Of those, 35755 are unaltered (and therefore have known values), and the other 29781 pixels have unknown values. Figure 7-22(b) displays the locations of pixels with known DFT values as white dots and those with unknown values as black dots.

In the least-squares reconstruction method, all of the pixels with unknown values are set to zero, and then the inverse 2-D DFT is computed. The resulting image, shown in Fig. 7-22(c), is a poor rendition of the original image in part (a) of the figure.

### (b) ISTA reconstruction

The ISTA reconstruction process consists of two steps:

(1) Measurement vector  $\mathbf{y}$ , representing the 35755 DFT pixels with known values, is used to estimate the (entire 65536) wavelet transform vector  $\mathbf{z}$  by applying the recipe outlined in Section 7-11.3 with  $\lambda = 0.01$  and 1000 iterations. The relationship between  $\mathbf{y}$  and  $\mathbf{z}$  is given by  $\mathbf{y} = \mathbf{A}_w \mathbf{z}$ , with  $\mathbf{A}_w = \mathbf{B} \mathbf{W}$  and some rows of  $\mathbf{B}$  deleted.

(2) Vector  $\mathbf{z}$  is then used to reconstruct  $\mathbf{x}$  by applying the relation  $\mathbf{x} = \mathbf{W}^{-1} \mathbf{z}$ .

The reconstructed image, displayed in Fig. 7-22(d), is an excellent rendition of the original image.

It is important to note that while  $\mathbf{B}$  and  $\mathbf{W}^{-1}$  are each  $(N^2 \times N^2)$ , with  $N = 65536$ , neither matrix is ever computed or

stored during the implementation of the reconstruction process. Multiplication by  $\mathbf{W}^{-1}$  is implemented by a 2-D filter bank, and multiplication by  $\mathbf{B}$  is implemented by a 2-D FFT. Consequently, ISTA is a very fast algorithm.

### 7-12.2 Image Inpainting

In an image inpainting problem, some of the pixel values of an image are unknown, either because those pixel values have been corrupted, or because they represent some unwanted feature of the image that we wish to remove. The goal is to restore the image to its original version, in which the unknown pixel values are replaced with the, hitherto, unknown pixel values of the original image. This can be viewed as a kind of interpolation problem.

It is not at all evident that this can be done at all—how can we restore unknown pixel values? But under the assumption that the Daubechies wavelet transform of the image is sparse (mostly zero-valued), image inpainting can be formulated as a compressed sensing problem. Let  $\mathbf{y}$  be the vector of the known pixel values and  $\mathbf{x}$  be the vector of the wavelet transform of the image. Note that all elements of  $\mathbf{x}$  are unknown, even though some of the pixel values are actually known. Then the problem can be formulated as an underdetermined linear system  $\mathbf{y} = \mathbf{Ax}$ .

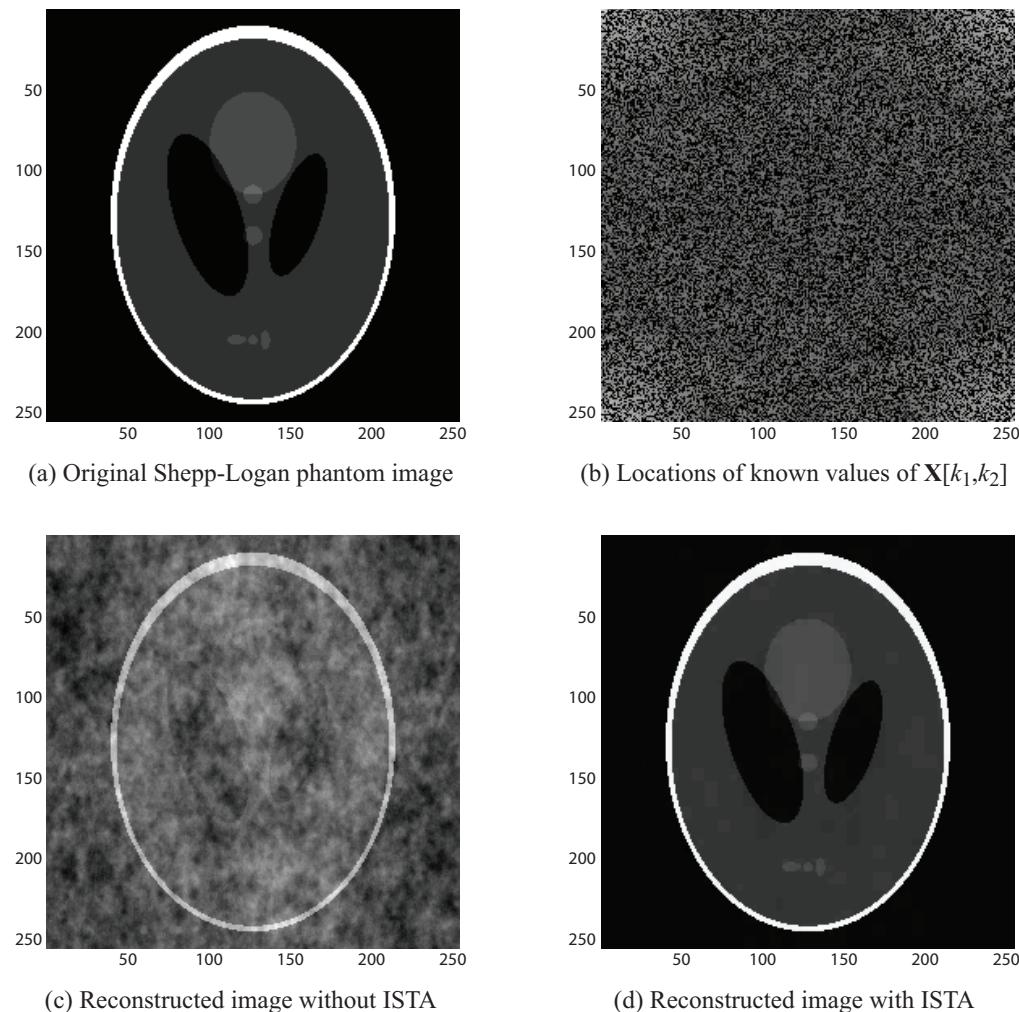
For example, if  $\mathbf{x}$  is a column vector of length five, and only the first, third, and fourth elements of  $\mathbf{x}$  are known, the problem can be formulated as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}.$$

One application of image inpainting is to restore a painting in which the paint in some regions of the painting has been chipped off, scraped off, damaged by water or simply faded, but most of the painting is unaffected. Another application is to remove unwanted letters or numbers from an image. Still another application is “wire removal” in movies, the elimination of wires used to suspend actors or objects used for an action stunt in a movie scene.

In all of these cases, damage to the painting, or presence of unwanted objects in the image, has made some small regions of the painting or image unknown. The goal is to fill in the unknown values to restore the (digitized) painting or image to its original version.

Using a  $(200 \times 200) = 40000$ -pixel clown image, 19723 pixels were randomly selected and their true values were deleted.

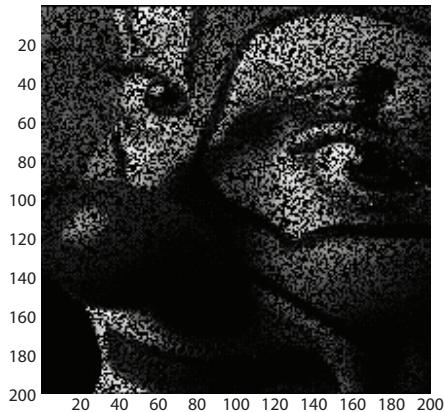


**Figure 7-22** (a) Original Shepp-Logan phantom image, (b) 2-D DFT image with locations of pixels of known values displayed in white and those of unknown values displayed in black, (c) reconstructed image using available DFT pixels and (d) reconstructed image after filling in missing DFT pixel values with estimates provided by ISTA.

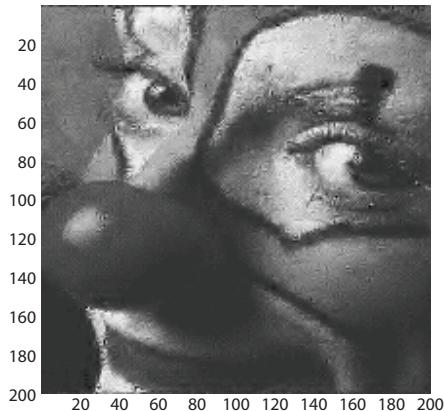
In the image shown in **Fig. 7-23(a)**, the locations of pixels with unknown values are painted black, while the remaining half (approximately) have their correct values. The goal is to reconstruct the clown image from the remaining half.

In terms of the formulation  $\mathbf{y} = \mathbf{AW}^T\mathbf{z}$ ,  $M = 20277$  and  $N = 40000$ , so that just over half of the clown image pixel

values are known. The ISTA is a good algorithm to solve this compressed sensing problem, since the matrix vector multiplication  $\mathbf{y} = \mathbf{AW}^T\mathbf{z}$  can be implemented quickly by taking the inverse wavelet transform of the current iteration (multiplication by  $\mathbf{W}^T$ ), and then selecting a subset of the pixel values (multiplication by  $\mathbf{A}$ ). The result, after 1000 iterations, is shown in



(a) Locations of known values of image



(b) Reconstructed (inpainting) image

**Figure 7-23** (a) Locations of known values of clown image in white and those of unknown values in black; (b) restored image.

$(M+L-1) \times (M+L-1)$  blurred image  $y[n,m]$ . The process is reversible: the blurred image can be deconvolved to reconstruct  $x[n,m]$  by subjecting  $y[n,m]$  to a Wiener filter, as described earlier in Section 6-4.3. To do so, however, requires that all of  $y[n,m]$  be known.

Often, we encounter deconvolution applications where only a fraction of  $y[n,m]$  is known, specifically, the part of  $y[n,m]$  called the **valid convolution**. This is the part whose convolution computation does not require the image  $x[n,m]$  to be zero-valued outside the square  $0 \leq n, m \leq M-1$ .

For  $L < M$  (image larger than PSF), the valid 2-D convolution of  $h[n,m]$  and  $x[n,m]$  is defined as

$$\begin{aligned} y_V[n,m] &= \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} x[i,j] h[n-i, m-j] \\ &= h[n,m] * x[n,m], \text{ restricted to} \\ &\quad \{L-1 \leq n, m \leq M-1\}. \end{aligned} \quad (7.153)$$

A valid convolution omits all end effects in 1-D convolution and all edge effects in 2-D convolution. Consequently, the size of  $y_V[n,m]$  is  $(M-L+1) \times (M-L+1)$ , instead of  $(M+L-1) \times (M+L-1)$  for the complete convolution  $y[n,m]$ .

To further illustrate the difference between  $y[n,m]$  and  $y_V[n,m]$ , let us consider the following example:

$$x[n,m] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

and

$$h[n,m] = \begin{bmatrix} 11 & 12 \\ 13 & 14 \end{bmatrix}.$$

Since  $M = 3$  and  $L = 2$ , the 2-D convolution is

$$(M+L-1) \times (M+L-1) = 4 \times 4,$$

and  $y[n,m]$  is given by

$$y[n,m] = \begin{bmatrix} 11 & 34 & 57 & 36 \\ 57 & 143 & 193 & 114 \\ 129 & 293 & 343 & 192 \\ 91 & 202 & 229 & 126 \end{bmatrix}.$$

In contrast, the size of the valid 2-D convolution is

$$(M-L+1) \times (M-L+1) = 2 \times 2,$$

### 7-12.3 Valid 2-D Deconvolution

#### (a) Definition of valid convolution

Given an  $M \times M$  image  $x[n,m]$  and an  $L \times L$  point spread function (PSF)  $h[n,m]$ , their 2-D convolution generates an

and  $yv[n, m]$  is given by

$$yv[n, m] = \begin{bmatrix} 143 & 193 \\ 293 & 343 \end{bmatrix}.$$

The valid convolution  $yv[n, m]$  is the central part of  $y[n, m]$ , obtained by deleting the edge rows and columns from  $y[n, m]$ . In MATLAB, the valid 2-D convolution of  $X$  and  $H$  can be computed using the command

```
Y=conv2(X, H, 'valid').
```

### (b) Reconstruction from $yv[n, m]$

The valid 2-D deconvolution problem is to reconstruct an unknown image from its valid 2-D convolution with a known PSF. The 2-D DFT and Wiener filter cannot be used here, since not all of the blurred image  $y[n, m]$  is known. It may seem that we may simply ignore, or set to zero, the unknown parts of  $y[n, m]$  and still obtain a decent reconstructed image using a Wiener filter, but as we will demonstrate with an example, such an approach does not yield fruitful results.

The valid 2-D deconvolution problem is clearly underdetermined, since the  $(M-L+1) \times (M-L+1)$  portion of the blurred image is smaller than the  $M \times M$  unknown image. But if  $x[n, m]$  is sparsifiable, then valid 2-D deconvolution can be formulated as a compressed sensing problem and solved using the ISTA. The matrix  $A$  turns out to be a **block Toeplitz with Toeplitz blocks** matrix, but multiplication by  $A$  is implemented as a valid 2-D convolution. Multiplication by  $A^T$  is implemented as a valid 2-D convolution.

The valid 2-D convolution can be implemented as  $yv = Ax$ , where

$$\begin{aligned} x &= [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9]^T, \\ yv &= [143 \ 193 \ 293 \ 343]^T, \end{aligned}$$

and the matrix  $A$  is composed of the elements of  $h[n, m]$  as follows:

$$A = \begin{bmatrix} 14 & 13 & 0 & 12 & 11 & 0 & 0 & 0 & 0 \\ 0 & 14 & 13 & 0 & 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 14 & 13 & 0 & 12 & 11 & 0 \\ 0 & 0 & 0 & 0 & 14 & 13 & 0 & 12 & 11 \end{bmatrix}.$$

Note that  $A$  is a  $2 \times 3$  block matrix of  $2 \times 3$  blocks. Each block is constant along its diagonals, and the blocks are constant along block diagonals. This is the block Toeplitz with Toeplitz blocks structure. Also note that images  $x[n, m]$  and  $yv[n, m]$  have been unwrapped row by row, starting with the top row, and the

transposes of the rows are stacked into a column vector. Finally, note that multiplication by  $A^T$  can be implemented as a valid 2-D convolution with the doubly reversed version of  $h[n, m]$ . For example, if

$$z[n, m] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

and

$$g[n, m] = \begin{bmatrix} 14 & 13 \\ 12 & 11 \end{bmatrix} = h[1-n, 1-m], \quad \text{with } n, m = 0, 1,$$

then the valid 2-D convolution of  $z[n, m]$  and  $g[n, m]$  is

$$wv[n, m] = \begin{bmatrix} 184 & 234 & 284 \\ 384 & 434 & 484 \\ 584 & 634 & 684 \end{bmatrix}.$$

This valid 2-D convolution can also be implemented as  $w = A^T z$  where

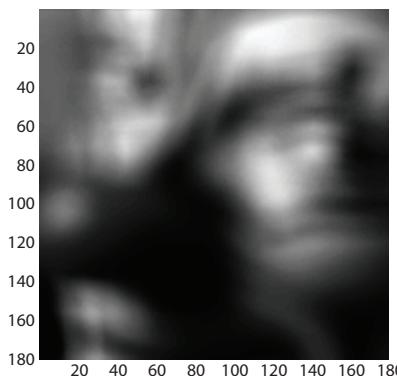
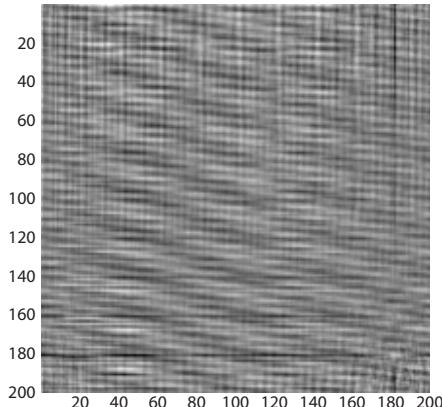
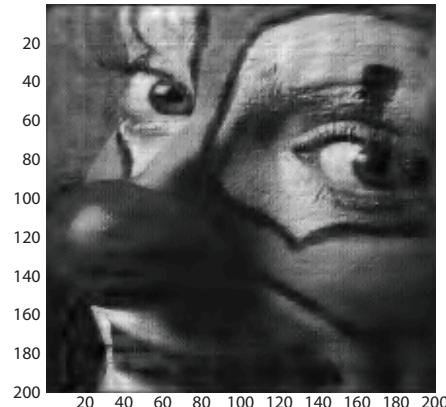
$$z = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16]^T$$

and

$$w = [184 \ 234 \ 284 \ 384 \ 434 \ 484 \ 584 \ 634 \ 684]^T.$$

To illustrate the process with an image, we computed the valid 2-D convolution of the  $(200 \times 200)$  clown image with a  $(20 \times 20)$  PSF. The goal is to reconstruct the clown image from the  $(181 \times 181)$  blurred image shown in Fig. 7-24(a). The db3 Daubechies wavelet function was used to sparsify the image. Here,  $M = 200$  and  $L = 20$ , so the valid 2-D convolution has size  $(M-L+1) \times (M-L+1) = 181 \times 181$ . In terms of  $yv = Ax$ ,  $A$  is  $(181^2 \times 200^2) = 32761 \times 40000$ .

Parts (b) and (c) of Fig. 7-24 show reconstructed versions of the clown image, using a Wiener filter and ISTA, respectively. Both images involve deconvolution using the restricted valid convolution data  $yv[n, m]$ . In the Wiener-image approach, the unknown parts of the blurred image (beyond the edges of  $yv[n, m]$ ) were ignored, and the resultant image bears no real resemblance to the original clown image. In contrast, the ISTA approach provides excellent reconstruction of the original image. This is because ISTA is perfectly suited for solving underdetermined systems of linear equations with sparse solutions.

(a) Valid 2-D convolution  $y_v[m,n]$  of clown image(b) 2-D deconvolution using  $y_v[m,n]$  and Wiener filter(c) 2-D deconvolution using  $y_v[m,n]$  and ISTA**Figure 7-24** (a) Valid 2-D convolution  $y_v[n,m]$  of clown image, (b) deconvolution using Wiener filter, and (c) deconvolution using ISTA.

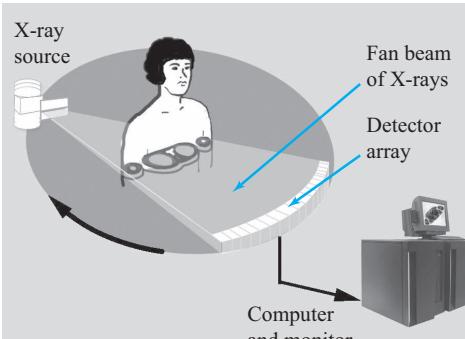
### 7-12.4 Computed Axial Tomography (CAT)

The basic operation of the CAT scanner was described in the opening chapter of this book using **Fig. 1-24**, which we reproduce here as **Fig. 7-25**. For a source-to-detection path through a body at a radius  $r$  and at orientation  $\theta$ , the **path attenuation** is given by Eq. (1.18) as

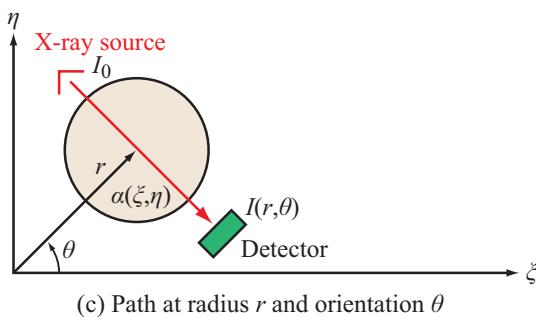
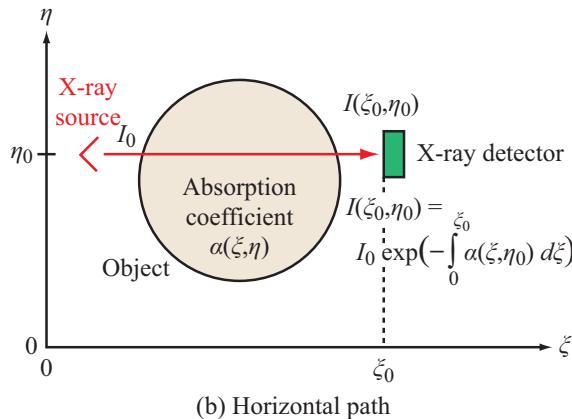
$$p(r, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \alpha(\xi, \eta) \delta(r - \xi \cos \theta - \eta \sin \theta) d\xi d\eta, \quad (7.154)$$

where  $\alpha(\xi, \eta)$  is the absorption coefficient of the body under test at location  $(\xi, \eta)$  in Cartesian coordinates or  $(r, \theta)$  in polar coordinates. The impulse function  $\delta(r - \xi \cos \theta - \eta \sin \theta)$  dictates that only those points in the  $(\xi, \eta)$  plane that fall along the path specified by fixed values of  $(r, \theta)$  are included in the integration.

The relation between  $p(r, \theta)$  and  $\alpha(\xi, \eta)$  is known as the 2-D **Radon transform** of  $\alpha(\xi, \eta)$ . The goal of CAT is to reconstruct  $\alpha(\xi, \eta)$  from the measured path attenuations  $p(r, \theta)$ , by inverting the Radon transform given by Eq. (7.154). We do so with the help of the Fourier transform.



(a) CAT scanner



**Figure 7-25** (a) CAT scanner, (b) X-ray path along  $\xi$ , and (c) X-ray path along arbitrary direction.

Recall from entry #1 in **Table 2-5** that for variable  $r$ ,  $\mathcal{F}\{\delta(r)\} = 1$ , and from entry #3 in **Table 2-4** that the shift property is

$$\mathcal{F}\{x(r - r_0)\} = \mathbf{X}(f) e^{-j2\pi f r_0}.$$

The combination of the two properties leads to

$$\begin{aligned} \mathcal{F}\{\delta(r - \xi \cos \theta - \eta \sin \theta)\} &= \int_0^\infty \delta(r - \xi \cos \theta - \eta \sin \theta) e^{-j2\pi f r} dr \\ &= e^{-j2\pi f (\xi \cos \theta + \eta \sin \theta)} = e^{-j2\pi(\mu\xi + \nu\eta)}, \end{aligned} \quad (7.155)$$

where we define spatial frequencies  $\mu$  and  $\nu$  as

$$\mu = f \cos \theta, \quad (7.156a)$$

$$\nu = f \sin \theta. \quad (7.156b)$$

Next, let us define  $\mathbf{A}(\mu, \nu)$  as the 2-D Fourier transform of the absorption coefficient  $\alpha(\xi, \eta)$  using the relationship given by Eq. (3.16a):

$$\mathbf{A}(\mu, \nu) = \int_{-\infty}^\infty \int_{-\infty}^\infty \alpha(\xi, \eta) e^{-j2\pi\mu\xi} e^{-j2\pi\nu\eta} d\xi d\eta. \quad (7.157)$$

If we know  $\mathbf{A}(\mu, \nu)$ , we can perform an inverse 2-D Fourier transform to retrieve  $\alpha(\xi, \eta)$ . To do so, we need to relate  $\mathbf{A}(\mu, \nu)$  to the measured path attenuation profiles  $p(r, \theta)$ . To that end, we use Eq. (7.154) to compute  $\mathbf{P}(f, \theta)$ , the 1-D Fourier transform of  $p(r, \theta)$ :

$$\begin{aligned} \mathbf{P}(f, \theta) &= \int_0^\infty p(r, \theta) e^{-j2\pi f r} dr \\ &= \int_0^\infty \left[ \int_{-\infty}^\infty \int_{-\infty}^\infty \alpha(\xi, \eta) \right. \\ &\quad \left. \cdot \delta(r - \xi \cos \theta - \eta \sin \theta) d\xi d\eta \right] e^{-j2\pi f r} dr. \end{aligned} \quad (7.158)$$

By reversing the order of integration, we have

$$\begin{aligned} \mathbf{P}(f, \theta) &= \int_{-\infty}^\infty \int_{-\infty}^\infty \alpha(\xi, \eta) \\ &\quad \cdot \left[ \int_0^\infty \delta(r - \xi \cos \theta - \eta \sin \theta) e^{-j2\pi f r} dr \right] d\xi d\eta. \end{aligned} \quad (7.159)$$

We recognize the integral inside the square bracket as the

Fourier transform of the shifted impulse function, as given by Eq. (7.155). Hence, Eq. (7.159) simplifies to

$$\mathbf{P}(f, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \alpha(\xi, \eta) e^{-j(2\pi f\xi + 2\pi v\eta)} d\xi d\eta, \quad (7.160)$$

which is identical to Eq. (7.157). Hence,

$$\mathbf{A}(\mu, v) = \mathbf{P}(f, \theta), \quad (7.161)$$

where  $\mathbf{A}(\mu, v)$  is the 2-D Fourier transform of  $\alpha(\xi, \eta)$ , and  $\mathbf{P}$  is the 1-D Fourier transform (with respect to  $r$ ) of  $p(r, \theta)$ . The variables  $(\mu, v)$  and  $(f, \theta)$  are related by Eq. (7.156).

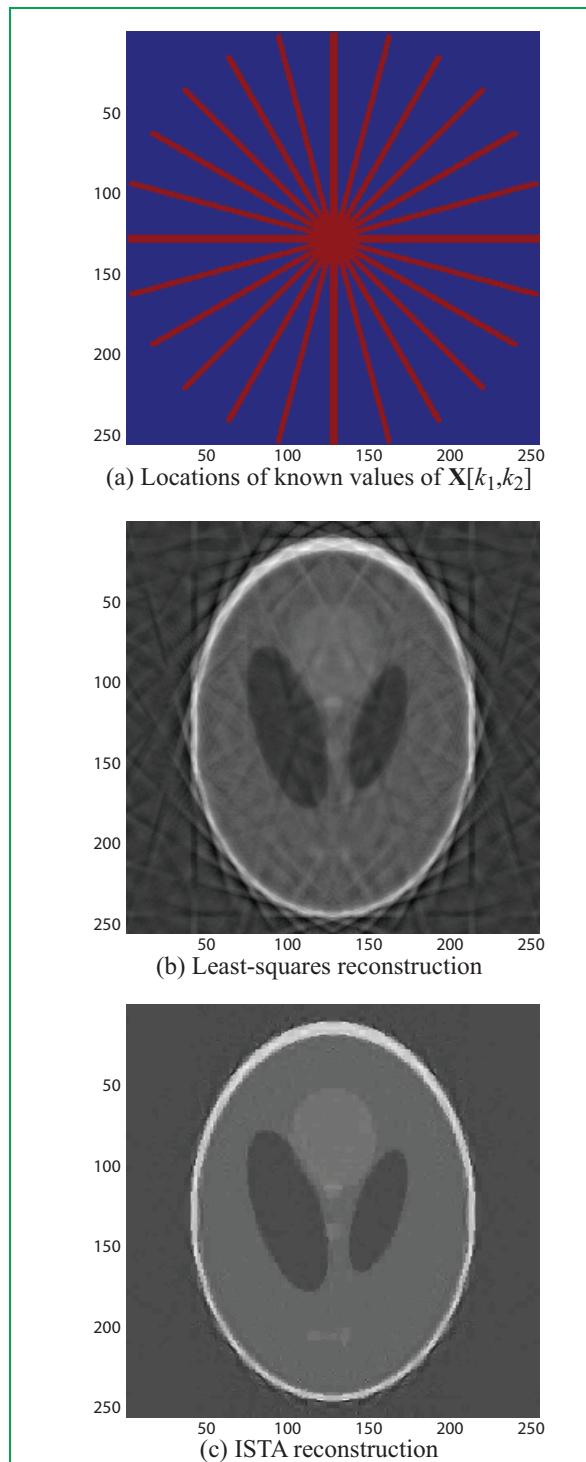
If  $p(r, \theta)$  is measured for all  $r$  across the body of interest and for all directions  $\theta$ , then its 1-D Fourier transform  $\mathbf{P}(f, \theta)$  can be computed, and then converted to  $\mathbf{A}(\mu, v)$  using Eq. (7.156). The conversion is called the **projection-slice theorem**. In practice, however,  $p(r, \theta)$  is measured for only a finite number of angles  $\theta$ , so  $\mathbf{A}(\mu, v)$  is known only along radial slices in the 2-D wavenumber domain  $(\mu, v)$ . Reconstruction to find  $\alpha(\xi, \eta)$  from a subset of its 2-D Fourier transform values is a perfect example of compressed sensing.

### Image reconstruction from partial radial slices

To demonstrate the CAT reconstruction process, we computed the 2-D DFT  $\mathbf{X}[k_1, k_2]$  of a  $256 \times 256$  Shepp-Logan phantom image, and then retained the data values corresponding to only 12 radial slices, as shown in Fig. 7-26(a). These radial slices simulate  $\mathbf{P}(f, \theta)$ , corresponding to 12 radial measurements  $p(r, \theta)$ . In terms of  $\mathbf{y} = \mathbf{Ax}$ , the number of pixels in the frequency domain image is  $N = 65536$ , and the number of values contained in the 12 radial slices is  $M = 11177$ . The Haar transform was used to sparsify the image.

**(a) Least-squares reconstruction:** Unknown values of  $\mathbf{X}[k_1, k_2]$  were set to zero, and then the inverse 2-D DFT was computed. The resulting image is displayed in Fig. 7-26(b).

**(b) ISTA reconstruction:** Application of ISTA with  $\lambda = 0.01$  for 1000 iterations led to the image in Fig. 7-26(c), which bears very good resemblance to the original image.



**Figure 7-26** Shepp-Logan phantom image reconstruction from partial radial slices of its 2-D DFT: (a) radial slices of  $\mathbf{X}[k_1, k_2]$ , (b) least-squares reconstruction, and (c) ISTA reconstruction.

## Summary

### Concepts

- The wavelet transform of an image is an orthonormal expansion of the image using basis functions that are localized in wavenumber or space.
- The 1-D wavelet transform of a piecewise-polynomial signal is sparse (mostly zero-valued). This is why it is useful.
- The wavelet and inverse wavelet transforms are implemented using filter banks and cyclic convolutions. This makes their computation very fast.
- The filters in the tree-structured filter banks used to implement wavelet and inverse wavelet transforms must satisfy the Smith-Barnwell condition for perfect reconstruction, and also form a quadrature-mirror pair.
- An image can be compressed by thresholding its 2-D

- wavelet transform.
- An image can be denoised by thresholding and shrinking its 2-D wavelet transform. This preserves edges while reducing noise. Thresholding and shrinkage minimizes the LASSO cost functional, which favors sparsity.
- Compressed sensing allows an image to be reconstructed from fewer linear combinations of its pixel values than the number of pixel values, using the ISTA algorithm or (rarely) basis pursuit.
- The ISTA algorithm applies thresholding and shrinkage at each iteration of the Landweber algorithm.
- Applications of compressed sensing include: tomography, image inpainting, and valid deconvolution.

### Mathematical Formulae

#### Zero-stuffing

$$y[n] = \begin{cases} x[n/2] & \text{for } n \text{ even} \\ 0 & \text{for } n \text{ odd} \end{cases}$$

#### Decimation

$$y[n] = x[2n]$$

#### QMF relation

$$g[n] = -(-1)^n h[L-n]$$

#### Smith-Barnwell condition

$$(h[n]*h[-n])(-1)^n + (h[n]*h[-n]) = 2\delta[n]$$

#### Haar functions

$$g[n] = \frac{1}{\sqrt{2}} \{1, 1\} \quad \text{and} \quad h[n] = \frac{1}{\sqrt{2}} \{1, -1\}$$

### Important Terms

Provide definitions or explain the meaning of the following terms:

average image

decimation

orthonormal basis

sparse

basis pursuit

detail image

quadrature mirror filter

subband decomposition

compressed sensing

Haar

pair

thresholding

cyclic convolution

ISTA algorithm

Shepp-Logan phantom

tree-structured filter

Daubechies

Landweber algorithm

shrinkage

banks

dbK

LASSO functional

Smith-Barnwell condition

zero-stuffing

## PROBLEMS

### Section 7-2: Expansions of Signals in Orthogonal Basis Functions

**7.1** The *continuous-time* Haar functions are defined as

$$\phi(t) = \begin{cases} 1 & \text{for } 0 < t < 1, \\ 0 & \text{otherwise,} \end{cases}$$

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

$$\psi_{m,n}(t) = 2^{m/2} \psi(2^m t - n).$$

Let  $B = \{\phi(t), \psi_{m,n}(t), m, n \text{ integers}\}$  and let  $F$  be the set of piecewise-constant functions with support (nonzero region)  $0 \leq t \leq 1$  whose values change at  $t = m/2^N$ .

- (a) Show that any member of  $F$  is a linear combination of elements of  $B$ .
- (b) Show that  $B$  is an orthonormal basis for  $F$ . Hint: Draw pictures.

**7.2** Let  $B = \{e^{j2\pi kt}, k \text{ integer}, 0 \leq t \leq 1\}$  and  $F$  be the set of continuous functions with support (nonzero region)  $0 \leq t \leq 1$ . Show that  $B$  is an orthonormal basis for  $F$ .

### Section 7-4: Haar Wavelet Transforms

**7.3** Let  $x[n] = \{4, 4, 4, 1, 1, 1, 1, 7, 7, 7, 7, 5, 5, 5, 4\}$ .

- (a) Compute all of the signals in the Haar analysis filter bank [Fig. 7-8](#).
  - (b) Check your answers using Rayleigh's (Parseval's) theorem. You may use MATLAB.
- 7.4** Let  $x[n] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$ .
- (a) Compute all of the signals in the Haar analysis filter bank [Fig. 7-8](#).
  - (b) Check your answers using Rayleigh's (Parseval's) theorem. You may use MATLAB.

### Section 7-5: Discrete-Time Wavelet Transforms

**7.5**  $h[n] = \{\underline{a}, b, \frac{1}{2}, -\frac{1}{2}\}$ . Find  $a, b$  such that  $h[n]$  satisfies Smith-Barnwell condition given by Eq. (7.57).

**7.6** If  $h[n] = \{\underline{a}, b, c, d\}$ , find  $g[n]$  such that  $g[n]$  and  $h[n]$  are a QMF pair given by Eq. (7.52a).

**7.7** Why are time-reversals used in the synthesis filters? Show that using  $g[n]$  and  $h[n]$  instead of  $g[-n]$  and  $h[-n]$  for the synthesis filters and then  $h[n] = (-1)^n g[n]$  for the QMF, perfect reconstruction is possible only if  $h[n]$  and  $g[n]$  have DTFTs

$$\mathbf{G}(\Omega) = \begin{cases} 1 & \text{for } 0 \leq \Omega < \frac{\pi}{2}, \\ 0 & \text{for } \frac{\pi}{2} < \Omega < \pi, \end{cases}$$

and

$$\mathbf{H}(\Omega) = \begin{cases} 0 & \text{for } 0 \leq \Omega < \frac{\pi}{2}, \\ 1 & \text{for } \frac{\pi}{2} < \Omega < \pi, \end{cases}$$

which constitutes an octave-band filter bank. Note that  $h[-n] = h[n]$  and  $g[-n] = g[n]$ . Hint: Replace  $g[-n]$  with  $g[n]$  and  $h[-n]$  with  $h[n]$  in Eq. (7.48).

**7.8** Repeat Problem 7.7, except now change the synthesis filters from  $g[-n]$  and  $h[-n]$  to  $g[n]$  and  $-h[n]$ , and then use  $h[n] = (-1)^n g[n]$  for the new QMF.

- (a) Show that one equation for perfect reconstruction is now automatically satisfied.
- (b) Show that the other equation still cannot be satisfied by any  $g[n]$ .

Hint: Replace  $g[-n]$  with  $g[n]$  and  $h[-n]$  with  $-h[n]$  in Eq. (7.48) and use Eq. (7.34).

### Section 7-6: Sparsification Using Wavelets of Piecewise Polynomial Signals

**7.9** Use the Smith-Barnwell condition given by Eq. (7.62) to design the db2 Daubechies wavelet function. Confirm that your answer matches the coefficients listed in [Table 7-3](#). Do this by equating coefficients of time  $n$ . You should get a large linear system of equations and a small nonlinear system of equations.

**7.10** Use the Smith-Barnwell condition given by Eq. (7.62) to design the db2 Daubechies wavelet function. Confirm that your answer matches the coefficients listed in [Table 7-3](#). Use  $q[n] = \{q[0], q[1]\} = q[0]\{\underline{1}, b\}$ , where  $b = q[1]/q[0]$ . This avoids the large linear systems of equations and the simultaneous quadratic equations of the previous problem.

### Section 7-7: 2-D Wavelet Transform

**7.11** Use [haar.m](#) to compute the 2-D Haar transform of the image in [letters.mat](#). Set `sigma=0` and `lambda=0` in the first line of [haar.m](#). Also depict the image reconstructed from the wavelet transform.

**7.12** Use daub.m to compute the 2-D db3 transform of the SAR image in sar.mat. Change the first line to

```
load sar.mat; sigma=0; lambda=0;
```

Also depict the image reconstructed from the wavelet transform.

### Section 7-8: Wavelet-Based Denoising by Thresholding and Shrinkage

**7.13** This problem investigates denoising the letters image using the wavelet transform, by thresholding and shrinking the 2-D Haar transform of the noisy image.

(a) Run haar.m. This adds noise to the image in letters.mat, computes its 2-D Haar transform, thresholds and shrinks this wavelet transform, computes the inverse 2-D Haar wavelet transform of the result, and displays images. The threshold and shrinkage uses  $\lambda = 70$ .

(b) Why does this work better than the 2-D DFT or convolution with a lowpass filter?

**7.14** This problem investigates denoising an MRI head image using the wavelet transform, by thresholding and shrinking the 2-D Haar transform of the noisy image.

(a) Run the program haar.m. This adds noise to the MRI image, computes its 2-D Haar transform, thresholds and shrinks this wavelet transform, computes the inverse 2-D Haar wavelet transform of the result, and displays images. Change load letters.mat to load mri.mat;. The threshold and shrinkage uses  $\lambda = 150$ .

(b) Why does this work better than the 2-D DFT or convolution with a lowpass filter?

**7.15** This problem investigates denoising an MRI head image using the wavelet transform, by thresholding and shrinking the 2-D db3 transform of the noisy image.

(a) Download and run the program daub.m. This adds noise to the MRI image, computes its 2-D db3 transform, thresholds and shrinks this wavelet transform, computes the inverse 2-D db3 wavelet transform of the result, and displays images. Change the first line to

```
load mri.mat; sigma=50; lambda=100;
```

The threshold and shrinkage uses  $\lambda = 100$ .

(b) Why does this work better than the 2-D DFT or convolution with a lowpass filter?

**7.16** This problem investigates denoising an SAR image using wavelet transforms, by thresholding and shrinking the 2-D db3 transform of the noisy image.

(a) Download and run the program daub.m. This adds noise to the SAR image, computes its 2-D db3 transform, thresholds and shrinks this wavelet transform, computes the inverse 2-D db3 wavelet transform of the result, and displays images. Change the first line to

```
load sar.mat; sigma=50; lambda=100;
```

The threshold and shrinkage uses  $\lambda = 100$ , and the signal-to-noise ratio is about 6.1.

(b) Why does this work better than the 2-D DFT or convolution with a lowpass filter?

### Section 7-9: Compressed Sensing

**7.17** Even if a compressed sensing problem is only slightly underdetermined, and it has a mostly sparse solution, there is no guarantee that the sparse solution is unique. The worst case for compressed sensing is as follows. Let:

(a)  $a_{m,n} = e^{-j2\pi mn/N}$ ,  $n = 0, \dots, N-1$ ,  $m = 0, \dots, N-1$ ; skip every multiple of  $N/L$ .

(b)

$$x_n = \begin{cases} 1 & \text{for } n \text{ is a multiple of } L, \\ 0 & \text{for } n \text{ not a multiple of } L. \end{cases}$$

(c) For  $N = 12$  and  $L = 4$ :  $m = \{1, 2, 4, 5, 7, 8, 10, 11\}$  and  $\{n = 0, 4, 8\}$ , so that in this case  $A$  is  $8 \times 12$  and the sparsity (number of nonzero  $x_n$ ) is  $K = 3$ .

Show that  $y_n = 0$ !  $\{x_n\}$  is called the *Dirac comb*. Its significance: Let  $x_n$  be any signal of length 12 with nonzero elements only at  $\{n = 0, 4, 8\}$ . Let  $y_m = \sum_{n=0}^{11} a_{m,n} x_n$ . Then adding the Dirac comb to  $\{x_n\}$  won't alter  $\{y_m\}$ , so  $\{x_n\}$  plus any multiple of the Dirac comb is a  $K$ -sparse solution.

### Section 7-10: Computing Solutions to Underdetermined Systems

**7.18** Derive the pseudoinverse given by Eq. (7.124), which is the solution  $\hat{x}$  to the overdetermined linear system of equations  $\underline{y} = A\underline{x}$  that minimizes  $E = \|\underline{y} - A\underline{x}\|_2^2$ . Perform the derivation by letting  $\underline{x} = \hat{\underline{x}} + \underline{\delta}$  for an arbitrary  $\underline{\delta}$  and showing that the coefficient of  $\underline{\delta}$  must be zero.

**7.19** Derive Eq. (7.125), which is the solution  $\hat{\underline{x}}$  to the underdetermined linear system of equations  $\underline{y} = A\underline{x}$  that minimizes  $E = \|\hat{\underline{x}}\|_2^2$ . Perform the derivation by minimizing the Tikhonov criterion given by Eq. (7.128), letting parameter  $\lambda \rightarrow 0$ , applying the pseudoinverse given by Eq. (7.124), and using the matrix identity  $(A^T A + \lambda^2 I)^{-1} A^T = A^T (A A^T + \lambda^2 I)^{-1}$ .

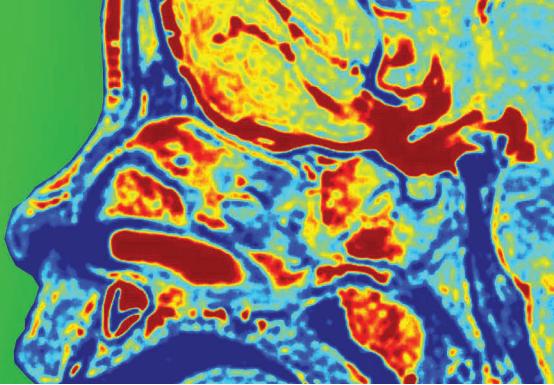
## Section 7-12: Compressed Sensing Examples

**7.20** *Free the clown from his cage.* Run the program `P7.20.m`. This sets horizontal and vertical bands of the clown image to zero, making it appear that the clown is confined to a cage. *Free the clown:* The program then uses inpainting to replace the bands of zeros with pixels by regarding the bands of zeros as unknown pixel values of the clown. Change the widths of the bands of zeros and see how this affects the reconstruction.

**7.21** *De-square the clown image.* Run program `P7.21.m`. This sets 81 small squares of the clown image to zero, decrating it. The program then uses inpainting to replace the small squares with pixels by regarding the 81 small squares as unknown pixel values of the clown. Change the sizes of the squares and see how this affects the reconstruction.

**7.22** The tomography example at the end of Chapter 7 used 24 rays. Download and run the program `P7.22.m`, which generated this example. Change the number of rays (`M`) from 24 to 32, and then 16. Compare the compressed sensing reconstruction results to the least-squares reconstructions (all unknown DFT values set to zero).

# CHAPTER 8



8

## Random Variables, Processes, and Fields

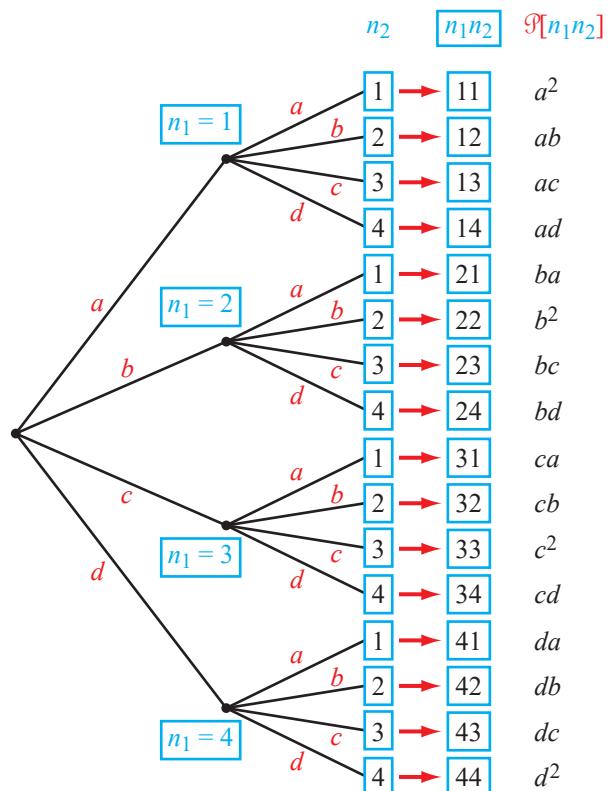
### Contents

- Overview, 255
- 8-1** Introduction to Probability, 255
- 8-2** Conditional Probability, 259
- 8-3** Random Variables, 261
- 8-4** Effects of Shifts on Pdfs and Pmfs, 263
- 8-5** Joint Pdfs and Pmfs, 265
- 8-6** Functions of Random Variables, 269
- 8-7** Random Vectors, 272
- 8-8** Gaussian Random Vectors, 275
- 8-9** Random Processes, 278
- 8-10** LTI Filtering of Random Processes, 282
- 8-11** Random Fields, 285
- Problems, 288

### Objectives

Learn to:

- Compute conditional probabilities and density functions.
- Compute means, variances, and covariances for random variables and vectors.
- Compute autocorrelations and power spectral densities of wide-sense-stationary random processes in continuous and discrete time.
- Use thresholding and shrinkage of its wavelet transform to denoise an image.
- Compute autocorrelations and power spectral densities of wide-sense-stationary random fields.



This chapter supplies a quick review of probability, random variables, vectors, processes, and fields for use in Chapter 9, which reviews estimation theory and applies it to image estimation. Readers already familiar with these topics may skip this chapter.

## Overview

In this chapter, we offer a brief review of **random variables**, **random processes**, and **random fields**. A random field is a random process in 2-D. These topics will all be used in Chapter 9, which covers **estimation** and **Markov random fields**. To begin our review, Section 8-1 provides a primer on the nomenclature used in the language of probability, illustrated with several examples.

Section 8-2 presents conditional probability, which will play an important role in Chapter 9. In Section 8-3, we introduce probability density functions (pdf) and probability mass functions (pmf) for describing the distributions of 1-D random variables, and then these tools are extended to 2-D in Sections 8-5 and 8-6. Sections 8-7 and 8-8 treat random vectors (vectors of random variables), which are then extended to 1-D random processes and 2-D random fields in later sections. A particular emphasis is placed on wide-sense stationary (WSS) random processes and fields.

## 8-1 Introduction to Probability

A probability experiment is an experiment in which the outcome is uncertain (**random**), but each possible outcome (**event**) has a given or computable probability (**likelihood**) of occurring.

**Sample Space  $\mathcal{S}$ :** The set of all distinguishable outcomes. For a coin flipped once,  $\mathcal{S} = \{H, T\}$ , where  $H$  denotes a “head” outcome and  $T$  denotes a “tail” outcome. In this case,  $\mathcal{S}$  consists of  $M = 2$  elements. For a coin flipped twice in a row,  $M = 4$  and

$$\mathcal{S} = \{E_1, E_2, E_3, E_4\}, \quad (8.1)$$

where  $E_1$  to  $E_4$  are outcomes (events) defined as:

$$\begin{aligned} E_1 &= \{H_1 H_2\}, & E_2 &= \{H_1 T_2\}, \\ E_3 &= \{T_1 H_2\}, & E_4 &= \{T_1 T_2\}, \end{aligned}$$

where  $H_1$  and  $H_2$  denote that the results of the first and second flips are heads, and similarly for tails  $T_1$  and  $T_2$ .

**Event Probability  $\mathbb{P}$ :** Each element of  $\mathcal{S}$ , such as  $E_1$  through  $E_4$ , is called an **event**. Events may also include combinations of elements, such as

$$E_a = \{E_2, E_4\} = \{H_1 T_2, T_1 T_2\},$$

which in this case represents the outcome that the second coin flip is a “tail,” regardless of the outcome of the first coin flip. Associated with each event is a certain probability determined by the conditions and constraints of the experiment. If each time

a coin is flipped, it is equally likely to produce a head or a tail, then the probabilities of all four events are the same, namely

$$\mathbb{P}[E_1] = \mathbb{P}[E_3] = \mathbb{P}[E_2] = \mathbb{P}[E_4] = \frac{1}{4}.$$

**Event Space  $\mathcal{A}$ :** The set of all subsets of sample space  $\mathcal{S}$  to which probabilities can be computed on the basis of the probabilities assigned to the elements of  $\mathcal{S}$ . If  $\mathcal{S}$  has a finite number of elements  $M$ , then  $\mathcal{A}$  has  $2^M$  elements, each of which is a subset of  $\mathcal{S}$ . Included in  $\mathcal{A}$  are both  $\mathcal{S}$  itself and the null set  $\emptyset$ ; that is,  $\mathcal{S} \in \mathcal{A}$  and  $\emptyset \in \mathcal{A}$ .

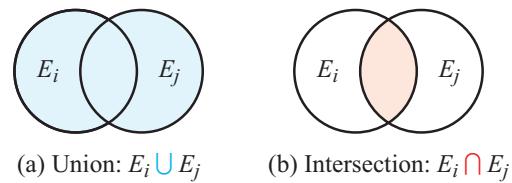
### 8-1.1 Complement, Union, and Intersection

Event  $E_2 = \{H_1 T_2\}$  represents the outcome that the first coin flip is a head and the second one is a tail, and its associated probability of occurrence is  $\mathbb{P}[E_2]$ . The **complement** of event  $E_2$  is  $E'_2$ , and it denotes the condition that the outcome is *not*  $H_1 T_2$ . Thus, the complement of an event includes all possible outcomes of  $\mathcal{S}$  except for that particular event:

$$\mathbb{P}[E'_2] = \mathbb{P}[\mathcal{S}] - \mathbb{P}[E_2] = 1 - \mathbb{P}[E_2], \quad (8.2)$$

where we used the probability axiom (Section 8-1.2) that the total probability of all events constituting  $\mathcal{S}$  is 1. Probability  $\mathbb{P}[E'_2]$  equals  $\mathbb{P}[E_1] + \mathbb{P}[E_3] + \mathbb{P}[E_4]$ , since only one of  $\{E_1, E_3, E_4\}$  can occur.

► The **union** of two events  $E_i$  and  $E_j$  is an *OR* statement:  $E_i$  occurring,  $E_j$  occurring, or both, as illustrated by **Fig. 8-1(a)**. ◀



**Figure 8-1** (a) The event of the **union** of two events  $E_i$  and  $E_j$  encompasses the combined elements of both events, whereas (b) the event of their **intersection** includes only the elements common to both of them.

► The **intersection** of events  $E_i$  and  $E_j$  is an AND statement (Fig. 8-1(b)), it is denoted  $E_i \cap E_j$ , and its associated probability  $\mathcal{P}[E_i \cap E_j]$  represents the condition that *both*  $E_i$  and  $E_j$  occur. ◀

► Probabilities are assigned by the user for all elements of the sample space  $S$ , consistent with the axioms of probability. Using these axioms, the probability of any element of the event space  $\mathcal{A}$  can then be computed. ◀

For the two-time coin-flip experiment, the occurrence of any one of the four events,  $E_1$  through  $E_4$ , negates the possibility of occurrence of the other three. Hence, if  $E_1 = H_1H_2$  and  $E_2 = H_1T_2$ ,

$$\mathcal{P}[E_1 \cap E_2] = \mathcal{P}[H_1H_2 \cap H_1T_2] = 0.$$

When the intersection  $E_i \cap E_j = \emptyset$ , the associated probability is

$$\mathcal{P}[E_i \cap E_j] = 0 \quad (\text{disjoint}), \quad (8.3)$$

in which case events  $E_i$  and  $E_j$  are said to be **disjoint**.

## 8-1.2 Axioms of Probability

The axioms of probability are rules for computing  $\mathcal{P}[E]$  for any member  $E$  of event space  $\mathcal{A}$ . The user must first assign probabilities to each member of the sample space  $S$  that satisfies those axioms.

**Axiom 1:** Probability of any event  $E_i$  is bounded between 0 and 1:

$$0 \leq \mathcal{P}[E_i] \leq 1. \quad (8.4)$$

**Axiom 2:** Total probability for all distinguishable events comprising  $S$  is 1:

$$\mathcal{P}[S] = \sum_{i=1}^M \mathcal{P}[E_i] = 1, \quad (8.5)$$

where  $M$  is the total number of elements in  $S$ .

**Axiom 3:** If  $E_i$  and  $E_j$  are disjoint, then  $\mathcal{P}[E_i \cap E_j] = 0$  and

$$\mathcal{P}[E_i \cup E_j] = \mathcal{P}[E_i] + \mathcal{P}[E_j] \quad (E_i \text{ and } E_j \text{ disjoint}). \quad (8.6)$$

Also, the probability of the union of an event  $E_i$  with its complement  $E'_i$  is 1:

$$\mathcal{P}[E_i \cup E'_i] = 1, \quad (8.7a)$$

thus stating that the total probability of the presence and absence of an event is 1. Similarly, the probability of the intersection of an event  $E_i$  and its complement  $E'_i$  is zero:

$$\mathcal{P}[E_i \cap E'_i] = 0. \quad (8.7b)$$

## 8-1.3 Properties of Unions and Intersections

### Commutative property

$$E_i \cup E_j = E_j \cup E_i, \quad (8.8a)$$

$$E_i \cap E_j = E_j \cap E_i. \quad (8.8b)$$

### Associative property

$$(E_i \cup E_j) \cup E_k = E_i \cup (E_j \cup E_k), \quad (8.9a)$$

$$(E_i \cap E_j) \cap E_k = E_i \cap (E_j \cap E_k). \quad (8.9b)$$

### Distributive property

$$(E_i \cup E_j) \cap E_k = (E_i \cap E_k) \cup (E_j \cap E_k), \quad (8.10a)$$

$$(E_i \cap E_j) \cup E_k = (E_i \cup E_k) \cap (E_j \cup E_k). \quad (8.10b)$$

### De Morgan's law

For two events  $E_i$  and  $E_j$ , De Morgan's law states:

$$(E_1 \cup E_2)' = E'_1 \cap E'_2, \quad (8.11a)$$

$$(E_1 \cap E_2)' = E'_1 \cup E'_2, \quad (8.11b)$$

and hence,

$$\mathcal{P}[(E_1 \cup E_2)'] = \mathcal{P}[E'_1 \cap E'_2], \quad (8.12a)$$

$$\mathcal{P}[(E_1 \cap E_2)'] = \mathcal{P}[E'_1 \cup E'_2], \quad (8.12b)$$

where as noted earlier, **the prime denotes the complement (absence) of the specified event**. De Morgan's law also leads to these useful relationships:

$$(E_1 \cap E_2) \cup (E'_1 \cap E_2) = (E_1 \cup E'_1) \cap E_2 = E_2, \quad (8.13a)$$

$$(E_1 \cap E_2) \cap (E'_1 \cap E_2) = (E_1 \cap E'_1) \cap E_2 = \emptyset. \quad (8.13b)$$

Because  $(E_1 \cap E_2)$  and  $(E'_1 \cap E_2)$  are disjoint, the probability relationship corresponding to Eq. (8.13a) is

$$\mathcal{P}[E_1 \cap E_2] + \mathcal{P}[E'_1 \cap E_2] = \mathcal{P}[E_2]. \quad (8.14)$$

### Probability of Union and Intersection Events

For two events, the probability of their union is equal to the sum of their individual probabilities minus the probability of their intersection:

$$\mathcal{P}[E_i \cup E_j] = \mathcal{P}[E_i] + \mathcal{P}[E_j] - \mathcal{P}[E_i \cap E_j]. \quad (8.15)$$

The relationship given by Eq. (8.15) can be derived by using **Fig. 8-1(a)** to write  $(E_i \cup E_j)$  as the union of three intersections:

$$E_i \cup E_j = (E_i \cap E'_j) \cup (E'_i \cap E_j) \cup (E_i \cap E_j), \quad (8.16)$$

and the corresponding probability is given by

$$\mathcal{P}[E_i \cup E_j] = \mathcal{P}[E_i \cap E'_j] + \mathcal{P}[E'_i \cap E_j] + \mathcal{P}[E_i \cap E_j]. \quad (8.17)$$

Upon setting  $E_1$  and  $E_2$  in Eq. (8.14) as  $E_i$  and  $E_j$ , respectively, we obtain

$$\mathcal{P}[E_i \cap E_j] + \mathcal{P}[E'_i \cap E_j] = \mathcal{P}[E_j]. \quad (8.18a)$$

Repeating the process but with  $E_1$  and  $E_2$  set in Eq. (8.14) as  $E_j$  and  $E_i$  (instead of as  $E_i$  and  $E_j$ ), respectively, leads to

$$\mathcal{P}[E_j \cap E_i] + \mathcal{P}[E'_j \cap E_i] = \mathcal{P}[E_i]. \quad (8.18b)$$

Using Eqs. (8.18a and b) in Eq. (8.17) leads to Eq. (8.15).

To illustrate the meaning of Eq. (8.14), let us consider the example of a coin toss with  $N = 2$  times. If we use  $E_1$  to denote that the first toss is a head and  $E_2$  to denote that the second toss is a tail, then Eq. (8.14) becomes

$$\mathcal{P}[H_1 \cap T_2] + \mathcal{P}[H'_1 \cap T_2] = \mathcal{P}[T_2]. \quad (8.19)$$

The first term is the probability that the first toss resulted in a head and the second toss resulted in a tail, and the second term is the probability that the first toss did not result in a head, but the second one did result in a tail. The sum of the two probabilities is equal to the probability that the second toss is a tail,  $\mathcal{P}[T_2]$ , regardless of the outcome of the first toss.

### 8-1.4 Probability Tree for Coin-Flip Experiment

Suppose a coin is flipped  $N$  times, and the result of the  $k$ th flip—with  $1 \leq k \leq N$ —is either a head and designated  $H_k$ , or a tail and designated  $T_k$ .

**Independent flips:** The result of any coin flip has no effect on the result of any other flip.

**Biased coin:** The coin is *not* necessarily a “fair” coin, meaning that the probability of heads is not necessarily equal to the probability of tails. Suppose the probability of heads for any coin flip is a known value  $a$ , where  $0 \leq a \leq 1$ . Thus,

$$\mathcal{P}[H_k] = a, \quad (8.20a)$$

$$\mathcal{P}[T_k] = 1 - a, \quad (8.20b)$$

$$1 \leq k \leq N.$$

For an unbiased coin,  $a = 0.5$ . A set designates a specific event, such as  $H_1 T_2$ , which can be written as

$$H_1 T_2 = H_1 \cap T_2.$$

For  $N = 2$ , the sample space  $S$  has  $2^N = 2^2 = 4$  elements:

$$S = \{H_1 \cap H_2, H_1 \cap T_2, T_1 \cap H_2, T_1 \cap T_2\},$$

and the event space  $A$  has  $2^{2^N} = 2^4 = 16$  elements:

$$\begin{aligned} & \{S, H_2, T_2, T_1 H_2, T_1 T_2, (T_1 T_2)', (T_1 H_2)', H_1 T_2 \cup T_1 H_2\} \cup \\ & \{\emptyset, H_1, T_1, H_1 H_2, H_1 T_2, (H_1 H_2)', (H_1 T_2)', H_1 H_2 \cup T_1 T_2\}. \end{aligned}$$

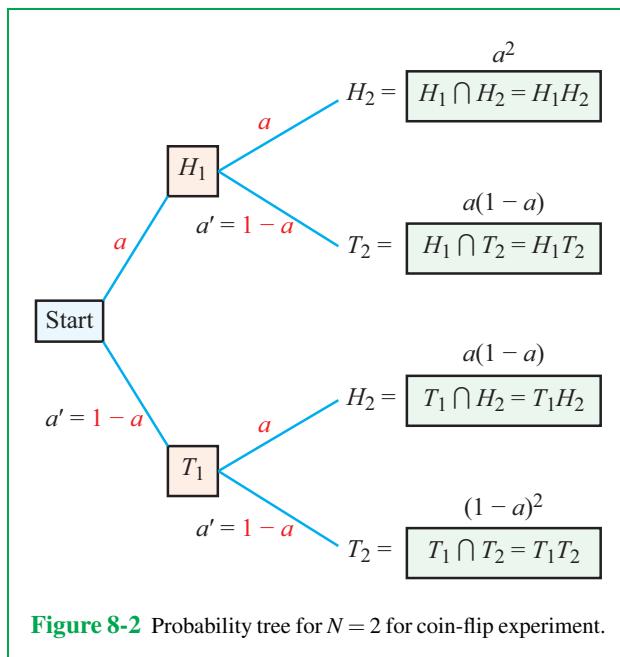
The **probability tree** of  $S$  is shown in **Fig. 8-2**. Each element of  $S$  and its assigned probability are denoted at the end of each branch of the tree. Examples of computing probabilities of union events are given in Example 8-1.

#### Example 8-1: Probability Tree for Coin-Flip Experiment

For the coin-flip experiment with  $N = 2$  and  $\mathcal{P}[H_k] = a$ , compute the probabilities of the following events: (a)  $H_1 \cup H_2$ , (b)  $H_1 H_2 \cup T_1 T_2$ , and (c)  $H_1 T_2 \cup T_1 H_2$ .

**Solution:** (a) According to Eq. (8.15),

$$\mathcal{P}[H_1 \cup H_2] = \mathcal{P}[H_1] + \mathcal{P}[H_2] - \mathcal{P}[H_1 \cap H_2].$$

**Figure 8-2** Probability tree for  $N = 2$  for coin-flip experiment.

From the tree in **Fig. 8-1**,

$$\mathbb{P}[H_1] = \mathbb{P}[H_2] = a$$

and

$$\mathbb{P}[H_1 \cap H_2] = a^2.$$

Hence,

$$\mathbb{P}[H_1 \cup H_2] = a + a - a^2 = a(2 - a).$$

**(b)**

$$\mathbb{P}[H_1H_2 \cup T_1T_2] = \mathbb{P}[H_1H_2] + \mathbb{P}[T_1T_2] - \mathbb{P}[H_1H_2 \cap T_1T_2].$$

From the tree in **Fig. 8-2**,  $\mathbb{P}[H_1H_2] = a^2$  and  $\mathbb{P}[T_1T_2] = (1 - a)^2$ . Furthermore,  $H_1H_2$  and  $T_1T_2$  are disjoint events, since they cannot both occur. Hence, the probability of their intersection is zero,

$$\mathbb{P}[H_1H_2 \cap T_1T_2] = 0,$$

and therefore

$$\mathbb{P}[H_1H_2 \cup T_1T_2] = a^2 + (1 - a)^2.$$

**(c)**

$$\mathbb{P}[H_1T_2 \cup T_1H_2] = \mathbb{P}[H_1T_2] + \mathbb{P}[T_1H_2] - \mathbb{P}[H_1T_2 \cap T_1H_2].$$

Using the tree in **Fig. 8-2** and noting that  $H_1T_2$  and  $T_1H_2$  are disjoint events, we have

$$\mathbb{P}[H_1T_2 \cup T_1H_2] = a(1 - a) + (1 - a)a = 2a(1 - a).$$

### 8-1.5 Probability Tree for Tetrahedral Die Experiment

A **tetrahedral die** is a four-sided object with one of the following four numbers: 1, 2, 3, 4, printed on each of its four sides (**Fig. 8-3**). When the die is rolled, the outcome is the number printed on the bottom side.

**Condition 1:** The result of any die roll has no effect on the result of any other roll.

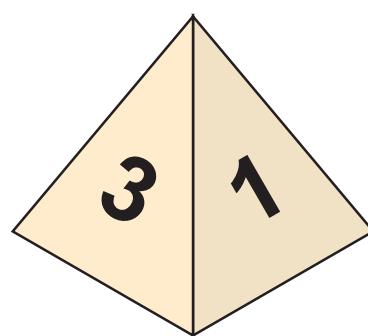
**Condition 2:** The die is not necessarily a “fair” die. If we denote  $n$  as the number that appears on the bottom of the die after it has been rolled, the probabilities that  $n = 1, 2, 3$ , or 4 are

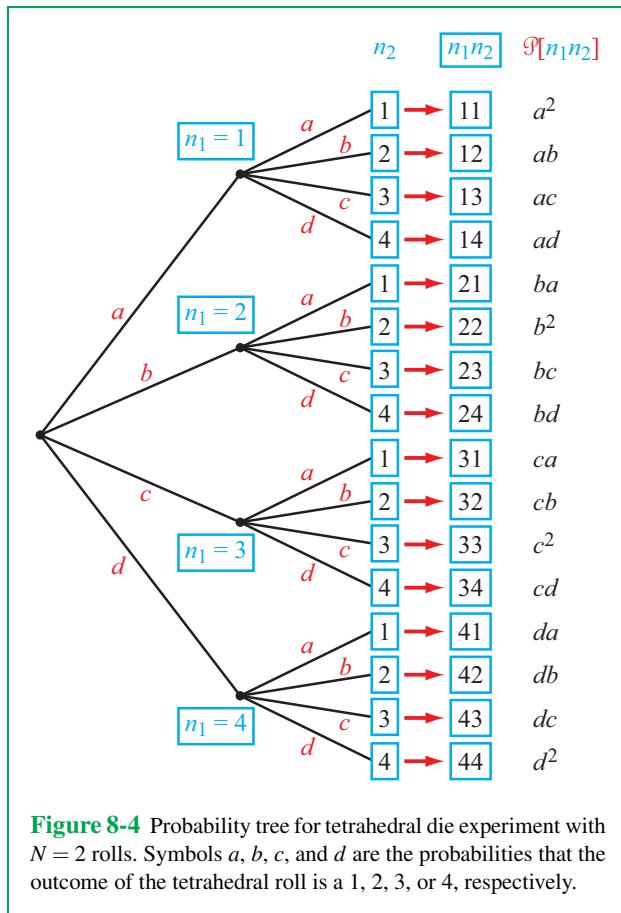
$$\begin{aligned}\mathbb{P}[n = 1] &= a, \\ \mathbb{P}[n = 2] &= b, \\ \mathbb{P}[n = 3] &= c, \\ \mathbb{P}[n = 4] &= d,\end{aligned}$$

with the constraints that  $0 \leq a, b, c, d \leq 1$  and

$$a + b + c + d = 1.$$

For a fair die,  $a = b = c = d = 1/4$ .

**Figure 8-3** Tetrahedral die with 4 sides displaying numerals 1 to 4. When the die is rolled, the outcome is the numeral on the bottom side.



**Figure 8-4** Probability tree for tetrahedral die experiment with  $N = 2$  rolls. Symbols  $a, b, c$ , and  $d$  are the probabilities that the outcome of the tetrahedral roll is a 1, 2, 3, or 4, respectively.

When rolled  $N$  times, the sample space  $\mathbb{S}$  of the tetrahedral die has  $4^N$  elements, each of which has the form  $\{n_1, n_2, \dots, n_N\}$ , where each  $n$  is one of the four numbers  $\{1, 2, 3, 4\}$ . For  $N = 2$ ,  $\mathbb{S}$  has  $4^2 = 16$  elements:

$$\{n_1 n_2\} = \{11, 12, 13, 14, 21, 22, 23, 24, 31, 32, 33, 34, 41, 42, 43, 44\}.$$

The probability tree for the tetrahedral die experiment is shown in **Fig. 8-4** for  $N = 2$ .

### Example 8-2: Tetrahedral Die Probabilities

For the tetrahedral die experiment with  $N = 2$ ,  $a = 0.1$ ,  $b = 0.2$ ,

$c = 0.3$ , and  $d = 0.4$ , compute the probabilities of (a)  $n_1$  is an even number and (b)  $n_1 + n_2 = 3$ .

**Solution:** (a) The probability of  $n_1 = \text{even number}$  is the sum of the probabilities for  $n_1 = 2$  and  $n_1 = 4$ :

$$\begin{aligned} P[n_1 = \text{even}] &= P[n_1 = 2] + P[n_1 = 4] \\ &= 0.2 + 0.4 = 0.6. \end{aligned}$$

(b) Using the tree in **Fig. 8-3**,

$$\begin{aligned} P[n_1 + n_2 = 3] &= P[n_1 = 1 \cap n_2 = 2] + P[n_1 = 2 \cap n_2 = 1] \\ &= ab + ba = 2ab = 2 \times 0.1 \times 0.2 = 0.04. \end{aligned}$$

**Concept Question 8-1:** What, exactly, is a probability tree?

**Exercise 8-1:** For the coin flip experiment, what is the pmf for  $n = \# \text{ flips of all tails followed by the first head}$ ?

**Answer:**  $p[n] = a(1 - a)^{(n-1)}$  since we require  $n - 1$  consecutive tails before the first head. This is called a **geometric pmf**.

## 8-2 Conditional Probability

For a given event, say  $E_1$ , its probability of occurrence is denoted  $P[E_1]$ . The value of  $P[E_1]$  is calculated assuming no foreknowledge that any other event has already occurred. Sometimes, the prior occurrence of another event, say  $E_3$ , may impact the probability of occurrence of the current event under consideration. Such a probability is called a **conditional probability**, designated

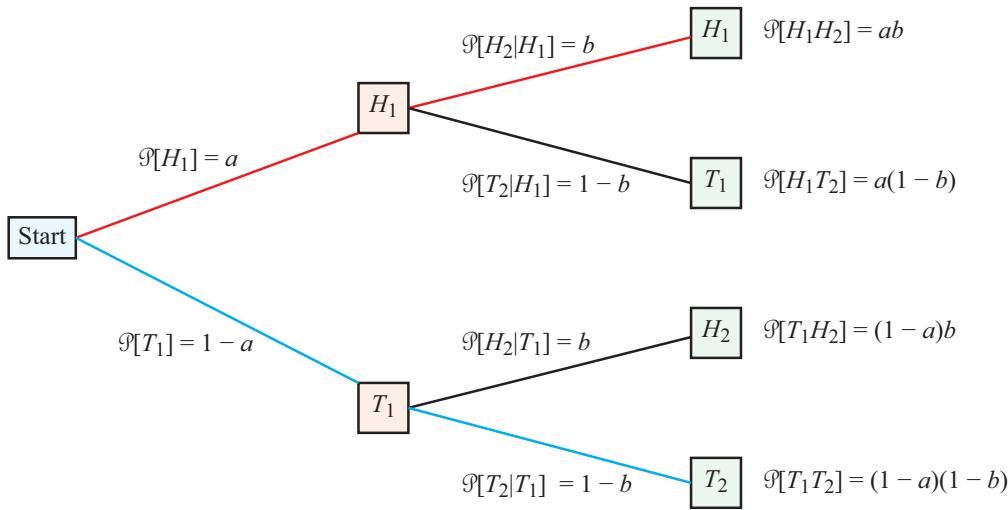
$$P[E_1 | E_3],$$

and is read as “probability of  $E_1$  given  $E_3$ .” **Figure 8-5** displays the conditional probability tree for the coin flip experiment with  $N = 2$ , and through Examples 8-3 and 8-4 we illustrate how to compute the conditional probability for specific scenarios.

### 8-2.1 Conditional Probability Formula

The general form of the conditional probability formula for events  $E_1$  and  $E_2$  is

$$P[E_1 | E_2] = \frac{P[E_1 \cap E_2]}{P[E_2]} = \frac{P[E_1 \cap E_2]}{P[E_1 \cap E_2] + P[E'_1 \cap E_2]}, \quad (8.21)$$



**Figure 8-5** Conditional probability tree for a coin-flip experiment with  $N = 2$ . The top red path represents the outcome  $H_1H_2$  and the bottom blue path represents the outcome  $T_1T_2$ .

where we used Eq. (8.14) in the second step.

and

$$\mathcal{P}[E_i \cup E_j] = \mathcal{P}[E_i] + \mathcal{P}[E_j] \quad (\text{mutually exclusive}). \quad (8.23b)$$

### 8-2-2 Independent and Mutually Exclusive Events

Two events  $E_i$  and  $E_j$  are said to be **independent** if the probability of occurrence of either has no effect on the occurrence of the other:

$$\mathcal{P}[E_i|E_j] = \mathcal{P}[E_i], \quad (8.22a)$$

$$\mathcal{P}[E_j|E_i] = \mathcal{P}[E_j], \quad (8.22b)$$

and

$$\mathcal{P}[E_i \cap E_j] = \mathcal{P}[E_i] \mathcal{P}[E_j]. \quad (8.22c)$$

(independent events)

In the coin-flip experiment, it was assumed that the outcome of each flip was independent of the outcomes of other flips. Hence,  $\mathcal{P}[H_1]$  is independent of  $\mathcal{P}[T_2]$ .

Two events are **mutually exclusive** if the occurrence of one of them negates the possibility of the occurrence of the other. Consequently,

$$\mathcal{P}[E_i \cap E_j] = 0 \quad (\text{mutually exclusive}) \quad (8.23a)$$

**Table 8-1** provides a summary of the major terms and symbols used in this section.

### Example 8-3: Coin-Flip Conditional Probability

Given that the result of the coin-flip experiment for  $N = 2$  was two heads or two tails, compute the conditional probability that the result was two heads. Use  $a = b = 0.4$ .

#### Solution:

$$\begin{aligned} E_1 &= H_1H_2 && (\text{both heads}), \\ E_2 &= H_1H_2 \cup T_1T_2, && (\text{both heads or both tails}). \end{aligned}$$

The conditional probability that the result is both heads is

$$\begin{aligned} \mathcal{P}[E_1|E_2] &= \frac{\mathcal{P}[E_1 \cap E_2]}{\mathcal{P}[E_2]} = \frac{\mathcal{P}[H_1H_2]}{\mathcal{P}[H_1H_2 \cup T_1T_2]} \\ &= \frac{\mathcal{P}[H_1H_2]}{\mathcal{P}[H_1H_2] + \mathcal{P}[T_1T_2]}, \end{aligned}$$

**Table 8-1** Probability symbols and terminology.

Term	Notation
Sample space	$\mathcal{S}$
Event (examples)	$E_1, E_2, \dots$
Outcome (examples)	$H, T; x_1 x_2$
Empty set (impossible event)	$\emptyset$
Complement of $E$ ("not $E$ ")	$E'$
Union of $E_i$ and $E_j$ (" $E_i$ or $E_j$ ")	$E_i \cup E_j$
Intersection of $E_i$ and $E_j$ (" $E_i$ and $E_j$ ")	$E_i \cap E_j$
$E_i$ and $E_j$ are independent	$\mathbb{P}[E_i \cap E_j] = \mathbb{P}[E_i] \mathbb{P}[E_j]$
$E_i$ and $E_j$ are mutually exclusive	$\mathbb{P}[E_i \cup E_j] = \mathbb{P}[E_i] + \mathbb{P}[E_j]$

where we used the relation  $(H_1 H_2) \cap (T_1 T_2) = 0$ . Using the probability tree in **Fig. 8-5** gives

$$\mathbb{P}[E_1 | E_2] = \frac{a^2}{a^2 + (1-a)^2} = \frac{0.4^2}{0.4^2 + (1-0.4)^2} = 0.31.$$

#### Example 8-4: Tetrahedral Die Conditional Probability

Given that the result of the tetrahedral die experiment of Example 8-2 was  $n_1 + n_2 = 3$ , compute the probability that  $n_1 = 1$ . Use  $a = 0.1$ ,  $b = 0.2$ ,  $c = 0.3$ , and  $d = 0.4$ .

**Solution:** From **Fig. 8-4**, there are two ways to obtain  $n_1 + n_2 = 3$ , namely

$$\begin{aligned} E_1 &= n_1 n_2 = 12, \\ E_2 &= n_1 n_2 = 21. \end{aligned}$$

We need to compute the conditional probability

$$\mathbb{P}[n_1 = 1 | n_1 + n_2 = 3].$$

Since satisfying both conditions requires that  $n_2 = 2$ , it follows

that

$$\begin{aligned} \mathbb{P}[n_1 = 1 | n_1 + n_2 = 3] &= \frac{\mathbb{P}[n_1 = 1 \cap n_2 = 2]}{\mathbb{P}[n_1 + n_2 = 3]} \\ &= \frac{\mathbb{P}[n_1 = 1 \cap n_2 = 2]}{\mathbb{P}[n_1 n_2 = 12] + \mathbb{P}[n_1 n_2 = 21]} \\ &= \frac{ab}{ab + ba} = 0.5. \end{aligned}$$

The last entries were obtained from the probability tree in **Fig. 8-4**. Note that because of the *a priori* knowledge that  $n_1 + n_2 = 3$ , the probability of  $n_1 = 1$  increased from 0.1 to 0.5.

**Concept Question 8-2:** Why does the conditional probability formula require division by  $\mathbb{P}[B]$ ?

**Exercise 8-2:** Given that the result of the coin flip experiment for  $N = 2$  was one head and one tail, compute  $\mathbb{P}[H_1]$ .

**Answer:**  $\mathbb{P}[H_1 | H_1 T_2 \cup T_1 H_2] = 0.5$ . (See [IP](#)).

## 8-3 Random Variables

The number of heads  $n$  among  $N$  flips is a **random variable**. A random variable is a number assigned to each possible outcome of a random experiment. The range of values that  $n$  can assume is from zero (no heads) to  $N$  (all heads). Another possible random variable is the number of consecutive pairs of heads among the  $N$  flips.

For the tetrahedral die experiment, our random variable might be the number of times among  $N$  tosses that the outcome is the number 3, or the number of times the number 3 is followed by the number 2, or many others. In all of these cases, the random variables have real discrete values, so we refer to them as **discrete random variables**. This is in contrast to **continuous random variables** in which the random variable may assume any value over a certain continuous range. We will examine the properties of both types of variables.

### 8-3.1 Probability Distributions for Continuous Random Variables

Since a continuous random variable can take on a continuum of values, such as the set of real numbers in an interval, the probability of a continuous random variable taking on a specific value is zero. To describe the distribution of a continuous random variable, we must use a density function, called a **probability**

**density function (pdf).** A pdf is *not* a probability; its units are the same as those of the random variable (e.g., distance). Instead, it describes the *relative* likelihood of the random variable taking on a specific value. It is analogous to mass density, rather than pure mass.

Let us consider the example shown in **Fig. 8-6**. Part (a) of the figure displays the height profile measured by a laser ranger as the beam was moved across 1 m of a ground surface. Relative to the mean surface, the height varies between about  $-20$  mm and  $+20$  mm, which means that the probability that the ground surface height may exceed this range is zero. If we count the number of times that each height occurs, we end up with the pdf shown in **Fig. 8-6(b)**. The measured pdf is discrete, but if the discretization is made infinitesimally small, the pdf becomes continuous. In the present case, the pdf is approximately Gaussian in shape. The horizontal axis denotes the surface height  $x$  (relative to the mean surface) and the vertical axis denotes  $p(x)$ , the probability density function. If a point on the surface is selected randomly, the probability that it has a height between  $5$  mm and  $5.001$  mm is  $0.001$  times the value of the pdf at  $5$  mm, which is  $0.0004$ .

Formally,  $p(x)$  for a continuous random variable is defined as

$$p(x') = \lim_{\delta \rightarrow 0} \frac{1}{\delta} \mathbb{P}[x' \leq x < x' + \delta]. \quad (8.24)$$

Thus, the probability of  $x$  lying within a narrow interval  $[x' < x < x' + \delta]$  is  $p(x) \delta$ . The **interval probability** that  $x$  lies between values  $a$  and  $b$  is

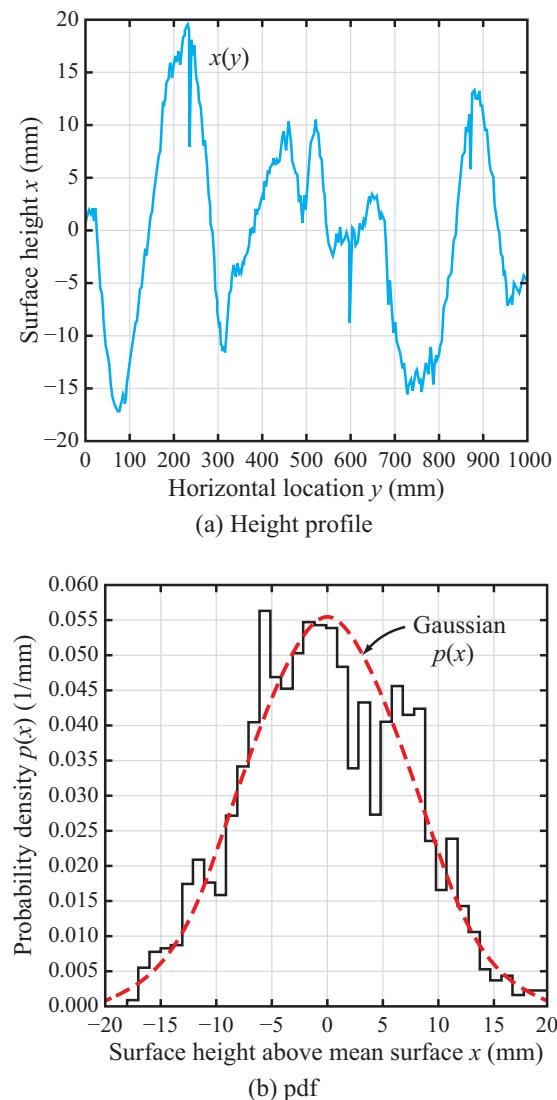
$$\mathbb{P}[a \leq x < b] = \int_a^b p(x') dx'. \quad (8.25a)$$

and the total probability over all possible values of  $x$  is

$$\int_{-\infty}^{\infty} p(x') dx' = 1. \quad (8.25b)$$

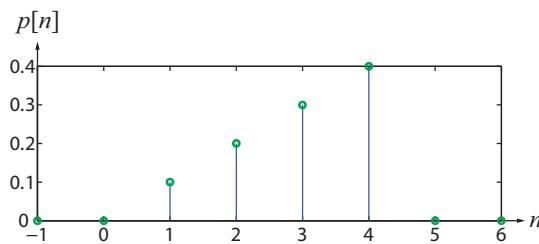
### 8-3.2 Probability Distributions for Discrete Random Variables

The term *pdf* is used to describe the probability function associated with a continuous random variable. The analogous function for a discrete random variable  $n$  is the **probability mass function (pmf)**,  $p[n]$ .



**Figure 8-6** (a) Measured height profile  $x(y)$  and (b) pdf of digitized height profile  $p(x)$ .

► Note that we use *curved brackets* for a pdf  $p(x)$  and *square brackets* for a pmf  $p[n]$ . ◀

Figure 8-7 pmf  $p[n]$  for the tetrahedral die experiment.

For a discrete random variable  $n$ ,

$$p[n'] = \mathbb{P}[n = n'], \quad (8.26)$$

where  $\mathbb{P}[n = n']$  is the probability that  $n$  has the value  $n'$ . By way of an example, Fig. 8-7 displays  $p[n]$  for a single toss of the tetrahedral die, with the random variable  $n$  representing the outcome of the toss, namely the number 1, 2, 3, or 4.

The **interval probability** that the value of  $n$  is between  $n' = n_a$  and  $n' = n_b - 1$ , inclusive of those limits, is

$$\mathbb{P}[n_a \leq n < n_b] = \sum_{n'=n_a}^{n_b-1} p[n'], \quad (8.27a)$$

and the total probability over all possible values of  $n'$  is

$$\sum_{n'=-\infty}^{\infty} p[n'] = 1. \quad (8.27b)$$

► The notation and properties of continuous and discrete random variables are summarized in Table 8-2. ◀

**Concept Question 8-3:** If  $x$  is a continuous random variable, what is  $\mathbb{P}[x = c]$  for any constant  $c$ ?

**Exercise 8-3:** Random variable  $x$  has the pdf

$$p(x) = \begin{cases} 2x & \text{for } 0 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Compute  $\mathbb{P}[x < 1/2]$ .

**Answer:**  $\mathbb{P}[x < 1/2] = \int_0^{1/2} 2x \, dx = 1/4$ .

**Exercise 8-4:** Random variable  $n$  has the pmf  $p[n] = (\frac{1}{2})^n$  for integers  $n \geq 1$ . Compute  $\mathbb{P}[n \leq 5]$ .

**Answer:**  $\mathbb{P}[n \leq 5] = \sum_{n=1}^5 (\frac{1}{2})^n = 31/32$ .

## 8-4 Effects of Shifts on Pdfs and Pmfs

### 8-4.1 Continuous Random Variable

If a random variable  $x$  characterized by a pdf  $p(x)$  is shifted by a constant amount  $\tau$  to form a new random variable  $y$ , with

$$y = x - \tau, \quad (8.28a)$$

then according to the rules of probability, the pdf of  $y$  is the pdf of  $x$  shifted by  $\tau$ :

$$p(y) = p(x - \tau). \quad (8.28b)$$

When computing mean values and other moments, the dummy variables used in integrations should be related by

$$y' = x' - \tau. \quad (8.28c)$$

### 8-4.2 Discrete Random Variable

For an integer  $k$  and two discrete random variables  $n$  and  $m$  related by

$$m = n - k, \quad (8.29a)$$

the pmf of  $m$  is related to the pmf of  $n$  by

$$\begin{aligned} p[m = m'] &= \mathbb{P}[m = m'] \\ &= \mathbb{P}[n - k = m'] \\ &= \mathbb{P}[n = m' + k] = p[n = n'] = p[n'], \end{aligned} \quad (8.29b)$$

where dummy variables  $n'$  and  $m'$  are related by

$$m' = n' - k. \quad (8.29c)$$

**Table 8-2** Notation and properties of continuous and discrete random variables.**A. Continuous Random Variable**

	$x$
pdf of $x$	$p(x)$
mean value of $x$	$\bar{x} = E[x] = \int_{-\infty}^{\infty} x' p(x') dx'$
mean value of $x^2$	$\bar{x^2} = E[x^2] = \int_{-\infty}^{\infty} (x')^2 p(x') dx'$
standard deviation of $x$	$\sigma_x = \sqrt{\bar{x^2} - \bar{x}^2}$
variance of $x$	$\sigma_x^2 = \bar{x^2} - \bar{x}^2$
interval probability over $(a, b)$	$P[a \leq x < b] = \int_a^b p(x') dx'$
shift property: $y = x - \tau$ , with $\tau = \text{constant}$	$p(y) = p(x)$ , with $y = x - \tau$
covariance of $x$ and $y$	$\lambda_{x,y} = \bar{xy} - \bar{x}\bar{y}$

**B. Discrete Random Variable**

	$n$
pmf of $n$	$p[n]$
mean value of $n$	$\bar{n} = \sum_{n'=-\infty}^{\infty} n' p[n']$
mean value of $n^2$	$\bar{n^2} = \sum_{n'=-\infty}^{\infty} (n')^2 p[n']$
standard deviation of $n$	$\sigma_n = \sqrt{\bar{n^2} - \bar{n}^2}$
variance of $n$	$\sigma_n^2 = \bar{n^2} - \bar{n}^2$
interval probability over $[n_a \leq n < n_b]$	$P[n_a \leq n < n_b] = \sum_{n'=n_a}^{n_b-1} p[n']$
shift property: $m = n - k$	$p[m] = p[n]$ , with $m = n - k$
covariance of $n$ and $m$	$\lambda_{n,m} = \bar{nm} - \bar{n}\bar{m}$

**Example 8-5: Binomial Pmf**

In the coin-flip experiment, define random variable  $n$  as the number of heads in  $N$  flips. Compute pmf  $p[n]$ , given that for any coin flip, the probability for a head outcome is  $a$ .

**Solution:** Since the results of different coin flips are inde-

pendent, the probability of any given sequence of results is the product of the probabilities of the individual results. The probability for a head is  $a$  and the probability for a tail is  $(1 - a)$ , and if the number of heads is  $n$ , the number of tails is  $(N - n)$ . Since multiplication of probabilities is commutative, the probability of any specific sequence of  $n$  heads and  $(N - n)$

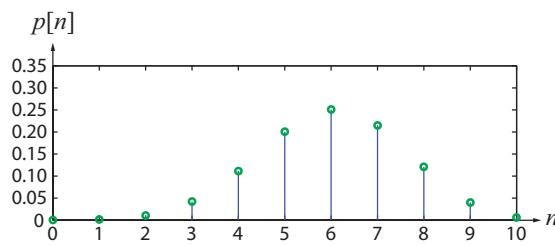


Figure 8-8 Binomial pmf  $p[n]$  for  $N = 10$  and  $a = 0.6$ .

tails, *in any order*, is

$$\mathcal{P}[n] = a^n (1-a)^{N-n} \quad (\text{single sequence}). \quad (8.30a)$$

The number of such possible sequences, denoted  $\binom{N}{n}$ , is

$$\binom{N}{n} = \frac{N!}{n!(N-n)!} \quad (\text{number of sequences}). \quad (8.30b)$$

The pmf  $p(n)$  is the product of the probability for an individual sequence and the number of sequences:

$$p(n) = \mathcal{P}[n] \binom{N}{n} = \frac{a^n (1-a)^{N-n} N!}{n!(N-n)!}. \quad (8.31)$$

This is known as the **binomial probability density function**. A plot of  $p(n)$  is shown in Fig. 8-8 for  $N = 10$  and  $a = 0.6$ .

The expression given by Eq. (8.31) is the pmf for a specific value of  $n$ . If we were to add the probabilities for all possible values of  $n$  (between 0 (no heads) and  $N$  (all heads)), among  $N$  tosses, then the sum should be 1. Indeed, a bit of algebra leads to the conclusion

$$\begin{aligned} \sum_{n'=0}^N p(n) &= \sum_{n'=0}^N \frac{a^{n'} (1-a)^{N-n'} N!}{n'!(N-n')!} \\ &= (1-a)^N + a(1-a)^{N-1}N \\ &\quad + \frac{a^2(1-a)^{N-2}N(N-1)}{2} + \cdots + a^N \\ &= (a + (1-a))^N = 1. \end{aligned}$$

In the last step, we used the binomial expansion of  $(a + (1-a))^N$ .

## 8-5 Joint Pdfs and Pmfs

This section introduces the joint pdf and joint pmf for two random variables, in continuous and discrete format. Atmospheric temperature and pressure are each a random variable that varies with time and the prevailing atmospheric conditions. Hence, each has its own pdf or pmf. The study of atmospheric phenomena requires knowledge of the statistics of both variables, including their joint pdf or pmf.

### 8-5.1 Continuous Random Variables

The **joint pdf** of two continuous random variables  $x$  and  $y$  is defined as

$$p(x=x', y=y') = \lim_{\delta \rightarrow 0} \frac{1}{\delta^2} \mathcal{P}[x' \leq x < x' + \delta, y' \leq y < y' + \delta]. \quad (8.32)$$

The definition is extendable to any number of random variables.

The interval probability over the range  $(a_x \leq x < b_x)$  and  $(a_y \leq y < b_y)$  is

$$\mathcal{P}[a_x \leq x < b_x, a_y \leq y < b_y] = \int_{a_x}^{b_x} \int_{a_y}^{b_y} p(x', y') dx' dy'. \quad (8.33)$$

Additionally, if we extend the limits to  $\pm\infty$ , we have

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x', y') dx' dy' = 1. \quad (8.34)$$

The **marginal pdfs**  $p(x)$  and  $p(y)$  are related to the joint pdf through

$$p(x) = \int_{-\infty}^{\infty} p(x, y') dy' \quad (\text{marginal pdf for } x), \quad (8.35a)$$

$$p(y) = \int_{-\infty}^{\infty} p(x', y) dx' \quad (\text{marginal pdf for } y). \quad (8.35b)$$

The **conditional pdf** for random variable  $x$ , given random variable  $y$ , is given by

$$\begin{aligned} p(x=x' | y=y') &= \lim_{\delta \rightarrow 0} \frac{\frac{1}{\delta^2} \mathcal{P}[x' \leq x < x' + \delta, y' \leq y < y' + \delta]}{\frac{1}{\delta} \mathcal{P}[y' \leq y < y' + \delta]} \\ &= \frac{p(x, y)}{p(y)} \quad (\text{conditional pdf}). \quad (8.36a) \end{aligned}$$

Similarly,

$$p(y|x) = \frac{p(x,y)}{p(x)}. \quad (8.36b)$$

If variables  $x$  and  $y$  are statistically **independent**, then

$$p(x,y) = p(x) p(y), \quad (8.37a)$$

in which case

$$p(x|y) = p(x) \quad (\text{independent variables}). \quad (8.37b)$$

The **Gaussian** (or **normal**) pdf is an important density function because it is applicable to the probabilistic behavior of many physical variables. If random variables  $x$  and  $y$  are each characterized by a Gaussian pdf, then

$$p(x) = \frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-(x-\bar{x})^2/2\sigma_x^2} \quad (8.38a)$$

and

$$p(y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-(y-\bar{y})^2/2\sigma_y^2}, \quad (8.38b)$$

where  $\bar{x}$  and  $\bar{y}$  are the mean values of  $x$  and  $y$ , respectively, and  $\sigma_x$  and  $\sigma_y$  are their associated standard deviations. Moreover, if  $x$  and  $y$  are independent, then their joint pdf is

$$\begin{aligned} p(x,y) &= p(x) p(y) \\ &= \frac{1}{2\pi\sigma_x\sigma_y} e^{-(x-\bar{x})^2/2\sigma_x^2} e^{-(y-\bar{y})^2/2\sigma_y^2}. \end{aligned} \quad (8.39)$$

Part (a) of **Fig. 8-9** displays a 1-D plot of the Gaussian pdf with  $\bar{x} = 2$  and  $\sigma_x = 0.45$ , and part (b) displays a 2-D Gaussian with  $\bar{x} = \bar{y} = 0$  and  $\sigma_x = \sigma_y = 1.2$ .

### Example 8-6: Triangle-Like Joint Pdf

Given that

$$p(x,y) = \begin{cases} C & \text{for } 0 \leq x \leq y \leq 1, \\ 0 & \text{otherwise,} \end{cases} \quad (8.40)$$

compute (a) constant  $C$ , (b) the marginal pdf  $p(x)$ , and (c) the conditional pdf  $p(y|x)$  at  $x = 3/4$ .

**Solution:** (a) We note from the definition of  $p(x,y)$  that  $x$  can extend between 0 and 1, but  $y$  extends only between whatever

value  $x$  is and up to 1. The domain of  $p(x,y)$  is the shaded triangle in **Fig. 8-10**.

Since the total probability is 1, it follows that

$$\begin{aligned} 1 &= \int_0^1 \left[ \int_{x'}^1 p(x',y') dy' \right] dx' \\ &= \int_0^1 \left[ \int_{x'}^1 C dy' \right] dx' \\ &= \int_0^1 \left( Cy' \Big|_{x'}^1 \right) dx' = \int_0^1 C(1-x') dx' = \frac{C}{2}. \end{aligned} \quad (8.41)$$

Hence,  $C = 2$ .

(b) With  $C = 2$ , application of Eq. (8.35a) leads to

$$\begin{aligned} p(x) &= \int_x^1 p(x,y') dy' \\ &= \int_x^1 2 dy' = \begin{cases} 2(1-x) & 0 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \quad (8.42)$$

(c) Using the results of parts (a) and (b) in Eq. (8.36b), the conditional pdf  $p(y|x)$  is

$$p(y|x) = \frac{p(x,y)}{p(x)} = \begin{cases} \frac{2}{2(1-x)} & 0 \leq x \leq y \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

For  $x = 3/4$ ,

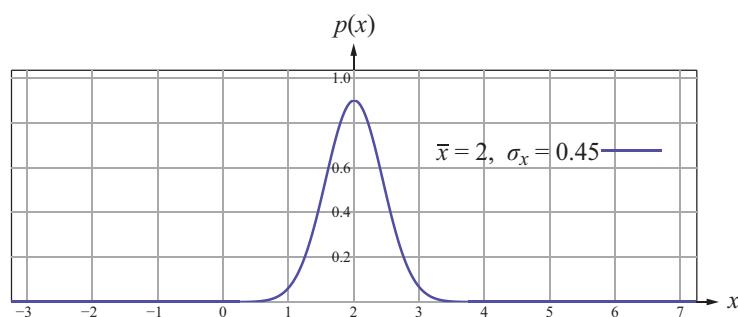
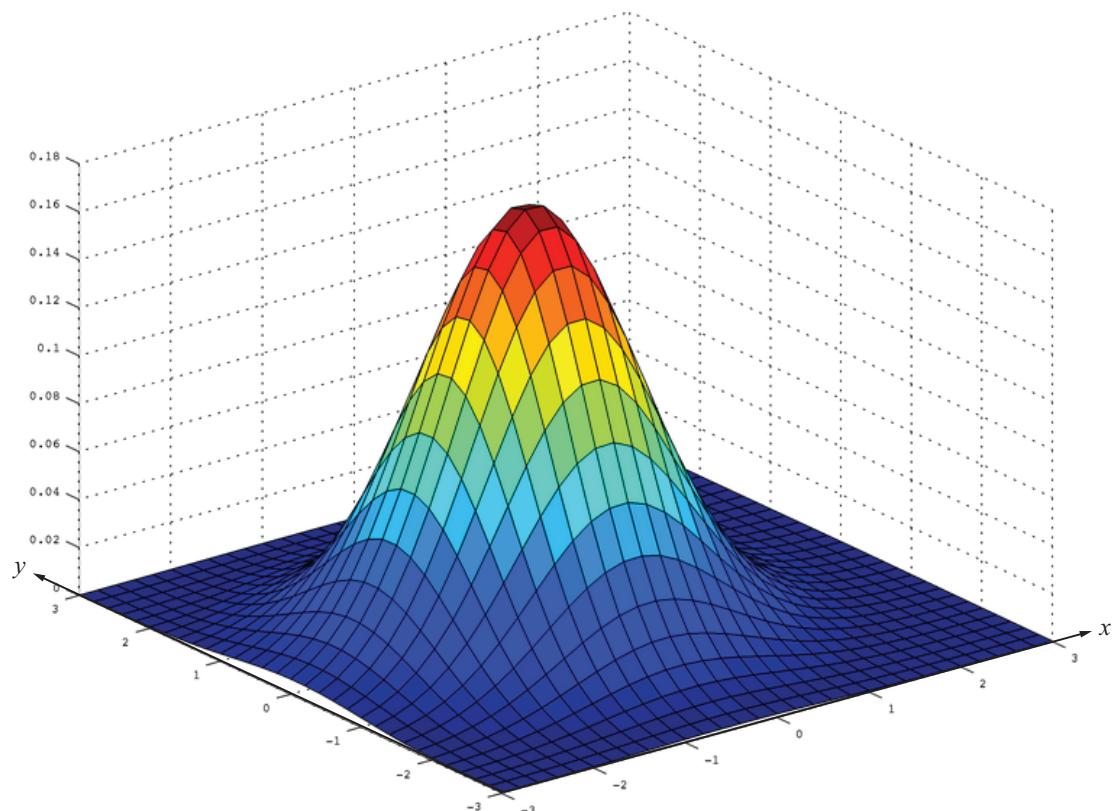
$$p(y|x=3/4) = \begin{cases} 4 & 3/4 \leq y \leq 1, \\ 0 & \text{otherwise.} \end{cases} \quad (8.43)$$

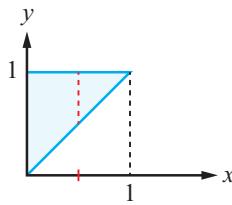
Note that both  $p(x)$  and  $p(y|x=3/4)$  integrate to 1, as required by Eq. (8.5):

$$\begin{aligned} \int_0^1 p(x') dx' &= \int_0^1 2(1-x') dx' = 1, \\ \int_{3/4}^1 p(y' | x=3/4) dy' &= \int_{3/4}^1 4 dy' = 1. \end{aligned}$$

## 8-5.2 Discrete Random Variables

For two discrete random variables  $n$  and  $m$ , their **joint probability mass function** is  $p[n,m]$ . The probability that  $n$  is in the range between  $n_a$  and  $n_b - 1$ , inclusive of those limits, and  $m$  is in the

(a) 1-D Gaussian plot with  $\bar{x}=2$  and  $\sigma_x=0.45$ .(b) 2-D Gaussian pdf with  $\bar{x}=\bar{y}=0$  and  $\sigma_x=\sigma_y=1.2$ .**Figure 8-9** Gaussian pdfs in (a) 1-D and (b) 2-D.



**Figure 8-10** The domain of joint pdf  $p(x,y)$  of Example 8-6; for each value of variable  $x$ , variable  $y$  extends from that value to 1.

range between  $m_a$  and  $m_b - 1$ , also inclusive of those limits, is

$$\mathbb{P} \left[ \begin{array}{l} n_a \leq n < n_b \\ m_a \leq m < m_b \end{array} \right] = \sum_{n'=n_a}^{n_b-1} \sum_{m'=m_a}^{m_b-1} p[n', m']. \quad (8.44a)$$

As required by the axioms of probability, when the limits on the double sum are extended to  $\pm\infty$ , the total probability adds up to 1:

$$\sum_{n'=-\infty}^{\infty} \sum_{m'=-\infty}^{\infty} p[n', m'] = 1. \quad (8.44b)$$

In analogy with the expressions given by Eq. (8.35) for the marginal pdfs in the continuous case, the **marginal pmfs**  $p[n]$  and  $p[m]$  in the discrete case are related to the joint pmf by

$$p[n] = \sum_{m'=-\infty}^{\infty} p[n, m'], \quad (8.45a)$$

$$p[m] = \sum_{n'=-\infty}^{\infty} p[n', m]. \quad (8.45b)$$

Finally, the **conditional pmf** for random variable  $n$ , given random variable  $m$ , is

$$p[n|m] = \frac{p[n, m]}{p[m]}. \quad (8.46)$$

Of course, if  $n$  and  $m$  are independent random variables, then  $p[n, m] = p[n] p[m]$ , in which case

$$p[n|m] = p[n] \quad (\text{n and m independent}). \quad (8.47)$$

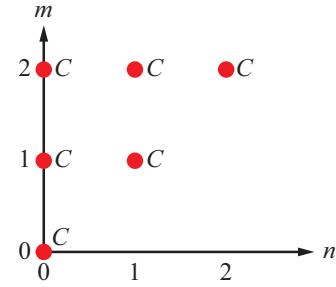
### Example 8-7: Discrete Version of Example 8-6

Given that

$$p[n, m] = \begin{cases} C & \text{for } 0 \leq n \leq m \leq 2, \\ 0 & \text{otherwise,} \end{cases}$$

compute (a) constant  $C$ , (b) the marginal pmf  $p[n]$ , and (c) the conditional pmf  $p[m|n]$  at  $n = 1$ .

**Solution:** (a) The joint pmf, depicted in **Fig. 8-11**, is the discrete equivalent of the joint pdf displayed in **Fig. 8-10**.



**Figure 8-11** Depiction of joint pmf  $p[n, m]$  in Example 8-7.

Since the total probability is 1, it follows that

$$1 = \sum_{n'=0}^2 \sum_{m'=0}^2 p[n', m'] = \underbrace{\sum_{m'=0}^2}_{n'=0} C + \underbrace{\sum_{m'=1}^2}_{n'=1} C + \underbrace{\sum_{m'=2}^2}_{n'=2} C = 3C + 2C + C = 6C.$$

Hence,  $C = 1/6$ .

(b) With  $C = 1/6$ , application of Eq. (8.45a) leads to

$$p[n] = \sum_{m'=n}^2 p[n, m'] = \sum_{m'=n}^2 \frac{1}{6} = \begin{cases} 3/6 & \text{for } n = 0, \\ 2/6 & \text{for } n = 1, \\ 1/6 & \text{for } n = 2. \end{cases}$$

(c) Using Eq. (8.46), but with  $n$  and  $m$  interchanged, the conditional probability  $p[m|n]$  is

$$p[m|n] = \frac{p[m, n]}{p[n]} = \frac{1/6}{p[n]}.$$

To evaluate  $p[m|n]$  at  $n = 1$ , we should note that since  $n \leq m$ , the range of  $m$  becomes limited to  $[1, 2]$ . Hence,

$$p[m|1] = \frac{1/6}{p[n=1]} = \begin{cases} \frac{1/6}{2/6} = \frac{1}{2} & \text{for } m = 1, \\ \frac{1/6}{2/6} = \frac{1}{2} & \text{for } m = 2, \\ 0 & \text{otherwise.} \end{cases}$$

We note that the total conditional probability adds up to 1:

$$\sum_{m=1}^2 p[m | n = 1] = \frac{1}{2} + \frac{1}{2} = 1.$$

**Concept Question 8-4:** Since  $\mathbb{P}[y = c] = 0$  for any continuous random variable  $y$  and constant  $c$ , how can  $p(x|y)$  make sense?

**Exercise 8-5:** In Example 8-6, compute the marginal pdf  $p(y)$  and conditional marginal pdf  $p(x|y)$ .

**Answer:**

$$p(y) = \int_0^y 2 dx = \begin{cases} 2y & \text{for } 0 \leq y \leq 1, \\ 0 & \text{otherwise,} \end{cases}$$

$$p(x|y) = \frac{p(x,y)}{p(y)} = \begin{cases} \frac{2}{2y} & \text{for } 0 \leq x < y \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

As expected, this becomes an impulse if  $y = 0$ .

**Exercise 8-6:** In Example 8-7, compute the marginal pmf  $p[m]$  and conditional marginal pmf  $p[n|m]$ .

**Answer:**

$$p[m] = \sum_{n'=0}^m \frac{1}{6} = \begin{cases} 1/6 & \text{for } m = 0, \\ 2/6 & \text{for } m = 1, \\ 3/6 & \text{for } m = 2. \end{cases}$$

Hence,

$$p[m] = \frac{m+1}{6} \quad \text{for } m = 0, 1, 2,$$

$$p[n|m] = \frac{p[n,m]}{p[m]} = \frac{1/6}{(m+1)/6} = \frac{1}{m+1} \quad \text{for } m = 0, 1, 2,$$

which sums to 1, as required for a pmf.

## 8-6 Functions of Random Variables

### 8-6.1 Mean Value

The **mean value**, or simply the **mean** or **expectation**, of a continuous random variable  $x$  characterized by a pdf  $p(x)$  is defined as

$$\bar{x} = E[x] = \int_{-\infty}^{\infty} x' p(x') dx'. \quad (8.48a)$$

Here, we use  $E[x]$  to denote the “expected value” of  $x$ , synonymous with the abbreviated notation  $\bar{x}$ .

Similarly, for a discrete random variable  $n$  characterized by a pmf  $p(n)$ , the mean value of  $n$  is

$$\bar{n} = E[n] = \sum_{n'=-\infty}^{\infty} n' p[n']. \quad (8.48b)$$

For a function  $f(x,y)$  of two continuous random variables  $x$  and  $y$ , the expectation (mean value) of  $f(x,y)$  is computed using the joint pdf of  $(x,y)$ , namely  $p(x,y)$ , as follows:

$$E[f(x,y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x',y') p(x',y') dx' dy'. \quad (8.49a)$$

Similarly, for the discrete case

$$E[f[n,m]] = \sum_{n'=-\infty}^{\infty} \sum_{m'=-\infty}^{\infty} f[n',m'] p[n',m']. \quad (8.49b)$$

Expectation is a **linear operator**: for any two continuous random variables  $x$  and  $y$ , and any two constants  $a$  and  $b$ ,

$$\begin{aligned} E[ax+by] &= \int_{-\infty}^{\infty} (ax'+by') p(x',y') dx' dy' \\ &= a \int_{-\infty}^{\infty} x' dx' \int_{-\infty}^{\infty} p(x',y') dy' \\ &\quad + b \int_{-\infty}^{\infty} y' dy' \int_{-\infty}^{\infty} p(x',y') dx'. \end{aligned} \quad (8.50)$$

In view of the relations given by Eq. (8.35) for the marginal pdfs, Eq. (8.50) can be rewritten as

$$\begin{aligned} E[ax+by] &= a \int_{-\infty}^{\infty} x' dx' p(x) + b \int_{-\infty}^{\infty} y' dy' p(y') \\ &= aE[x] + bE[y] = a\bar{x} + b\bar{y}. \end{aligned} \quad (8.51)$$

- The mean value of the weighted sum of two random variables is equal to the sum of their weighted means. ◀

A similar relationship applies to two discrete random variables  $n$  and  $m$ :

$$E[an + bm] = a\bar{n} + b\bar{m}. \quad (8.52)$$

## 8-6.2 Conditional Mean

The **conditional expectation**, also known as the conditional mean, of random variable  $x$ , given that  $y = y'$  uses the conditional pdf  $p(x|y = y')$ :

$$E[x | y = y'] = \int_{-\infty}^{\infty} x' p(x'|y') dx'. \quad (8.53a)$$

Note that the conditional mean is defined at a specific value of the second random variable, namely  $y = y'$ .

For the discrete case,

$$E[n | m = m'] = \sum_{n'=-\infty}^{\infty} n' p[n'|m']. \quad (8.53b)$$

## 8-6.3 Variance and Standard Deviation

The **variance** of a random variable  $x$  with mean value  $\bar{x}$  is the mean value of  $(x - \bar{x})^2$ , where  $(x - \bar{x})$  is the deviation of  $x$  from its mean  $\bar{x}$ . Denoted  $\sigma_x^2$ , the variance is defined as

$$\begin{aligned} \sigma_x^2 &= E[(x - \bar{x})^2] \\ &= E[x^2 - 2x\bar{x} + \bar{x}^2] \\ &= E[x^2] - 2\bar{x}E[x] + \bar{x}^2 = \bar{x}^2 - \bar{x}^2, \end{aligned} \quad (8.54)$$

where  $\bar{x}^2 = E[x^2]$ . Also,  $\sigma_{ax}^2 = a^2\sigma_x^2$ .

The square root of the variance is the **standard deviation**:

$$\sigma_x = \sqrt{\bar{x}^2 - \bar{x}^2}. \quad (8.55)$$

A pdf with a small standard deviation is relatively narrow in shape, whereas one with a large value for  $\sigma_x$  is relatively broad (**Fig. 8-12**).

The **covariance**  $\lambda_{x,y}$  of two random variables  $x$  and  $y$  is defined as the mean value of the product of the deviation of  $x$  from  $\bar{x}$  and  $y$  from  $\bar{y}$ :

$$\begin{aligned} \lambda_{x,y} &= E[(x - \bar{x})(y - \bar{y})] \\ &= E[xy] - \bar{x}E[y] - \bar{y}E[x] + \bar{x}\bar{y} = \bar{xy} - \bar{x}\bar{y}, \end{aligned} \quad (8.56)$$

where  $\bar{xy} = E[xy]$ .

Unlike the mean, the variance of the linear sum of two random variables is *not* a linear operator. Consider the variable  $x + y$ :

$$\begin{aligned} \sigma_{(x+y)}^2 &= E[((x+y) - E[x+y])^2] \\ &= E[(x+y)^2 - 2(x+y)E[x+y] + (E[x+y])^2] \\ &= E[x^2] + E[2xy] + E[y^2] \\ &\quad - 2E[x]E[x+y] - 2E[y]E[x+y] + (E[x+y])^2 \\ &= \bar{x}^2 + 2\bar{xy} + \bar{y}^2 - 2\bar{x}(\bar{x}+\bar{y}) - 2\bar{y}(\bar{x}+\bar{y}) + (\bar{x}+\bar{y})^2. \end{aligned} \quad (8.57)$$

The expectation of  $(x+y)$  is linear; i.e.,

$$(\bar{x}+\bar{y}) = \bar{x} + \bar{y}. \quad (8.58)$$

Use of Eq. (8.58) in Eq. (8.57) leads to

$$\sigma_{(x+y)}^2 = \bar{x}^2 + 2\bar{xy} + \bar{y}^2 - 2\bar{x}(\bar{x}+\bar{y}) - 2\bar{y}(\bar{x}+\bar{y}) + (\bar{x}+\bar{y})^2, \quad (8.59)$$

which simplifies to

$$\sigma_{(x+y)}^2 = (\bar{x}^2 - \bar{x}^2) + (\bar{y}^2 - \bar{y}^2) + 2(\bar{xy} - \bar{x}\bar{y}). \quad (8.60)$$

In view of the definitions given by Eqs. (8.54) and (8.56), Eq. (8.60) can be rewritten as

$$\sigma_{(x+y)}^2 = \sigma_x^2 + \sigma_y^2 + 2\lambda_{x,y}. \quad (8.61)$$

## 8-6.4 Properties of the Covariance

(1) The covariance between a random variable and itself is its variance:

$$\lambda_{x,x} = \sigma_x^2. \quad (8.62)$$

(2) The degree of **correlation** between two random variables is defined by the **correlation coefficient**  $\rho_{xy}$ :

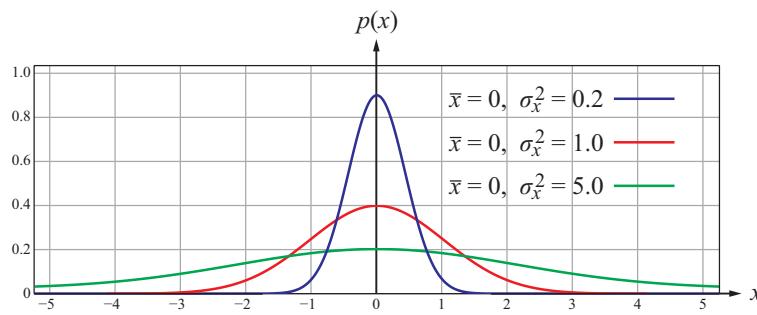
$$\rho_{x,y} = \frac{\lambda_{x,y}}{\sigma_x \sigma_y}. \quad (8.63)$$

If  $x$  and  $y$  are **uncorrelated**, then

$$E[xy] = E[x]E[y] = \bar{x}\bar{y}, \quad (8.64)$$

in which case Eq. (8.56) yields the result

$$\lambda_{x,y} = 0 \quad (\text{x and y uncorrelated}), \quad (8.65)$$



**Figure 8-12** Three Gaussian distributions, all with  $\bar{x} = 0$  but different standard deviations.

and, consequently,  $\rho_{x,y} = 0$ .

(3) Two random variables  $x$  and  $y$  are **uncorrelated** if their covariance is zero, and they are **independent** if their joint pdf is separable into the product of their individual pdfs:

$$p(x,y) = p(x) p(y) \quad (\text{x and y independent}). \quad (8.66)$$

- Uncorrelated random variables may or may not be independent, but independent random variables are uncorrelated. ◀

Consider two independent random variables  $x$  and  $y$ . The mean of their product is

$$\begin{aligned} \bar{xy} &= E[xy] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x'y' p(x',y') dx' dy' \\ &= \int_{-\infty}^{\infty} x'y' p_x(x') p_y(y') dx' dy' \\ &= \int_{-\infty}^{\infty} x' p(x') dx' \int_{-\infty}^{\infty} y' p(y') dy' \\ &= \bar{x}\bar{y} \quad (\text{x and y independent}), \end{aligned} \quad (8.67)$$

which leads to  $\lambda_{x,y} = \bar{xy} - \bar{x}\bar{y} = 0$ , and hence uncorrelation between  $x$  and  $y$ .

For readers interested in mechanical systems, we note the following correspondences:

Probabilistic Term	Mechanical Term
Expectation (mean)	Center of mass
Variance $\sigma_x^2$	Moment of inertia
Standard deviation $\sigma_x$	Radius of gyration
Covariance $\lambda_{x,y}$	Cross moment of inertia

### Example 8-8: Triangle-Like Pdf II

Given the joint pdf introduced earlier in Example 8-6, namely

$$p(x,y) = \begin{cases} C & \text{for } 0 \leq x \leq y \leq 1, \\ 0 & \text{otherwise,} \end{cases}$$

compute (a)  $\bar{x}$ , (b)  $\sigma_x^2$ , (c)  $E[y | x = 3/4]$ , and (d)  $\sigma_y^2 |_{x=3/4}$ .

**Solution:** (a) From Eq. (8.42),

$$p(x) = \begin{cases} 2(1-x), & 0 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, the mean value of  $x$  is

$$\bar{x} = E[x] = \int_0^1 x' p(x') dx' = \int_0^1 2x'(1-x') dx' = \frac{1}{3}.$$

(b) To compute  $\sigma_x^2$ , we start by computing  $\bar{x^2}$ :

$$\bar{x^2} = E[x^2] = \int_0^1 (x')^2 p(x') dx' = \int_0^1 2(x')^2(1-x') dx' = \frac{1}{6}.$$

Hence, by Eq. (8.54),

$$\sigma_x^2 = \bar{x}^2 - \bar{x}^2 = \frac{1}{6} - \left(\frac{1}{3}\right)^2 = \frac{1}{18}.$$

(c) With  $x$  and  $y$  interchanged, the expression for the conditional expectation given by Eq. (8.53a) is

$$E[y | x = x'] = \int_{-\infty}^{\infty} y' p(y' | x') dy'.$$

Using the expression given by Eq. (8.43) and setting  $x' = 3/4$  leads to

$$E[y | x = 3/4] = \int_{3/4}^1 y' 4 dy' = \frac{7}{8},$$

which is the midpoint between  $3/4$  and 1.

(d) By Eq. (8.54), the variance  $\sigma_{y|x=3/4}^2$  is given by

$$\sigma_{y|x=3/4}^2 = E[y^2 | x = 3/4] - (E[y | x = 3/4])^2.$$

The first term computes to

$$E[y^2 | x = 3/4] = \int_{3/4}^1 (y')^2 4 dy' = \frac{37}{48}.$$

Hence,

$$\sigma_{y|x=3/4}^2 = \frac{37}{48} - \left(\frac{7}{8}\right)^2 = \frac{1}{192}.$$

## 8-6.5 Uniform Pdf

A pdf is **uniform** if it has a constant value over a specified range, such as  $a$  to  $b$ , and zero otherwise:

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{for } a < x < b, \\ 0 & \text{otherwise.} \end{cases} \quad (8.68)$$

The mean and variance of a random variable characterized by a uniform pdf are

$$\bar{x} = \int_a^b x' p(x') dx' = \int_a^b \left(\frac{1}{b-a}\right) x' dx' = \frac{a+b}{2} \quad (8.69)$$

and

$$\sigma_x^2 = \bar{x}^2 - \bar{x}^2 = \int_a^b (x')^2 \left(\frac{1}{b-a}\right) dx' - \left(\frac{a+b}{2}\right)^2 = \frac{(b-a)^2}{12}. \quad (8.70)$$

**Concept Question 8-5:** When is the variance of the sum of two random variables the sum of the variances?

**Concept Question 8-6:** Does “ $x$  and  $y$  are uncorrelated” imply “ $x$  and  $y$  are independent” or is it the other way around?

**Exercise 8-7:** In Example 8-6, compute (a) the mean  $E(y)$  and (b) variance  $\sigma_y^2$ .

**Answer:** (a)

$$p(y) = \int_0^y 2 dx = \begin{cases} 2y & \text{for } 0 \leq y \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$

$$E[y] = \int_0^1 y(2y) dy = 2/3.$$

(b)

$$E[y^2] = \int_0^1 y^2(2y) dy = 1/2.$$

$$\text{Hence } \sigma_y^2 = \frac{1}{2} - \left(\frac{2}{3}\right)^2 = \frac{1}{18}.$$

**Exercise 8-8:** Show that for the joint pdf given in Example 8-8,  $\lambda_{x,y} = 1/36$ .

**Answer:** (See [IP](#)).

## 8-7 Random Vectors

Suppose we wish to quantify the probability of an event—specifically, the *time* at which students at a certain university wake up in the morning—in terms of three random variables, namely the *age* of an individual student, the *temperature* of the room in which the student was sleeping, and the *number* of credit hours that the student is enrolled in. Many other factors impact the time that a student wakes up in the morning, but let us assume we are focused on only these three random variables.

From the perspective of the prime subject of this book, namely image processing, our multiple random variables might be the image intensities of multiple images of a scene acquired at different wavelengths, different times, or under different illumination conditions. The objective might be to assign each image pixel to one of a list of possible classes on the basis of the intensities of the multiple images for that pixel. If we are dealing with multi-wavelength satellite images of Earth’s surface, for

example, the classes would be water, urban, forest, etc. And if the multiple images are ultrasound medical images of the same scene but acquired under different conditions or different times, the classes would be bone, tissue, etc.

When two or more random variables are associated with an event of interest, we form a **random vector** (*RV*). A random vector  $\mathbf{x}$  of length  $N$  is a column vector comprising  $N$  random variables  $\{x_1, x_2, \dots, x_N\}$ :

$$\mathbf{x} = [x_1, x_2, \dots, x_N]^T. \quad (8.71)$$

We write  $\mathbf{x}$  in bold to denote that it is a vector, and here the superscript “T” denotes the **transpose** operation, which in this case converts a horizontal vector into a column vector.

- ▶ Throughout this chapter, we assume that all random variables represent real quantities. ◀

The distribution of a random vector of continuous or discrete random variables is described respectively by a joint pdf or joint pmf of the random variables. The dummy variable vector associated with the random vector  $x$  is

$$\mathbf{x}' = [x'_1, x'_2, \dots, x'_N]^T, \quad (8.72)$$

and the joint pdf of vector  $\mathbf{x}$  is  $p(\mathbf{x})$ .

- ▶ The notation and properties of random vectors are summarized in **Table 8-3**. ◀

## 8-7.1 Mean Vector and Covariance Matrix

The **mean vector**  $\bar{\mathbf{x}}$  of random vector  $\mathbf{x}$  is the vector of the mean values of the components of  $\mathbf{x}$ :

$$\bar{\mathbf{x}} = E[\mathbf{x}] = [\bar{x}_1, \bar{x}_2, \dots, \bar{x}_N]^T. \quad (8.73)$$

In analogy with Eq. (8.56), the **covariance matrix**  $\mathbf{K}_{\mathbf{x}}$  of random vector  $\mathbf{x}$  comprises the covariances between  $x_i$  and  $x_j$ :

$$\begin{aligned} \mathbf{K}_{\mathbf{x}} &= E[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T] \\ &= E[\mathbf{x}\mathbf{x}^T] - \bar{\mathbf{x}}\bar{\mathbf{x}}^T = \begin{bmatrix} \lambda_{x_1,x_1} & \lambda_{x_1,x_2} & \cdots & \lambda_{x_1,x_N} \\ \lambda_{x_2,x_1} & \lambda_{x_2,x_2} & \cdots & \lambda_{x_2,x_N} \\ \vdots & & & \\ \lambda_{x_N,x_1} & \lambda_{x_N,x_2} & \cdots & \lambda_{x_N,x_N} \end{bmatrix}, \end{aligned} \quad (8.74)$$

where  $\lambda_{x_i,x_j} = \bar{x}_i x_j - \bar{x}_i \bar{x}_j$ . Similarly, the **cross-covariance matrix**  $\mathbf{K}_{\mathbf{x},\mathbf{y}}$  between random vector  $\mathbf{x}$  of length  $N$  and random vector  $\mathbf{y}$  of length  $M$  is given by the  $(N \times M)$  matrix

$$\begin{aligned} \mathbf{K}_{\mathbf{x},\mathbf{y}} &= E[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{y} - \bar{\mathbf{y}})^T] \\ &= E[\mathbf{x}\mathbf{y}^T] - \bar{\mathbf{x}}\bar{\mathbf{y}}^T = \begin{bmatrix} \lambda_{x_1,y_1} & \lambda_{x_1,y_2} & \cdots & \lambda_{x_1,y_M} \\ \lambda_{x_2,y_1} & \lambda_{x_2,y_2} & \cdots & \lambda_{x_2,y_M} \\ \vdots & & & \\ \lambda_{x_N,y_1} & \lambda_{x_N,y_2} & \cdots & \lambda_{x_N,y_M} \end{bmatrix}. \end{aligned} \quad (8.75)$$

We note that  $\mathbf{K}_{\mathbf{x},\mathbf{x}} = \mathbf{K}_{\mathbf{x}} = \mathbf{K}_{\mathbf{x}}^T$  and  $\mathbf{K}_{\mathbf{y},\mathbf{x}} = \mathbf{K}_{\mathbf{x},\mathbf{y}}^T$ . Additionally, if  $\mathbf{x}$  and  $\mathbf{y}$  are **uncorrelated**,

$$\mathbf{K}_{\mathbf{y},\mathbf{x}} = [\mathbf{0}] \quad (\mathbf{x} \text{ and } \mathbf{y} \text{ uncorrelated}). \quad (8.76)$$

## 8-7.2 Random Vector $\mathbf{Ax}$

Given a random vector  $\mathbf{x}$  of length  $N$  and a constant  $(M \times N)$  matrix  $\mathbf{A}$ , we now examine how to relate the attributes of  $\mathbf{x}$  to those of random vector  $\mathbf{y}$  given by

$$\mathbf{y} = \mathbf{Ax}. \quad (8.77)$$

The mean value of  $\mathbf{y}$  is simply

$$\bar{\mathbf{y}} = E[\mathbf{y}] = E[\mathbf{Ax}] = \mathbf{A}\bar{\mathbf{x}}. \quad (8.78)$$

To demonstrate the validity of Eq. (8.78), we rewrite  $\mathbf{y}$  in terms of its individual elements:

$$y_i = \sum_{j=1}^N A_{i,j} x_j, \quad 1 \leq i \leq M, \quad (8.79)$$

where  $A_{i,j}$  is the  $(i, j)$ th element of  $\mathbf{A}$ . Taking the expectation of  $y_i$ , while recalling from Eq. (8.51) that the expectation is a linear operator, gives

$$E[y_i] = \sum_{j=1}^N A_{i,j} E[x_j], \quad 1 \leq i \leq M. \quad (8.80)$$

Since this result is equally applicable to all random variables  $y_i$  of random vector  $\mathbf{y}$ , it follows that Eq. (8.78) is true.

Using the matrix algebra property

$$(\mathbf{Ax})^T = \mathbf{x}^T \mathbf{A}^T, \quad (8.81)$$

**Table 8-3** Notation and properties of random vectors.

<b>A. Continuous Random Vector</b>	
$\mathbf{x} = [x_1, x_2, \dots, x_N]^T$	
pdf of $\mathbf{x}$	$p(\mathbf{x})$
mean value of $\mathbf{x}$	$\bar{\mathbf{x}} = E[\mathbf{x}] = [\bar{x}_1, \bar{x}_2, \dots, \bar{x}_N]^T$
covariance matrix of $\mathbf{x}$	$\mathbf{K}_x = E[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T] = \overline{\mathbf{x}\mathbf{x}^T} - \bar{\mathbf{x}}\bar{\mathbf{x}}^T$
cross-covariance matrix	$\mathbf{K}_{x,y} = E[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{y} - \bar{\mathbf{y}})^T] = \overline{\mathbf{x}\mathbf{y}^T} - \bar{\mathbf{x}}\bar{\mathbf{y}}^T$

<b>B. Discrete Random Vector</b>	
$\mathbf{n} = [n_1, n_2, \dots, n_N]^T$	
pmf of $\mathbf{n}$	$p[\mathbf{n}]$
mean value of $\mathbf{n}$	$\bar{\mathbf{n}} = E[\mathbf{n}] = [\bar{n}_1, \bar{n}_2, \dots, \bar{n}_N]^T$
covariance matrix of $\mathbf{n}$	$\mathbf{K}_n = E[(\mathbf{n} - \bar{\mathbf{n}})(\mathbf{n} - \bar{\mathbf{n}})^T] = \overline{\mathbf{n}\mathbf{n}^T} - \bar{\mathbf{n}}\bar{\mathbf{n}}^T$
cross-covariance matrix	$\mathbf{K}_{n,m} = E[(\mathbf{n} - \bar{\mathbf{n}})(\mathbf{m} - \bar{\mathbf{m}})^T] = \overline{\mathbf{n}\mathbf{m}^T} - \bar{\mathbf{n}}\bar{\mathbf{m}}^T$

we derive the following relation for the covariance matrix of  $\mathbf{y}$ :

$$\begin{aligned}\mathbf{K}_y &= E[(\mathbf{Ax})(\mathbf{Ax})^T] - E[(\mathbf{Ax})]E[\mathbf{Ax}]^T \\ &= \mathbf{A}E[\mathbf{x}\mathbf{x}^T]\mathbf{A}^T - \mathbf{A}E[\mathbf{x}]E[\mathbf{x}]^T\mathbf{A}^T = \mathbf{A}\mathbf{K}_x\mathbf{A}^T.\end{aligned}\quad (8.82)$$

Similarly, the cross-covariance matrix between  $\mathbf{y}$  and  $\mathbf{x}$  is obtained by applying the basic definition for  $\mathbf{K}_{y,x}$  (as given in the second step of Eq. (8.75), after interchanging  $\mathbf{x}$  and  $\mathbf{y}$ ):

$$\mathbf{K}_{y,x} = E[\mathbf{y}\mathbf{x}^T] - \bar{\mathbf{y}}\bar{\mathbf{x}}^T = E[\mathbf{A}\mathbf{x}\mathbf{x}^T] - \mathbf{A}\bar{\mathbf{x}}\bar{\mathbf{x}}^T = \mathbf{A}\mathbf{K}_x\mathbf{A}^T,\quad (8.83)$$

where we used Eq. (8.74) in the last step.

Noting that  $(a+b)(c+d) = ac + ad + bc + bd$ , we can show that the covariance matrix of the sum of two random vectors  $\mathbf{x}$  and  $\mathbf{y}$  is

$$\mathbf{K}_{(x+y)} = \mathbf{K}_x + \mathbf{K}_{x,y} + \mathbf{K}_{y,x} + \mathbf{K}_y.\quad (8.84)$$

**Table 8-4** provides a summary of the covariance relationships between random vectors  $\mathbf{x}$  and  $\mathbf{y}$ .

**Concept Question 8-7:** If  $\mathbf{x}$  is a random vector and  $\mathbf{y} = \mathbf{Ax}$ , how is  $\mathbf{K}_y$  related to  $\mathbf{K}_x$ ? Is it  $\mathbf{K}_y = \mathbf{A}\mathbf{K}_x\mathbf{A}^T$  or  $\mathbf{K}_y = \mathbf{A}^T\mathbf{K}_x\mathbf{A}$ ?

**Table 8-4** Properties of two real-valued random vectors.

### Random Vectors $\mathbf{x}$ and $\mathbf{y}$

Covariance matrix  $\mathbf{K}_x = E[\mathbf{x}\mathbf{x}^T] - \bar{\mathbf{x}}\bar{\mathbf{x}}^T$

Cross-covariance matrix  $\mathbf{K}_{y,x} = E[\mathbf{x}\mathbf{y}^T] - \bar{\mathbf{x}}\bar{\mathbf{y}}^T$ ,  
any 2 random vectors  $\mathbf{x}$  and  $\mathbf{y}$

Cross-covariance matrix  $\mathbf{K}_{y,x} = \mathbf{K}_{x,y}^T$ ,  
any 2 random vectors  $\mathbf{x}$  and  $\mathbf{y}$

Cross-covariance matrix  $\mathbf{K}_{x,x} = \mathbf{K}_x = \mathbf{K}_x^T$ , for  $\mathbf{x} = \mathbf{y}$

Cross-covariance matrix  $\mathbf{K}_{x,y} = \mathbf{0}$ ,  
if  $\mathbf{x}$  and  $\mathbf{y}$  are uncorrelated

Covariance of sum matrix

$$\mathbf{K}_{(x+y)} = \mathbf{K}_x + \mathbf{K}_{x,y} + \mathbf{K}_{y,x} + \mathbf{K}_y$$

### Random Vectors $\mathbf{x}$ and $\mathbf{y} = \mathbf{Ax}$

Mean value  $\bar{\mathbf{y}} = \mathbf{A}\bar{\mathbf{x}}$

Covariance matrix  $\mathbf{K}_y = \mathbf{A}\mathbf{K}_x\mathbf{A}^T$

Cross-covariance matrix  $\mathbf{K}_{y,x} = \mathbf{A}\mathbf{K}_x$

Cross-covariance matrix  $\mathbf{K}_{x,y} = \mathbf{K}_x\mathbf{A}^T$

**Exercise 8-9:** Random vector  $\mathbf{x}$  has the covariance matrix  $\mathbf{K}_x = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}$ . If  $\mathbf{y} = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \mathbf{x}$ , find  $\mathbf{K}_y$  and  $\mathbf{K}_{x,y}$ .

**Answer:**

$$\mathbf{K}_y = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}^T = \begin{bmatrix} 115 & 51 \\ 51 & 23 \end{bmatrix}.$$

$$\mathbf{K}_{x,y} = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 13 \\ 9 & 5 \end{bmatrix}.$$

## 8-8 Gaussian Random Vectors

The expression given by Eq. (8.39) describes the joint pdf for a **Gaussian random vector** consisting of two independent random variables. For the general case, if  $\mathbf{x}$  is a random vector of length  $N$ , as defined by Eq. (8.71), and if its  $N$  random variables are **jointly Gaussian**, then it is considered a Gaussian random vector and its joint pdf is given by

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{N/2} (\det \mathbf{K}_x)^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}})^T \mathbf{K}_x^{-1} (\mathbf{x}-\bar{\mathbf{x}})}, \quad (8.85)$$

where  $(\det \mathbf{K}_x)$  is the determinant of the covariance matrix  $\mathbf{K}_x$ , and  $\mathbf{K}_x^{-1}$  is the inverse of matrix  $\mathbf{K}_x$ .

The random variables  $\{x_1, x_2, \dots, x_N\}$  are said to be **jointly Gaussian random variables**. Often, the label gets abbreviated to Gaussian random variables, but this can be misleading because even though each individual random variable may have a Gaussian pdf, the combined set of random variables need not be *jointly* Gaussian, nor form a Gaussian random vector. Hence, to avoid ambiguity, we always include the term “jointly,” when applicable, to random variables

► The adjective “jointly” is not attached to a Gaussian random vector because, by definition, the random variables constituting the Gaussian random vector are jointly Gaussian. However, two or more Gaussian random vectors can be jointly Gaussian, in which case we refer to their combination as jointly Gaussian vectors. ◀

The quantities defining the expression for  $p(\mathbf{x})$  given by Eq. (8.85) are the length  $N$  of the random vector, the mean-

value vector  $\bar{\mathbf{x}}$ , and the covariance matrix  $\mathbf{K}_x$ . Hence, a jointly Gaussian random vector  $\mathbf{x}$  often is described by the **shorthand notation**:

$$\mathbf{x} \sim \mathcal{N}(\bar{\mathbf{x}}, \mathbf{K}_x), \quad (8.86)$$

where “ $\mathcal{N}$ ” stands for “Normal Distribution,” another name for the Gaussian distribution. Next, we examine some of the important properties of the jointly Gaussian random vectors.

### 8-8.1 If $\mathbf{x}$ Is Gaussian, Then $\mathbf{y} = \mathbf{Ax}$ Is Gaussian

Per the notation given by Eq. (8.86), if  $\mathbf{x}$  is a Gaussian random vector, then random vector  $\mathbf{y} = \mathbf{Ax}$  also is Gaussian and given by

$$\mathbf{y} \sim \mathcal{N}(\mathbf{A}\bar{\mathbf{x}}, \mathbf{A}\mathbf{K}_x\mathbf{A}^T). \quad (8.87)$$

To demonstrate the validity of Eq. (8.87), we resort to the use of the ***N-dimensional continuous-space Fourier transform (N-D CSFT)***. Consider an  $N$ -dimensional image  $f(\mathbf{x})$ , with  $\mathbf{x}$  a random vector comprising  $N$  random variables (corresponding to  $N$  different pixels). The N-D CSFT of  $f(\mathbf{x})$  is  $\mathbf{F}(\boldsymbol{\mu})$ , where  $\boldsymbol{\mu}$  is an ***N-dimensional spatial frequency vector***

$$\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_N]^T, \quad (8.88)$$

and

$$\mathbf{F}(\boldsymbol{\mu}) = \underbrace{\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty}}_{N \text{ integrals}} f(\mathbf{x}) e^{-j2\pi\boldsymbol{\mu}^T \mathbf{x}} d\mathbf{x}, \quad (8.89)$$

with

$$\boldsymbol{\mu}^T \mathbf{x} = \sum_{n=1}^N \mu_n x_n. \quad (8.90)$$

For  $N = 2$ ,  $\mathbf{F}(\boldsymbol{\mu})$  reduces to the 2-D CSFT given by Eq. (3.16a), with  $\boldsymbol{\mu} = [\mu_1, \mu_2] = [\mu, v]$ .

The form of the transformation represented by Eq. (8.89) is equally applicable for computing the  $N$ -D CSFT of the joint pdf  $p(\mathbf{x})$ , which is called the ***characteristic function***\*  $\Phi_x(\boldsymbol{\mu})$ :

$$\Phi_x(\boldsymbol{\mu}) = \mathcal{F}\{p(\mathbf{x})\} = \underbrace{\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty}}_{N \text{ integrals}} p(\mathbf{x}') e^{-j2\pi\boldsymbol{\mu}^T \mathbf{x}'} d\mathbf{x}', \quad (8.91)$$

For a single random variable  $x$  with pdf  $p(x)$ , the mean value of

\*The term “characteristic function” is usually associated with the *mathematical* definition of the Fourier transform, which uses a “+” sign in the exponent in Eq. (8.91). For consistency with the *engineering* definition of the Fourier transform used throughout this book, we use a “-” sign instead.

any function of  $x$ , such as  $g(x)$ , is by definition

$$E[g(x)] = \int_{-\infty}^{\infty} g(x') p(x') dx'. \quad (8.92)$$

By extension, Eq. (8.91) reduces to

$$\Phi_x(\mu) = E[e^{-j2\pi\mu^T x}]. \quad (8.93)$$

Since  $\mathbf{x}$  is a Gaussian random vector, use of its pdf, as defined by Eq. (8.85), in Eq. (8.91) can be shown to lead to

$$\Phi_x(\mu) = e^{-j2\pi\mu^T \bar{x} - 2\pi^2 \mu^T K_x \mu}. \quad (8.94)$$

We note that the first term is Eq. (8.94) is the  $N$ -dimensional generalization of entry #3 in **Table 3-1** and the second term is the generalization of entry #13 in the same table.

Now, we wish to derive the characteristic function  $\Phi_y(\mu)$  of vector  $\mathbf{y} = \mathbf{Ax}$ . We start by introducing the spatial frequency vector  $\tilde{\mu}$  and its transpose:

$$\tilde{\mu} = \mathbf{A}^T \mu \quad (8.95a)$$

and

$$\tilde{\mu}^T = \mu^T \mathbf{A}. \quad (8.95b)$$

Next, rewriting Eq. (8.93) for  $\mathbf{y}$  instead of  $\mathbf{x}$ , we have

$$\Phi_y(\mu) = E[e^{-j2\pi\mu^T y}] = E[e^{-j2\pi\mu^T (\mathbf{Ax})}] = E[e^{-j2\pi\tilde{\mu}^T x}]. \quad (8.96)$$

In analogy to the correspondence between the two forms of  $\Phi_x(\mu)$  given by Eqs. (8.93) and (8.94), the form of Eq. (8.94) is applicable to Eq. (8.96) is

$$\Phi_y(\mu) = e^{-j2\pi\tilde{\mu}^T \bar{x} - 2\pi^2 \tilde{\mu}^T K_x \tilde{\mu}} \quad (8.97)$$

$$= e^{-j2\pi\mu^T (\mathbf{A}\bar{x}) - 2\pi^2 \mu^T (\mathbf{A}K_x \mathbf{A}^T)\mu}, \quad (8.98)$$

which is the characteristic function for a Gaussian random vector with mean vector  $(\mathbf{A}\bar{x})$  and covariance matrix  $K_y = \mathbf{A}K_x \mathbf{A}^T$ . Hence,

$$\mathbf{y} \sim \mathcal{N}(\mathbf{A}\bar{x}, \mathbf{A}K_x \mathbf{A}^T). \quad (8.99)$$

This result is consistent with the form of Eq. (8.86); upon replacing  $\mathbf{x}$  with  $\mathbf{y}$ , and using Eqs. (8.78) and (8.82), we have

$$\mathbf{y} \sim \mathcal{N}(\bar{y}, K_y) \sim \mathcal{N}(\mathbf{A}\bar{x}, \mathbf{A}K_x \mathbf{A}^T).$$

## 8-8.2 Properties of Gaussian Random Vectors

### A. Marginals of Gaussian random vectors are Gaussian

Let  $\mathbf{x}$  be a Gaussian random vector partitioned into two vectors  $\mathbf{y}$  and  $\mathbf{z}$ :

$$\mathbf{x} = [\mathbf{y}^T, \mathbf{z}^T]^T = \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix}. \quad (8.100)$$

Next, if we define matrix  $\mathbf{A}$  as

$$\mathbf{A} = [\mathbf{I} \quad \mathbf{0}], \quad (8.101)$$

it follows that  $\mathbf{y}$  is related to  $\mathbf{x}$  by

$$\mathbf{y} = \mathbf{Ax} = [\mathbf{I} \quad \mathbf{0}] \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix}. \quad (8.102)$$

According to the result given by Eq. (8.99), if  $\mathbf{x}$  is a Gaussian random vector, so is  $\mathbf{y}$ , which proves that the marginal of a Gaussian vector is itself a Gaussian random vector.

### B. Uncorrelated jointly Gaussian random vectors are independent

As demonstrated by the result of Eq. (8.67), independent random variables are uncorrelated:

independent always uncorrelated.

The converse, however, is not necessarily true; in general, uncorrelated random variables may or may not be independent:

uncorrelated sometimes independent.

But, for Gaussian random vectors, the relationships are always bidirectional:

uncorrelated independent **(Gaussian)**.

To demonstrate the validity of this assertion, let us define a Gaussian random vector  $\mathbf{z}$ , partitioned into two vectors  $\mathbf{x}$  and  $\mathbf{y}$ ,

$$\mathbf{z} = [\mathbf{x}^T \mathbf{y}^T]^T = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}, \quad (8.103)$$

with  $\mathbf{x}$  of length  $N_x$ ,  $\mathbf{y}$  of length  $N_y$ , and  $\mathbf{z}$  of length  $N = N_x + N_y$ . Our task is to demonstrate that if  $\mathbf{x}$  and  $\mathbf{y}$  are uncorrelated, then they are independent. Uncorrelation means that  $\mathbf{K}_{x,y} = [\mathbf{0}]$  and independence means that  $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}) p(\mathbf{y})$ .

The joint pdf of  $\mathbf{z}$  has the form of Eq. (8.85) with  $\mathbf{x}$  replaced with  $\mathbf{z}$ :

$$p(\mathbf{z}) = \frac{1}{(2\pi)^{N/2}(\det \mathbf{K}_z)^{1/2}} e^{-\frac{1}{2}(\mathbf{z}-\bar{\mathbf{z}})^T \mathbf{K}_z^{-1} (\mathbf{z}-\bar{\mathbf{z}})}. \quad (8.104)$$

The mean vector and covariance matrix of  $\mathbf{z}$  are given by

$$\bar{\mathbf{z}} = E[\mathbf{z}] = \begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\mathbf{y}} \end{bmatrix} \quad (8.105a)$$

and

$$\mathbf{K}_z = \begin{bmatrix} \mathbf{K}_x & \mathbf{K}_{x,y} \\ \mathbf{K}_{x,y}^T & \mathbf{K}_y \end{bmatrix}. \quad (8.105b)$$

Since  $\mathbf{x}$  and  $\mathbf{y}$  are uncorrelated,  $\mathbf{K}_{x,y} = \mathbf{0}$ , in which case  $\mathbf{K}_z$  becomes a block diagonal matrix:

$$\mathbf{K}_z = \begin{bmatrix} \mathbf{K}_x & \mathbf{0} \\ \mathbf{0}^T & \mathbf{K}_y \end{bmatrix}, \quad (8.106a)$$

$$\det \mathbf{K}_z = (\det \mathbf{K}_x)(\det \mathbf{K}_y), \quad (8.106b)$$

and

$$\mathbf{K}_z^{-1} = \begin{bmatrix} \mathbf{K}_x^{-1} & \mathbf{0} \\ \mathbf{0}^T & \mathbf{K}_y^{-1} \end{bmatrix}. \quad (8.106c)$$

Moreover, the exponent of Eq. (8.104) becomes

$$\begin{aligned} -\frac{1}{2} \begin{bmatrix} \mathbf{x} - \bar{\mathbf{x}} \\ \mathbf{y} - \bar{\mathbf{y}} \end{bmatrix}^T \begin{bmatrix} \mathbf{K}_x^{-1} & \mathbf{0} \\ \mathbf{0}^T & \mathbf{K}_y^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{x} - \bar{\mathbf{x}} \\ \mathbf{y} - \bar{\mathbf{y}} \end{bmatrix} \\ = -\frac{1}{2}(\mathbf{x} - \bar{\mathbf{x}})^T \mathbf{K}_x^{-1} (\mathbf{x} - \bar{\mathbf{x}}) - \frac{1}{2}(\mathbf{y} - \bar{\mathbf{y}})^T \mathbf{K}_y^{-1} (\mathbf{y} - \bar{\mathbf{y}}). \end{aligned} \quad (8.107)$$

In view of the results represented by Eqs. (8.106b) and (8.107), the pdf of  $\mathbf{z}$  simplifies to

$$\begin{aligned} p(\mathbf{z}) &= \frac{1}{(2\pi)^{N_x/2}(2\pi)^{N_y/2}(\det \mathbf{K}_x)^{1/2}(\det \mathbf{K}_y)^{1/2}} \\ &\times e^{-\frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}})^T \mathbf{K}_x^{-1} (\mathbf{x}-\bar{\mathbf{x}})} e^{-\frac{1}{2}(\mathbf{y}-\bar{\mathbf{y}})^T \mathbf{K}_y^{-1} (\mathbf{y}-\bar{\mathbf{y}})} \\ &= p(\mathbf{x}) p(\mathbf{y}), \end{aligned} \quad (8.108)$$

where we used the relation  $N = N_x + N_y$ .

► The result given by Eq. (8.108) confirms that if  $\mathbf{x}$  and  $\mathbf{y}$  are jointly Gaussian and uncorrelated, then they are independent. ◀

### C. Conditional Gaussian random vectors

If  $\mathbf{z}$  is a Gaussian random vector partitioned into vectors  $\mathbf{x}$  and  $\mathbf{y}$  as defined by Eq. (8.103), namely

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}, \quad (8.109)$$

then the conditional pdf  $p(\mathbf{x} | \mathbf{y} = \mathbf{y}')$  also is Gaussian:

$$p(\mathbf{x} | \mathbf{y} = \mathbf{y}') \sim \mathcal{N}(\bar{\mathbf{x}} | \mathbf{y} = \mathbf{y}', \mathbf{K}_{x|y}), \quad (8.110)$$

with

$$\bar{\mathbf{x}} | \mathbf{y} = \mathbf{y}' = E[\mathbf{x} | \mathbf{y} = \mathbf{y}'] = \bar{\mathbf{x}} + \mathbf{K}_{x,y} \mathbf{K}_y^{-1} (\mathbf{y}' - \bar{\mathbf{y}}) \quad (8.111a)$$

and

$$\mathbf{K}_{x|y} = \mathbf{K}_x - \mathbf{K}_{x,y} \mathbf{K}_y^{-1} \mathbf{K}_{x,y}^T. \quad (8.111b)$$

Interchanging  $\mathbf{x}$  and  $\mathbf{y}$  everywhere in Eqs. (8.110) and (8.111) provides expressions for the conditional pdf  $p(\mathbf{y} | \mathbf{x} = \mathbf{x}')$ .

Deriving these relationships involves a rather lengthy mathematical process, which we do not include in here (see Problem 8-13).

### Example 8-9: Gaussian Random Vectors

Random vector  $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$  has a joint pdf

$$p(\mathbf{x}) \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_x), \quad (8.112)$$

with a covariance matrix

$$\mathbf{K}_x = \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}. \quad (8.113)$$

Random vector  $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$  is related to  $\mathbf{x}$  by

$$y_1 = 2x_1 + 3x_2, \quad (8.114a)$$

$$y_2 = 4x_1 + 5x_2. \quad (8.114b)$$

Also, random variable  $z$  is related to  $x_1$  and  $x_2$  by

$$z = x_1 + 2x_2. \quad (8.115)$$

Determine: (a)  $\sigma_{x_1}^2$  and  $\lambda_{x_1, x_2}$ , (b)  $\sigma_z^2$ , (c)  $\mathbf{K}_y$ , (d)  $\mathbf{K}_{y,x}$ , and

(e)  $\mathbf{K}_{y|x}$ .

**Solution:** (a) The covariance matrix given by Eq. (8.113) represents

$$\mathbf{K}_x = \begin{bmatrix} \lambda_{x_1,x_1} & \lambda_{x_1,x_2} \\ \lambda_{x_2,x_1} & \lambda_{x_2,x_2} \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}. \quad (8.116)$$

The variance  $\sigma_{x_1}^2$  of  $x_1$  is the same as the covariance  $\lambda_{x_1,x_1}$ . Hence,

$$\sigma_{x_1}^2 = 5,$$

and from Eq. (8.116),

$$\lambda_{x_1,x_2} = 2.$$

(b) Random variable  $z = x_1 + 2x_2$  is related to  $\mathbf{x}$  by

$$z = [1 \ 2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{B}\mathbf{x},$$

with

$$\mathbf{B} = [1 \ 2].$$

Application of Eq. (8.77) leads to

$$\sigma_z^2 = K_z = \mathbf{B}\mathbf{K}_x\mathbf{B}^T = [1 \ 2] \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 17.$$

Alternatively, using Eq. (8.61),

$$\sigma_z^2 = \sigma_{x_1}^2 + 4\sigma_{x_2}^2 + 2\lambda_{x_1,x_2} = 5 + 4 \times 2 + 2 \times 2 = 17.$$

(c) In view of the coefficients in Eq. (8.114), we can relate  $\mathbf{y}$  to  $\mathbf{x}$  by

$$\mathbf{y} = \mathbf{Ax},$$

with

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}.$$

By Eq. (8.77),

$$\mathbf{K}_y = \mathbf{AK}_x\mathbf{A}^T = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix} = \begin{bmatrix} 53 & 99 \\ 99 & 185 \end{bmatrix}.$$

(d) From Eq. (8.83)

$$\mathbf{K}_{y,x} = \mathbf{AK}_x = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 16 & 7 \\ 30 & 13 \end{bmatrix}.$$

(e) Interchanging  $\mathbf{x}$  and  $\mathbf{y}$  in Eq. (8.106b) gives

$$\begin{aligned} \mathbf{K}_{y|x} &= \mathbf{K}_y - \mathbf{K}_{y,x}\mathbf{K}_x^{-1}\mathbf{K}_{y,x}^T \\ &= \begin{bmatrix} 53 & 99 \\ 99 & 185 \end{bmatrix} - \begin{bmatrix} 16 & 7 \\ 30 & 13 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ -2 & 5 \end{bmatrix} \begin{bmatrix} 16 & 30 \\ 7 & 13 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \end{aligned}$$

This result makes perfect sense; if  $\mathbf{x}$  is known then  $\mathbf{y} = \mathbf{Ax}$  also is known, and hence the covariance of  $\mathbf{y}$  given  $\mathbf{x}$ , is zero.

**Concept Question 8-8:** If  $\{x_1, x_2, \dots, x_N\}$  are all Gaussian random variables, is  $[x_1, x_2, \dots, x_N]^T$  necessarily a Gaussian random vector?

## 8-9 Random Processes

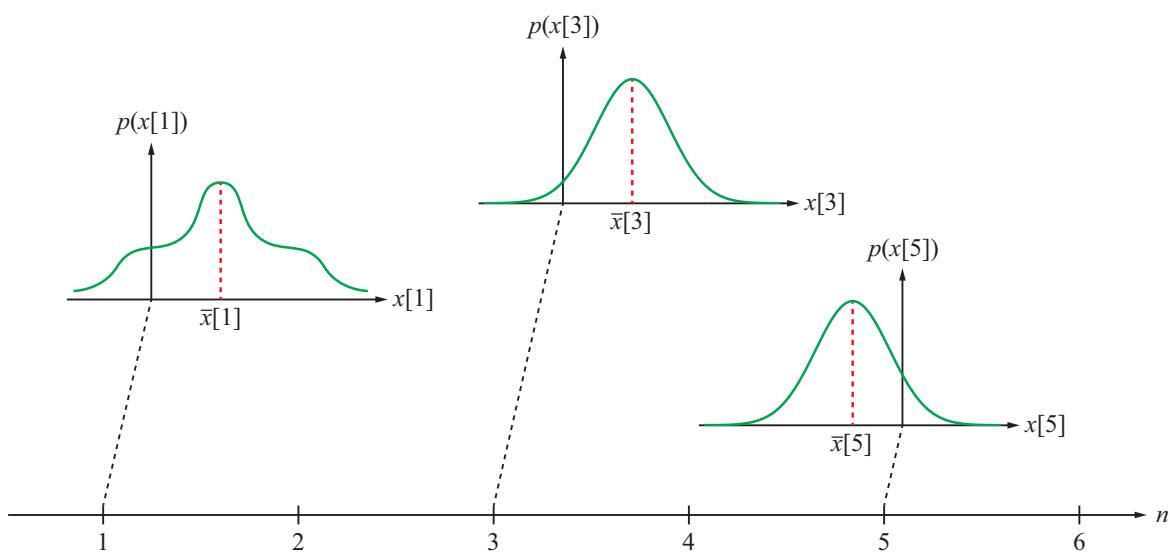
**Figure 8-13** displays pdfs for the air temperature  $x[n]$  at discrete times  $n = 1, 3$ , and  $5$ . The air temperatures  $x[1]$ ,  $x[3]$ , and  $x[5]$ —as well as  $x[n]$  for all other values of  $n$ —are each a random variable characterized by its own pdf, generated from a probabilistic model that involves several atmospheric variables. For example, the pdf of  $x[3]$  is  $p(x[3])$  and its mean value is  $\bar{x}[3]$ , and similar attributes apply to  $x[n]$  for every  $n$ . The set encompassing all of these random variables for all times  $n$  is called a **random process**, and since in the present example they are indexed by discrete time, the process is called a **discrete-time random process**. Otherwise, had the indexing been continuous in time, we would have called it a **continuous-time random process**.

The random process  $x[n]$  consists of the infinitely long random vector:

$$\mathbf{x} = [\dots, x[-2], x[-1], x[0], x[1], \dots]^T, \quad (8.117)$$

and it is characterized by the joint infinite-dimensional pdf  $p(\mathbf{x})$ . As we will see shortly, different members of  $\mathbf{x}$  may or may not be correlated with each other, or their correlation may be a function of the time separation between them.

► A **sample function** or **realization** of a random process is a deterministic function that results from a specific outcome of the probabilistic experiment generating the random process. Random processes are also known as **stochastic processes**. ◀



**Figure 8-13**  $x[n]$  is the air temperature at discrete time  $n$ . At time  $n = 1$ , atmospheric conditions lead to a probabilities model for random variable  $x[1]$  given by a pdf  $p(x[1])$  and a mean value  $\bar{x}[1]$ . Similar models characterize  $x[n]$  at each  $n$ . The sequence of random variables  $\{ \dots, x[-2], x[-1], x[0], x[1], \dots \}$  constitutes a **discrete-time random process**.

### 8-9.1 Examples of Discrete-Time Random Processes

#### A. Independent and identically distributed (IID)

The joint pdf or pmf of two independent random variables is equal to the product of their individual pdfs or pmfs:  $p(x,y) = p(x)p(y)$ . By extension, if all of the elements of vector  $\mathbf{x}$  are statistically independent of each other, and if in addition, they all have the same generic pdf  $p(x)$  then  $\mathbf{x}$  is said to be an **independent and identically distributed (IID)** random process characterized by

$$p(\mathbf{x}) = \prod_{n'=-\infty}^{\infty} p(x[n']). \quad (8.118)$$

#### B. Gaussian random process

A random process is **Gaussian** if each finite subset of the infinite set of random variables  $\{x[n]\}$  is a jointly Gaussian set of random variables, and therefore they can be stacked into a

Gaussian random vector. That is,

$$\mathbf{x} = [x[n_1], x[n_2], \dots, x[n_N]]^T \quad (8.119)$$

is a Gaussian random vector for any  $N$  integer-valued times  $\{n_1, n_2, \dots, n_N\}$ , and for any  $N$ . The joint pdf has the form given by Eq. (8.85).

### 8-9.2 Functions of Random Processes

The following definitions pertain to discrete-time random processes. Analogous expressions apply to continuous-time random processes.

#### Mean value

$$\bar{x}[n] = E[x[n]]. \quad (8.120)$$

A zero-mean random process is a process with  $\bar{x}[n] = 0$ .

## Autocovariance function

$$\begin{aligned} K_x[i, j] &= \lambda_{x[i], x[j]} = E[(x[i] - \bar{x}[i])(x[j] - \bar{x}[j])] \\ &= \overline{x[i] x[j]} - \bar{x}[i] \bar{x}[j]. \end{aligned} \quad (8.121)$$

## Cross-covariance function

$$\begin{aligned} K_{xy}[i, j] &= \lambda_{x[i], y[j]} = E[(x[i] - \bar{x}[i])(y[j] - \bar{y}[j])] \\ &= \overline{x[i] y[j]} - \bar{x}[i] \bar{y}[j]. \end{aligned} \quad (8.122)$$

Random processes  $x[n]$  and  $y[n]$  are **uncorrelated** if

$$K_{xy}[n, m] = 0 \quad \text{for all } n \text{ and } m \quad (\text{uncorrelated}).$$

## Autocorrelation function

$$R_x[i, j] = E[x[i] x[j]] = K_x[i, j] + \bar{x}[i] \bar{x}[j]. \quad (8.123)$$

For a zero-mean random process with  $\bar{x}[i] = \bar{x}[j] = 0$ ,

$$R_x[i, j] = K_x[i, j] \quad (\text{zero-mean process}). \quad (8.124)$$

## Cross-correlation function

$$R_{xy}[i, j] = E[x[i] y[j]] = K_{xy}[i, j] + \bar{x}[i] \bar{y}[j]. \quad (8.125a)$$

For zero-mean random processes

$$R_{xy}[i, j] = K_{xy}[i, j] \quad (\text{zero-mean process}). \quad (8.125b)$$

## 8.9.3 Wide-Sense Stationary (WSS) Random Process

### A. Discrete time

A random process is considered **wide-sense stationary (WSS)**, also known as **weak-sense stationary**, if it has the following three properties:

- (a) The mean  $\bar{x}[n]$  is constant for all values of  $n$ .
- (b) The autocovariance function  $K_x[i, j]$  is a function of the difference  $(i - j)$ , rather than  $i$  or  $j$  explicitly. That is,

$$K_x[i, j] = K_x[i - j]. \quad (8.126)$$

- (c) The autocorrelation function also is a function of  $(i - j)$ :

$$R_x[i, j] = R_x[i - j]. \quad (8.127)$$

Since the constant mean is known or can be estimated (Section 9-1), it can be subtracted off from the random process, thereby producing a zero-mean random process, in which case Eqs. (8.124) and (8.125b) become applicable.

► For the sake of simplicity, we will henceforth assume that all WSS random processes have zero mean. ◀

For a zero-mean random process, with  $\bar{x}[i] = \bar{y}[j] = 0$  for all  $i$  and  $j$ , the combination of Eqs. (8.121) through (8.127) leads to

$$R_x[i - j] = K_x[i - j] = E[x[i] x[j]] \quad (8.128a)$$

and

$$R_{xy}[i - j] = K_{xy}[i - j] = E[x[i] y[j]]. \quad (8.128b)$$

Changing variables to  $n = i - j$  gives, for any  $j$ ,

$$R_x[n] = E[x[n + j] x[j]], \quad (8.129a)$$

$$R_{xy}[n] = E[x[n + j] y[j]]. \quad (8.129b)$$

If the process is IID,  $x[n + j]$  and  $x[j]$  are independent random variables, except for  $n = 0$ . Hence,  $R_x[n]$  becomes

$$R_x[n] = \begin{cases} E[x^2[j]] & \text{for } n = 0, \\ E[x[n + j]] E[x[j]] & \text{for } n \neq 0. \end{cases} \quad (8.130)$$

Since the process is presumed to be zero-mean, which means that  $\bar{x}[i] = 0$  for any  $i$ ,  $R_x[n]$  simplifies to

$$R_x[n] = \sigma^2 \delta[n], \quad (8.131)$$

where the variance is

$$\sigma^2 = \overline{x^2[j]}. \quad (8.132)$$

### B. Continuous-time processes

All of the definitions and relationships introduced earlier for discrete-time random processes generalize directly to continuous-time random processes. For example,  $x(t)$  is a Gaussian random process if

$$\{x(t_1), x(t_2), \dots, x(t_N)\} \quad (8.133)$$

are jointly Gaussian random variables for any  $N$  real-valued times  $\{t_1, t_2, \dots, t_N\}$ , and any integer  $N$ . For the discrete-time

WSS random process, we defined the autocorrelation and cross-correlation functions in terms of the discrete-time difference  $n = i - j$ . By analogy, we define  $\tau = t_i - t_j$  for the continuous-time case and then we generalize by replacing  $t_j$  with simply  $t$ , which leads to

$$R_x(\tau) = K_x(\tau) = E[x(\tau + t)x(t)] \quad (\text{zero-mean WSS}) \quad (8.134a)$$

and

$$R_{xy}(\tau) = K_{xy}(\tau) = E[x(\tau + t)y(t)] \quad (\text{zero-mean WSS}). \quad (8.134b)$$

These expressions are for a zero-mean WSS random process.

Furthermore, if the process is also IID,

$$R_x(\tau) = \sigma^2 \delta(\tau). \quad (8.135)$$

## 8-9.4 Power Spectral Density

### A. Continuous-time deterministic signal

For a deterministic (non-random) continuous-time signal  $x(t)$ , the signal power at time  $t$  is simply  $|x(t)|^2$ , and the total energy of the signal is, from Eq. (2.7),

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt. \quad (8.136)$$

In image processing, the image intensity is real-valued, so we will continue to treat 1-D signals and 2-D images as real-valued, in which case  $|x(t)|^2 = x^2(t)$ .

### B. Continuous-time random signal

The power at time  $t$  of a continuous-time random process  $x(t)$  is defined not only in terms of  $x^2(t)$  but also in terms of the probability of that specific value of  $x(t)$ . That is,

$$P(t) = E[x^2(t)] = \int_{x'=-\infty}^{\infty} (x'(t))^2 p(x'(t)) dx'(t). \quad (8.137)$$

If  $x(t)$  is a zero-mean WSS random process, we can express the power in terms of the autocorrelation function  $R_x(\tau)$ , as defined by Eq. (8.134a), for  $\tau = 0$ :

$$E[x^2(t)] = R_x(0). \quad (8.138)$$

In general,  $R_x(\tau)$  is related to the **power spectral** density of the signal,  $S_x(f)$ , by the Fourier transform:

$$S_x(f) = \mathcal{F}\{R_x(\tau)\} = \int_{-\infty}^{\infty} R_x(\tau') e^{-j2\pi f \tau'} d\tau' \quad (8.139a)$$

and

$$R_x(\tau) = \mathcal{F}^{-1}\{S_x(f)\} = \int_{-\infty}^{\infty} S_x(f') e^{j2\pi f' \tau} df'. \quad (8.139b)$$

Setting  $\tau = 0$  leads to

$$E[x^2(t)] = R_x(0) = \int_{-\infty}^{\infty} S_x(f') df'. \quad (8.140)$$

For the special case where  $S_x(f)$  is zero at all frequencies except over an infinitesimally narrow band of width  $B$  centered at  $f' = f_0$ , the expression given by Eq. (8.140) reduces to

$$E[x^2(t)] = S_x(f_0) B. \quad (8.141)$$

Since  $x(t)$  is real-valued,  $R_x(\tau) = R_x(-\tau)$  and  $S_x(f) = S_x(-f)$ , so the **bilateral power spectral density** of  $x(t)$  at  $f_0$  is  $2S_x(f_0)$ .

Finally, for two zero-mean jointly WSS random processes  $x$  and  $y$ , the cross-correlation function  $R_{xy}(\tau)$  and the cross-spectral density  $S_{xy}(f)$  are related by

$$S_{xy}(f) = \mathcal{F}\{R_{xy}(\tau)\} = \int_{-\infty}^{\infty} R_{xy}(\tau') e^{-j2\pi f \tau'} d\tau' \quad (8.142a)$$

and

$$R_{xy}(\tau) = \mathcal{F}^{-1}\{S_{xy}(f)\} = \int_{-\infty}^{\infty} S_{xy}(f') e^{j2\pi f' \tau} df'. \quad (8.142b)$$

### C. Discrete-time random process

Using the expressions given in Eq. (8.129) for the autocorrelation and cross-correlation functions  $R_x[n]$  and  $R_{xy}[n]$  for zero-mean WSS random processes, application of the DTFT (instead of the Fourier transform) leads to

$$S_x(\Omega) = \sum_{n=-\infty}^{\infty} R_x[n] e^{-j\Omega n} \quad (8.143a)$$

and

$$S_{xy}(\Omega) = \sum_{n=-\infty}^{\infty} R_{xy}[n] e^{-j\Omega n}. \quad (8.143b)$$

### D. White random process

According to Eq. (8.135), the autocorrelation function for a zero-mean WSS random process is given by  $R_x(\tau) = \sigma^2 \delta(\tau)$ .

From Eq. (8.139a), the corresponding power spectral density is

$$\begin{aligned} \mathbf{S}_x(f) &= \int_{-\infty}^{\infty} R_x(\tau') e^{-j2\pi f\tau'} d\tau' = \int_{-\infty}^{\infty} \sigma^2 \delta(\tau') e^{-j2\pi f\tau'} d\tau' \\ &= \sigma^2. \end{aligned} \quad (8.144)$$

Hence,  $\mathbf{S}_x(f)$  is constant across all frequencies  $f$ . By analogy to white light, which consists of all colors of the spectrum, a frequency independent power spectral density is called **white** and  $x(t)$  is called a **white random process**. One of the properties of such a process is that  $x(t_1)$  and  $x(t_2)$  are uncorrelated random variables (and also independent if  $x(t)$  is Gaussian), which means that the value of  $x(t_1)$  at time  $t_1$  carries no information about the value of  $x(t_2)$  at time  $t_2$  no matter how small  $|t_2 - t_1|$  is, so long as  $t_2 \neq t_1$ .

Often, a white random process is illustrated by a plot like the one depicted in Fig. 8-14(a). However, such a plot is incorrect because it displays time continuity of  $x(t)$ . The correct plot should look more like the one in part (b) of the figure.

Similarly, for a discrete-time WSS white random processes with  $R_x[n] = \sigma^2 \delta[n]$ , use of Eq. (8.131) in Eq. (8.143a) leads to

$$\mathbf{S}_x(\Omega) = \sigma^2 \quad (8.145)$$

for all  $\Omega$ . Furthermore,  $x[n_1]$  and  $x[n_2]$  are uncorrelated for any times  $n_1 \neq n_2$ , and also independent if  $x[n]$  is a Gaussian random process.

**Concept Question 8-9:** Are IID random processes wide-sense stationary?

**Concept Question 8-10:** Why can we assume that a WSS random process has zero mean?

**Exercise 8-10:** A zero-mean WSS random process has power spectral density

$$\mathbf{S}_x(f) = \frac{4}{(2\pi f)^2 + 4}.$$

What is its autocorrelation function?

**Answer:**  $R(\tau) = \mathcal{F}^{-1}\{\mathbf{S}_x(f)\}$ . From entry #3 in Table 2-5,  $R(\tau) = e^{-2|\tau|}$ .

**Exercise 8-11:** A zero-mean WSS random process has the Gaussian autocorrelation function  $R(\tau) = e^{-\pi\tau^2}$ . What is its power spectral density?

**Answer:**  $\mathbf{S}_x(f) = \mathcal{F}\{R(\tau)\}$ . From entry #6 in Table 2-5,  $\mathbf{S}_x(f) = e^{-\pi f^2}$ .

## 8-10 LTI Filtering of Random Processes

### 8-10.1 Continuous-Time Random Process

Consider an LTI system with a real-valued impulse response  $h(t)$ , a continuous-time random process  $x(t)$  at its input, and an output  $y(t)$ :

$$x(t) \rightarrow \boxed{h(t)} \rightarrow y(t). \quad (8.146)$$

Since  $x(t)$  is a random process, it can exhibit many sample functions (manifestations), and each sample function  $x(t)$  is filtered by  $h(t)$  to produce a sample function  $y(t)$ . Hence,  $y(t)$  also is a random process:

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(\alpha) x(t - \alpha) d\alpha. \quad (8.147a)$$

Now we examine the relation between the autocorrelations of  $x(t)$  and  $y(t)$ , as well as the cross-correlation between  $x(t)$  and  $y(t)$ . We assume  $x(t)$  is a zero-mean WSS random process, which implies that  $R_x(t_1, t_2) = R_x(t_1 - t_2)$ . Taking the expectation  $E[\cdot]$  of Eq. (8.147a) gives

$$E[y(t)] = \int_{-\infty}^{\infty} h(\alpha) E[x(t - \alpha)] d\alpha = 0, \quad (8.147b)$$

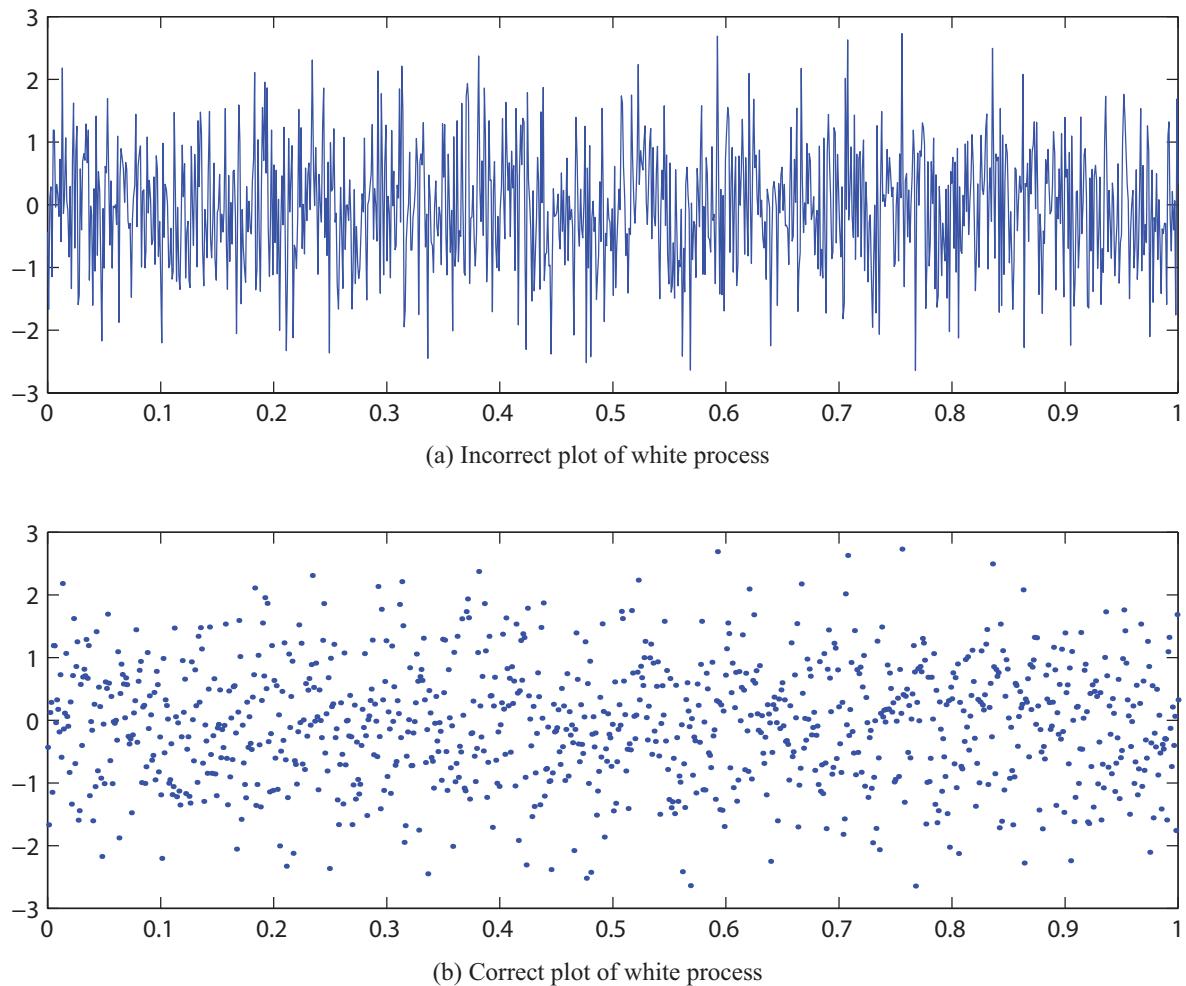
where we assumed that  $E[x(t - \alpha)] = 0$ , based on our earlier assumption that  $x(t)$  is a zero-mean WSS random process. Hence,  $y(t)$  is zero-mean.

#### A. Autocorrelation of output

Let us consider  $y(t)$  at times  $t_1$  and  $t_2$ ; upon replacing  $t$  with  $t_1$  and dummy variable  $\alpha$  with  $\alpha_1$ , and then repeating the process at  $t_2$ , we have

$$y(t_1) = h(t_1) * x(t_1) = \int_{-\infty}^{\infty} h(\alpha_1) x(t_1 - \alpha_1) d\alpha_1 \quad (8.148a)$$

and



**Figure 8-14** A white process cannot be continuous in time.

$$y(t_2) = h(t_2) * x(t_2) = \int_{-\infty}^{\infty} h(\alpha_2) x(t_2 - \alpha_2) d\alpha_2. \quad (8.148b)$$

Multiplication of  $y(t_1)$  by  $y(t_2)$  gives

$$y(t_1) y(t_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(\alpha_1) h(\alpha_2) x(t_1 - \alpha_1) x(t_2 - \alpha_2) d\alpha_1 d\alpha_2. \quad (8.149)$$

Taking the expectation  $E[\cdot]$  of both sides gives

$$R_y(t_1, t_2) = \int \int_{-\infty}^{\infty} h(\alpha_1) h(\alpha_2) R_x(t_1 - t_2 - \alpha_1 + \alpha_2) d\alpha_1 d\alpha_2. \quad (8.150)$$

Since we have already shown that  $E[y(t)] = 0$ , it follows that  $R_y(t_1, t_2) = R_y(t_1 - t_2)$  and  $y(t)$  is WSS. Upon defining

$\tau = t_1 - t_2$ , we have

$$\begin{aligned} R_y(\tau) &= \int \int_{-\infty}^{\infty} h(\alpha_1) h(\alpha_2) R_x(\tau - \alpha_1 + \alpha_2) d\alpha_1 d\alpha_2 \\ &= h(\tau) * h(-\tau) * R_x(\tau). \end{aligned} \quad (8.151)$$

Taking the Fourier transform of both sides leads to the following expression for the power spectral density of  $y(t)$ :

$$\mathbf{S}_y(f) = \mathbf{H}(f) \mathbf{H}(-f) \mathbf{S}_x(f) = |\mathbf{H}(f)|^2 \mathbf{S}_x(f). \quad (8.152)$$

## B. Cross-correlation between input and output

We begin by rewriting Eq. (8.147a) with the dummy variable  $\alpha$  changed to  $\alpha_1$ :

$$y(t) = \int_{-\infty}^{\infty} h(\alpha_1) x(t - \alpha_1) d\alpha_1. \quad (8.153)$$

Next, we multiply both sides by  $x(t - \alpha_2)$ :

$$y(t) x(t - \alpha_2) = \int_{-\infty}^{\infty} h(\alpha_1) x(t - \alpha_1) x(t - \alpha_2) d\alpha_1. \quad (8.154)$$

Taking the expectation  $E[\cdot]$  of both sides—while keeping in mind that  $x(t)$  is a zero-mean WSS process—leads to

$$R_{yx}(t, t - \alpha_2) = \int_{-\infty}^{\infty} h(\alpha_1) R_x(\alpha_2 - \alpha_1) d\alpha_1, \quad (8.155)$$

and

which states that  $x(t)$  and  $y(t)$  are jointly WSS. Noting that

$$R_{yx}(t, t - \alpha_2) = R_{yx}(t - (t - \alpha_2)) = R_{yx}(\alpha_2),$$

and the integral in Eq. (8.155) is in the form of a convolution leads to the result

$$R_{yx}(\alpha_2) = h(\alpha_2) * R_x(\alpha_2). \quad (8.156)$$

The frequency domain equivalent of Eq. (8.156) is

$$\mathbf{S}_{yx}(f) = \mathbf{H}(f) \mathbf{S}_x(f). \quad (8.157)$$

## 8-10.2 Discrete-Time Random Processes

The relations obtained in the preceding subsection are all for continuous-time random processes. If, instead, the zero-mean WSS random process  $x[n]$  is passed through an LTI system with impulse response  $h[n]$  to produce a random process  $y[n]$ , the

filtering process is a convolution depicted as

$$x[n] \rightarrow \boxed{h[n]} \rightarrow y[n], \quad (8.158)$$

and equivalent to

$$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} h[i] x[n-i]. \quad (8.159)$$

The discrete-time counterparts of the autocorrelation and cross-correlation relations given by Eqs. (8.151) and (8.156) are derived similarly and given by

$$R_y[n] = h[n] * h[-n] * R_x[n] \quad (8.160a)$$

and

$$R_{yx}[n] = h[n] * R_x[n]. \quad (8.160b)$$

These relationships presume our earlier characterization that  $x[n]$  and  $y[n]$  are zero-mean, jointly WSS processes.

The DTFT maps convolutions in the discrete-time domain to products in the frequency domain:

$$\mathbf{S}_y(\Omega) = |\mathbf{H}(\Omega)|^2 \mathbf{S}_x(\Omega) \quad (8.161a)$$

$$\mathbf{S}_{yx}(\Omega) = \mathbf{H}(\Omega) \mathbf{S}_x(\Omega) \quad (8.161b)$$

**Example 8-10: Continuous-Time Random Process**

The input  $x(t)$  to an LTI system defined by

$$\frac{dy}{dt} + 7y(t) = 5x(t)$$

is a zero-mean, white random process with  $R_x(\tau) = 3\delta(\tau)$ . Compute the power spectral density and autocorrelation function of the output  $y(t)$ .

**Solution:** Application of the Fourier transform to the system equation gives

$$(j2\pi f + 7) \mathbf{Y}(f) = 5\mathbf{X}(f),$$

where use was made of property #5 in **Table 2-4**. The system's

frequency response is

$$\mathbf{H}(f) = \frac{\mathbf{Y}(f)}{\mathbf{X}(f)} = \frac{5}{j2\pi f + 7}.$$

From Eq. (8.144),  $\mathbf{S}_x(f) = \sigma^2$  when  $x(t)$  is a white WSS with  $R_x(\tau) = \sigma^2 \delta(\tau)$ . Hence, in the present case,  $\mathbf{S}_x(f) = 3$ , and the power spectral density of  $y(t)$  is, from Eq. (8.152),

$$\mathbf{S}_y(f) = |\mathbf{H}(f)|^2 \mathbf{S}_x(f) = \left| \frac{5}{j2\pi f + 7} \right|^2 \times 3 = \frac{75}{4\pi^2 f^2 + 49}.$$

Using entry #3 in **Table 2-5**, the inverse Fourier transform of  $\mathbf{S}_y(f)$  is

$$R_y(\tau) = \frac{75}{14} e^{-7|\tau|}.$$

### Example 8-11: Discrete-Time Random Process

The input  $x[n]$  to a discrete-time LTI system given by

$$y[n] + 2y[n - 1] = 3x[n] + 4x[n - 1]$$

is a zero-mean white random process with  $R_x[n] = 3\delta[n]$ . Compute the power spectral density of  $y[n]$ .

**Solution:** Taking the DTFT of the system equation gives

$$(1 + 2e^{-j\Omega}) \mathbf{Y}(\Omega) = (3 + 4e^{-j\Omega}) \mathbf{X}(\Omega),$$

where use was made of entry #2 in **Table 2-7**. The system's frequency response is then

$$\mathbf{H}(\Omega) = \frac{\mathbf{Y}(\Omega)}{\mathbf{X}(\Omega)} = \frac{3 + 4e^{-j\Omega}}{1 + 2e^{-j\Omega}}.$$

Given that  $x[n]$  is a zero-mean white process with  $R_x[n] = 3\delta[n]$ , it follows that  $\mathbf{S}_x(\Omega) = 3$ , and by Eq. (8.161a),

$$\begin{aligned} \mathbf{S}_y(\Omega) &= |\mathbf{H}(\Omega)|^2 \mathbf{S}_x(\Omega) = \left| \frac{3 + 4e^{-j\Omega}}{1 + 2e^{-j\Omega}} \right|^2 \times 3 \\ &= \frac{3[(3 + 4\cos\Omega)^2 + 16\sin^2\Omega]}{(1 + 2\cos\Omega)^2 + 4\sin^2\Omega}. \end{aligned}$$

**Exercise 8-12:**  $x(t)$  is a white process with  $R_x(\tau) = 5\delta(\tau)$  and

$$y(t) = \int_{t-1}^{t+1} x(\tau) d\tau.$$

Compute the power spectral density of  $y(t)$ .

**Answer:**  $\mathbf{S}_y(f) = 20 \operatorname{sinc}^2(2f)$ . (See [IP](#)).

## 8-11 Random Fields

The preceding sections highlighted the properties and relations for 1-D random processes, in both continuous and discrete time. Now, we extend those results to 2-D space, and to distinguish between 1-D and 2-D processes, we refer to the latter as a **random field**, instead of a random process, and we also change our symbols to match the notation we used in earlier chapters to represent 2-D images.

### 8-11.1 Random Field Notation

(a) **Continuous-space** random field  $f(x, y)$  with 2-D continuous spatial dimensions  $x$  and  $y$ .

(b) **Discrete-space** random field  $f[n, m]$  with 2-D discrete spatial dimensions  $n$  and  $m$ .

(c) **Probability density and mass functions**

$p(f(x, y) = f'(x, y))$  = probability density that random field  $f(x, y)$  at  $(x, y)$  has value  $f'(x, y)$  (**continuous space**),  
 $p[f[n, m] = f'[n, m]]$  = probability mass that random field  $f[n, m]$  at  $[n, m]$  has value  $f'[n, m]$  (**discrete space**).

(d) **Mean values** for each  $(x, y)$  and  $[n, m]$

$$E[f(x, y)] = \int_{-\infty}^{\infty} f'(x, y) p(f'(x, y)) df', \quad (8.162a)$$

(**continuous space**)

$$E[f[n, m]] = \sum_{f'=-\infty}^{\infty} f'[n, m] p[f'[n, m]], \quad (8.162b)$$

(**discrete space**)

where  $p(f'(x, y))$  is the pdf of variable  $f'(x, y)$  and  $p[f'[n, m]]$  is the pmf of  $f'[n, m]$ .

## (e) Autocorrelation functions

$$R_f(x_1, y_1; x_2, y_2) = E[f(x_1, y_1) f(x_2, y_2)], \quad (8.163a)$$

**(continuous space)**

$$R_f(n_1, m_1; n_2, m_2) = E[f[n_1, m_1] f[n_2, m_2]]. \quad (8.163b)$$

**(discrete space)**

and

$$E[f^2[n, m]] = R_f(0, 0) = \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} S_f(\Omega_1, \Omega_2) d\Omega_1 d\Omega_2. \quad (8.165b)$$

**(discrete-space zero-mean WSS)**

As in 1-D, if the mean values  $E[f(x, y)]$  and  $E[f[n, m]]$  are zero, the autocorrelation functions are equal to the **autocovariance functions**:

$$R_f(x_1, y_1; x_2, y_2) = K_f(x_1, y_1; x_2, y_2) \quad \text{(continuous space),}$$

$$R_f[n_1, m_1; n_2, m_2] = K_f[n_1, m_1; n_2, m_2] \quad \text{(discrete space).}$$

## 8-11.2 WSS Random Fields

The attributes of 1-D WSS random processes were discussed earlier in Section 8-9.3. Generalizing to 2-D, a continuous-space zero-mean random field  $f(x, y)$  is WSS if

$$(1) \quad E[f(x, y)] = \text{constant for all } (x, y), \text{ and}$$

$$(2) \quad R_f(x, y) = E[f(x' + x, y' + y) f(x', y')].$$

The corresponding relations for discrete-space random fields are

$$(1) \quad E[f[n, m]] = \text{constant for all } [n, m], \text{ and}$$

$$(2) \quad R_f[n, m] = E[f[n' + n, m' + m] f[n', m']].$$

The power spectral density  $S_f(\mu, v)$  in the continuous-space frequency domain  $(\mu, v)$  is related to  $R_f(x, y)$  by

$$S_f(\mu, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} R_f(x, y) e^{-j2\pi(\mu x + vy)} dx dy,$$

**(continuous-space zero-mean WSS)** (8.164a)

and the 2-D equivalent of Eq. (8.140) is

$$E[f^2(x, y)] = R_f(0, 0) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S_f(\mu, v) d\mu dv.$$

**(continuous-space zero-mean WSS)** (8.164b)

For a discrete-space WSS random field with zero mean,

$$S_f(\Omega_1, \Omega_2) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} e^{-j(\Omega_1 n + \Omega_2 m)},$$

**(discrete-space zero-mean WSS)** (8.165a)

## 8-11.3 Filtering WSS Random Fields

Extending the material on 1-D random processes (Section 8-10) to 2-D, we have

$$f(x, y) \rightarrow \boxed{h(x, y)} \rightarrow g(x, y) = h(x, y) * * f(x, y),$$

**(continuous space)** (8.166a)

$$f[n, m] \rightarrow \boxed{h[n, m]} \rightarrow g[n, m] = h[n, m] * * f[n, m].$$

**(discrete space)** (8.166b)

If  $f(x, y)$  and  $f[n, m]$  are zero-mean WSS random fields, then

$$R_g(x, y) = h(x, y) * * h(-x, -y) * * R_f(x, y), \quad (8.167a)$$

$$R_g[n, m] = h[n, m] * * h[-n, -m] * * R_f[n, m], \quad (8.167b)$$

$$R_{gf}(x, y) = h(x, y) * * R_f(x, y), \quad (8.167c)$$

$$R_{gf}[n, m] = h[n, m] * * R_f[n, m], \quad (8.167d)$$

$$S_g(\mu, v) = |\mathbf{H}(\mu, v)|^2 S_f(\mu, v), \quad (8.167e)$$

$$S_g(\Omega_1, \Omega_2) = |\mathbf{H}(\Omega_1, \Omega_2)|^2 S_f(\Omega_1, \Omega_2), \quad (8.167f)$$

$$S_{gf}(\mu, v) = \mathbf{H}(\mu, v) S_f(\mu, v), \quad (8.167g)$$

$$S_{gf}(\Omega_1, \Omega_2) = \mathbf{H}(\Omega_1, \Omega_2) S_f(\Omega_1, \Omega_2). \quad (8.167h)$$

**Exercise 8-13:** A white random field  $f(x, y)$  with autocorrelation function  $R_f(x, y) = 2\delta(x)\delta(y)$  is filtered by an LSI system with PSF  $h(r) = \frac{1}{r}$ , where  $r$  is the radius in  $(x, y)$  space. Compute the power spectral density of the output random field  $g(x, y)$ .

**Answer:**

$$S_g(\mu, v) = \frac{2}{\mu^2 + v^2}.$$

(See [IP](#)).

## Summary

### Concepts

- A random variable is a number assigned to each random outcome.
- A probability density function (pdf) is the probability that a random variable  $x$  lies in an interval of length  $\delta x$ .
- A Gaussian random variable is described by its mean and variance.
- A random vector is a vector of random variables. Mean and variance generalize to mean vector and covariance

- matrix.
- A random process is a set of random variables indexed by time.
  - A wide-sense stationary (WSS) random process has constant mean and autocorrelation  $R_x[i, j] = R_x[i - j]$ . The constant mean is subtracted off.
  - A random field is a 2-D random process in 2-D space.

### Mathematical Formulae

#### Conditional probability

$$\mathbb{P}[E_1|E_2] = \frac{\mathbb{P}[E_1 \cap E_2]}{\mathbb{P}[E_2]}$$

#### Interval probability

$$\mathbb{P}[a \leq x < b] = \int_a^b p(x') dx'$$

#### Interval probability

$$\mathbb{P}[a_x \leq x < b_x, a_y \leq y < b_y] = \int_{a_x}^{b_x} \int_{a_y}^{b_y} p(x', y') dx' dy'$$

#### Conditional pdf

$$p(x|y) = \frac{p(x,y)}{p(y)}$$

#### Expectation

$$E[f(x)] = \bar{f}(x) = \int_{-\infty}^{\infty} f(x') p(x') dx'$$

#### Variance

$$\sigma_x^2 = E[x^2] - \bar{x}^2$$

#### Covariance

$$\lambda_{x,y} = E[xy] - \bar{x}\bar{y}$$

#### Gaussian pdf

$$p(x) = \frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-(x-\bar{x})^2/(2\sigma_x^2)}$$

#### Mean vector

$$E[\mathbf{x}] = \bar{\mathbf{x}} = [\bar{x}_1, \dots, \bar{x}_N]^T$$

#### Covariance matrix

$$\mathbf{K}_x = E[\mathbf{x}\mathbf{x}^T] - \bar{\mathbf{x}}\bar{\mathbf{x}}^T$$

#### Covariance matrix

$$\mathbf{K}_{Ax} = \mathbf{A}\mathbf{K}_x\mathbf{A}^T$$

#### Gaussian random vector

$$\mathbf{x} \sim \mathcal{N}(\bar{\mathbf{x}}, \mathbf{K}_x)$$

$$p(\mathbf{x}) = \frac{e^{-(1/2)(\mathbf{x}-\bar{\mathbf{x}})^T \mathbf{K}_x^{-1} (\mathbf{x}-\bar{\mathbf{x}})}}{(2\pi)^{N/2} (\det \mathbf{K}_x)^{1/2}}$$

#### Conditional Gaussian expectation

$$E[\mathbf{x}|\mathbf{y}] = E[\mathbf{x}] + \mathbf{K}_{x,y}\mathbf{K}_y^{-1}(\mathbf{y} - E[\mathbf{y}])$$

#### Autocovariance function

$$K_x[i, j] = E[x[i] x[j]] - E[x[i]] E[x[j]]$$

#### Autocorrelation function

$$R_x[i, j] = E[x[i] x[j]]$$

$$R_x(t_1, t_2) = E[x(t_1) x(t_2)]$$

#### Wide-sense stationary random process

$$R_x(t_1, t_2) = R_x(t_1 - t_2); \bar{x}(t) = m$$

#### Power spectral density

$$S_y(f) = \mathcal{F}\{R_y(t)\} = |\mathbf{H}(f)|^2 S_x(f)$$

#### Cross-spectral density

$$S_{yx}(f) = \mathcal{F}\{R_{x,y}(t)\} = \mathbf{H}(f) S_x(f)$$

**Important Terms**

Provide definitions or explain the meaning of the following terms:

autocovariance function  
axioms of probability  
conditional probability  
covariance matrix  
cross-covariance function

disjoint  
iid random process  
independent  
mutually exclusive  
pdf

pmf  
power spectral density  
probability and trees  
random field  
realization

sample and event spaces  
sample function  
vector random variable  
white random process  
wide-sense stationary

**PROBLEMS****Section 8-2: Conditional Probability**

**8.1 Three-Card Monte** (a game hucksters play with chumps—don't be a chump!). There are three cards. Card #1 is red on both sides. Card #2 is black on both sides. Card #3 is red on one side and black on the other side. The cards are shuffled and one chosen at random. The top of the chosen card is red. What is  $\mathbb{P}[\text{bottom of the chosen card is also red}]$ ? (The chump bets even money.)

**8.2 A Tale of Two Tosses.** Two coins have  $\mathbb{P}[\text{heads}]$  as follows: Coin A has  $\mathbb{P}[\text{heads}] = 1/3$ . Coin B has  $\mathbb{P}[\text{heads}] = 3/4$ . The results of all flips are independent. We choose a coin at random and flip it. Given that it came up heads, compute  $\mathbb{P}[\text{it was coin A}]$ . Hint: Draw a probability tree.

**8.3 Three Coins in the Fountain** (an old movie). Three coins have  $\mathbb{P}[\text{heads}]$ : Coin A has  $\mathbb{P}[\text{heads}] = 2/3$ . Coin B has  $\mathbb{P}[\text{heads}] = 3/4$ . Coin C has  $\mathbb{P}[\text{heads}] = 4/5$ . The results of all flips are independent. Coin A is flipped. Then:

- If coin A lands heads, we flip coin B.
- If coin A lands tails, we flip coin C.

Let H2 denote that the second coin flipped (whatever it is) lands heads.

- (a) Compute  $\mathbb{P}[\text{H2}]$ .
- (b) Compute  $\mathbb{P}[\text{Coin A landed heads, given H2}]$ . Now the second coin is flipped  $n - 1$  more times (for a total of  $n$  flips). Let H2n denote that all  $n$  flips of the second coin land heads.
- (c) Compute  $\mathbb{P}[\text{H2n}]$ .
- (d) Compute  $\mathbb{P}[\text{Coin A landed heads, given H2n}]$ .
- (e) What happens to your answer to (d) as  $n \rightarrow \infty$ ? Explain this.

Hint: Draw a probability tree.

**8.4 Bayes's Rule.** Widgets are made by two factories. Factory A makes 6000 widgets per year, 2% of which are defective. Factory B makes 4000 widgets per year, 3% of which are defective. A widget is chosen at random from the 10,000 widgets made in a year.

- (a) Compute  $\mathbb{P}[\text{the chosen widget is defective}]$ .
- (b) Compute  $\mathbb{P}[\text{the chosen widget came from factory A, given that it is defective}]$ .

**8.5 Bayes's Rule.** Widgets are made by two factories. Factory A makes 7000 widgets per year, 1% of which are defective. Factory B makes 3000 widgets per year, 2% of which are defective. A widget is chosen at random from the 10,000 widgets made in a year.

- (a) Compute  $\mathbb{P}[\text{the chosen widget is defective}]$ .
- (b) Compute  $\mathbb{P}[\text{the chosen widget came from factory A, given that it is defective}]$ .

**Section 8-3: Random Variables**

**8.6** Random variables  $x$  and  $y$  have the joint pdf

$$p(x,y) = \begin{cases} cx & \text{if } 1 < x < y < 2, \\ 0 & \text{otherwise,} \end{cases}$$

where  $c$  is a constant to be determined.

- (a) Compute the constant  $c$  in the pdf  $p(x,y)$ .
- (b) Are  $x$  and  $y$  independent? Explain your answer.
- (c) Compute the marginal pdf  $p(y)$ .
- (d) Compute the conditional pdf  $p(x|y)$  at  $y = 3/2$ .

**8.7** Random variables  $x$  and  $y$  have the joint pdf

$$p(x,y) = \begin{cases} cxy & \text{if } 0 < y < x < 1, \\ 0 & \text{otherwise,} \end{cases}$$

where  $c$  is a constant to be determined.

- (a) Compute the constant  $c$  in the pdf  $p(x,y)$ .  
 (b) Are  $x$  and  $y$  independent? Explain your answer.  
 (c) Compute the marginal pdf  $p(x)$ .  
 (d) Compute the conditional pdf  $p(y|x)$  at  $x = 1/2$ .

**8.8** Random variable  $x$  has the exponential pdf

$$\begin{cases} p(x) = \lambda e^{-\lambda x} & \text{for } x > 0, \\ 0 & \text{for } x < 0. \end{cases}$$

- (a) Confirm  $\int_{-\infty}^{\infty} p(x) dx = 1$ .  
 (b) Compute the expectation  $\bar{x}$ .  
 (c) Compute the variance  $\sigma_x^2$ .

### Section 8-5: Joint Pdfs and Pmfs

**8.9** Random variables  $m$  and  $n$  have the joint pmf

$$p[m,n] = \begin{cases} c & \text{if } 0 \leq n \leq m \leq 4, \\ 0 & \text{otherwise,} \end{cases}$$

where  $c$  is a constant to be determined and  $m$  and  $n$  are integers.

- (a) Compute the constant  $c$  in the pmf  $p[m,n]$ .  
 (b) Are  $m$  and  $n$  independent? Explain your answer.  
 (c) Compute the marginal pmf  $p[m]$ .  
 (d) Compute the conditional pmf  $p[n|m]$  at  $m = 2$ .  
 (e) Compute the conditional mean  $E[n|m]$  at  $m = 2$ .  
 (f) Compute the conditional variance  $\sigma_{n|m}^2$  at  $m = 2$ .

**8.10** Random variables  $m$  and  $n$  have the joint pmf

$$\begin{cases} p[m,n] = cm & \text{if } 0 \leq n \leq m \leq 4, \\ 0 & \text{otherwise,} \end{cases}$$

where  $c$  is a constant to be determined and  $m$  and  $n$  are integers.

- (a) Compute the constant  $c$  in the pmf  $p[m,n]$ .  
 (b) Compute the marginal pmf  $p[n]$ .  
 (c) Compute the conditional pdf  $p[n|m]$  at  $m = 2$ .

### Section 8-7: Random Vectors

**8.11** Random variables  $x$  and  $y$  are zero-mean and jointly Gaussian, with variances  $\sigma_x^2 = \sigma_y^2 = 1$  and covariance  $\lambda_{x,y} = \rho$  for some constant  $\rho \neq \pm 1$ .

- (a) Write out the formula for the joint pdf  $p(x,y)$ .  
 (b) Show that changing variables from  $(x,y)$  to  $(z,w)$ , where

$$\begin{bmatrix} z \\ w \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

yields two decorrelated ( $\lambda_{z,w} = 0$ ) random variables  $z$  and  $w$ .

**8.12** Random variables  $\{x_1, x_2, x_3\}$  are all zero-mean and jointly Gaussian, with variances  $\sigma_{x_i}^2 = 2$  and covariances  $\lambda_{x_i, x_j} = -1$  for  $i \neq j$ .

- (a) Write out the joint pdf  $p(x_1, x_2, x_3)$ . Use Eq. (8.85) notation.  
 (b) Write out the joint marginal pdf  $p(x_1, x_3)$ . Use Eq. (8.85) notation.  
 (c) Let  $y = x_1 + 2x_2 + 3x_3$ . Compute  $\sigma_y^2$ .  
 (d) Show that the conditional pdf  $p(x_1|x_2 = 2, x_3 = 3) = \delta(x_1 + 5)$ . Hint: Compute the eigenvalues and eigenvectors of the covariance matrix.

**8.13** Prove Eq. (8.111a) and Eq. (8.111b), which are:

$$\begin{aligned} E[\mathbf{x}|\mathbf{y} = \mathbf{y}'] &= E[\mathbf{x}] + \mathbf{K}_{\mathbf{x}, \mathbf{y}} \mathbf{K}_{\mathbf{y}}^{-1} (\mathbf{y}' - E[\mathbf{y}]), \\ \mathbf{K}_{\mathbf{x}|\mathbf{y}} &= \mathbf{K}_{\mathbf{x}} - \mathbf{K}_{\mathbf{x}, \mathbf{y}} \mathbf{K}_{\mathbf{y}}^{-1} \mathbf{K}_{\mathbf{x}, \mathbf{y}}^T. \end{aligned}$$

Hint: Define random vector  $\mathbf{w} = \mathbf{x} - \mathbf{K}_{\mathbf{x}, \mathbf{y}} \mathbf{K}_{\mathbf{y}}^{-1} \mathbf{y}$  and show that  $\mathbf{w}$  and  $\mathbf{y}$  are jointly Gaussian and uncorrelated, hence independent, random vectors.

### Section 8-9: LTI Filtering of Random Processes

**8.14**  $x(t)$  is a zero-mean WSS random process with power spectral density

$$S_x(f) = \frac{1}{9 + (2\pi f)^2}.$$

$x(t)$  is passed through the LTI system  $y(t) = \frac{dx}{dt} + 3x(t)$ .

- (a) Compute the power spectral density  $S_y(f)$ .  
 (b) Compute the cross-spectral density  $S_{yx}(f)$ .

**8.15**  $x(t)$  is a zero-mean WSS random process with power spectral density  $S_x(f) = 1$ .  $x(t)$  is passed through the LTI system  $\frac{dy}{dt} + 4y(t) = 3x(t)$ .

- (a) Compute the power spectral density  $S_y(f)$ .  
 (b) Compute the cross-spectral density  $S_{yx}(f)$ .

**8.16**  $x(t)$  is a zero-mean WSS random process with power spectral density  $\mathbf{S}_x(f) = 1$ .  $x(t)$  is passed through an LTI system with impulse response

$$h(t) = \begin{cases} 2e^{-3t} & \text{for } t > 0, \\ 0 & \text{for } t < 0. \end{cases}$$

- (a) Compute the power spectral density  $\mathbf{S}_y(f)$ .
- (b) Compute the cross-spectral density  $\mathbf{S}_{yx}(f)$ .

### Section 8-11: Random Fields

**8.17**  $f(x, y)$  is a zero-mean WSS Gaussian random field with power spectral density  $\mathbf{S}_f(\mu, \nu) = 1$ .  $g(x, y)$  is  $f(x, y)$  filtered by a brick-wall lowpass filter with cutoff spatial frequency  $\frac{1}{2}$ .

- (a) Compute the pdf of  $g(7, 5)$ .
- (b) Compute the joint pdf of  $\{g(7, 5), g(2, 4)\}$ .

**8.18**  $f(x, y)$  is a zero-mean WSS random field with power spectral density  $\mathbf{S}_f(\mu, \nu) = 1$ .  $g(x, y)$  is  $f(x, y)$  blurred by a Gaussian PSF, so  $g(x, y) = f(x, y) \ast \ast e^{-\pi(x^2+y^2)}$ .

- (a) Compute the variance  $\sigma_{g(3,5)}^2$ .
- (b) Compute the covariance  $\lambda_{g(3,5), g(2,4)}$ .

**8.19** A **fractal** random field  $f(x, y)$  has a power spectral density of form  $\mathbf{S}_f(\rho, \phi) = c/\rho^d$  for two constants  $c$  and  $d$ . Show that the Laplacian operator  $g(x, y) = \nabla^2 f(x, y)$  defined in Eq. (5.5) “whitens”  $f(x, y)$  (produces a white random field output  $g(x, y)$ ) if  $d = 4$ .

**8.20**  $f[n, m]$  is a zero-mean WSS white random field with power spectral density  $\mathbf{S}_f(\Omega_1, \Omega_2) = 1$ .  $g[n, m]$  is  $f[n, m]$  filtered with a Laplacian, so  $g[n, m] = \nabla^2 f[n, m]$ .  $\nabla^2 f[n, m]$  is the discrete-space Laplacian (Eq. (5.9)).  $g[n, m] = f[n, m] \ast \ast h_{\text{Laplace}}[n, m]$ . Eq. (5.10):

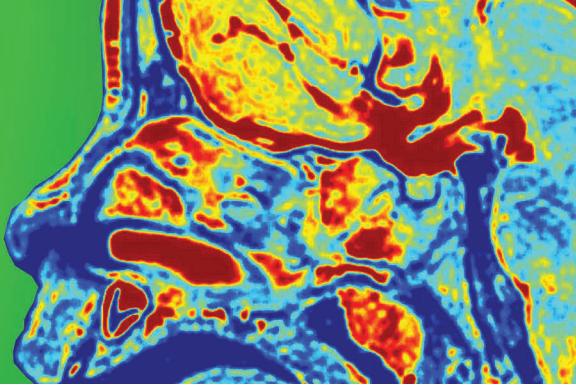
$$h_{\text{Laplace}}[n, m] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Compute the power spectral density of  $g[n, m]$ .

**8.21**  $f[n, m]$  is a zero-mean WSS Gaussian random field with autocorrelation function  $R_f[m, n] = 2^{6-|m|-|n|}$ .

- (a) Compute the pdf of  $f[5, 4]$
- (b) Compute the joint pdf of  $\{f[5, 4], f[3, 1]\}$ .

# CHAPTER 9



## 9

# Stochastic Denoising and Deconvolution

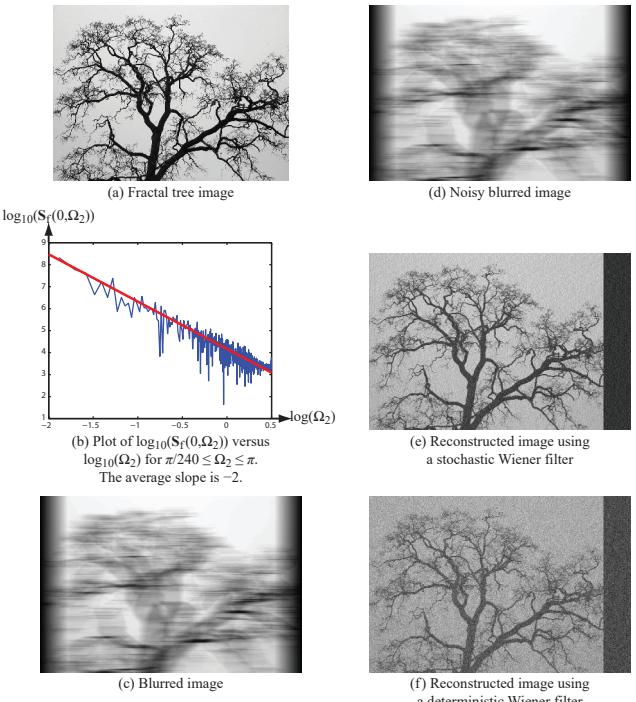
## Contents

- Overview, 292
- 9-1** Estimation Methods, 292
- 9-2** Coin-Flip Experiment, 298
- 9-3** 1-D Estimation Examples, 300
- 9-4** Least-Squares Estimation, 303
- 9-5** Deterministic versus Stochastic Wiener Filtering, 307
- 9-6** 2-D Estimation, 309
- 9-7** Spectral Estimation, 313
- 9-8** 1-D Fractals, 314
- 9-9** 2-D Fractals, 320
- 9-10** Markov Random Fields, 322
- 9-11** Application of MRF to Image Segmentation, 327
- Problems, 331

## Objectives

Learn to:

- Compute MLE, MAP, and LS estimates for small analytic problems.
- Estimate parameters of a fractal power spectral density from a signal or image.
- Denoise and deconvolve fractal signals and images using stochastic filters.
- Use thresholding and shrinkage of its wavelet transform to denoise an image.
- Use the ICM algorithm to segment an image.



This chapter provides a quick review of estimation theory, including MLE, MAP, LS and LLSE estimators. It then derives stochastic versions of the deterministic denoising and deconvolution filters presented in earlier chapters. Incorporating *a priori* information, in the form of power spectral density, is shown to greatly improve the performance of denoising and deconvolution filters on 1-D and 2-D problems. A very quick presentation of Markov random fields and the ICM algorithm for image segmentation, with examples, are also provided.

## Overview

In a **deterministic** (non-random) **inverse problem**, the goal is to compute an unknown 1-D signal  $x(t)$  or  $x[n]$ , or a 2-D image  $f(x,y)$  or  $f[n,m]$ , from observation of a corresponding signal  $y(t)$  or  $y[n]$ , or corresponding image  $g(x,y)$  or  $g[n,m]$ , wherein the unknown quantity and its observed counterpart are linked by a known model. The inversion process may also take into consideration **side information** about the unknown quantity, such as non-negativity, or *a priori* information, in the form of a pdf or pmf for it. The solution of an inverse problem may be difficult, but a well-posed inverse problem always has a unique solution.

An **estimation** problem is a **stochastic** (random) version of the inverse problem. The observation is a 1-D random variable or random process, or a 2-D random field. The random character of the observation may be associated with the inherent nature of the observing system itself—an example of which is a laser imager and certain radar systems—or due to additive noise introduced in the system's receiver. An image generated by a monochromatic laser imager or synthetic-aperture radar usually exhibits a speckle-like randomness superimposed on the true image intensity that would have been measured by a widebandwidth sensor.

Based on knowledge of the sources and mechanisms responsible for the randomness associated with the observed signal or image, we can incorporate randomness (**stochasticity**) in the model that relates the unknown quantity to the observation by modeling the unknown quantity as a random variable with a characteristic pdf or pmf. This pdf or pmf represents *a priori* information about the unknown quantities. Because of the stochastic nature of the inverse problem, we call it an **estimation problem**. The formulation of an estimation problem usually leads to a **likelihood function**, and the goal of the estimation problem becomes to maximize the likelihood function; the value of the unknown signal or image that maximizes the likelihood function is deemed the solution of the estimation problem.

In Section 9-1, we introduce three common approaches to estimation:

- (a) Maximum Likelihood Estimation (**MLE**),
- (b) Maximum *A Posteriori* Probability (**MAP**) Estimation,
- (c) Least-Squares Estimation (**LSE**).

In all three methods, the observation is presumed to be random in nature, and the unknown quantity also is presumed to be random but only in MAP and LSE; the unknown quantity is presumed to be non-random in MLE.

All three estimation methods are applied in Section 9-2 to the coin-flip experiment presented earlier in Section 8-1, obtaining three different estimators of  $P[\text{heads}]$ . This is then followed in Sections 9-3 to 9-5 with applications of the estimation methods to four 1-D estimation problems: (1) estimating the mean of a wide-sense stationary (WSS) random process, in the context of polling, (2) denoising a signal containing additive noise, (3) denoising a signal known to be sparse, and (4) deconvolving a signal from its noisy convolution with a known impulse response.

The treatment is extended to 2-D in Section 9-6, and in Section 9-7 we review the periodogram method for estimating power spectral densities of random processes and random fields. The periodogram can be used to obtain parametric forms of power spectral densities for classes of signals and images, such as fractals, which can then be used in the formulations developed in earlier sections. The procedures are outlined in Sections 9-8 for signals and 9-9 for images. The last two sections of the chapter are focused on an introduction to **Markov random fields (MRF)** and their application to image segmentation. An MRF incorporates **a priori information** (prior knowledge) about the relationships between the value of a given pixel and those of its neighbors.

## 9-1 Estimation Methods

This section introduces three estimation methods: MLE, MAP, and LSE, and even though our ultimate goal is to apply these methods to 2-D images (which we do in later sections), we will, for the present, limit the presentation to estimating a scalar **unknown**  $x$  from a scalar **observation**  $y_{\text{obs}}$  of a random variable  $y$ . In later sections we generalize the formulations to random vectors, processes, and fields.

An estimation problem consists of the following ingredients:

- (a)  $x$ : the unknown quantity to be estimated, which may have a constant value or it may be a random variable with a known pdf  $p(x)$ .
- (b)  $y_{\text{obs}}$ : the **observed value** of random variable  $y$ .
- (c)  $p(y | x = x')$ : the conditional pdf of  $y$ , given that  $x = x'$ , provided by a model.

In a typical situation,  $y$  and  $x$  are related by a model of the form

$$y(t) = h(t) * x(t) + v(t) \quad (\text{continuous time}),$$

or

$$y[n] = h[n] * x[n] + v[n] \quad (\text{discrete time}),$$

where  $h(t)$  and  $h[n]$  are the continuous-time and discrete-time impulse responses of the observing system and  $v(t)$  and  $v[n]$  represent random noise added by the measurement process. To obtain a good estimate of  $x(t)$  (or  $x[n]$ ), we need to filter out the noise and to deconvolve  $y(t)$  (or  $y[n]$ ).

We now introduce the basic structure of each of the three estimation methods.

### 9-1.1 Maximum Likelihood Estimation (MLE)

Among the three estimation methods, the **maximum likelihood estimation (MLE)** method is the one usually used when the unknown quantity  $x$  has a constant value, as opposed to being a random variable. The basic idea behind MLE is to choose the value of  $x$  that makes what actually happened, namely the observation  $y = y_{\text{obs}}$ , the most likely outcome. MLE maximizes the likelihood of  $y = y_{\text{obs}}$  (hence the name MLE) by applying the following recipe:

- (1) Set  $y = y_{\text{obs}}$  in the conditional pdf  $p(y|x)$  provided by the model to obtain  $p(y_{\text{obs}}|x)$ .
- (2) Choose the value of  $x$  that maximizes the **likelihood** function  $p(y_{\text{obs}}|x)$  and denote it  $\hat{x}_{\text{MLE}}$ .
- (3) If **side information** about  $x$  is available, such as  $x$  is non-negative or  $x$  is bounded within a specified interval, then incorporate that information in the maximization process.

In practice it is often easier to maximize the natural logarithm,  $\ln(p(y_{\text{obs}}|x))$ , rather than to maximize  $p(y_{\text{obs}}|x)$  itself. Henceforth,  $\ln(p(y_{\text{obs}}|x))$  will be referred to as the **log-likelihood function**.

- $p(y_{\text{obs}}|x)$  = likelihood function  
 $p(x)$  = *a priori* pdf  
 $p(x|y_{\text{obs}})$  = *a posteriori* pdf  
 $\ln(p(y_{\text{obs}}|x))$  = log-likelihood function

### 9-1.2 Maximum A Posteriori (MAP) Estimation

If the unknown quantity  $x$  is a random variable, the two methods most commonly used to estimate  $x$ , given an observation  $y = y_{\text{obs}}$ , are the **maximum a posteriori (MAP)** estimator and the **least-squares estimator (LSE)**. This subsection covers MAP and the next one covers LSE.

The observation  $y = y_{\text{obs}}$  is called **a posteriori** information because it is about the outcome. This is in contrast to **a priori**

information about the unknown (input) random variable  $x$  in the form of its pdf  $p(x)$ . For example,  $x$  may be known to have a **Gaussian pdf** (Fig. 9-1(a))

$$p(x) = \frac{e^{-(x-\bar{x})^2/(2\sigma_x^2)}}{\sqrt{2\pi\sigma_x^2}} \quad (\text{Gaussian}), \quad (9.1)$$

where  $\bar{x} = E[x]$  is the mean value of  $x$  and  $\sigma_x^2$  is its variance. Or  $x$  might be known to be a value of a sparse signal with **Laplacian pdf** (Fig. 9-1(b))

$$p(x) = \frac{e^{-\sqrt{2}|x|/\sigma_x}}{\sqrt{2}\sigma_x} \quad (\text{Laplacian}). \quad (9.2)$$

Gaussian and Laplacian *a priori* pdfs will be used later in Sections 9-3 and 9-4.

The idea behind MAP estimation is to determine the *most probable* value of  $x$ , given  $y = y_{\text{obs}}$ , which requires maximizing the **a posteriori pdf**  $p(x|y_{\text{obs}})$ . This is in contrast to the MLE method introduced earlier, which sought to maximize the likelihood function  $p(y_{\text{obs}}|x)$ . As we see shortly, in order to maximize  $p(x|y_{\text{obs}})$ , we need to know not only  $p(y_{\text{obs}}|x)$ , but also the **a priori pdf**  $p(x)$ .

As noted earlier, usually we know or have expressions for  $p(x)$  and the likelihood function  $p_{\text{obs}}(y|x)$ , but not for the *a posteriori* pdf  $p(x|y)$ . To obtain an expression for the latter (so we may maximize it), we use the conditional pdf relation given by Eq. (8.36b) to relate the joint pdf  $p(x,y)$  to each of the two conditional pdfs:

$$p(x,y) = p(y|x) p(x) \quad (9.3a)$$

and

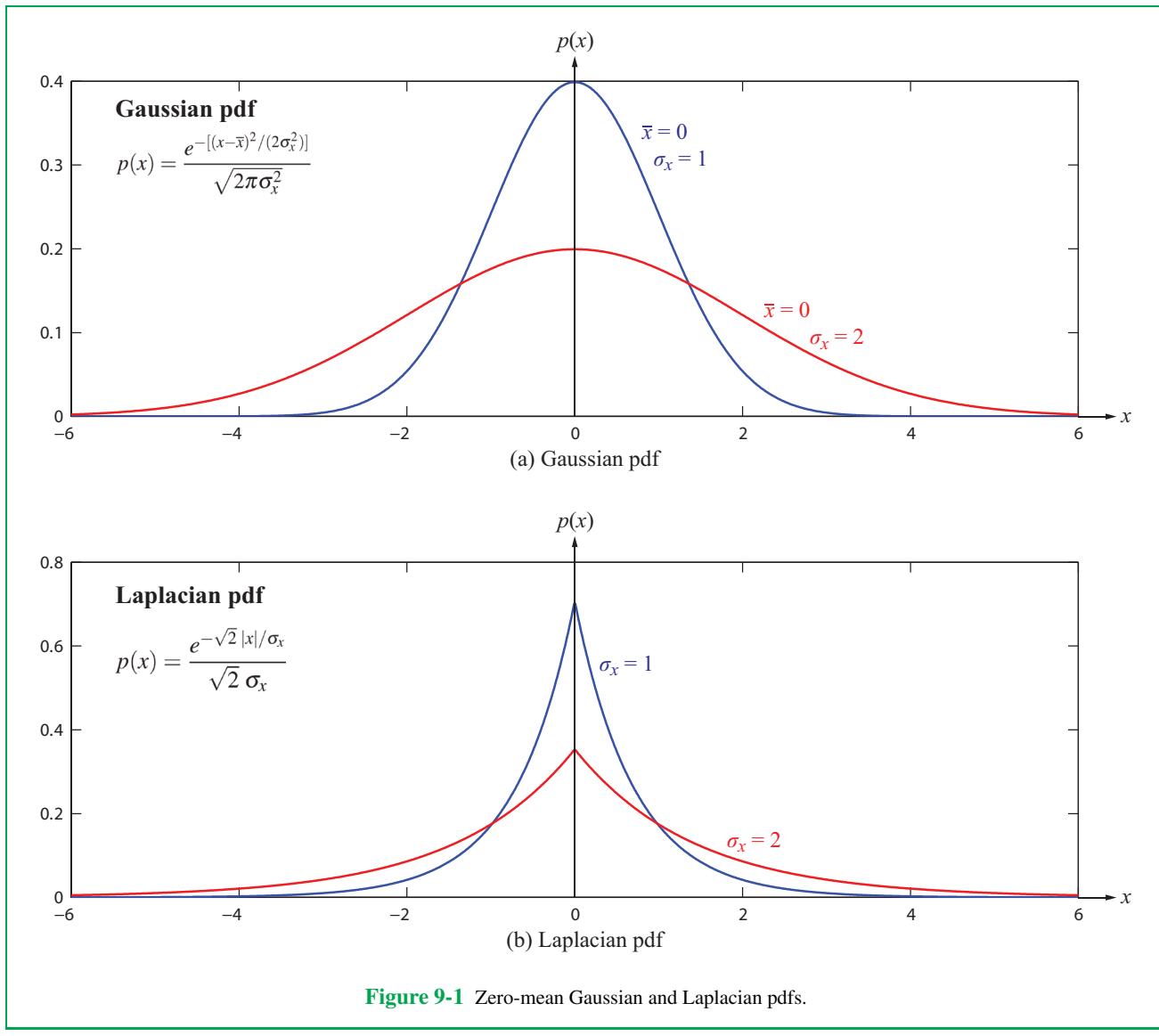
$$p(x,y) = p(x|y) p(y). \quad (9.3b)$$

Combining the two relations gives **Bayes's rule** for pdfs:

$$p(x|y) = p(y|x) \frac{p(x)}{p(y)}. \quad (9.4)$$

Bayes's rule relates the *a posteriori* pdf  $p(x|y_{\text{obs}})$  to the likelihood function  $p(y_{\text{obs}}|x)$ .

The goal of the MAP estimator is to choose the value of  $x$  that maximizes the *posteriori* pdf  $p(x|y_{\text{obs}})$ , given the *a priori* pdf  $p(x)$  and the likelihood function  $p(y_{\text{obs}}|x)$ . The third pdf in Eq. (9.4), namely  $p(y)$ , has no influence on the maximization process because it is not a function of  $x$ . Hence,  $p(y)$  can be set equal to any arbitrary constant value  $C$ .



**Figure 9-1** Zero-mean Gaussian and Laplacian pdfs.

The MAP estimator recipe consists of the following steps:

given by Eq. (9.4) and set  $p(y) = C$ :

$$p(x|y_{\text{obs}}) = p(y_{\text{obs}}|x) \frac{p(x)}{C} . \quad (9.5)$$

- (1) Set  $y = y_{\text{obs}}$  in the expression for the *a posteriori* pdf  $p(x|y)$

- (2) Choose the value of  $x$  that maximizes the *a posteriori* pdf  $p(x|y_{\text{obs}})$  and denote it  $\hat{x}_{\text{MAP}}$ .
- (3) If side information about  $x$  is available, incorporate that information in the estimation process.

As noted with the MLE method in the preceding subsection, it is often easier to maximize the natural logarithm of the *a posteriori* pdf:

$$\ln(p(x|y_{\text{obs}})) = \ln p(y_{\text{obs}}|x) + \ln p(x) - \ln C, \quad (9.6)$$

which is the sum of the logarithms of the likelihood function  $p(y_{\text{obs}}|x)$  and the *a priori* pdf  $p(x)$ . Subtracting  $\ln C$  does not affect the process of maximizing the value of  $x$ .

### 9-1.3 Least-Squares Estimator (LSE)

Whereas the MAP estimator sought to estimate  $x$  by maximizing the *a posteriori* pdf  $p(x|y_{\text{obs}})$  for a given observed value  $y_{\text{obs}}$ , the **least-squares estimator (LSE)** estimates the most probable value of  $x$  by minimizing the **mean square error (MSE)** between the estimated value  $\hat{x}$  and the random variable  $x$ . The MSE is defined as

$$\text{MSE} = E[(x - \hat{x}(y))^2] = \iint (x' - \hat{x}(y'))^2 p(x', y') dx' dy'. \quad (9.7)$$

Of course,  $\hat{x}$  is a function of  $y$ , and as we will show shortly, the value of  $x$  estimated by the LSE method, for a given value  $y = y_{\text{obs}}$ , is given by

$$\hat{x}_{\text{LS}} = E[x | y = y_{\text{obs}}] = \int x' p(x'|y_{\text{obs}}) dx', \quad (9.8)$$

where  $p(x|y_{\text{obs}})$  is the *a posteriori* pdf given by Eq. (9.4) with  $y = y_{\text{obs}}$ .

We now prove that  $\hat{x}_{\text{LS}}$  is indeed the value of  $x$  that minimizes the MSE by introducing the perturbation  $\varepsilon(y)$  as the deviation of  $\hat{x}(y)$  from  $E[x|y]$ :

$$\hat{x}(y) = E[x|y] - \varepsilon(y). \quad (9.9)$$

For  $y = y_{\text{obs}}$ ,  $E[x|y_{\text{obs}}]$  is the LS estimator  $\hat{x}_{\text{LS}}$  given by Eq. (9.8), so our task is to demonstrate that the MSE given by Eq. (9.7) is at its minimum when  $\varepsilon(y) = 0$ . To that end, let us replace  $\hat{x}(y)$  in the MSE expression given by Eq. (9.7) with the *perturbed* definition given by Eq. (9.9):

$$\text{MSE} = E[(x - \hat{x}(y))^2] = E[(x - E[x|y] + \varepsilon(y))^2]. \quad (9.10)$$

From the definition on the right-hand side of Eq. (9.7), it is

apparent from the integration that both  $x'$  and  $y'$  are treated as random variables, so we need to keep that in mind in what follows.

Since expectation is a linear operator,

$$\begin{aligned} E[(x - \hat{x}(y))^2] &= E[(x - E[x|y] + \varepsilon(y))^2] \\ &= E[(x - E[x|y])^2] + E[\varepsilon^2(y)] \\ &\quad + 2E[x\varepsilon(y)] - 2E[E[x|y]\varepsilon(y)]. \end{aligned} \quad (9.11)$$

We now show that the final two terms cancel each other. Using Eq. (8.36a), namely

$$p(x', y') = p(x'|y') p(y'), \quad (9.12)$$

the third term in Eq. (9.11) becomes

$$\begin{aligned} 2E[x\varepsilon(y)] &= 2 \iint x' \varepsilon(y') p(x', y') dx' dy' \\ &= 2 \iint x' \varepsilon(y') p(x'|y') p(y') dx' dy' \\ &= 2 \int p(y') \varepsilon(y') \int x' p(x'|y') dx' dy' \\ &= 2 \int p(y') \varepsilon(y') E[x | y = y'] dy' \\ &= 2E[\varepsilon(y) E[x|y]]. \end{aligned} \quad (9.13)$$

This result shows that the third and fourth terms in Eq. (9.11) are identical in magnitude but opposite in sign, thereby canceling one another out. Hence, the MSE is

$$\text{MSE} = E[(x - \hat{x}(y))^2] = E[(x - E[x|y])^2] + E[\varepsilon^2(y)]. \quad (9.14)$$

The MSE is minimum when the term  $E[\varepsilon^2(y)] = 0$ , which, in turn, requires that  $\varepsilon(y) = 0$ . This result proves that the conditional mean given by Eq. (9.8) is indeed the solution that minimizes the MSE.

Readers familiar with mechanics will recognize the result given by Eq. (9.14) as the **parallel-axis theorem**: The **moment of inertia** of a planar object is minimized when the axis of rotation goes through the **center of mass** of the object. Here, MSE represents the moment of inertia and  $\hat{x}(y)$  represents the center of mass.

Now that we have shown that  $\hat{x}_{\text{LS}}$  is given by Eq. (9.8), let us examine how to compute it. Using Bayes's rule from Eq. (9.4),

Eq. (9.8) becomes

$$\begin{aligned}\hat{x}_{\text{LS}} &= \int x' p(y_{\text{obs}}|x') \frac{p(x')}{p(y=y_{\text{obs}})} dx' \\ &= \frac{1}{p(y=y_{\text{obs}})} \int x' p(y_{\text{obs}}|x') p(x') dx'.\end{aligned}\quad (9.15)$$

The quantity  $p(y=y_{\text{obs}})$  can be determined from

$$p(y=y_{\text{obs}}) = \int p(y_{\text{obs}}|x') p(x') dx', \quad (9.16)$$

leading to the final expression

$$\hat{x}_{\text{LS}} = \frac{\int x' p(y_{\text{obs}}|x') p(x') dx'}{\int p(y_{\text{obs}}|x') p(x') dx'}. \quad (9.17)$$

As noted at the outset of Section 9-1, the pdfs  $p(x)$  and  $p(y|x)$  are made available by the model describing the random variable  $x$  and its relationship to random variable  $y$ . Hence, computing  $\hat{x}_{\text{LS}}$  using Eq. (9.17) should be a straightforward task.

Generalizing to the case where  $\mathbf{x}$  and  $\mathbf{y}$  are random vectors and jointly Gaussian, the LS vector-equivalent of Eq. (9.8) is

$$\hat{\mathbf{x}}_{\text{LS}} = E[\mathbf{x} | \mathbf{y} = \mathbf{y}_{\text{obs}}] = \bar{\mathbf{x}} + \mathbf{K}_{\mathbf{x}, \mathbf{y}} \mathbf{K}_{\mathbf{y}}^{-1} (\mathbf{y}_{\text{obs}} - \bar{\mathbf{y}}), \quad (9.18)$$

#### (jointly Gaussian vectors)

where we used Eq. (8.111a). Here,  $\bar{\mathbf{x}}$  and  $\bar{\mathbf{y}}$  are the mean values of vectors  $\mathbf{x}$  and  $\mathbf{y}$ , and  $\mathbf{K}_{\mathbf{y}}$  and  $\mathbf{K}_{\mathbf{x}, \mathbf{y}}$  are the autocorrelation and cross-correlation matrices defined in Section 8-7.1. We note that  $\hat{\mathbf{x}}_{\text{LS}}$  is linearly related to the observation vector  $\mathbf{y}_{\text{obs}}$ , which will prove useful later.

We should note that the MAP and LS methods are sometimes called *Bayesian estimators* because they take advantage of *a priori* information about  $x$  in the form of  $p(x)$ , whereas the MLE method does not use  $p(x)$ . The possible drawback of the Bayesian methods is that if  $p(x)$  is incorrect, their estimates may prove to be inferior to that of the MLE estimate, but if  $p(x)$  is correct and applicable, the Bayesian methods are likely to produce more accurate estimates of  $x$  than MLE.

**Exercise 9-1:** How does Eq. (9.18) simplify when random vectors  $\mathbf{x}$  and  $\mathbf{y}$  are replaced with random variables  $x$  and  $y$ ?

**Answer:**

$$\hat{x}_{\text{LS}} = \bar{x} + \frac{\lambda_{x,y}}{\sigma_y^2} (y_{\text{obs}} - \bar{y}).$$

## 9-1.4 Discrete-Value Estimators

The formulations associated with the three estimation methods covered in the preceding subsections are specific to continuous-value quantities, wherein the symbol  $x$  denotes the unknown quantity we wish to estimate and  $y$  denotes the observation  $y_{\text{obs}}$ . Now we consider discrete-value variables  $m$  and  $n$ , with:

(a)  $m$ : the unknown quantity to be estimated, which may have a constant value or it may be a random variable with a known pmf  $p[m]$ .

(b)  $n$ : the *observed value* of a discrete random variable  $n$ .

Also available from the model relating  $m$  to  $n$  is:

(c)  $p[n|m]$ : the conditional pmf.

Occasionally, we may encounter scenarios in which the observation  $n$  is a discrete random variable, but the unknown is a continuous-value quantity. In such cases, we continue to use  $n$  as the discrete-value observation, but we use  $x$  and  $p(x)$  instead of  $m$  and  $p[m]$  for the continuous-value random variable.

Transforming from  $(x, y)$  notation to  $[m, n]$  notation and from integrals to sums leads to the formulations that follow.

### A. MLE

$\hat{m}_{\text{MLE}}$  = the MLE-estimated value of  $m$ , determined by maximizing the likelihood function (or its logarithm):

$$p[n_{\text{obs}}|m] \quad (\text{both } n \text{ and } m \text{ are discrete}). \quad (9.19a)$$

To maintain notational consistency, if the unknown quantity is continuous,  $m$  should be replaced with  $x$ :

$$p[n_{\text{obs}}|x] \quad (\text{x continuous and } n \text{ discrete}). \quad (9.19b)$$

### B. MAP

$\hat{m}_{\text{MAP}}$  = the MAP-estimated value of  $m$ , determined by maximizing the *a posteriori* pdf

$$p[m|n_{\text{obs}}] = \frac{p[n_{\text{obs}}|m] p[m]}{p[n_{\text{obs}}]}. \quad (9.20a)$$

(**both  $n$  and  $m$  are discrete**)

As noted in connection with Eq. (9.5),  $p[n_{\text{obs}}]$  exercises no effect on the maximization process, so it can be set to any arbitrary

constant value  $C$ . Also, if the unknown quantity is continuous,  $m$  and  $p[m]$  should be replaced with  $x$  and  $p(x)$ :

$$p(x|n_{\text{obs}}) = \frac{p[n_{\text{obs}}|x] p(x)}{p[n_{\text{obs}}]} \quad (\text{x continuous and n discrete}). \quad (9.20b)$$

## C. LSE

$$\hat{m}_{\text{LS}} = \frac{\sum_{m'=-\infty}^{\infty} m' p[n_{\text{obs}}|m'] p[m']}{\sum_{m'=-\infty}^{\infty} p[n_{\text{obs}}|m'] p[m']} . \quad (9.21a)$$

(both  $m$  and  $n$  are discrete)

This expression, which is the discrete counterpart of the expression given by Eq. (9.17), is applicable only if both  $m$  and  $n$  are discrete random variables. If the unknown quantity is a continuous random variable and  $n$  is a discrete random variable, then  $m$  and  $p[m]$  should be replaced with  $x$  and  $p(x)$ , respectively, and the summations in Eq. (9.21a) should be converted to integrals, namely

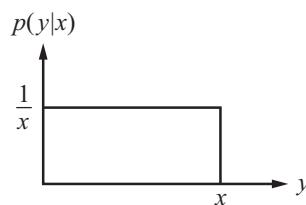
$$\hat{x}_{\text{LS}} = \frac{\int x' p[n_{\text{obs}}|x'] p(x') dx'}{\int p[n_{\text{obs}}|x'] p(x') dx'} . \quad (9.21b)$$

(x continuous and n discrete)

### Example 9-1: MLE

We are given  $N$  independent observations of a random variable  $y$  with uniform pdf:

$$p(y|x) = \begin{cases} 1/x & \text{for } 0 \leq y \leq x, \\ 0 & \text{otherwise.} \end{cases}$$



Compute  $\hat{x}_{\text{MLE}}$  for (a)  $N = 1$  and (b) arbitrary  $N$ .

**Solution: (a)** For  $N = 1$ , we are given a single observation,  $y_1$  of  $y$ . Since  $p(y|x) = 0$  for  $y > x$ , we require  $\hat{x}_{\text{MLE}} > y_1$ . The height of the pdf for  $0 \leq y \leq x$  is  $1/x$ . This height is maximized by choosing  $\hat{x}_{\text{MLE}}$  to be as small as possible. The smallest possible value of  $\hat{x}_{\text{MLE}}$  for which  $p(y_1|x) \neq 0$  is  $\hat{x}_{\text{MLE}} = y_1$ .

**(b)** Repeating this argument for each of  $N$  independent observations  $\{y_1, \dots, y_N\}$ , we require the smallest value of  $\hat{x}_{\text{MLE}}$  such that  $\hat{x}_{\text{MLE}} > \{y_1, \dots, y_N\}$ . This is  $\hat{x}_{\text{MLE}} = \max\{y_1, \dots, y_N\}$ .

The likelihood function is nonzero in the  $N$ -dimensional hypercube  $0 \leq y_n \leq x$ :

$$p(y_1, \dots, y_N|x) = \begin{cases} 1/x^N & \text{for } 0 \leq y_n \leq x, \\ 0 & \text{otherwise,} \end{cases}$$

which is maximized by minimizing  $x^N$  subject to  $0 \leq y_n \leq x$ . Hence  $\hat{x}_{\text{MLE}} = \max\{y_1, \dots, y_N\}$ .

**Concept Question 9-1:** What is the essential difference between MLE and MAP estimation?

**Exercise 9-2:** Determine  $\hat{x}_{\text{LS}}$  and  $\hat{x}_{\text{MAP}}$ , given that only the *a priori* pdf  $p(x)$  is known.

**Answer:**  $\hat{x}_{\text{LS}} = \bar{x}$ , and  $\hat{x}_{\text{MAP}} = \text{value of } x \text{ at which } p(x) \text{ is the largest.}$

**Exercise 9-3:** We observe random variable  $y$  whose pdf is  $p(y|\lambda) = \lambda e^{-\lambda y}$  for  $y > 0$ . Compute the maximum likelihood estimate of  $\lambda$ .

**Answer:** The log-likelihood function is

$$\ln p(y_{\text{obs}}|\lambda) = \ln \lambda - \lambda y_{\text{obs}}.$$

Setting its derivative to zero gives  $\frac{1}{\lambda} - y_{\text{obs}} = 0$ . Solving gives

$$\hat{\lambda}_{\text{MLE}}(y_{\text{obs}}) = \frac{1}{y_{\text{obs}}} .$$

**Exercise 9-4:** We observe random variable  $y$  whose pdf is  $p(y|\lambda) = \lambda e^{-\lambda y}$  for  $y > 0$  and  $\lambda$  has the *a priori* pdf  $p(\lambda) = 3\lambda^2$  for  $0 \leq \lambda \leq 1$ . Compute the maximum *a posteriori* estimate of  $\lambda$ .

**Answer:** The *a posteriori* pdf is

$$p(\lambda | y_{\text{obs}}) = \frac{p(y_{\text{obs}}|\lambda) p(\lambda)}{p(y_{\text{obs}})}.$$

Its logarithm is  $\ln \lambda - \lambda y_{\text{obs}} + 2 \ln \lambda + \ln(3) - \ln(p(y_{\text{obs}}))$ . Setting its derivative to zero gives  $\frac{3}{\lambda} - y_{\text{obs}} = 0$ . Solving gives

$$\hat{\lambda}_{\text{MAP}}(y_{\text{obs}}) = \frac{3}{y_{\text{obs}}}$$

if  $3/y_{\text{obs}} < 1$  since  $\lambda < 1$ . Note that  $\lambda_{\text{MAP}}(y_{\text{obs}}) = 3\lambda_{\text{MLE}}(y_{\text{obs}})$  as  $\lambda$  is likely to be large.

## 9-2 Coin-Flip Experiment

In Section 8-1.4, we described a coin-flip experiment in which a coin is flipped  $N$  times. The result of the  $k$ th flip, with  $1 \leq k \leq N$ , is either a head event and designated by  $H_k$ , or a tail event and designated by  $T_k$ . The result of any flip has no effect on the result of any other flip, but the coin can be *biased* towards heads or tails. For any given coin, the probability of heads for any flip is an unknown number  $a$ :

$$\begin{aligned}\mathbb{P}[H_k] &= a, \\ \mathbb{P}[T_k] &= 1 - a,\end{aligned}$$

and  $0 \leq a \leq 1$ . For a *fair* coin,  $a = 0.5$ , but we wish to consider the more general case where the coin can be such that  $a$  can assume any desired value between 0 (all tails) and 1 (all heads).

We now use the coin-flip experiment as a vehicle to test the three estimation methods of the preceding section. In the context of the coin-flip experiment, our unknown and observation quantities are:

- (1)  $x$  = unknown probability of heads, which we wish to estimate on the basis of a finite number of observations,  $N$ . If we were to flip the coin an infinite number of times, the estimated value of  $x$  should be  $a$ , the true probability of heads, but in our experiment  $N$  is finite. For the sake of the present exercise, we set  $N = 10$ .

- (2)  $n_{\text{obs}}$  = number of heads observed in  $N = 10$  flips.

As noted earlier, the MLE method does not use an *a priori* pdf for  $x$ , but the MAP and LS methods do. Consequently, the three estimation methods result in three different estimates  $\hat{x}(n_{\text{obs}})$ .

### 9-2.1 MLE Coin Estimate

For the coin-flip experiment, the MLE likelihood function given by Eq. (9.19b) is the conditional binomial pmf given by Eq. (8.31), with the notation changed from  $p(n)$  to  $p(n_{\text{obs}}|x)$  and  $a$  replaced with  $x$ . The pmf given by Eq. (8.31) presumes that  $a$  is a known quantity, whereas in the present case  $a$  is the unknown value of the quantity  $x$ . The notation conversion gives

$$p[n_{\text{obs}}|x] = \frac{x^{n_{\text{obs}}} (1-x)^{N-n_{\text{obs}}} N!}{n_{\text{obs}}! (N-n_{\text{obs}})!}. \quad (9.22)$$

Here,  $n$  is a discrete-value random variable, but  $x$  is continuous. The log-likelihood function is

$$\begin{aligned}\ln(p[n_{\text{obs}}|x]) &= n_{\text{obs}} \ln x + (N-n_{\text{obs}}) \ln(1-x) \\ &\quad + \ln N! - \ln n_{\text{obs}}! - \ln(N-n_{\text{obs}})!. \quad (9.23)\end{aligned}$$

Only the first two terms are functions of  $x$ . The log-likelihood function is maximized by setting its partial derivative with respect to  $x$  to zero:

$$0 = n_{\text{obs}} \frac{\partial}{\partial x} (\ln x) + (N-n_{\text{obs}}) \frac{\partial}{\partial x} (\ln(1-x)) = \frac{n_{\text{obs}}}{x} - \frac{N-n_{\text{obs}}}{1-x}, \quad (9.24)$$

where we used the relations

$$\frac{\partial}{\partial x} (\ln x) = \frac{1}{x}$$

and

$$\frac{\partial}{\partial x} (\ln(1-x)) = -\frac{1}{1-x}.$$

Solving Eq. (9.24) for  $x$  gives

$$\hat{x}_{\text{MLE}} = x = \frac{n_{\text{obs}}}{N}, \quad (9.25)$$

and for the coin-flip experiment with  $N = 10$ ,  $\hat{x}_{\text{MLE}} = n_{\text{obs}}/10$ .

This result not only satisfies the condition  $0 \leq x \leq 1$ , but it is also intuitively obvious. It says that if we flip the coin 10 times and 6 of those flips turn out to be heads, then the most-likelihood estimate of the probability that any individual flip is a head is 0.6. As we will see shortly, the MAP and LS estimators provide slightly different estimates.

### 9-2.2 MAP Coin Estimate

Whereas the MLE method does not use an *a priori* probabilistic information about the unknown quantity  $x$ , the MAP method uses the pmf  $p[m]$  if  $m$  is a discrete-value random variable or

the pdf  $p(x)$  if  $x$  is a continuous-value random variable. For the coin-flip experiment, let us assume that  $x$  is continuous, with  $0 \leq x \leq 1$ , and that we know the coin is biased such that values of  $x$  closer to 1 are more likely. Specifically, we are given the *a priori* pdf

$$p(x) = \begin{cases} 2x & \text{for } 0 \leq x \leq 1, \\ 0 & \text{otherwise,} \end{cases} \quad (9.26)$$

which states that the higher the value of  $x$  is, the greater is its pdf. This is in contrast to the *uniform pdf*  $p(x) = 1$  for  $0 \leq x \leq 1$ , which allocates the same probability density to all values of  $x$  within the specified range. We note that the expression given by Eq. (9.26) satisfies the condition that

$$\int_0^1 p(x) dx = 1.$$

The MAP estimator obtains  $\hat{x}_{\text{MAP}}$  by maximizing the *a posteriori* pdf given by Eq. (9.20b):

$$p(x|n_{\text{obs}}) = \frac{1}{C} p[n_{\text{obs}}|x] p(x) \quad (9.27)$$

$$= \frac{2x}{C} \frac{x^{n_{\text{obs}}} (1-x)^{N-n_{\text{obs}}} N!}{n_{\text{obs}}! (N-n_{\text{obs}})!}. \quad (9.28)$$

where we used Eqs. (9.22) and (9.26), and  $p[n]$  has been set to a constant value because it has no effect on the maximization process. The natural logarithm of the pdf is

$$\begin{aligned} \ln(p(x|n_{\text{obs}})) &= \ln\left(\frac{2N!}{Cn_{\text{obs}}! (N-n_{\text{obs}})!}\right) \\ &\quad + \ln((x)^{n_{\text{obs}}} + 1) + (N - n_{\text{obs}}) \ln(1 - x) \\ &= \ln\left(\frac{2N!}{Cn_{\text{obs}}! (N-n_{\text{obs}})!}\right) \\ &\quad + (n_{\text{obs}} + 1) \ln x + (N - n_{\text{obs}}) \ln(1 - x). \end{aligned} \quad (9.29)$$

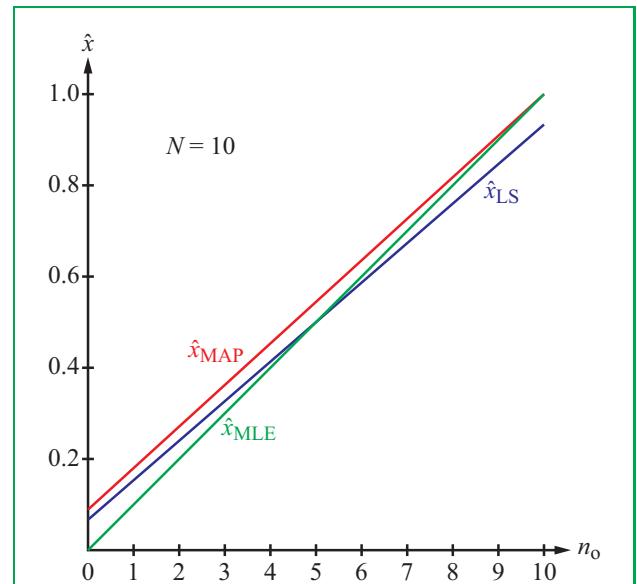
Only the last two terms are functions of  $x$ , so when we take the partial derivative with respect to  $x$  and set it equal to zero, we get

$$0 = \frac{n_{\text{obs}} + 1}{x} - \frac{N - n_{\text{obs}}}{1 - x},$$

whose solution for  $x$  yields the MAP estimate

$$\hat{x}_{\text{MAP}} = x = \frac{n_{\text{obs}} + 1}{N + 1}. \quad (9.30)$$

For the specific case where  $N = 10$ ,  $\hat{x}_{\text{MAP}} = (n_{\text{obs}} + 1)/11$ ,



**Figure 9-2** Variation of  $\hat{x}_{\text{MLE}}$ ,  $\hat{x}_{\text{MAP}}$ , and  $\hat{x}_{\text{LS}}$  with  $n_{\text{obs}}$ , the number of coin flips that were observed to be heads, for  $N = 10$  coin flips and  $p(x') = 2x'$  for  $0 \leq x' \leq 1$ .

which is different from the MLE estimate  $\hat{x}_{\text{MLE}} = n_{\text{obs}}/10$ . **Figure 9-2** displays a plot of  $\hat{x}_{\text{MLE}}$ ,  $\hat{x}_{\text{MAP}}$ , and  $\hat{x}_{\text{LS}}$  as a function of  $n_{\text{obs}}$ , all for  $N = 10$ . We observe that even when none of the 10 coin flips is a head (i.e.,  $n_{\text{obs}} = 0$ ), the MAP estimator predicts  $\hat{x}_{\text{MAP}} = 1/11 \approx 0.09$ , but as  $n_{\text{obs}}$  approaches  $N = 10$ , both estimators approach the same limit of 1.

## 9-2.3 LS Coin Estimate

Since  $x$  is a continuous random variable and  $n$  a discrete random variable, the applicable expression for  $\hat{x}_{\text{LS}}$  is the one given by Eq. (9.21b):

$$\hat{x}_{\text{LS}} = \frac{\int x' p[n_{\text{obs}}|x'] p[x'] dx'}{\int p[n_{\text{obs}}|x'] p[x'] dx'}. \quad (9.31)$$

After inserting the expression for  $p[n_{\text{obs}}|x]$  given by Eq. (9.22) and the expression for  $p(x)$  given by Eq. (9.26) into Eq. (9.31), and then canceling terms that are common to the numerator and

denominator but not functions of  $x$ , we have

$$\hat{x}_{\text{LS}} = \frac{\int_0^1 (x')^{n_{\text{obs}}+1} (1-x')^{N-n_{\text{obs}}} dx'}{\int_0^1 (x')^{n_{\text{obs}}} (1-x')^{N-n_{\text{obs}}} dx'}. \quad (9.32)$$

The integration is somewhat cumbersome, but it leads to the result

$$\hat{x}_{\text{LS}} = \frac{n_{\text{obs}} + 1}{N + 2}. \quad (9.33)$$

For  $N = 10$ ,  $\hat{x}_{\text{LS}} = (n_{\text{obs}} + 1)/12$ . The variation of  $\hat{x}_{\text{LS}}$  with  $n_{\text{obs}}$ , which is displayed in [Fig. 9-2](#), starts at a value slightly smaller than  $\hat{x}_{\text{MAP}}$  at  $n_{\text{obs}} = 0$  and concludes at  $\hat{x}_{\text{LS}} = 0.92$  at  $n_{\text{obs}} = 10$ .

The same coin-flip experiment has produced three different estimators, depending on the estimation criterion and use or not use of the *a priori* pdf.

**Answer:** The *a posteriori* pdf is

$$p(x|n_{\text{obs}}) = \frac{p[n_{\text{obs}}|x] p(x)}{p[n_{\text{obs}}]}.$$

Its logarithm, excluding constants, is

$$\ln p(x|n_{\text{obs}}) = n_{\text{obs}} \ln x + (N - n_{\text{obs}} + 1) \ln(1 - x).$$

Setting its derivative to zero gives

$$\frac{n_{\text{obs}}}{x} - \frac{N - n_{\text{obs}} + 1}{1 - x} = 0.$$

Solving gives

$$\hat{x}_{\text{MAP}}(n_{\text{obs}}) = \frac{n_{\text{obs}}}{N + 1}.$$

Note that  $\hat{x}_{\text{MAP}}(n_{\text{obs}}) < \hat{x}_{\text{MLE}}(n_{\text{obs}})$  as  $x$  is likely to be small.

## 9-3 1-D Estimation Examples

In Section 9-2, we introduced three different approaches to estimation: MLE, MAP, and LS. We used the coin-flip problem as an example for demonstrating all three approaches, computing different estimates of  $\mathbb{P}[\text{heads}]$  based on observations of the number of heads in ten flips of the coin. Now we apply the three estimation methods to the following discrete-time 1-D estimation scenarios: (1) estimating the mean value of an *independent and identically distributed (IID)* random process (as described in Section 8-9.1 A), in the context of polling, (2) denoising a random process containing additive white Gaussian noise, given its power spectral density, (3) deconvolving a random process containing additive white Gaussian noise, and (4) deconvolving a sparse random process containing additive white Gaussian noise. We present these 1-D scenarios as a prelude to similar presentations for 2-D images in Section 9-4.

Denoising and deconvolution were presented in Chapter 6, so they should be familiar. The problem of estimating the expectation of an IID Gaussian random process leads to an estimator called the *sample mean*, which we present in the context of combining poll results.

**Concept Question 9-2:** How could we get three different estimates of  $\mathbb{P}[\text{heads}]$  from the same number of heads?

**Exercise 9-5:** Compute the MAP estimator for the coin-flip experiment assuming the *a priori* uniform pdf  $p(x) = 1$ , for  $0 \leq x \leq 1$ .

**Answer:**  $\hat{x}_{\text{MAP}} = \hat{x}_{\text{MLE}} = n_{\text{obs}}/N$ .

**Exercise 9-6:** Compute the MAP estimator for the coin-flip experiment with  $N = 10$  and the *a priori* pdf  $p(x) = 3x^2$ , for  $0 \leq x \leq 1$ .

**Answer:**  $\hat{x}_{\text{MAP}} = (n_{\text{obs}} + 2)/12$ .

**Exercise 9-7:** Compute the MAP estimator for the coin-flip experiment with a triangular *a priori* pdf  $p(x) = 2(1-x)$  for  $0 \leq x \leq 1$ .

### 9-3.1 Polling Scenario

Suppose an election is to take place involving two candidates, A and B. The fraction of voters who will vote for candidate A is the unknown constant  $\mu$ , which we wish to estimate. The goal of

a pollster is to estimate  $\mu$  from a subset of  $M$  polled voters, each of whom tells the pollster for which candidate he or she will vote. If the subset of voters is representative of the entire voting population, and if the choice made by each voter is independent of the choices made by other voters, the number of voters in the subset of  $M$  voters who say they will vote for candidate A is a discrete random variable  $m$  and its pmf is the binomial pmf defined in Eq. (8.31), namely

$$\begin{aligned} p[m] &= \binom{M}{m} \mu^m (1-\mu)^{M-m} \\ &= \frac{\mu^m (1-\mu)^{M-m} M!}{m! (M-m)!}, \quad m = 0, \dots, M. \end{aligned} \quad (9.34)$$

For large values of  $M$  (where  $M$  is the number of polled voters),  $p[m]$  can be approximated as a Gaussian pdf with a mean  $M\mu$  and a variance  $\sigma_m^2 = M\mu(1-\mu)$ . In shorthand notation,

$$m \sim \mathcal{N}(M\mu, \sigma_m^2). \quad (9.35)$$

where  $\mathcal{N}$  is shorthand for **normal distribution**.

Here, random variable  $m$  represents the number of voters who indicated they plan to vote for candidate A, out of a total of  $M$  polled voters. Instead of  $m$ , the pollster can formulate the estimation problem in terms of the normalized random variable  $z$ , where

$$z = \frac{m}{M}. \quad (9.36)$$

Furthermore,  $z$  can be expressed as

$$z = \mu + v, \quad (9.37)$$

with  $v$  as a zero-mean Gaussian random variable:

$$v \sim \mathcal{N}(0, \sigma_z^2), \quad (9.38)$$

and  $\sigma_z^2$  is related to  $\sigma_m^2$  by

$$\sigma_z^2 = \frac{\sigma_m^2}{M^2} = \frac{M\mu(1-\mu)}{M^2} = \frac{\mu(1-\mu)}{M}. \quad (9.39)$$

If the polling indicates that the vote is likely to be close (between the two candidates), then the pollster might set  $\mu \approx 1/2$ , in which case  $\sigma_z^2 \approx 1/(4M)$ . Additionally, the value of the unknown constant  $\mu$  is estimated as

$$\hat{\mu} = z_{\text{obs}} = \frac{m}{M}, \quad (9.40)$$

where  $z_{\text{obs}}$  is the **observed value** of  $z$ , namely the number of

voters who indicated they plan to vote for candidate A and  $M$  is the total number of polled voters.

For a single poll, the estimation process is straightforward, but what if we have  $N$  different polls, each comprised of  $M$  voters, resulting in  $N$  estimates of  $\mu$ ? Instead of only one random variable  $z$  with only one estimate  $\hat{\mu}$ , we now have  $N$  random variables,  $z[1]$  through  $z[N]$ , and  $N$  estimates of  $\mu$ , namely

$$\hat{\mu}[i] = z_{\text{obs}}[i] = \frac{m[i]}{M}, \quad 0 \leq i \leq N. \quad (9.41)$$

How do we combine the information provided by the  $N$  polls to generate a single estimate of  $\mu$ ? As we see next, the answer to the question depends on which estimation method we use.

## 9-3.2 MLE Estimate of Sample Mean

The  $N$  polling processes can be represented by  $N$  random variables  $z[i]$  given by

$$z[i] = \mu + v[i], \quad 0 \leq i \leq N, \quad (9.42)$$

where  $\mu$  is the true mean we wish to estimate and  $v[i]$  is the zero-mean Gaussian random variable associated with polling process  $z[i]$ . Our first step involves combining the  $N$  random variables into an  $N$ -dimensional random vector  $\mathbf{z}$ :

$$\mathbf{z} = [z[1], z[2], \dots, z[N]]^T. \quad (9.43)$$

Since the various polls are independent processes, we assume that the  $\{z[i]\}$  are independent random variables. Furthermore, we assume that  $M$ , the number of polled voters, is sufficiently large to justify describing the random variables  $\{z[i]\}$  by Gaussian pdfs with the same unknown mean  $\mu$  and variance  $\sigma_z^2$ . The joint conditional pdf  $p(\mathbf{z}|\mu)$  of vector  $\mathbf{z}$  is the product of the marginal pdfs of all  $N$  members of  $\mathbf{z}$ :

$$\begin{aligned} p(\mathbf{z}|\mu) &= \prod_{i=1}^N p(z[i]) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi\sigma_z^2}} e^{-(z[i]-\mu)^2/(2\sigma_z^2)} \\ &= \frac{1}{(2\pi\sigma_z^2)^{N/2}} \prod_{i=1}^N e^{-(z[i]-\mu)^2/(2\sigma_z^2)}. \end{aligned} \quad (9.44)$$

Setting  $z[i] = z_{\text{obs}}[i] =$  the observed result of the  $i$ th poll and  $\mu = \hat{\mu} =$  the estimate of  $\mu$  to be computed in Eq. (9.44) gives the likelihood function. Taking its natural logarithm gives the

**log-likelihood function:**

$$\ln(p(\mathbf{z}_{\text{obs}}|\hat{\mu})) = -\frac{N}{2} \ln(2\pi\sigma_z^2) - \frac{1}{2\sigma_z^2} \sum_{i=1}^N (z_{\text{obs}}[i] - \hat{\mu})^2. \quad (9.45)$$

The log-likelihood function is maximized by setting its partial derivative with respect to  $\hat{\mu}$  to zero:

$$0 = \frac{\partial}{\partial \hat{\mu}} \ln(p(\mathbf{z}_{\text{obs}}|\hat{\mu})) = \frac{1}{2\sigma_z^2} \sum_{i=1}^N (z_{\text{obs}}[i] - \hat{\mu}). \quad (9.46)$$

Solving for  $\hat{\mu}$  gives the MLE estimate of  $\mu$ :

$$\hat{\mu}_{\text{MLE}} = \frac{1}{N} \sum_{i=1}^N z_{\text{obs}}[i] = \frac{1}{N} \sum_{i=1}^N \frac{m[i]}{M} = \frac{1}{NM} \sum_{i=1}^N m[i], \quad (9.47)$$

which is equal to the total number of voters (among all  $N$  polls) who selected candidate  $A$ , divided by the total number of polled voters,  $NM$ . The result given by Eq. (9.47) is called the **sample mean** of  $\{z_{\text{obs}}[i]\}$ . As with the MLE estimator in Eq. (9.25) for the coin-flip problem, this is the “obvious” estimator of the mean  $\mu$ .

Note that Eq. (9.42) can be interpreted as a set of  $N$  observations of a white Gaussian random process with unknown mean  $\mu$ . So the sample mean can be used to estimate the unknown mean of a white Gaussian random process.

### 9-3.3 MAP Estimate of Sample Mean

We again have  $N$  polls, represented by  $\{z[i]\}$ , but we also have some additional information generated in earlier polls of candidate  $A$  versus candidate  $B$ . The data generated in earlier polls show that the fraction  $\mu$  of voters who preferred candidate  $A$  varied from one poll to another, and that collectively  $\mu$  behaves like a Gaussian random variable with a mean  $\mu_p$  and a variance  $\sigma_p^2$ :

$$\mu \sim \mathcal{N}(\mu_p, \sigma_p^2). \quad (9.48)$$

Since the information about the statistics of  $\mu$  is available ahead of the  $N$  polls, we refer to the pdf of  $\mu$  as an *a priori* pdf:

$$p(\mu) = \frac{1}{\sqrt{2\pi\sigma_p^2}} e^{-(\mu-\mu_p)^2/(2\sigma_p^2)}. \quad (9.49)$$

Our plan in the present subsection is to apply the MAP method outlined in Section 9-1.2, wherein the goal is to maximize the *a posteriori* pdf  $p(\text{unknown mean } \mu \mid \text{observation vector } \mathbf{z}_{\text{obs}}) = p(\mu|\mathbf{z}_{\text{obs}})$ . Using the form of Eq. (9.5),  $p(\mu|\mathbf{z}_{\text{obs}})$  can

be expressed as

$$p(\mu|\mathbf{z}_{\text{obs}}) = p(\mathbf{z}_{\text{obs}}|\mu) \frac{p(\mu)}{p(\mathbf{z}_{\text{obs}})} = p(\mathbf{z}_{\text{obs}}|\mu) \frac{p(\mu)}{C}, \quad (9.50)$$

where we set  $p(\mathbf{z}_{\text{obs}}) = C$  because it is not a function of the unknown mean  $\mu$ .

Inserting Eqs. (9.44), with  $\mathbf{z} = \mathbf{z}_{\text{obs}}$ , and (9.49) into Eq. (9.50) leads to

$$p(\mu|\mathbf{z}_{\text{obs}}) = \frac{1}{C(2\pi\sigma_z^2)^{N/2}} \times \frac{1}{\sqrt{2\pi\sigma_p^2}} e^{-(\mu-\mu_p)^2/(2\sigma_p^2)} \\ \times \prod_{i=1}^N e^{-(z_{\text{obs}}[i]-\mu)^2/(2\sigma_z^2)}. \quad (9.51)$$

The log-likelihood function is

$$\ln(p(\mu|\mathbf{z}_{\text{obs}})) = -\ln C - \frac{N}{2} \ln(2\pi\sigma_z^2) - \frac{1}{2} \ln(2\pi\sigma_p^2) \\ - \frac{1}{2\sigma_p^2} (\mu - \mu_p)^2 - \frac{1}{2\sigma_z^2} \sum_{i=1}^N (z_{\text{obs}}[i] - \mu)^2. \quad (9.52)$$

The *a posteriori* estimate  $\hat{\mu}_{\text{MAP}}$  is the value of  $\mu$  that maximizes the log-likelihood function, which is obtained by setting its partial derivative with respect to  $\mu$  to zero:

$$0 = \frac{\partial}{\partial \mu} \left[ -\frac{1}{2\sigma_p^2} (\mu - \mu_p)^2 \right] + \frac{\partial}{\partial \mu} \left[ -\frac{1}{2\sigma_z^2} \sum_{i=1}^N (z_{\text{obs}}[i] - \mu)^2 \right].$$

The solution leads to

$$\hat{\mu}_{\text{MAP}} = \mu = \frac{\left(\frac{\sigma_z^2}{\sigma_p^2}\right) \mu_p + \sum_{i=1}^N z_{\text{obs}}[i]}{N + \frac{\sigma_z^2}{\sigma_p^2}}. \quad (9.53)$$

The MAP estimate is the sample mean of the following:  $\{z[i]\}$ , augmented with  $(\sigma_z^2/\sigma_p^2)$  copies of  $\mu_p$ , for a total of  $(N + \sigma_z^2/\sigma_p^2)$  “observations.” For the polling problem, if  $M$  is the number of voters polled in each of the  $N$  polls and  $M'$  is the number of voters polled in each of the earlier polls, then  $(\sigma_z^2/\sigma_p^2) = M'/M$ , so if the previous polls polled more voters, its estimate  $\mu_p$  of  $\mu$  is weighed more heavily.

### Example 9-2: Polling Example

Five different polls of 1000 voters each were taken. The fractions of voters supporting candidate A were:

$$z_{\text{obs}} = \{0.51, 0.55, 0.53, 0.54, 0.52\}.$$

Compute (a) the MLE estimate  $\hat{\mu}_{\text{MLE}}$ , and (b) the MAP estimate  $\hat{\mu}_{\text{MAP}}$ , given that 2000 voters had been polled in an earlier poll in which 60% of voters selected candidate A.

**Solution: (a)** The MLE estimate is simply the sample mean:

$$\begin{aligned}\hat{\mu}_{\text{MLE}} &= \frac{1}{5} \sum_{i=1}^5 z_{\text{obs}}[i] \\ &= \frac{1}{5}(0.51 + 0.55 + 0.53 + 0.54 + 0.52) = 0.53.\end{aligned}$$

**(b)** The MAP estimate is given by Eq. (9.53):

$$\hat{\mu}_{\text{MAP}} = \frac{\left(\frac{\sigma_z^2}{\sigma_v^2}\right)\mu_p + \sum_{i=1}^5 z_{\text{obs}}[i]}{N + \frac{\sigma_z^2}{\sigma_v^2}}.$$

From the available information,  $N = 5$ ,  $\mu_p = 0.6$  (earlier poll),  $M = 1000$  votes, and  $M' = 2000$  voters (earlier *a priori* information). Also,  $\sigma_z^2/\sigma_v^2 = M'/M = 2$ . Using these values leads to

$$\hat{\mu}_{\text{MAP}} = \frac{2 \times 0.6 + \sum_{i=1}^5 z_{\text{obs}}[i]}{5 + 2} = 0.55.$$

The *a priori* information biased the estimate by raising it from  $\hat{\mu}_{\text{MLE}} = 0.53$  to  $\hat{\mu}_{\text{MAP}} = 0.55$ .

**Concept Question 9-3:** Another term for the sample mean of a bunch of numbers is their what?

**Exercise 9-8:** Compute the mean and variance of the sample mean of  $N$  iid random variables  $\{x_i\}$ , each of which has mean  $\bar{x}$  and variance  $\sigma_x^2$ .

**Answer:** From Chapter 8, the variance of the sum of uncorrelated random variables is the sum of their variances, independent random variables are uncorrelated, and  $\sigma_{ax}^2 = a^2\sigma_x^2$ . Let  $\mu = \frac{1}{N} \sum_{i=1}^N x_i$  be the sample mean of the  $\{x_i\}$ . Then

$$\sigma_\mu^2 = \frac{1}{N^2} N\sigma_x^2 = \frac{\sigma_x^2}{N}.$$

Also,  $E[\mu] = \frac{1}{N} \sum_{i=1}^N E[x_i] = \bar{x}$ , so the sample mean is an unbiased estimator of  $\bar{x}$ . So the sample mean converges to the actual mean  $\bar{x}$  of the  $\{x_i\}$  as  $N \rightarrow \infty$ . This is why the sample mean is useful, and why polling works.

## 9-4 Least-Squares Estimation

### 9-4.1 Gaussian Random Vectors

Generalizing Eq. (9.18) for two jointly Gaussian random vectors  $\mathbf{x}$  and  $\mathbf{y}$  (instead of  $\mathbf{x}$  and  $\mathbf{y}_{\text{obs}}$ ), the least-squares estimate  $\hat{\mathbf{x}}_{\text{LS}}$  of vector  $\mathbf{x}$ , given vector  $\mathbf{y}$ , is given by the conditional mean

$$\hat{\mathbf{x}}_{\text{LS}} = E[\mathbf{x}|\mathbf{y}] = \bar{\mathbf{x}} + \mathbf{K}_{\mathbf{x},\mathbf{y}}\mathbf{K}_{\mathbf{y}}^{-1}(\mathbf{y} - \bar{\mathbf{y}}), \quad (9.54)$$

where  $\bar{\mathbf{x}}$  and  $\bar{\mathbf{y}}$  are the means of vectors  $\bar{\mathbf{x}}$  and  $\bar{\mathbf{y}}$ , respectively,  $\mathbf{K}_{\mathbf{x},\mathbf{y}}$  is the cross-covariance matrix defined by Eq. (8.75), and  $\mathbf{K}_{\mathbf{y}}$  is the covariance matrix given by the form of Eq. (8.74). If  $\mathbf{x}$  and  $\mathbf{y}$  are zero-mean, Eq. (9.54) simplifies to

$$\begin{aligned}\hat{\mathbf{x}}_{\text{LS}} &= \mathbf{K}_{\mathbf{x},\mathbf{y}}\mathbf{K}_{\mathbf{y}}^{-1}\mathbf{y} = \mathbf{R}_{\mathbf{x},\mathbf{y}}\mathbf{R}_{\mathbf{y}}^{-1}\mathbf{y}, \\ &\text{(zero-mean jointly Gaussian)}\end{aligned} \quad (9.55)$$

where—because  $\mathbf{x}$  and  $\mathbf{y}$  are zero-mean— $\mathbf{K}_{\mathbf{x},\mathbf{y}}$  was replaced with the cross-correlation function  $\mathbf{R}_{\mathbf{x},\mathbf{y}}$  and  $\mathbf{K}_{\mathbf{y}}$  was replaced with the auto-correlation function  $\mathbf{R}_{\mathbf{y}}$ .

Vector  $\mathbf{x}$  represents the unknown quantity, vector  $\hat{\mathbf{x}}_{\text{LS}}$  represents the LS estimate of  $\mathbf{x}$ , and the difference between them,  $\mathbf{e} = (\mathbf{x} - \hat{\mathbf{x}}_{\text{LS}})$ , represents the *estimation error*. Also, vector  $\mathbf{y}$  represents the observation. The expectation of the product of the estimation error and the observation is

$$\begin{aligned}E[(\mathbf{x} - \hat{\mathbf{x}}_{\text{LS}})\mathbf{y}^T] &= E[\mathbf{x}\mathbf{y}^T] - E[\hat{\mathbf{x}}_{\text{LS}}\mathbf{y}^T] \\ &= \mathbf{R}_{\mathbf{x},\mathbf{y}} - \mathbf{R}_{\mathbf{x},\mathbf{y}}\mathbf{R}_{\mathbf{y}}^{-1}E[\mathbf{y}\mathbf{y}^T] \\ &= \mathbf{R}_{\mathbf{x},\mathbf{y}} - \mathbf{R}_{\mathbf{x},\mathbf{y}}\mathbf{R}_{\mathbf{y}}^{-1}\mathbf{R}_{\mathbf{y}} \\ &= \mathbf{R}_{\mathbf{x},\mathbf{y}} - \mathbf{R}_{\mathbf{x},\mathbf{y}} = 0.\end{aligned} \quad (9.56)$$

► The result given by Eq. (9.56) is a statement of **orthogonality**: The estimation error  $\mathbf{e} = (\mathbf{x} - \hat{\mathbf{x}}_{\text{LS}})$  is uncorrelated with the observation  $\mathbf{y}$  used to produce the LS estimate  $\hat{\mathbf{x}}_{\text{LS}}$ . ◀

► The result given by Eq. (9.59) is another statement of orthogonality: When the expectation of the product of two random variables—in this case, the error  $e[n]$  and the observation  $y[n-j]$ —is zero, it means that those two random variables are uncorrelated. We conclude from this observation that if  $x[n]$  and  $y[n]$  are jointly Gaussian WSS random processes, then  $\hat{x}_{\text{LLSE}}[n] = \hat{x}_{\text{LS}}[n]$ . ◀

## 9-4.2 Linear Least-Squares Estimation

We now consider a scenario in which the estimator is constrained to be a *linear* function of the observation. As we shall see shortly, the derived **linear least-squares estimate**  $\hat{x}_{\text{LLSE}}$  is equal to the LS estimate for Gaussian random vectors of the preceding subsection.

Given that  $x[i]$  and  $y[i]$  are zero-mean, jointly wide-sense-stationary (**WSS**, defined in Section 8-9.3) random processes, our present goal is to compute the linear least-squares estimate  $\hat{x}_{\text{LLSE}}[n]$  of  $x[n]$  at discrete time  $n$  from the infinite set of observations  $\{y[i], -\infty < i < \infty\}$ . The estimator is constrained to be a linear function of the  $\{y[i]\}$ , which means that  $\hat{x}_{\text{LLSE}}[n]$  and  $\{y[i]\}$  are related by a linear sum of the form

$$\hat{x}_{\text{LLSE}}[n] = \sum_{i=-\infty}^{\infty} h[i] y[n-i] = h[n] * y[n]. \quad (9.57)$$

Here,  $h[n]$  is an unknown function (filter) that is yet to be determined.

Let us consider the error  $e[n]$  between  $x[n]$  and the estimate  $\hat{x}_{\text{LLSE}}$ :

$$e[n] = x[n] - \hat{x}_{\text{LLSE}}[n] = x[n] - \sum_{i=-\infty}^{\infty} h[i] y[n-i]. \quad (9.58)$$

Next, let us square the error  $e[n]$ , take the derivative of its expectation with respect to  $h[j]$  and then set it equal to zero:

$$\begin{aligned} 0 &= \frac{\partial}{\partial h[j]} E[e[n]^2] \\ &= 2E \left[ e[n] \frac{\partial e[n]}{\partial h[j]} \right] \\ &= 2E \left[ e[n] \frac{\partial}{\partial h[j]} \left( x[n] - \sum_{j=-\infty}^{\infty} h[j] y[n-j] \right) \right] \\ &= 2E[e[n] (-y[n-j])] \\ &= -2E[e[n] y[n-j]]. \end{aligned} \quad (9.59)$$

## 9-4.3 1-D Stochastic Wiener Smoothing Filter

We now use the orthogonality relationship to derive 1-D stochastic versions of the deterministic Wiener deconvolution filter presented in Chapter 6 and the deterministic sparsifying denoising filter presented in Chapter 7. The stochastic versions of these filters assume that the signals are random processes instead of just functions. This allows *a priori* information about the signals, in the form of their power spectral densities, to be incorporated into the filters. If all of the random processes are white, the stochastic filters reduce to the deterministic filters of Chapters 6 and 7, as we show later.

Another advantage of the stochastic forms of these filters is that the trade-off parameter  $\lambda$  in the Tikhonov and LASSO criteria can now be interpreted as an inverse signal-to-noise ratio, as we also show later. We derive the 1-D forms of stochastic filters so we may generalize them to 2-D later.

Our task in the present subsection is to determine the filter  $h[i]$  introduced in Eq. (9.57). To that end, we insert Eq. (9.58) into Eq. (9.59) and apply the distributive property of the expectation:

$$\begin{aligned} 0 &= E[e[n] y[n-j]] \\ &= E \left[ \left( x[n] - \sum_{i=-\infty}^{\infty} h[i] y[n-i] \right) y[n-j] \right] \\ &= E[x[n] y[n-j]] - \sum_{i=-\infty}^{\infty} h[i] E[y[n-i] y[n-j]] \\ &= R_{xy}[j] - \sum_{i=-\infty}^{\infty} h[i] R_y[j-i] \\ &= R_{xy}[j] - h[j] * R_y[j], \end{aligned} \quad (9.60)$$

where, in the last step, we used the standard definition of convolution of two discrete-time 1-D signals, Eq. (2.71a).

Taking the DTFT of Eq. (9.60) gives

$$0 = \mathbf{S}_{xy}(\Omega) - \mathbf{H}(\Omega) \mathbf{S}_y(\Omega).$$

The solution for  $\mathbf{H}(\Omega)$  is labeled  $\mathbf{H}_{\text{SDN}}(\Omega)$ :

$$\mathbf{H}_{\text{SDN}}(\Omega) = \frac{\mathbf{S}_{xy}(\Omega)}{\mathbf{S}_y(\Omega)}. \quad (9.61)$$

The subscript SDN stands for ***stochastic denoising*** Wiener filter.

The LS estimate of  $x[n]$  is obtained from observation  $y[n]$  by applying the recipe:

$$\hat{x}_{\text{LLSE}}[n] = \hat{x}_{\text{LS}}[n] = h_{\text{SDN}}[n] * y[n], \quad (9.62a)$$

with

$$h_{\text{SDN}}[n] = \text{DTFT}^{-1} \left\{ \frac{\mathbf{S}_{xy}(\Omega)}{\mathbf{S}_y(\Omega)} \right\}. \quad (9.62b)$$

Application examples follow in the next two subsections.

#### 9-4.4 Stochastic Wiener Denoising

Let us suppose that  $y[n]$  are noisy observations of  $x[n]$ :

$$y[n] = x[n] + v[n], \quad -\infty < n < \infty,$$

where  $v[n]$  is a zero-mean IID random noise process. Also,  $x[n]$  and  $v[n]$  are uncorrelated and jointly WSS Gaussian random processes. Our goal is to compute the linear least-squares estimate  $\hat{x}_{\text{LLSE}}[n]$  at discrete  $n$  from the infinite set of observations  $\{y[i], -\infty < i < \infty\}$ . In order to apply the recipe given by Eq. (9.62), we first need to obtain expressions for  $\mathbf{S}_{xy}(\Omega)$  and  $\mathbf{S}_y(\Omega)$ .

Since  $x[n]$  and  $v[n]$  are uncorrelated and WSS, it follows that the cross-correlation

$$\begin{aligned} R_{xy}[i, j] &= E[x[i](x[j] + v[j])] \\ &= E[x[i]x[j]] + E[x[i]v[j]] = R_x[i - j]. \end{aligned} \quad (9.63)$$

Similarly,

$$\begin{aligned} R_y[i, j] &= E[(x[i] + v[i])(x[j] + v[j])] \\ &= E[x[i]x[j]] + E[v[i]x[j]] + E[x[i]v[j]] + E[v[i]v[j]]. \end{aligned} \quad (9.64)$$

Since  $x[i]$  and  $v[j]$  are uncorrelated, the second and third terms are zero. Hence,

$$R_y[i, j] = R_x[i - j] + \sigma_v^2 \delta[i - j], \quad (9.65)$$

where  $\sigma_v^2$  is the variance of the noise  $v[n]$ . Equations (9.63) and (9.65) show that  $R_{xy}[i, j] = R_{xy}[i - j]$  and  $R_y[i, j] = R_y[i - j]$ .

Taking the DTFTs of Eqs. (9.63) and (9.65) gives

$$\mathbf{S}_{xy}(\Omega) = \mathbf{S}_x(\Omega), \quad (9.66a)$$

and

$$\mathbf{S}_y(\Omega) = \mathbf{S}_x(\Omega) + \sigma_v^2. \quad (9.66b)$$

The combination of the two results leads to

$$\mathbf{H}_{\text{SDN}}(\Omega) = \frac{\mathbf{S}_{xy}(\Omega)}{\mathbf{S}_y(\Omega)} = \frac{\mathbf{S}_x(\Omega)}{\mathbf{S}_x(\Omega) + \sigma_v^2}. \quad (9.67)$$

The expression given by Eq. (9.67) represents a ***stochastic denoising filter*** because at frequency components  $\Omega$  for which  $\mathbf{S}_x(\Omega) \gg \sigma_v^2$ ,  $\mathbf{H}_{\text{SDN}}(\Omega) \approx 1$  and  $\hat{x}_{\text{LS}}[n] \approx y[n]$ , but at frequency components for which  $\mathbf{S}_x(\Omega) \ll \sigma_v^2$ ,  $\mathbf{H}_{\text{SDN}}(\Omega) \approx 0$ , thereby filtering out the noise at frequencies where the noise drowns out the signal.

Assuming the power spectral density  $\mathbf{S}_x(\Omega)$  and the noise variance  $\sigma_v^2$  are known or can be estimated (as discussed later in Section 9-7), the denoising recipe consists of the following steps:

- (1) Equation (9.67) is used to compute  $\mathbf{H}_{\text{SDN}}(\Omega)$ .
- (2) The inverse DTFT is applied to obtain

$$h_{\text{SDN}}[n] = \text{DTFT}^{-1}\{\mathbf{H}_{\text{SDN}}(\Omega)\}.$$

- (3) The LS denoised estimate of random process  $x[n]$  at time  $n$  is

$$\hat{x}_{\text{LS}}[n] = h_{\text{SDN}}[n] * y_{\text{obs}}[n], \quad (9.68)$$

where  $y_{\text{obs}}[n]$  is the observation. Alternatively,  $y_{\text{obs}}[n]$  can be transformed to  $\mathbf{Y}_{\text{obs}}(\Omega)$ , then used to compute

$$\hat{x}_{\text{LS}}(\Omega) = \mathbf{H}_{\text{SDN}}(\Omega) \mathbf{Y}_{\text{obs}}(\Omega),$$

after which an inverse transformation of  $\hat{x}_{\text{LS}}(\Omega)$  yields  $\hat{x}_{\text{LS}}[n]$ .

#### 9-4.5 Stochastic Wiener Deconvolution Example

Now let  $y[n]$  be noisy observations of  $h_{\text{blur}}[n] * x[n]$ :

$$y[n] = h_{\text{blur}}[n] * x[n] + v[n], \quad -\infty < n < \infty, \quad (9.69)$$

where  $h_{\text{blur}}[n]$  is a known impulse response. The goal is to compute the linear least-square estimate  $\hat{x}_{\text{LLSE}}[n]$  at time  $n$  from the observations  $\{y[i], -\infty < i < \infty\}$ . The noise  $v[n]$  is a zero-mean IID random process uncorrelated and jointly WSS with  $x[n]$ , and  $E(v[n]^2) = \sigma_v^2 \delta[n]$ .

Replacing  $x[n]$  with  $h_{\text{blur}}[n] * x[n]$  in the derivation leading to Eq. (9.67) for the smoothing filter for noisy observations and

using Eq. (8.161)(a and b) gives

$$\mathbf{S}_y(\Omega) = |\mathbf{H}_{\text{blur}}(\Omega)|^2 \mathbf{S}_x(\Omega) + \sigma_v^2, \quad (9.70\text{a})$$

$$\mathbf{S}_{xy}(\Omega) = \mathbf{H}_{\text{blur}}^*(\Omega) \mathbf{S}_x(\Omega), \quad (9.70\text{b})$$

which leads to the **stochastic deconvolution (SDC)** Wiener filter:

$$\mathbf{W}_{\text{SDC}}(\Omega) = \frac{\mathbf{H}_{\text{blur}}^*(\Omega) \mathbf{S}_x(\Omega)}{|\mathbf{H}_{\text{blur}}(\Omega)|^2 \mathbf{S}_x(\Omega) + \sigma_v^2}. \quad (9.71\text{a})$$

For frequencies  $\Omega$  for which  $|\mathbf{H}_{\text{blur}}(\Omega)|^2 \mathbf{S}_x(\Omega) \gg \sigma_v^2$ ,

$$\mathbf{W}_{\text{SDC}}(\Omega) \approx \frac{\mathbf{H}_{\text{blur}}^*(\Omega) \mathbf{S}_x(\Omega)}{|\mathbf{H}_{\text{blur}}(\Omega)|^2 \mathbf{S}_x(\Omega)} = \frac{1}{\mathbf{H}_{\text{blur}}(\Omega)}, \quad (9.71\text{b})$$

which makes perfect sense for a noiseless deconvolution filter.

The Wiener deconvolution filter given by Eq. (9.71a) can be rewritten as

$$\begin{aligned} \mathbf{W}_{\text{SDC}}(\Omega) &= \frac{\mathbf{H}_{\text{blur}}^*(\Omega) \mathbf{S}_x(\Omega)}{|\mathbf{H}_{\text{blur}}(\Omega)|^2 \mathbf{S}_x(\Omega) + \sigma_v^2} \\ &= \frac{|\mathbf{H}_{\text{blur}}(\Omega)|^2 \mathbf{S}_x(\Omega)}{|\mathbf{H}_{\text{blur}}(\Omega)|^2 \mathbf{S}_x(\Omega) + \sigma_v^2} \times \frac{1}{\mathbf{H}_{\text{blur}}(\Omega)}, \end{aligned} \quad (9.72)$$

which has the nice interpretation of a Wiener denoising filter followed by a noiseless deconvolution filter.

Assuming  $\mathbf{H}_{\text{blur}}(\Omega)$ ,  $\mathbf{S}_x(\Omega)$ , and  $\sigma_v^2$  are known or can be estimated through other means, a reconstructed **stochastic Wiener** estimate can be computed

$$\hat{x}_{\text{SDC}}[n] = \text{DTFT}^{-1}[\mathbf{W}_{\text{SDC}}(\Omega) \mathbf{Y}_{\text{obs}}(\Omega)], \quad (9.73)$$

where  $\mathbf{Y}_{\text{obs}}(\Omega)$  is the DTFT of the noisy, convolved observation  $y_{\text{obs}}[n]$ .

#### 9-4-6 Stochastic MAP Sparsifying Denoising Estimator

We return to the denoising problem:

$$y[n] = x[n] + v[n], \quad 1 \leq n \leq N, \quad (9.74)$$

and we make the following assumptions:

- (1)  $v[n]$  is a zero-mean white Gaussian random process with  $R_v[n] = \sigma_v^2 \delta[n]$ .
- (2)  $x[n]$  and  $v[n]$  are IID and jointly WSS random processes.

(3)  $x[n]$  at each time  $n$  has a Laplacian *a priori* pdf given by

$$p(x[n]) = \frac{1}{\sqrt{2} \sigma_x} e^{-\sqrt{2}|x[n]|/\sigma_x}. \quad (9.75)$$

We will now show that these conditions lead to a sparse (mostly zero-valued) estimate of  $x[n]$ .

To best use the combined information, we form vectors  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{v}$ :

$$\mathbf{x} = [x[1], x[2], \dots, x[N]]^T, \quad (9.76\text{a})$$

$$\mathbf{y} = [y[1], y[2], \dots, y[N]]^T, \quad (9.76\text{b})$$

$$\mathbf{v} = [v[1], v[2], \dots, v[N]]^T. \quad (9.76\text{c})$$

Using Eq. (9.75), the *a priori* pdf of unknown vector  $\mathbf{x}$  is

$$p(\mathbf{x}) = \prod_{n=1}^N p(x[n]) = \prod_{n=1}^N \frac{1}{\sqrt{2} \sigma_x} e^{-\sqrt{2}|x[n]|/\sigma_x}. \quad (9.77)$$

For  $\mathbf{y} = \mathbf{y}_{\text{obs}}$ , where  $\mathbf{y}_{\text{obs}}$  is the observation vector, the conditional pdf  $p(\mathbf{y}_{\text{obs}}|\mathbf{x})$  is the same as the jointly Gaussian pdf of the noise vector  $\mathbf{v}$  with  $\mathbf{v}[n] = y_{\text{obs}}[n] - x[n]$ :

$$\begin{aligned} p(\mathbf{y}_{\text{obs}}|\mathbf{x}) &= p(\mathbf{v}) = \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{-v[n]^2/(2\sigma_v^2)} \\ &= \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{-(y_{\text{obs}}[n]-x[n])^2/(2\sigma_v^2)}. \end{aligned} \quad (9.78)$$

The MAP estimate  $\hat{x}_{\text{MAP}}[n]$  at time  $n$  is related to the *a posteriori* pdf  $p(\mathbf{x}|\mathbf{y}_{\text{obs}})$ , which is related to  $p(\mathbf{y}_{\text{obs}}|\mathbf{x})$  and  $p(\mathbf{x})$  by the vector equivalent of Eq. (9.5):

$$p(\mathbf{x}|\mathbf{y}_{\text{obs}}) = p(\mathbf{y}_{\text{obs}}|\mathbf{x}) \frac{p(\mathbf{x})}{C}. \quad (9.79)$$

Inserting Eqs. (9.77) and (9.78) into Eq. (9.79) and then taking the natural log leads to the MAP log-likelihood function:

$$\begin{aligned} \ln(p(\mathbf{x}|\mathbf{y}_{\text{obs}})) &= -\ln C - \frac{N}{2} \ln(2\pi\sigma_v^2) \\ &\quad - \frac{1}{2\sigma_v^2} \sum_{n=1}^N (y_{\text{obs}}[n] - x[n])^2 \\ &\quad - N \ln(\sqrt{2} \sigma_x) - \frac{\sqrt{2}}{\sigma_x} \sum_{n=1}^N |x[n]|, \end{aligned} \quad (9.80)$$

The usual procedure for obtaining the MAP estimate  $\hat{x}_{\text{MAP}}[n]$  involves taking the partial derivative of the log-likelihood function with respect to  $x[n]$ , equating the result to zero, and then solving for  $x[n]$ . In the present case, the procedure is not so straightforward because one of the terms includes the absolute value of  $x[n]$ . Upon ignoring the three terms in Eq. (9.80) that do not involve  $x[n]$  (because they exercise no impact on minimizing the log-likelihood function), and then multiplying the two remaining terms by  $(-\sigma_v^2)$ , we obtain a new cost functional

$$\Lambda = \frac{1}{2} \sum_{n=1}^N (y_{\text{obs}}[n] - x[n])^2 + \sqrt{2} \frac{\sigma_v^2}{\sigma_x} \sum_{n=1}^N |x[n]|. \quad (9.81)$$

The functional form of  $\Lambda$  is identical to that of the LASSO cost functional given in Eq. (7.106), and so is the form of the solution given by Eq. (7.109):

$$\hat{x}_{\text{MAP}}[n] = \begin{cases} y_{\text{obs}}[n] - \lambda & \text{for } y_{\text{obs}}[n] > \lambda, \\ y_{\text{obs}}[n] + \lambda & \text{for } y_{\text{obs}}[n] < -\lambda, \\ 0 & \text{for } |y_{\text{obs}}[n]| < \lambda, \end{cases} \quad (9.82)$$

where  $\lambda$  is the noise-to-signal ratio

$$\lambda = \sqrt{2} \frac{\sigma_v^2}{\sigma_x}. \quad (9.83)$$

**Concept Question 9-4:** How did we use the orthogonality principle of linear prediction in Section 9-4?

**Concept Question 9-5:** How is the Tikhonov parameter  $\lambda$  interpreted in Section 9-4?

**Exercise 9-9:** What is the MAP sparsifying estimator when the noise variance  $\sigma_v^2$  is much smaller than  $\sigma_x^2$ ?

**Answer:** When  $\sigma_v^2 \ll \sigma_x^2$ ,  $\lambda \rightarrow 0$ , and  $\hat{x}[n] = y_{\text{obs}}[n]$ .

**Exercise 9-10:** What is the MAP sparsifying estimator when noise variance  $\sigma_v^2$  is much greater than  $\sigma_x^2$ ?

**Answer:** When  $\sigma_v^2 \gg \sigma_x^2$ ,  $\lambda \rightarrow \infty$ , and  $\hat{x}_{\text{MAP}} = 0$ . This makes sense: the *a priori* information that  $x$  is sparse dominates the noisy observation.

## 9-5 Deterministic versus Stochastic Wiener Filtering

Through a speedometer example, we now compare results of speed estimates based on two Wiener filtering approaches, one using a deterministic filter and another using a stochastic filter. The speedometer accepts as input  $y(t)$ , a noisy measurement of the distance traveled at time  $t$ , measured by an odometer, and converts  $y(t)$  into an output speed  $r(t) = dy/dt$ . [We use symbol  $r$  (for rate) instead of  $s$ , to denote speed to avoid notational confusion in later sections.] Sampling  $y(t)$  and  $r(t)$  at a sampling interval  $\Delta$  converts them into discrete-time signals  $y[n]$  and  $r[n]$ :

$$y[n] = y(t = n\Delta), \quad (9.84a)$$

$$r[n] = \frac{y[n] - y[n-1]}{\Delta}. \quad (9.84b)$$

The observed distance  $y[n]$  contains a noise component  $v[n]$ ,

$$y[n] = s[n] + v[n], \quad (9.85)$$

where  $s[n]$  is the actual distance that the odometer would have measured had it been noise-free, and  $v[n]$  is a zero-mean white noise Gaussian random process with variance  $\sigma_v^2$ . An example of a slightly noisy odometer signal  $y[n]$  with  $\sigma_v^2 = 2$  is shown in Fig. 9-3(a), and the application of the differentiator given by Eq. (9.84b) with  $\Delta = 1$  s results in the *unfiltered* speed  $r_u[n]$  shown in Fig. 9-3(b). The expression for  $r_u[n]$  is, from Eq. (9.84b),

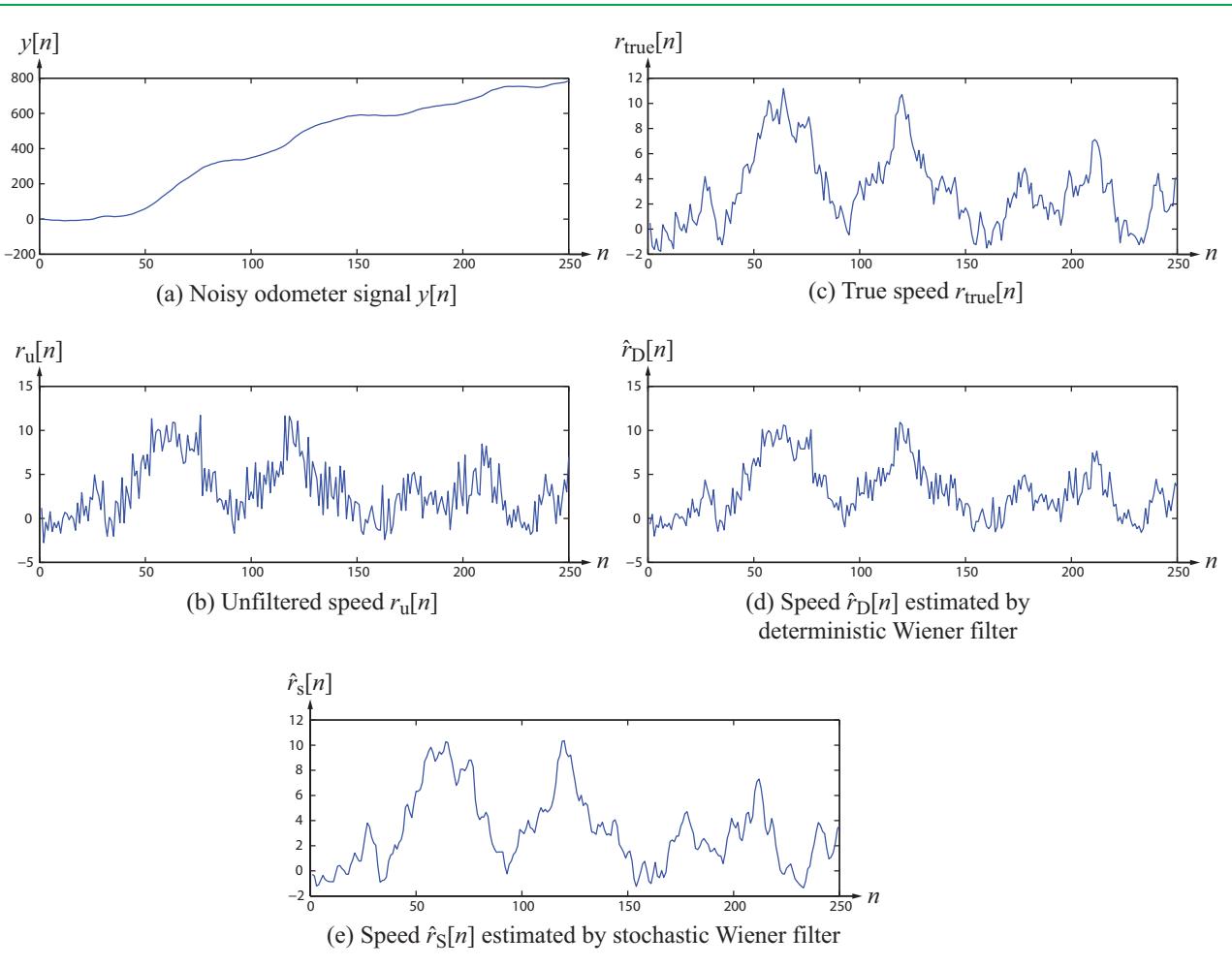
$$\begin{aligned} r_u[n] &= \frac{y[n] - y[n-1]}{\Delta} \\ &= \left( \frac{s[n] - s[n-1]}{\Delta} \right) + \left( \frac{v[n] - v[n-1]}{\Delta} \right) \\ &= r_{\text{true}}[n] + r_{\text{noise}}[n]. \end{aligned} \quad (9.86)$$

The goal of the Wiener filter is to estimate the true speed  $r_{\text{true}}[n]$ , so for the purpose of present example, we show in Fig. 9-2(c) a plot of the true speed  $r_{\text{true}}[n]$ , against which we will shortly compare the Wiener-filter results.

### 9-5.1 Deterministic Wiener Filter

For  $\Delta = 1$  s, the true speed  $r_{\text{true}}[n]$  is related to the true distance  $s[n]$  by

$$r_{\text{true}}[n] = s[n] - s[n-1]. \quad (9.87)$$



**Figure 9-3** Speedometer example: (a) noisy odometer signal  $y[n]$ , (b) unfiltered speed  $r_u[n] = (y[n] - y[n-1])/\Delta$ , with  $\Delta = 1$  s, (c) true speed  $r_{\text{true}}[n] = (s[n] - s[n-1])/\Delta$ , with  $\Delta = 1$  s, (d) deterministically estimated speed  $\hat{r}_D[n]$ , and (e) stochastically estimated speed  $\hat{r}_S[n]$ .

For a segment of length  $N$ , with  $\{n = 1, \dots, N\}$ , the  $N$ -point DFT of Eq. (9.87) gives

$$\mathbf{R}[k] = \mathbf{S}[k] - e^{-j2\pi k/N} \mathbf{S}[k] = (1 - e^{-j2\pi k/N}) \mathbf{S}[k], \quad (9.88)$$

where we used property #2 in **Table 2-9** to compute the second term of Eq. (9.88). The frequency response function  $\mathbf{H}[k]$  of the

noise-free system described by Eq. (9.87) is

$$\mathbf{H}[k] = \frac{\mathbf{S}[k]}{\mathbf{R}[k]} = \frac{1}{1 - e^{-j2\pi k/N}}. \quad (9.89)$$

The deterministic Wiener denoising/deconvolution filter was presented earlier in Section 6-4.4 for 2-D images. Converting the notation from 2-D to 1-D, as well as replacing the symbols in Eqs. (6.32a) and (6.32b) to match our current speedometer

problem, we obtain the following expression for the estimated DFT of the true speed  $r_{\text{true}}[n]$ :

$$\hat{\mathbf{R}}_D[k] = \mathbf{Y}[k] \mathbf{W}_{\text{DDC}}[k] \quad (9.90a)$$

with

$$\mathbf{W}_{\text{DDC}}[k] = \frac{\mathbf{H}^*[k]}{|\mathbf{H}[k]|^2 + \lambda^2}, \quad (9.90b)$$

where  $\mathbf{Y}[k]$  is the DFT of the noisy observation  $y[n]$ ,  $\mathbf{W}_{\text{DDC}}[k]$  is the **deterministic deconvolution** Wiener filter, and  $\lambda$  is the trade-off parameter in Tikhonov regularization. Combining Eqs. (9.89), (9.90a), and (9.90b), and multiplying the numerator and denominator by  $|1 - e^{j2\pi k/N}|^2$  leads to the deterministic estimate

$$\hat{\mathbf{R}}_D[k] = \frac{\mathbf{Y}[k] (1 - e^{-j2\pi k/N})}{1 + |1 - e^{-j2\pi k/N}|^2 \lambda^2}. \quad (9.91)$$

To obtain the “best” estimate of the speed  $r_D[n]$  using the deterministic Wiener filter, we need to go through the estimation process multiple time using different values of  $\lambda$ . The process entails the following steps:

- (1) Compute the DFT  $\mathbf{Y}[k]$  of the noisy observations  $y[n]$ .
- (2) Compute  $\hat{\mathbf{R}}_D[k]$  using Eq. (9.91) for various values of  $\lambda$ .
- (3) Perform an inverse  $N$ -point DFT to compute  $\hat{r}_D[n]$ .

For the speedometer example, the outcome with the “seemingly” best result is the one with  $\lambda = 0.1$ , and its plot is shown in **Fig. 9-3(d)**. It is an improvement over the unfiltered speed  $r_u[n]$ , but it still contains a noticeable noisy component.

## 9-5.2 Stochastic Wiener Filter

Based on the treatment given in Section 9-4.5, the stochastic Wiener approach to filtering the noisy signal  $y[n]$  uses a **stochastic denoising/deconvolution** Wiener filter  $\mathbf{W}_{\text{SDC}}(\Omega)$  as follows:

$$\hat{\mathbf{R}}_S(\Omega) = \mathbf{Y}(\Omega) \mathbf{W}_{\text{SDC}}(\Omega), \quad (9.92a)$$

with

$$\mathbf{W}_{\text{SDC}}(\Omega) = \frac{\mathbf{H}^*(\Omega) \mathbf{S}_s(\Omega)}{|\mathbf{H}(\Omega)|^2 \mathbf{S}_s(\Omega) + \sigma_v^2}, \quad (9.92b)$$

where  $\mathbf{H}(\Omega)$  is  $\mathbf{H}[k]$  of Eq. (9.89) with  $\Omega = 2\pi k/N$ ,  $\mathbf{S}_s(\Omega)$  is the power spectral density of  $s[n]$  and  $\sigma_v^2$  is the noise variance. Since the true distance  $s[n]$  is an unknown quantity, its power spectral density  $\mathbf{S}_s(\Omega)$  is unknown. In practice,  $\mathbf{S}_s(\Omega)$  is assigned a functional form based on experience with similar random processes.

As noted later in Section 9-7 and 9-8, a practical model for  $\mathbf{S}_s(\Omega)$  is

$$\mathbf{S}_s(\Omega) = \frac{C}{\Omega^2}, \quad (9.92c)$$

where  $C$  is a constant. Using Eqs. (9.89) and (9.92c) leads to the stochastic estimate

$$\hat{\mathbf{R}}_S(\Omega) = \frac{\mathbf{Y}(\Omega)(1 - e^{-j\Omega})}{1 + |1 - e^{-j\Omega}|^2 \Omega^2 \frac{\sigma_v^2}{C}}. \quad (9.93)$$

Implementation of the stochastic Wiener filter entails the following steps:

- (1) For the given observation distance signal  $y[n]$ , compute its DFT  $\mathbf{Y}[k]$ .
- (2) Convert Eq. (9.93) into a numerical format by replacing  $\Omega$  with  $2\pi k/N$  everywhere.
- (3) Using  $C' = \sigma_v^2/C$  as a parameter, compute  $\hat{\mathbf{R}}_S[k]$  for various values of  $C'$ .
- (4) Perform an inverse DFT to obtain  $\hat{r}_S[n]$  for each value of  $C'$ .
- (5) Select the value of  $C'$  that appears to provide the best result.

The result for the seemingly best value of  $C'$ , which turned out to be  $C' = 0.4$ , is displayed in **Fig. 9-3(e)**. Comparison of the plots for  $\hat{r}_D[n]$  and  $\hat{r}_S[n]$  reveals that the stochastic approach provides a much better rendition of the true speed  $r_{\text{true}}[n]$  than does the deterministic approach.

**Concept Question 9-6:** Why did the stochastic Wiener filter produce a much better estimate than the deterministic Wiener filter in Section 9-5?

**Exercise 9-11:** What happens to the stochastic Wiener filter when the observation noise strength  $\sigma_v^2 \rightarrow 0$ ?

**Answer:** From Eq. (9.93), letting  $\sigma_v^2 \rightarrow 0$  makes  $\hat{\mathbf{R}}_S(\Omega) = \mathbf{Y}(\Omega)(1 - e^{-j\Omega})$ , whose inverse DTFT is  $\hat{r}_S[n] = y[n] - y[n-1]$ , which is Eq. (9.84b).

## 9-6 2-D Estimation

We now extend the various methods that were introduced in earlier sections for estimating 1-D random processes to estimating 2-D random fields. The derivations of these estimators are direct generalizations of their 1-D counterparts.

### 9-6.1 2-D MLE Estimate of Sample Mean

Throughout Chapter 8, it was assumed that WSS random fields were zero-mean, so realized by subtracting the constant mean from the original random field. To perform the subtraction step, we need to either know the value of the mean  $\bar{f}$  or we need to estimate it from the ***observation*** of the random field  $f_{\text{obs}}[n, m]$ . Extending the result of the 1-D MLE estimate of the mean given by Eq. (9.47) to a 2-D random field of size  $N \times N$  gives

$$\bar{f}_{\text{MLE}} = \frac{1}{N^2} \sum_{n=1}^N \sum_{m=1}^N f_{\text{obs}}[n, m]. \quad (9.94)$$

- ▶ Estimating the mean  $\bar{f}$  and subtracting it from  $f_{\text{obs}}[n, m]$  to generate a zero-mean observation random field  $g_{\text{obs}}[n, m]$  is a required prerequisite step to the application of Wiener filters for image smoothing and deconvolution. ◀

### 9-6.2 2-D Wiener Filter

As in 1-D, the least-squares (LS) estimate for jointly Gaussian random fields—unknown  $x[n]$  and observation  $y[n]$ —is equal to the linear least-squares estimate (LLSE) for any pair of linearly related random fields, whether jointly Gaussian or not.

- ▶ Hence, for convenience and with no loss of generality, we shall assume that all random fields are jointly WSS Gaussian, for estimation purposes. ◀

Given an unknown random field  $f[n, m]$ , a zero-mean noise random field  $v[n, m]$ , and an observation random field  $g[n, m]$ , with

$$g[n, m] = f[n, m] + v[n, m], \quad 1 \leq n, m \leq N, \quad (9.95)$$

the 2-D version of the 1-D stochastic denoising ***Wiener filter*** given by Eq. (9.61) is

$$\mathbf{H}_{\text{SDN}}(\Omega_1, \Omega_2) = \frac{\mathbf{S}_{fg}(\Omega_1, \Omega_2)}{\mathbf{S}_g(\Omega_1, \Omega_2)}. \quad (9.96)$$

### 9-6.3 Stochastic 2-D Wiener Denoising Filter

Moreover, if  $v[n, m]$  is a zero-mean white Gaussian noise random field with variance  $\sigma_v^2$ , it follows that

$$\mathbf{S}_{fg}(\Omega_1, \Omega_2) = \mathbf{S}_f(\Omega_1, \Omega_2) \quad (9.97a)$$

and

$$\mathbf{S}_g(\Omega_1, \Omega_2) = \mathbf{S}_f(\Omega_1, \Omega_2) + \sigma_v^2, \quad (9.97b)$$

in which case the expression for  $\mathbf{H}_{\text{SDN}}(\Omega_1, \Omega_2)$  becomes

$$\mathbf{H}_{\text{SDN}}(\Omega_1, \Omega_2) = \frac{\mathbf{S}_f(\Omega_1, \Omega_2)}{\mathbf{S}_f(\Omega_1, \Omega_2) + \sigma_v^2}. \quad (9.98)$$

If the power spectral density  $\mathbf{S}_f(\Omega_1, \Omega_2)$  and the noise variance  $\sigma_v^2$  are known, a 2-D image can be denoised by applying the following recipe:

- (1) Equation (9.98) can be used to compute  $\mathbf{H}_{\text{SDN}}(\Omega_1, \Omega_2)$ .
- (2) The inverse DSFT is applied to obtain
- (3) Then, each element of  $f[n, m]$  is denoised by using the filtering operation

$$\hat{f}_s[n, m] = h_{\text{SDN}}[n, m] * * g_{\text{obs}}[n, m], \quad (9.100)$$

where  $g_{\text{obs}}[n, m]$  is the noisy observation of  $f[n, m]$ . As in 1-D,  $h_{\text{SDN}}[n, m]$  usually is windowed to a finite spatial extent.

### 9-6.4 Stochastic 2-D Wiener Deconvolution Filter

Let us assume we have a noisy 2-D image  $g[n, m]$ . In fact, in addition to the added noise  $v[n, m]$ , the true (unknown) image  $f[n, m]$  had been blurred by the measurement process. Thus,

$$g[n, m] = h_{\text{blur}}[n, m] * * f[n, m] + v[n, m], \quad (9.101)$$

where  $h_{\text{blur}}[n, m]$  is a blur filter with a known PSF, established through a calibration process.

Our goal is to compute the stochastic reconstruction  $\hat{f}_s[n, m]$  of  $f[n, m]$  at each location  $[n, m]$ , from noisy observations  $\{g_{\text{obs}}[i, j], -\infty \leq i, j \leq \infty\}$ . To that end, we introduce the 2-D version of Eqs. (9.70a and b):

$$\mathbf{S}_{fg}(\Omega_1, \Omega_2) = \mathbf{H}_{\text{blur}}^*(\Omega_1, \Omega_2) \mathbf{S}_f(\Omega_1, \Omega_2), \quad (9.102a)$$

$$\mathbf{S}_g(\Omega_1, \Omega_2) = |\mathbf{H}_{\text{blur}}(\Omega_1, \Omega_2)|^2 \mathbf{S}_f(\Omega_1, \Omega_2) + \sigma_v^2, \quad (9.102b)$$

and then we use them to obtain the ***stochastic deconvolution***

Wiener filter

$$\begin{aligned} \mathbf{W}_{\text{SDC}}(\Omega_1, \Omega_2) &= \frac{\mathbf{S}_{fg}(\Omega_1, \Omega_2)}{\mathbf{S}_g(\Omega_1, \Omega_2)} \\ &= \frac{\mathbf{H}_{\text{blur}}^*(\Omega_1, \Omega_2) \mathbf{S}_f(\Omega_1, \Omega_2)}{|\mathbf{H}_{\text{blur}}(\Omega_1, \Omega_2)|^2 \mathbf{S}_f(\Omega_1, \Omega_2) + \sigma_v^2}. \end{aligned} \quad (9.103)$$

As before, we assume that we know the power spectral density  $\mathbf{S}_f(\Omega_1, \Omega_2)$ , the noise variance  $\sigma_v^2$ , and the blur filter  $h_{\text{blur}}[n, m]$ , in which case we follow the recipe:

- (1) Obtain  $\mathbf{H}_{\text{blur}}(\Omega_1, \Omega_2)$  from

$$\mathbf{H}_{\text{blur}}(\Omega_1, \Omega_2) = \text{DSFT}\{h_{\text{blur}}[n, m]\}. \quad (9.104)$$

- (2) Compute  $\mathbf{W}_{\text{SDC}}(\Omega_1, \Omega_2)$  using Eq. (9.103).

- (3) Compute  $w_{\text{SDC}}[n, m]$  from

$$w_{\text{SDC}}[n, m] = \text{DSFT}^{-1}\{\mathbf{W}_{\text{SDC}}(\Omega_1, \Omega_2)\}. \quad (9.105)$$

- (4) Estimate  $\hat{f}_s[n, m]$  at each location  $[n, m]$  based on observations  $g_{\text{obs}}[n, m]$  by applying the convolution:

$$\hat{f}_s[n, m] = w_{\text{SDC}}[n, m] \ast g_{\text{obs}}[n, m]. \quad (9.106)$$

Does the stochastic Wiener deconvolution filter provide better results than the deterministic Wiener deconvolution filter described in Section 6-4.4? The answer is an emphatic yes, as demonstrated by the following example.

## 9-6.5 Deterministic versus Stochastic Deconvolution Example

This is a 2-D “replica” of the 1-D speedometer example of Section 9-5. The image in Fig. 9-4(a) is a high-resolution low-noise MRI image of a human head. We will treat it as an “original” image  $f[n, m]$ . To illustrate (a) the effects of blurring caused by the convolution of an original image with the PSF of the imaging system, and (b) the deblurring realized by Wiener deconvolution, we use a disk-shaped PSF with a uniform distribution given by

$$h_{\text{blur}}[n, m] = \begin{cases} 1 & \text{for } m^2 + n^2 < 145, \\ 0 & \text{for } m^2 + n^2 \geq 145, \end{cases} \quad (9.107)$$

which provides a good model of out-of-focus imaging. The circular PSF has a radius of 12 pixels. We also add noise in the

form of a zero-mean white Gaussian random field  $v[n, m]$ :

$$g[n, m] = h_{\text{blur}}[n, m] \ast f[n, m] + v[n, m]. \quad (9.108)$$

The outcome of the blurring and noise-addition processes is displayed in Fig. 9-4(b).

### A. Deterministic Wiener Deconvolution

By extending the 1-D speedometer recipe of Section 9-5.1 to the 2-D MRI image, we obtain the deterministic Wiener estimate  $\hat{f}_D[n, m]$  as follows:

- (1)  $\mathbf{H}_{\text{blur}}[k_1, k_2]$  is computed from  $h_{\text{blur}}[n, m]$  by taking the 2-D DFT of Eq. (9.107).

- (2) The deterministic deconvolution Wiener filter

$$\mathbf{W}_{\text{DDC}}[k_1, k_2] = \frac{\mathbf{H}_{\text{blur}}^*[k_1, k_2]}{|\mathbf{H}_{\text{blur}}[k_1, k_2]|^2 + \lambda^2} \quad (9.109)$$

is computed for several values of the parameter  $\lambda$ .

- (3)  $\hat{\mathbf{F}}_D[k_1, k_2]$  is estimated using

$$\hat{\mathbf{F}}_D[k_1, k_2] = \mathbf{G}[k_1, k_2] \mathbf{W}_{\text{DDC}}[k_1, k_2], \quad (9.110)$$

where  $\mathbf{G}[k_1, k_2]$  is the 2-D DFT of the blurred image  $g[n, m]$ .

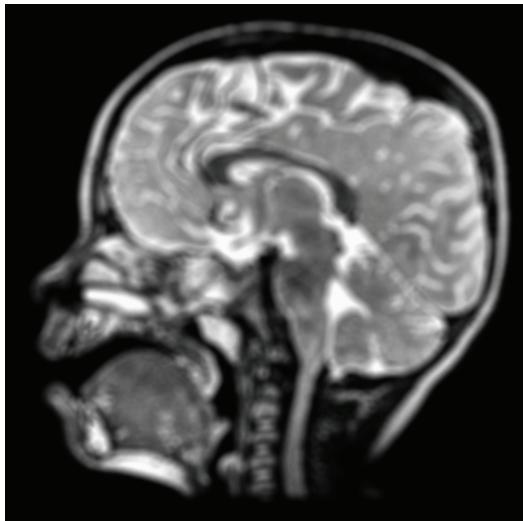
- (4) Application of the inverse DFT gives  $\hat{f}_D[n, m]$ . The “best” resultant image, shown in Fig. 9-4(c), used  $\lambda = 0.1$ .

### B. Stochastic Wiener Deconvolution

The 1-D stochastic Wiener solution to the speedometer problem is given in Section 9-5.2. The 2-D image deconvolution procedure using the stochastic Wiener approach is identical to the 1-D procedure outlined earlier in part A of Section 9-6.5, except for the form of the Wiener filter. For the stochastic case, the 2-D equivalent of the expression given by Eq. (9.92c) for the power spectral density is

$$\mathbf{S}_f(\Omega_1, \Omega_2) = \frac{C}{(\Omega_1^2 + \Omega_2^2)^2}. \quad (9.111)$$

Upon setting  $\Omega_1 = 2\pi k_1/N$  and  $\Omega_2 = 2\pi k_2/N$  in Eqs. (9.111) and (9.103), we obtain the stochastic Wiener deconvolution

(a) Original MRI image  $f[n,m]$ (c) Deconvolved image  $\hat{f}_D[n,m]$  using deterministic Wiener filter(b) Noisy blurred MRI image  $g[n,m]$ (d) Deconvolved image  $\hat{f}_s[n,m]$  using stochastic Wiener filter

**Figure 9-4** MRI images: (a) Original “true” image, (b) image blurred by imaging system, (c) image deconvolved by deterministic Wiener filter, and (d) image deconvolved by stochastic Wiener filter.

filter:

$$\mathbf{W}_{\text{SDC}}[k_1, k_2] = \frac{\mathbf{H}_{\text{blur}}^*[k_1, k_2]}{|\mathbf{H}_{\text{blur}}[k_1, k_2]|^2 + \left(\frac{2\pi}{N}\right)^4 \frac{\sigma_v^2}{C} (k_1^2 + k_2^2)^2}. \quad (9.112)$$

Repetition of the recipe given earlier for the deterministic filter but replacing  $\mathbf{W}_{\text{DDC}}[k_1, k_2]$  in Eq. (9.110) with  $\mathbf{W}_{\text{SDC}}[k_1, k_2]$  led to the image shown in [Fig. 9-4\(d\)](#). The process was optimized by repeating it for several different values of the parameter  $\sigma_v^2/C$ . The “best” result was for  $\sigma_v^2/C = 1$ .

Comparison of the images in [Fig. 9-4](#) leads to two conclusions:

(1) The image in part (c), generated by applying the deterministic Wiener filter, is far superior to the blurred image in part (b).

(2) Among the two filtering approaches, the stochastic approach (image in (d)) yields a sharper image than its deterministic counterpart, motivating the stochastic approach.

## 9-6.6 Stochastic 2-D MAP Sparsifying Estimator

The 2-D analogue of the 1-D sparsifying estimator of Section 9-4.6 can be summarized as follows:

(1)

$$g_{\text{obs}}[n, m] = f[n, m] + v[n, m], \quad (9.113)$$

with  $g_{\text{obs}}[n, m]$  representing the observation,  $f[n, m]$  representing the unknown random field, and  $v[n, m]$  representing a white Gaussian random field with known variance  $S_v(\Omega_1, \Omega_2) = \sigma_v^2$ .

(2)  $f[n, m]$  and  $v[n, m]$  are IID, jointly WSS random fields.

(3) Each  $f[n, m]$  has a 2-D Laplacian *a priori* pdf given by

$$p(f[n, m]) = \frac{1}{\sqrt{2} \sigma_f} e^{-\sqrt{2} |f[n, m]|/\sigma_f}. \quad (9.114)$$

The 2-D LASSO cost-functional equivalent to the 1-D expression given by Eq. (9.81) is

$$\Lambda = \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N (g_{\text{obs}}[n, m] - f[n, m])^2 + \sqrt{2} \frac{\sigma_v^2}{\sigma_f} \sum_{n=1}^N \sum_{m=1}^N |f[n, m]|, \quad (9.115)$$

and the MAP estimate is

$$\hat{f}_{\text{MAP}}[n, m] = \begin{cases} g_{\text{obs}}[n, m] - \lambda & \text{for } g_{\text{obs}}[n, m] > \lambda, \\ g_{\text{obs}}[n, m] + \lambda & \text{for } g_{\text{obs}}[n, m] < -\lambda, \\ 0 & \text{for } |g_{\text{obs}}[n, m]| < \lambda, \end{cases} \quad (9.116)$$

where  $\lambda$  is the noise-to-signal ratio

$$\lambda = \sqrt{2} \frac{\sigma_v^2}{\sigma_f}. \quad (9.117)$$

**Concept Question 9-7:** Why did the stochastic Wiener deconvolution filter produce a much better estimate than the deterministic Wiener deconvolution filter in Section 9-6?

**Exercise 9-12:** When  $\sigma_f \rightarrow \infty$ , the Laplacian pdf given by Eq. (9.114) approaches the uniform distribution. What does the MAP estimate reduce to when  $\sigma_f \rightarrow \infty$ ?

**Answer:** As  $\sigma_f \rightarrow \infty$ , the parameter  $\lambda \rightarrow 0$ , in which case  $\hat{f}_{\text{MAP}}[n, m] \rightarrow g_{\text{obs}}[n, m]$ .

## 9-7 Spectral Estimation

To apply the 1-D and 2-D stochastic denoising and deconvolution operations outlined in Sections 9-4 to 9-6, we need to know the power spectral densities  $S_x(\Omega)$  of the 1-D unknown random process  $x[n]$  and  $S_f(\Omega_1, \Omega_2)$  of the 2-D random field  $f[n, m]$ .

Had  $x[n]$  been available, we could have applied the 1-D  $N$ -order DFT to estimate  $S_x(\Omega)$  using

$$\hat{S}_x\left(\Omega = \frac{2\pi k}{N}\right) = \frac{1}{N} \left| \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \right|^2, \quad (9.118)$$

for  $k = 0, 1, \dots, N-1$ . The division by  $N$  converts energy spectral density to power spectral density. Similarly, application of the 2-D  $N$ th-order DFT to  $f[n, m]$  leads

$$\begin{aligned} \hat{S}_f\left(\Omega_1 = \frac{2\pi k_1}{N}, \Omega_2 = \frac{2\pi k_2}{N}\right) = \\ \frac{1}{N^2} \left| \sum_{n=0}^{N-1} f[n, m] e^{-j2\pi(nk_1 + mk_2)/N} \right|^2, \end{aligned} \quad (9.119)$$

for  $k_1 = 0, 1, \dots, N-1$ , and  $k_2 = 0, 1, \dots, N-1$ .

This estimation method is known as the **periodogram spectral**

**estimator.** Other spectral estimation methods exist as well, but the central problem is that  $x[n]$  and  $f[n, m]$  are the unknown quantities we wish to estimate, so we have no direct way to determine their power spectral densities. However, we can use parametric models to describe  $S_x(\Omega)$  and  $S_f(\Omega_1, \Omega_2)$ , which can then be used in the stochastic Wiener estimation recipes of Sections 9-4 to 9-6 to obtain the seemingly best outcome. As we shall see in the next section, many images and signals exhibit a **fractal-like** behavior with corresponding 1-D and 2-D power spectral densities of the form

$$S_x(\Omega) = \frac{C}{|\Omega|^a}, \quad \text{for } \Omega_{\min} < |\Omega| < \pi, \quad (9.120a)$$

$$S_f(\Omega_1, \Omega_2) = \frac{C}{(\Omega_1^2 + \Omega_2^2)^b}, \quad \text{for } \Omega_{\min} < |\Omega_1|, |\Omega_2| < \pi, \quad (9.120b)$$

where  $a$ ,  $b$ ,  $C$ , and  $\Omega_{\min}$  are adjustable constant parameters. The expressions given by Eq. (9.120) are known as power laws, an example of which with  $b = 2$  was used in Section 9-6.5 to deconvolve a blurred MRI image (Fig. 9-4).

**Exercise 9-13:** Suppose we estimate the autocorrelation function  $R_x[n]$  of data  $\{x[n], n = 0, \dots, N - 1\}$  using the sample mean over  $i$  of  $\{x[i] x[i - n]\}$ , zero-padding  $\{x[n]\}$  as needed. Show that the DFT of  $\hat{R}_x[n]$  is the periodogram (this is one reason why the periodogram works).

**Answer:**

$$\begin{aligned} \hat{R}_x[n] &= \frac{1}{N} \sum_{i=0}^{N-1} x[i] x[i - n] \\ &= \frac{1}{N} \sum_{i=0}^{N-1} x[i] x[-(n - i)] \\ &= \frac{1}{N} (x[n] * x[-n]). \end{aligned}$$

From entries #4 and #6 of Table 2-7, the DSFT of  $(x[n] * x[-n])$  is  $\mathbf{X}(\Omega) \mathbf{X}(-\Omega) = \mathbf{X}(\Omega) \mathbf{X}^*(\Omega) = |\mathbf{X}(\Omega)|^2$ , which is the periodogram defined in Eq. (9.118).

## 9-8 1-D Fractals

A fractal is a signal, image, or any object that exhibits **self-similarity** across different scales. A signal is self-similar if any segment of it resembles the overall signal *statistically*. The same

definition applies to an image in 2-D. Fractals are quite common in nature; examples include certain classes of trees, river deltas, and coastlines (Fig. 9-5). In a perfect fractal, self-similarity exists over an infinite number of scales, but for real objects the similarity is exhibited over a finite number of scales.

An example of a fractal signal is shown in Fig. 9-6; note the statistical resemblance between (a) the pattern of the entire signal extending over the range between 0 and 1 in part (a) of the figure and (b) the pattern in part (b) of only a narrow segment of the original, extending between 0.4 and 0.5 of the original scale.

### 9-8.1 Continuous-Time Fractals

Perfect self-similarity implies that a signal  $x(t)$  is identically equal to a scaled version of itself,  $x(at)$ , where  $a$  is a positive **scaling constant**, but **statistical self-similarity** implies that if  $x(t)$  is a zero-mean wide-sense stationary (WSS) random process, then its autocorrelation is self-similar. That is,

$$R_x(\tau) = C R_x(a\tau), \quad (9.121)$$

where  $\tau$  is the time shift between  $x(t)$  and  $x(t - \tau)$  in

$$R_x(\tau) = E[x(t) x(t - \tau)]. \quad (9.122)$$

According to Eq. (9.121), within a multiplicative constant  $C$ , the variation of the autocorrelation  $R_x(\tau)$  with the time shift  $\tau$  is the same as the variation of the autocorrelation of the time-scaled version  $R_x(a\tau)$  with the scaled time  $(a\tau)$ .

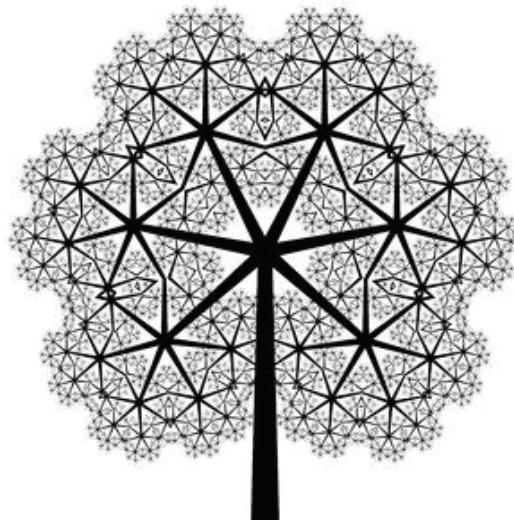
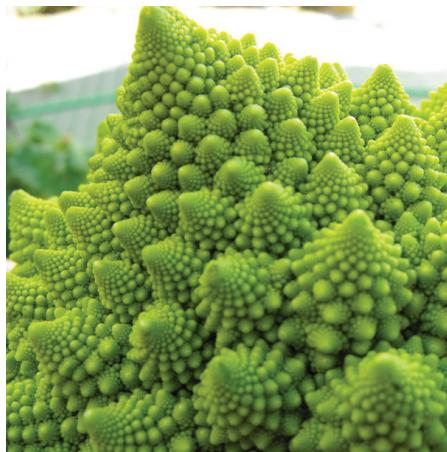
The self-similarity property extends to the power spectral density. According to Eq. (8.139a), the power spectral density  $S_x(f)$  of  $x(t)$  is related to its autocorrelation function  $R_x(\tau)$  by

$$S_x(f) = \int_{-\infty}^{\infty} R_x(\tau) e^{-j2\pi f \tau} d\tau. \quad (9.123)$$

Inserting Eq. (9.121) into Eq. (9.123) and then replacing  $\tau$  with  $\tau'/a$  leads to

$$\begin{aligned} S_x(f) &= C \int_{-\infty}^{\infty} R_x(a\tau) e^{-j2\pi f \tau} d\tau \\ &= \frac{C}{a} \int_{-\infty}^{\infty} R_x(\tau') e^{-j2\pi f \tau'/a} d\tau' = \frac{C}{a} S_x\left(\frac{f}{a}\right). \end{aligned} \quad (9.124)$$

Using functional analysis, it can be shown that strictly speaking, the only class of signals that are self-similar are **power laws** characterized by the form  $x(t) = Ct^a$ , where  $C$  and  $a$  are constants. For a 1-D fractal random process, its power spectral



**Figure 9-5** Fractal patterns in nature.

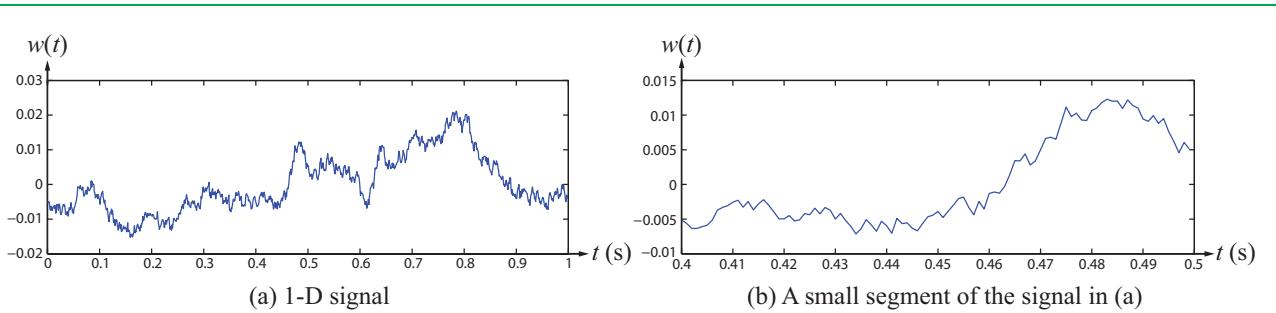
density has the form

$$\mathbf{S}_x(f) = \frac{C}{|f|^a}. \quad (9.125)$$

We should note that  $a > 0$  and  $\mathbf{S}(f) = \mathbf{S}(-f)$ .

Fractal random processes are often characterized using colors:

**White Fractal** ( $a = 0$ ): a random process whose power spectral density, as given by Eq. (9.125), has a **frequency exponent**  $a = 0$ . Hence,  $\mathbf{S}_x(f) = C$ , which means that all frequencies are present and weighted equally, just like *white light*.



**Figure 9-6** Fractal signal  $w(t)$ : Note the self-similarity between: (a) the entire signal and (b) a  $10 \times$  expanded scale version of the segment between  $t = 0.4$  s and  $0.5$  s.

**Pink Fractal** ( $a = 1$ ): a random process with  $\mathbf{S}_x(f) = C/|f|$ . Since higher frequencies (green and blue) are more heavily attenuated, the combination appears *pink* in color.

**Brown Fractal** ( $a = 2$ ): Brownian motion is characterized by  $S_x(f) = C/f^2$ , hence the name *brown*.

A true fractal signal is not realistic because it has infinite total power:

$$P = \int_{-\infty}^{\infty} \frac{C}{|f|^a} df \rightarrow \infty$$

for any  $a \geq 0$  and  $C \neq 0$ . Hence,  $\mathbf{S}_x(f)$  should be limited to a range of frequencies extending between  $f_{\min}$  and  $f_{\max}$ . Continuing with the *color* analogy,  $f_{\min}$  has to be greater than zero, because if  $a > 1$ ,

$$\int_{f_{\min}}^{\infty} \frac{C}{f^a} df \rightarrow \infty \quad (a > 1).$$

unless  $f_{\min} > 0$ . This unacceptable condition is known as the ***infrared catastrophe*** because, relative to frequencies in the viable part of the spectrum, infrared frequencies are considered to be very low. On the other end of the spectrum, the ***ultraviolet catastrophe*** represents the condition when  $0 < a < 1$  and

$$\int_0^{f_{\max}} \frac{C}{f^a} df \rightarrow \infty \quad (0 < a < 1),$$

unless  $f_{\max} < \infty$ . Hence, for a realistic fractal random process,

$$\mathbf{S}_x(f) = \frac{C}{|f|^a}, \quad f_{\min} < |f| < f_{\max}. \quad (9.126)$$

Since  $S_x(f)$  is symmetrical with respect to  $f = 0$ ,  $S_x(f) = S_x(-f)$ , and the total power of the fractal random

process is

$$P = 2 \int_{f_{\min}}^{f_{\max}} \mathbf{S}_x(f) df = 2 \int_{f_{\min}}^{f_{\max}} \frac{C}{f^a} df = \frac{2C}{1-a} (f_{\max}^{1-a} - f_{\min}^{1-a}). \quad (9.127)$$

The total power also is related to the zero-shift autocorrelation function  $R_x(0)$  and to the variance  $\sigma_x^2$  of the zero-mean Gaussian random process  $x(t)$  as in:

$$P = R_x(0) = \sigma_x^2. \quad (9.128)$$

## 9-8.2 Discrete-Time Fractals

As noted earlier, the power spectral density of a realistic fractal signal  $x(t)$  has to be bandlimited to within a range between  $f_{\min}$  and  $f_{\max}$ . Hence,  $x(t)$  can be converted to a discrete-time fractal random process  $x[n]$  by sampling it at a rate exceeding the Nyquist sampling rate of  $2f_{\max}$  samples/s. The self-similarity property of  $x(t)$  is equally present in  $x[n]$ , although it is more difficult to envision it in discrete time.

The functional form of the power spectral density of  $x[n]$  is analogous to that given by Eq. (9.125):

$$S_x(\Omega) = \frac{C}{|\Omega|^a}, \quad \Omega_{\min} < |\Omega| < \pi, \quad (9.129)$$

for some lower frequency bound  $\Omega_{\min}$ . Since  $\mathbf{S}_x(\Omega) = \mathbf{S}_x(-\Omega)$  the average power of  $x[n]$  is

$$P_{\text{av}} = 2 \times \frac{1}{2\pi} \int_{\Omega_{\min}}^{\pi} S_x(\Omega) d\Omega \\ = \frac{1}{\pi} \int_{\Omega_{\min}}^{\pi} \frac{C}{\Omega^a} d\Omega = \frac{C}{\pi(1-a)} (\pi^{1-a} - \Omega_{\min}^{1-a}). \quad (9.130)$$

### 9-8.3 Wiener Process

A **Wiener process**  $w(t)$  is a zero-mean **non-WSS** random process generated by integrating a white Gaussian random process  $z(t)$ :

$$w(t) = \int_{-\infty}^t z(\tau) d\tau. \quad (9.131)$$

The Wiener process is a 1-D version of Brownian motion, which describes the motion of particles or molecules in a solution. Often  $w(t)$  is initialized with  $w(0) = 0$ .

Even though the Wiener process is *not* WSS—and therefore it does not have a definable power spectral density, we will nonetheless proceed heuristically to obtain an expression for  $\mathbf{S}_w(f)$ . To that end, we start by converting Eq. (9.131) into the differential form

$$\frac{dw}{dt} = z(t). \quad (9.132)$$

Utilizing entry #5 in **Table 2-4**, the Fourier transform of the system described by Eq. (9.132) is

$$(j2\pi f) \mathbf{W}(f) = \mathbf{Z}(f), \quad (9.133)$$

which leads to the system response function  $\mathbf{H}(f)$  as

$$\mathbf{H}(f) = \frac{\mathbf{W}(f)}{\mathbf{Z}(f)} = \frac{1}{j2\pi f}. \quad (9.134)$$

As noted earlier in the definition for the Wiener process  $w(t)$ , the random process  $z(t)$  is white Gaussian, so the power spectral density of  $z(t)$  is constant:  $\mathbf{S}_z(f) = \sigma_z^2$ . For an LTI system with transfer function  $\mathbf{H}(f)$ , the power spectral density  $\mathbf{S}_w(f)$  of the Wiener random process  $w(t)$  is related to the power spectral density  $\mathbf{S}_z(f)$  of the white random process  $z(t)$  by

$$\mathbf{S}_w(f) = |\mathbf{H}(f)|^2 \mathbf{S}_z(f), \quad (9.135)$$

which leads to

$$\mathbf{S}_w(f) = \frac{\sigma_z^2}{4\pi^2} \frac{1}{f^2}. \quad (9.136)$$

Despite the non-rigorous analysis leading to Eq. (9.136), the result confirms that the power spectral density of a Wiener process varies as  $1/f^2$ .

### 9-8.4 Stochastic Wiener Filtering of Wiener Processes

The plot shown in **Fig. 9-7(a)** displays a realization of a Wiener process  $w(t)$  extending over a duration of 1 s. For simplicity, we

treat  $w(t)$  as a continuous-time signal, even though in reality it is a discrete-time version of  $w(t)$ . The power spectral density  $\hat{\mathbf{S}}_w(f)$  was estimated using the periodogram

$$\hat{\mathbf{S}}_w(f) = |\mathbf{W}(f)|^2 = \left| \int_{-\infty}^{\infty} w(t) e^{-j2\pi ft} dt \right|^2. \quad (9.137)$$

**Figure 9-7(b)** displays a plot of  $\mathbf{S}_w(f)$  as a function of  $f$  on a log-log scale over the range  $1 < f < 400$  Hz. Superimposed onto the actual spectrum (in blue) is a straight line in red whose slope in log-log scale is equivalent to  $1/f^2$ , confirming the applicability of the brown fractal model described by Eq. (9.136). From the intercept in **Fig. 9-7(b)**, it was determined that

$$\sigma_z^2 = 0.25.$$

Hence, Eq. (9.136) becomes

$$\hat{\mathbf{S}}_w(f) = \frac{0.25}{4\pi^2} \frac{1}{|f|^2}, \quad (1 < |f| < 400).$$

### A. Stochastic Wiener Denoising

To illustrate the utility of the stochastic denoising Wiener filter described in Section 9-4.4, we added random noise  $v(t)$  to  $w(t)$  to generate

$$y(t) = w(t) + v(t). \quad (9.138)$$

The sampled noisy signal  $y[n]$ , shown in **Fig. 9-7(c)**, exhibits large fluctuations because the signal-to-noise ratio is SNR = -0.21 dB. The negative sign is indicative that the noise contains more energy than the original signal  $w(t)$ . The corresponding signal-power to noise-power ratio is

$$\frac{P_s}{P_n} = 10^{(-0.21)/10} = 0.95.$$

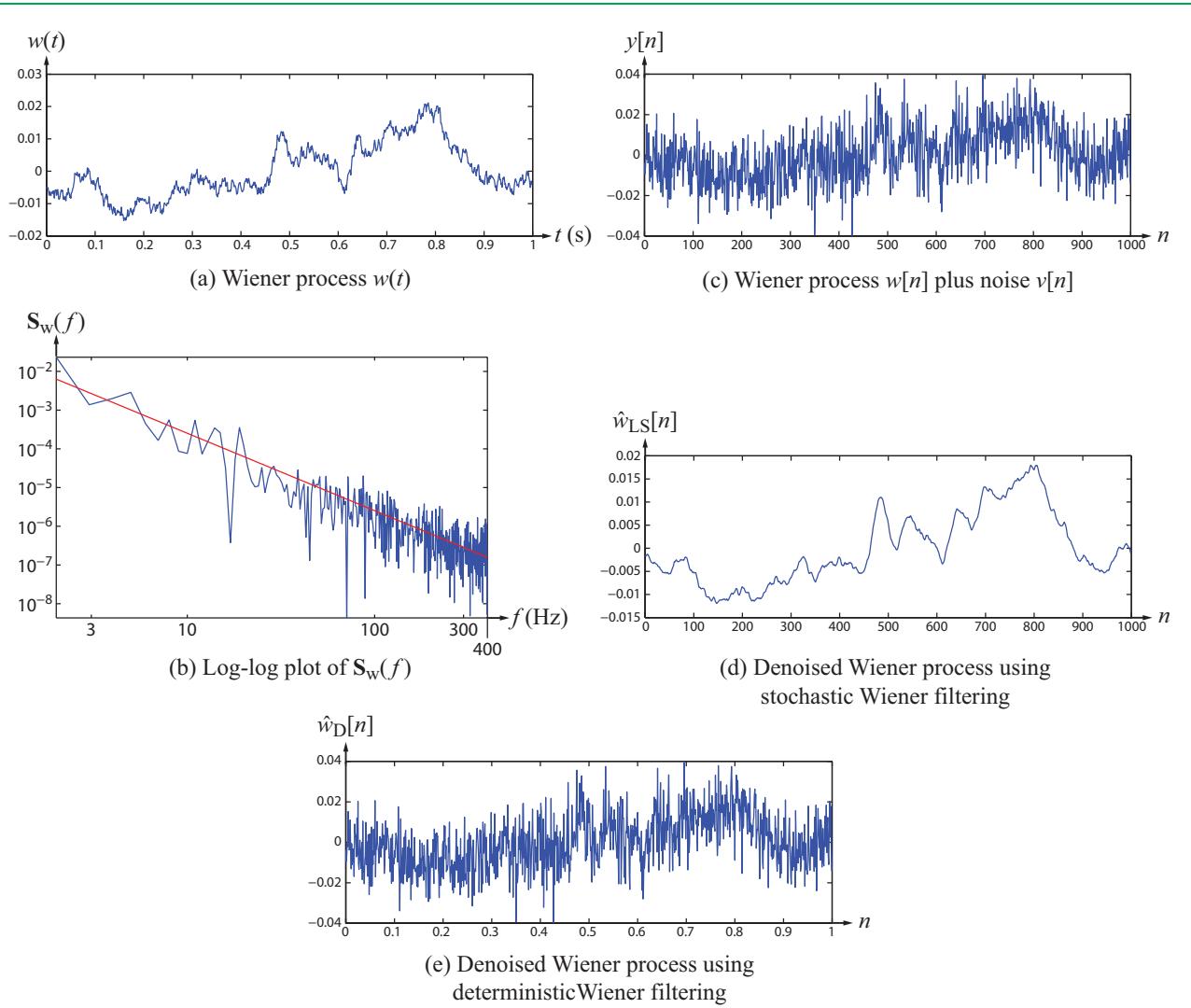
The variance of  $v(t)$  is  $\sigma_v^2 = 0.01$ .

Sampling  $y(t)$  at a sampling interval of 1 ms (corresponding to a sampling rate of 1000 samples/s) converts  $y(t)$  in Eq. (9.138) into the discrete-time signal

$$y[n] = w[n] + v[n]. \quad (9.139)$$

In the discrete-time frequency domain  $\Omega$ , the power spectral density corresponding to Eq. (9.136) is

$$\mathbf{S}_w(\Omega) = \frac{\sigma_z^2}{4\pi^2} \frac{1}{\Omega^2} = \frac{1}{16\pi^2\Omega^2}, \quad (9.140)$$



**Figure 9-7** (a) Wiener random process  $w(t)$ , (b) spectrum  $S_w(f)$  of  $w(t)$  plotted on a log-log scale, (c) noisy signal  $y[n] = w[n] + v[n]$ , (d) denoised estimate  $\hat{w}_{LS}[n]$  using stochastic filtering, and (e) deterministic estimate  $\hat{w}_D[n]$ , which resembles the noisy Wiener process in (c).

where we used  $\sigma_z^2 = 0.25$ . With subscript  $x$  in Eq. (9.67) changed to subscript  $w$ , and  $\sigma_v^2 = 0.01$ , use of Eq. (9.140) in

Eq. (9.67) gives

$$\begin{aligned} \mathbf{H}_{SDN}(\Omega) &= \frac{\mathbf{S}_w(\Omega)}{\mathbf{S}_w(\Omega) + \sigma_v^2} \\ &= \frac{1/(16\pi^2\Omega^2)}{1/(16\pi^2\Omega^2) + 0.01} = \frac{1}{1 + 1.58\Omega^2}, \end{aligned} \quad (9.141)$$

$$\hat{\mathbf{W}}_{LS}(\Omega) = \mathbf{H}_{SDN}(\Omega) \mathbf{Y}(\Omega), \quad (9.142)$$

and

$$\hat{w}_{LS}[n] = DTFT^{-1}\{\hat{\mathbf{W}}_{LS}(\Omega)\}, \quad (9.143)$$

where  $\mathbf{Y}(\Omega)$  is the DTFT of  $y[n]$ .

Application of the recipe outlined in Section 9-4.4 led to the plot shown in **Fig. 9-7(d)**. The estimated signal  $\hat{w}_{LS}[n]$  bears very close resemblance to the true signal  $w(t)$  shown in **Fig. 9-7(a)**, demonstrating the capability of the stochastic Wiener filter as a powerful tool for removing noise from noisy signals.

## B. Deterministic Wiener Denoising

In the deterministic approach to noise filtering, signal  $w(t)$  is treated as a white random process with  $S_w(f) = \sigma_w^2$ . Consequently, the expression for the deterministic denoising filter transfer function becomes

$$\mathbf{H}_{DDN}(\Omega) = \frac{S_w(\Omega)}{S_w(\Omega) + \sigma_v^2} = \frac{\sigma_w^2}{\sigma_w^2 + \sigma_v^2}. \quad (9.144)$$

Consequently, since  $\mathbf{H}_{DDN}(\Omega)$  is no longer a function of  $\Omega$ , implementation of the two steps in Eqs. (9.142) and (9.143) leads to the *deterministic estimate*

$$\hat{w}_D[n] = \left( \frac{\sigma_w^2}{\sigma_w^2 + \sigma_v^2} \right) y[n]. \quad (9.145)$$

Clearly,  $\hat{w}_D[n]$  is just a scaled version of  $y[n]$ , and therefore *no filtering is performed*. The plot shown in **Fig. 9-7(e)** is identical to that in **Fig. 9-7(c)** for  $y[n]$ . For display purposes, we chose to set the quantity  $\sigma_w^2/(\sigma_w^2 + \sigma_v^2) = 1$ .

## 9-8.5 Stochastic Deconvolution of the Wiener Process

To illustrate how the stochastic Wiener deconvolution filter of Section 9-4.5 can be used to deconvolve a random process, we selected the Wiener process  $w(t)$  shown in **Fig. 9-8(a)** and then convolved it with a blurring filter characterized by the rectangle impulse response

$$h_{blur}[n] = \begin{cases} 1000, & 0 < n < 100, \\ 0, & \text{otherwise.} \end{cases} \quad (9.146)$$

The convolved Wiener process  $z[n] = w[n] * h_{blur}[n]$  is displayed in **Fig. 9-8(b)**. The convolution process with the rectangle func-

tion is equivalent to integration in continuous time or summation in discrete time. That is, at a sampling rate of 1000 samples/s,

$$z[n] = 1000 \sum_{n=100}^n w[n], \quad (9.147a)$$

or equivalently,

$$z(t) = 1000 \int_{t-0.1}^t w(\tau) d\tau. \quad (9.147b)$$

To perform the summation (or integration), it is necessary to append  $w[n]$  with zeros for  $-100 \leq n < 0$  (or equivalently,  $-0.1 \leq t < 0$  for  $w(t)$ ). The integrated signal  $z[n]$  in **Fig. 9-8(b)** is a smoothed version of the original signal  $w[n]$ .

Next, white Gaussian noise  $v[n]$  was added to the system output  $z[n]$  to produce noisy observation

$$y[n] = z[n] + v[n] = h_{blur}[n] * w[n] + v[n]. \quad (9.148)$$

The noisy signal, plotted in **Fig. 9-8(c)**, is only slightly different from  $z[n]$  of **Fig. 9-8(b)**, because the signal-to-noise ratio is 36.9 dB, corresponding to an average signal power of about 4900 times that of the noise.

The stochastic Wiener deconvolution method uses the filter given by Eq. (9.71a), with subscript  $x$  changed to  $w$ :

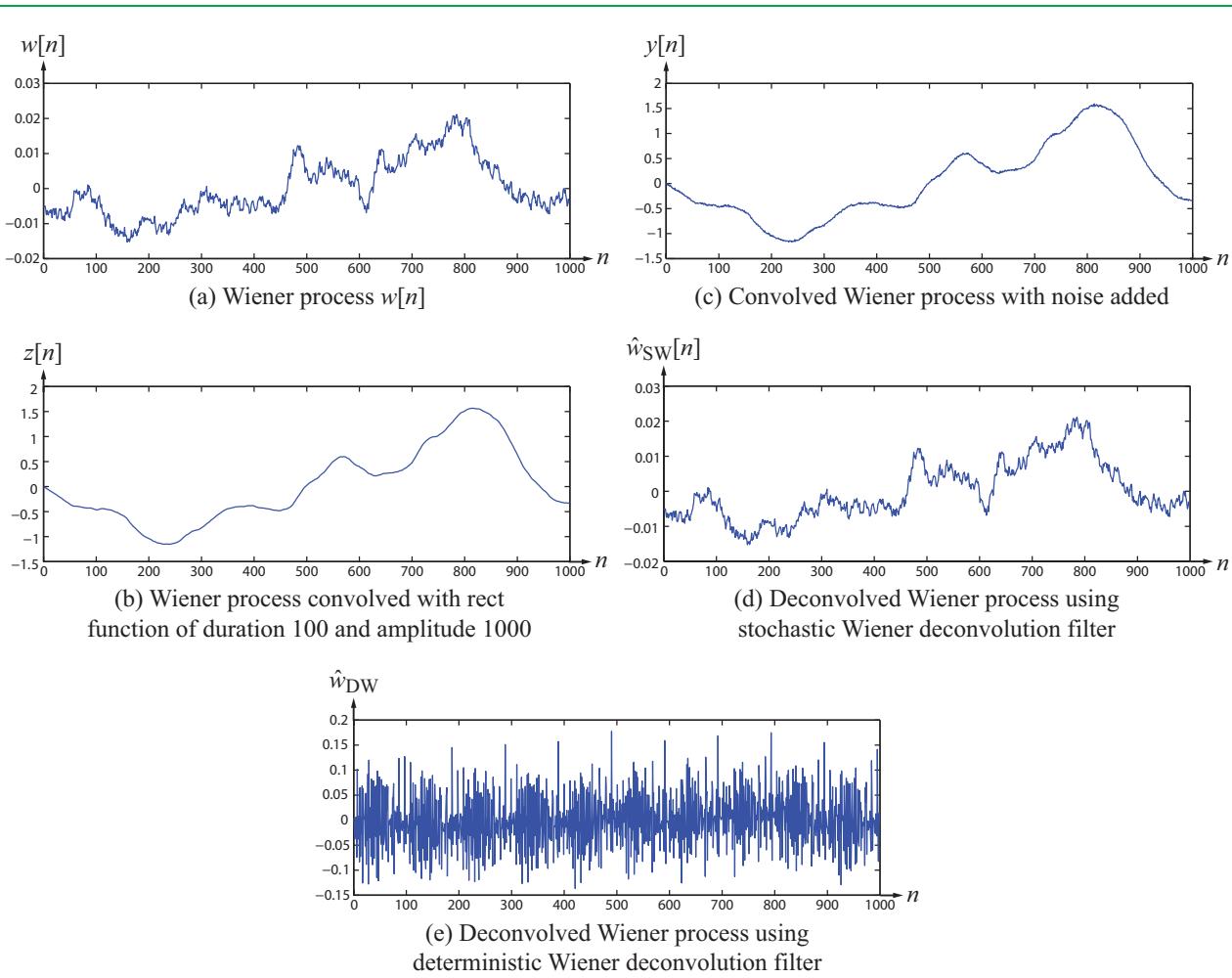
$$\mathbf{W}_{SDC}(\Omega) = \frac{\mathbf{H}_{blur}^*(\Omega) \mathbf{S}_w(\Omega)}{|\mathbf{H}_{blur}(\Omega)|^2 \mathbf{S}_w(\Omega) + \sigma_v^2}. \quad (9.149)$$

Here,  $\mathbf{H}_{blur}(\Omega)$  is the DTFT of  $h_{blur}[n]$  defined by Eq. (9.146),  $\mathbf{S}_w(\Omega)$  is given by Eq. (9.140), and from knowledge of the noise added to  $z[n]$ ,  $\sigma_v^2 = 0.01$ . Once  $\mathbf{W}_{SDC}(\Omega)$  has been computed,  $w[n]$  can be estimated using the form of Eq. (9.73), namely

$$\hat{w}_S[n] = DTFT^{-1}[\mathbf{W}_{SDC}(\Omega) \mathbf{Y}(\Omega)], \quad (9.150)$$

where  $\mathbf{Y}(\Omega)$  is the DTFT of the convolved noisy observation  $y[n]$ . Implementation of the stochastic convolution process led to the plot in **Fig. 9-8(d)**. The mean-square-error between  $w[n]$  and its reconstruction  $\hat{w}_S[n]$  over the interval  $0 < n < 1000$  is  $2.25 \times 10^{-6}$ .

For the sake of comparison, in part (e) of **Fig. 9-8** we again show  $\hat{w}_D[n]$ , the result of reconstructing  $w[n]$  using the deterministic Wiener filter. Clearly, the deterministic approach is not effective.



**Figure 9-8** Wiener deconvolution: (a) original Wiener process  $w[n]$ , (b) convolved signal  $z[n] = h[n] * w[n]$ , (c)  $y[n] = z[n] + v[n]$ , where  $v[n]$  is white Gaussian noise with  $S/N = 36.9$  dB, (d) deconvolved reconstruction  $\hat{w}_{SW}[n]$  using a stochastic Wiener filter, and (e) deconvolved reconstruction  $\hat{w}_{DW}[n]$  using a deterministic filter.

**Concept Question 9-8:** Why does the reciprocal power form of the power spectral density of a fractal random process need only hold over a finite range of frequencies, as in Eq. (9.126)?

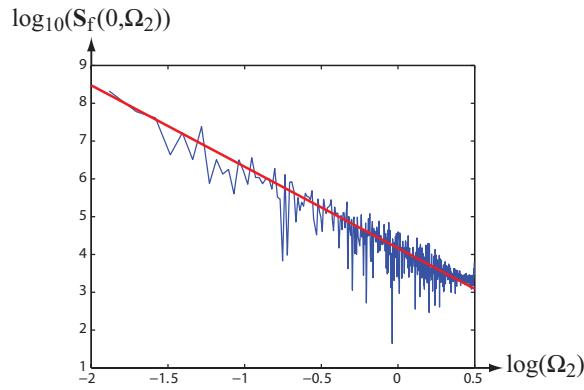
## 9-9 2-D Fractals

The 1-D fractal concepts introduced in the preceding section generalize readily to 2-D images, which we now demonstrate through an example:

**(1) Figure 9-9(a):**  $960 \times 1280$  image  $f[n, m]$  of a tree with a fractal-like branching structure.



(a) Fractal tree image



(d) Noisy blurred image

(e) Reconstructed image using  
a stochastic Wiener filter

(c) Blurred image

(f) Reconstructed image using  
a deterministic Wiener filter**Figure 9-9** 2-D deconvolution example.

**(2) Figure 9-9(b):** To ascertain the fractal character of the tree, the 2-D power spectral density  $\mathbf{S}_f(\Omega_1, \Omega_2)$  was estimated using

$$\hat{\mathbf{S}}_f(\Omega_1, \Omega_2) = \sum_{n=0}^{959} \sum_{m=0}^{1279} f[n, m] e^{-j(\Omega_1 n + \Omega_2 m)}, \quad (9.151)$$

and then  $\hat{\mathbf{S}}_f(0, \Omega_2)$  was plotted in **Fig. 9-9(b)** on a log-log scale. The average slope is  $-2$ , which means that  $\hat{\mathbf{S}}_f(0, \Omega_2)$  varies as  $1/\Omega_2^2$ . Generalizing to 2-D,

$$\mathbf{S}_f(\Omega_1, \Omega_2) = \frac{C}{\Omega_1^2 + \Omega_2^2}. \quad (9.152)$$

**(3) Figure 9-9(c):** To simulate motion blur in the horizontal direction, the image in **Fig. 9-9(a)** is convolved with a  $(1 \times 151)$  2-D PSF given by

$$h_{\text{blur}}[n, m] = \begin{cases} 1 & \text{for } 0 \leq n \leq 150 \text{ and } m = 1, \\ 0 & \text{otherwise.} \end{cases} \quad (9.153)$$

The resultant  $960 \times 1430$  blurred image given by

$$z[n, m] = h_{\text{blur}}[n, m] * * f[n, m] \quad (9.154)$$

is shown in **Fig. 9-9(c)**

**(4) Figure 9-9(d):** Addition of a slight amount of noise  $v[n, m]$  to  $z[n, m]$  gives

$$g[n, m] = z[n, m] + v[n, m] = h_{\text{blur}}[n, m] * * f[n, m] + v[n, m]. \quad (9.155)$$

The slightly noisy convolved image  $g[n, m]$  is shown in **Fig. 9-9(d)**.

**(5) Figure 9-9(e):** Generalizing Eq. (9.149) to 2-D gives

$$\mathbf{W}_{\text{SDC}}(\Omega_1, \Omega_2) = \frac{\mathbf{H}_{\text{blur}}^*(\Omega_1, \Omega_2) \mathbf{S}_f(\Omega_1, \Omega_2)}{|\mathbf{H}_{\text{blur}}(\Omega_1, \Omega_2)|^2 \mathbf{S}_f(\Omega_1, \Omega_2) + \sigma_v^2}, \quad (9.156)$$

where  $\mathbf{H}_{\text{blur}}(\Omega_1, \Omega_2)$  is the 2-D DSFT of  $h_{\text{blur}}[n, m]$  and  $\sigma_v^2$  is the noise variance. Application of the stochastic Wiener reconstruction recipe in 2-D gives

$$\hat{f}_s[n, m] = \text{DSFT}^{-1}[\mathbf{W}_{\text{SDC}}(\Omega_1, \Omega_2) \mathbf{G}(\Omega_1, \Omega_2)], \quad (9.157)$$

where  $\mathbf{G}(\Omega_1, \Omega_2)$  is the 2-D DSFT of the observed image  $g[n, m]$ . The reconstructed image is shown in **Fig. 9-9(e)**.

**(6) Figure 9-9(f):** Deterministic convolution involves the same steps represented by Eqs. (9.156) and (9.157) for the stochastic deconvolution process except for one single differ-

ence, namely that instead of using Eq. (9.152) for  $\mathbf{S}_f(\Omega_1, \Omega_2)$  the expression used is  $\mathbf{S}_f(\Omega_1, \Omega_2) = \sigma_f^2$ . The absence of the inverse frequency dependency leads to a fuzzier reconstruction than realized by the stochastic deconvolution filter.

## 9-10 Markov Random Fields

Medical applications—such as ultrasound imaging, MRI, and others—rely on the use of image processing tools to **segment** the images produced by those imaging sensors into regions of common features. The different regions may belong to different organs or different types of tissue, and the goal of the segmentation process is to facilitate the interpretation of the information contained in the images. Another name for segmentation is **classification**: assigning each pixel to one of a set of predefined classes on the basis of its own value as well as those of its neighbors. Similar tools are used to segment an image captured by a video camera, an infrared temperature sensor, or a weather radar system.

An important ingredient of the image segmentation process is a parameter estimation technique that models an image as a **Markov random field (MRF)**. The MRF model assigns each image pixel  $f[n, m]$  a conditional probability density based, in part, on the values of the pixels in its immediate neighborhood.

The purpose of the present section is to introduce the concept and attributes of MRFs and to demonstrate their applications through image examples.

### 9-10.1 1-D Markov Process

Before we delve into the 2-D case, let us first consider the 1-D case of a Markov random process  $x[n]$ . The value of  $x[n]$  is continuous, but it is sampled in time in discrete steps, generating the random vector  $\{ \dots, x[0], x[1], \dots, x[N], \dots \}$ . In Markov language,  $x[n]$  at time  $n$  is regarded as the *present*,  $x[n+1]$  is regarded as the *future*, and its values  $x[n-1], x[n-2], \dots, x[0]$  are regarded as the *past*. The Markov model assigns a conditional pdf to “the future value  $x[n+1]$  based on the present value  $x[n]$  but independent of past values,” which is equivalent to the mathematical statement

$$p(x[n+1] | \{x[0], x[1], \dots, x[n]\}) = p(x[n+1] | x[n]). \quad (9.158)$$

In 2-D, the “present value of  $x[n]$ ” becomes the values of the pixels in the neighborhood of pixel  $f[n, m]$ , and “past values” become pixels outside that neighborhood.

Using the Markov condition encapsulated by Eq. (9.158), and

after much algebra, it can be shown that

$$p(x[n]|\{\dots,x[N],x[N-1],\dots,x[n+1],x[n-1],\dots,x[0],\dots\}) = \frac{p(x[n+1]|x[n]) p(x[n]|x[n-1])}{\int p(x'[n+1]|x'[n]) p(x'[n]|x'[n-1]) dx'[n]}, \quad (9.159)$$

which states that the conditional pdf of  $x[n]$ , given all other values  $\{\dots,x[0],\dots,x[N],\dots\}$ , is governed by the product of two conditional pdfs, one relating  $x[n]$  to its immediate past  $x[n-1]$ , and another relating  $x[n]$  to its immediate future  $x[n+1]$ . An MRF generalizes this relationship from 1-D to 2-D using the concepts of neighborhoods and cliques.

## 9-10.2 Neighborhoods and Cliques

The **neighborhood** of a pixel at location  $[n,m]$  is denoted\*  $\Delta[n,m]$  and consists of a set of pixels surrounding pixel  $[n,m]$ , but excluding it. **Figure 9-10** displays the pixel locations included in 3-, 8-, 15-neighbor systems. Neighborhoods may also be defined in terms of *cliques*.

A **clique** is a set of locations such that any two members of the clique adjoin each other, either horizontally, vertically, or diagonally. **Figure 9-11** shows 10 cliques, one of which is a self-adjoining clique. A neighborhood can be represented as a disjoint union of cliques of various types.

In a typical image, pixel neighbors are more likely to have similar intensities than distant pixels. Different organs in a medical image or different objects in an imaged scene tend to have smooth features and the boundaries between them tend to have sharp transitions across them. **Image texture**, defined as the spatial variability within a given “homogeneous” region, might exhibit different statistical variations in different types of regions (classes). Image segmentation (classification) algorithms differ by the type of sensor used to produce the image, the type of scene, and the intended application. However, most of these algorithms share a common strategy, which we outline in this and the next section. Part of the strategy is to define the “relevant” neighborhood for a particular application, which entails defining the combination of cliques comprising such a neighborhood.

## 9-10.3 Likelihood Functions

The goal of image segmentation is to classify each pixel into one of  $K$  predefined classes. In a medical ultrasound image,

the classes pertain to different types of tissue, while in a video image of terrain, the classes might be roads, cars, trees, etc. The segmentation procedure involves several elements, the first of which is the likelihood function of the estimation method used to implement the segmentation.

Given the general image model

$$g[n,m] = f[n,m] + v[n,m], \quad (9.160)$$

where  $g[n,m]$  is the observed intensity of pixel  $[n,m]$ ,  $f[n,m]$  is the noise-free intensity, and  $v[n,m]$  is the additive noise, the estimated value  $\hat{f}[n,m]$  is obtained from  $g[n,m]$  by maximizing the likelihood function. The likelihood functions for the maximum likelihood estimation (**MLE**) method and the maximum *a posteriori* probability (**MAP**) method are given by

MLE:

$$p(g[n,m]|f[n,m]), \quad (9.161a)$$

MAP:

$$p(f[n,m]|g[n,m]) = \frac{p(g[n,m]|f[n,m]) p(f[n,m])}{p(g[n,m])}. \quad (9.161b)$$

### A. MLE Estimate $\hat{f}_{\text{MLE}}[n,m]$

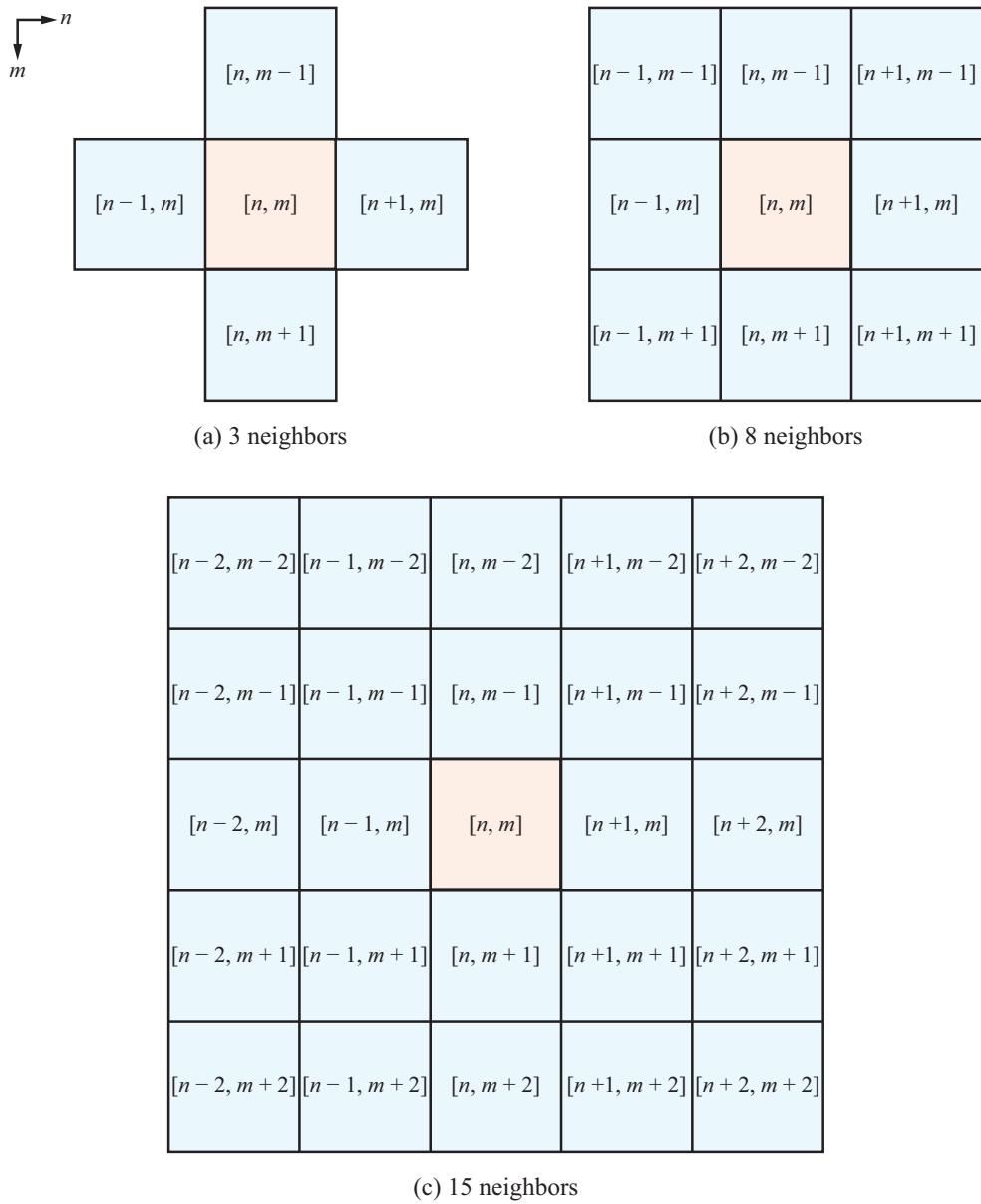
As shown earlier in Sections 9-2 and 9-3 for the 1-D case, the MLE maximization process entails: (a) introducing an appropriate model for the conditional probability of Eq. (9.161a), (b) taking the logarithm of the model expression, (c) computing the derivative of the likelihood function with respect to the unknown quantity  $f[n,m]$  and equating it to zero, and (d) solving for  $f[n,m]$ . The resultant value of  $f[n,m]$  is labeled  $\hat{f}_{\text{MLE}}[n,m]$ . The computed estimate is then used to assign pixel  $[n,m]$  to one of the  $K$  classes, in accordance with a **class-assignment algorithm** (introduced shortly).

### B. MAP Estimate $\hat{f}_{\text{MAP}}[n,m]$

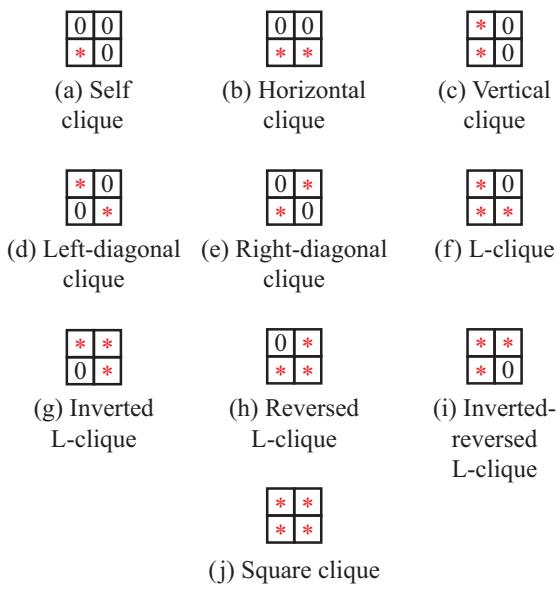
MAP requires models for two pdfs, namely the same conditional pdf used in MLE,  $p(g[n,m]|f[n,m])$ , as well as the pdf  $p(f[n,m])$ . The third pdf,  $p(g[n,m])$  in the denominator of Eq. (9.161b), is not needed and may be set equal to a constant. This is because it disappears in the maximization procedure (taking the log, differentiating with respect to  $f[n,m]$ , and equating to zero).

The type of models commonly used to describe the two pdfs in the numerator of Eq. (9.161b) are introduced shortly.

\*For purposes of clarity, we use the symbol  $\Delta$ , even though the Markov field literature use  $\partial$ .



**Figure 9-10** Examples of pixel neighborhoods.



**Figure 9-11** Ten types of pixel cliques. A red star denotes a location in a clique and a 0 denotes a location not in a clique.

#### 9-10.4 pdf $p(f[n, m])$

The **Hammersley-Clifford theorem** is sometimes called “the fundamental theorem of random fields.” It states that any conditional pdf for a set of random variables in a Markov random field is always given by the **Gibbs distribution**, provided the following conditions hold:

(1) **Positivity**:  $p(f[n, m]) > 0$ , which usually is true for images. Note that positivity states that the pmf  $p(f[n, m])$  of  $f[n, m]$  is always positive, but  $f[n, m]$  itself may assume both positive and negative values.

(2) **Stationarity**:  $p(f[n, m])$  does not vary with position. That is, the same statistics apply across the entire image.

(3) **Locality**: For a specified neighborhood  $\Delta[n, m]$ ,  $f[n, m]$ , is a Markov random field that obeys the **locality** condition

$$\begin{aligned} p(f[n, m] | \{ f[n', m'], n' \neq n, m' \neq m \}) \\ = p(f[n, m] | f_{\Delta}[n, m]), \end{aligned} \quad (9.162)$$

where  $f_{\Delta}[n, m]$  are the values of pixels in the specified neighborhood  $\Delta[n, m]$ . Thus, the pdf of  $f[n, m]$  at location  $[n, m]$  depends on only the values of the pixels within the neighborhood  $\Delta[n, m]$ .

The **Gibbs distribution** is a joint pdf for  $f[n, m]$  at all locations  $[n, m]$ , and it has the general form:

$$p(\{ f[n, m] \}) = \frac{1}{Z} e^{-U[n, m]}, \quad (9.163)$$

where  $\{ f[n, m] \}$  is  $f[n, m]$  at all locations  $[n, m]$  and  $Z$  is the **partition function** that normalizes the pdf so that it integrates to unity, as every pdf should, and  $U[n, m]$  is the sum of the **potential energy** over the cliques  $c$ :

$$U[n, m] = \sum_{\text{all cliques } c} U_c[n', m']. \quad (9.164)$$

The summation extends over the cliques defining the specified neighborhood  $\Delta[n, m]$ , and the model for the potential energy of an individual clique depends on the segmentation algorithm.

The **Ising model** is a popular image-processing algorithm used for identifying boundaries between different classes. The original Ising model, which was developed for characterizing the statistics of  $\pm 1$  electron spin in ferromagnetic materials, has been adapted to image segmentation by restricting the Markov random field  $f[n, m]$  to a few discrete values. Consequently,  $f[n, m]$  is described by a pmf, not a pdf. Each  $f[n, m]$  interacts only with its specified neighbors, such as its 8 immediate neighbors.

A particular form of the Ising model uses a **binary** assignment in which  $f[n, m]$  can assume either a value of 0 or a value of 1, and it interacts with its two horizontal neighbors and two vertical neighbors (**Fig. 9-10(a)**). Thus,

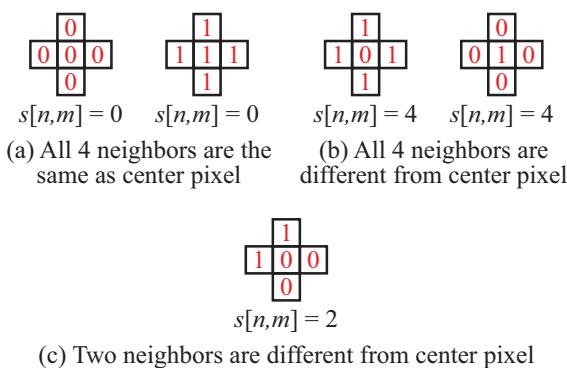
$$\Delta[n, m] = \{ [n \pm 1, m \pm 1] \}. \quad (9.165)$$

The **potential energy** of pixel  $[n, m]$  is defined as

$$U[n, m] = \beta s[n, m], \quad (9.166)$$

where  $\beta$  is a model parameter selected through experimentation (as discussed later in Section 9-11.3) and  $s[n, m]$  is a **dissimilarity index** that accounts for how many of the four pixels surrounding pixel  $[n, m]$  are different from pixel  $[n, m]$ . For example,  $s[n, m] = 0$  for the center pixel in **Fig. 9-12(a)** and  $s[n, m] = 4$  for the center pixel in **Fig. 9-12(b)**. The full range of dissimilarity index  $s[n, m]$  is between 0 for a pixel surrounded with like pixels and 4 for a pixel different from all of its four neighbors.

To compute  $s[n, m]$  for each pixel in the image, we first have to devise a scheme for assigning a value of 0 or 1 to each pixel. As we shall see in a later section, such an assignment can be realized using maximum likelihood segmentation. Consider, for



**Figure 9-12** Dissimilarity index  $s[n,m]$  is equal to the number of immediate horizontal and vertical pixels that are different from the pixel under consideration.

example, the noisy two-class image shown in 9-13(a). Because of the randomness associated with the pixel values, it is not easy to assign individual pixels to their correct class. When our eyes look at the image, however, we discern the boundaries between the four squares, even though some of the pixels in the predominantly dark squares are bright, and some of those in the predominantly bright squares are dark. The goal of segmentation is to assign each pixel to its correct class by incorporating

information about the neighborhood of that pixel, akin to how our eye-brain system perceives the boundaries between the squares in the image of Fig. 9-13(a).

By computing and then plotting the histogram of the observed image  $g[n,m]$ , as in Fig. 9-13(b), we can divide the range of values of  $g[n,m]$  into two segments and assign them values of 0 and 1. [If the image is to be classified into more than two classes, the approach can be extended accordingly, by assigning discrete values commensurate with the distances between the peaks in the distribution.] Using the 0/1 assignment, each pixel in the original image is then assigned a value of 0 or 1. We call this image a *binary image*.

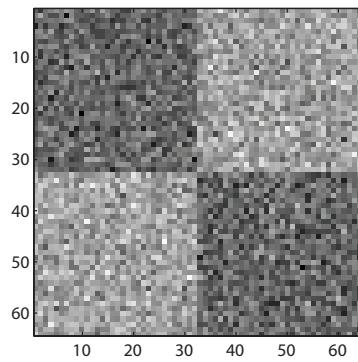
Once the dissimilarity index  $s[n,m]$  has been determined for each pixel, the *Ising-Gibbs distribution* given by Eq. (9.163) becomes

$$p(f[n,m]) = \frac{1}{Z} e^{-\beta s[n,m]}. \quad (9.167)$$

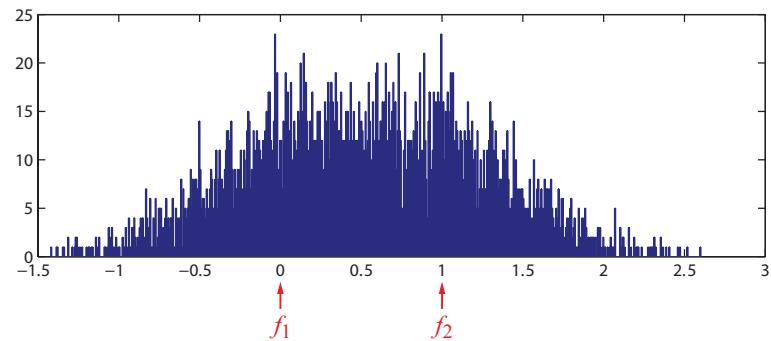
The negative exponential model favors shorter boundaries (smaller  $s[n,m]$ ) over longer boundaries. Its role in image segmentation is discussed shortly.

**Concept Question 9-9:** In general terms, what is a Markov random field model of an image?

**Concept Question 9-10:** How can a Markov random field model be useful?



(a) Noisy image



(b) Histogram of noisy image

**Figure 9-13** Noisy image of four squares and associated histogram. The noise-free image had only positive values, but the addition of random noise expands the range to negative values and to larger positive values.

## 9-11 Application of MRF to Image Segmentation

### 9-11.1 Image Histogram

Consider the  $320 \times 320$  noisy image shown in [Fig. 9-14\(a\)](#). Our goal is to segment the image into three classes: bone, other tissue, and background. To that end, we start by generating the histogram of the noisy image  $g[n, m]$ . The histogram displayed in [Fig. 9-14\(b\)](#) encompasses three clusters of pixels, a group concentrated around  $f[n, m] = f_1 = 0$ , another around  $f[n, m] = f_2 = 150$ , and a third around  $f[n, m] = f_3 = 300$ . The three clusters correspond to dark pixels in the background part of the image, pixels of non-bone tissue, and pixels of the bones in the five toes. In the present case, the values of  $f_1$  to  $f_3$  were extracted from the image histogram, but in practice more precise values are available from calibration experiments performed with the imaging sensor.

### 9-11.2 MLE Segmentation

The likelihood function for a noisy image  $g[n, m]$ , modeled as a Gaussian with variance  $\sigma^2$ , is given by

$$p(g[n, m]|f[n, m]) = \prod_{n=1}^N \prod_{m=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(1/2\sigma^2)(g[n, m]-f[n, m])^2}. \quad (9.168)$$

Here,  $N = 320$ . Taking the natural log of both sides gives

$$\begin{aligned} \ln(p(g[n, m]|f[n, m])) &= \\ &- \frac{N^2}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{n=1}^N \sum_{m=1}^N (g[n, m] - f[n, m])^2. \end{aligned} \quad (9.169)$$

In the present segmentation, each pixel is assigned to one of three classes:

- Class 1:  $f_1 = 0$
- Class 2:  $f_2 = 150$
- Class 3:  $f_3 = 300$

That is, for each pixel  $[n, m]$  with pixel value  $g[n, m]$ ,  $f[n, m]$  in Eq. (9.169) can assume only one of the three listed values. Because of the minus sign ahead of the second term in Eq. (9.169), the log of the MLE likelihood function is maximized by minimizing the difference  $(g[n, m] - f[n, m])^2$ . Hence, for pixel  $[n, m]$  with value  $g[n, m]$ , the MLE likelihood is maximized by assigning that pixel the value  $f_1$ ,  $f_2$ , or  $f_3$  depending on which one of them is closest to  $g[n, m]$ . Consequently, a pixel

with  $g[n, m] = 50$  is closer to  $f_1 = 0$  than to  $f_2 = 150$ , and therefore it gets classified as class 1 (black). Similarly, a pixel with  $g[n, m] = 100$  is closest to  $f_2 = 150$ , and therefore it gets classified as class 2 (tissue). The MLE segmentation process leads to the image shown in [Fig. 9-14\(c\)](#).

### 9-11.3 MAP Segmentation

Inserting Eqs. (9.167) and (9.168) into Eq. (9.161b) provides the MAP likelihood function

$$\begin{aligned} p(f[n, m]|g[n, m]) &= \frac{p(g[n, m]|f[n, m]) p(f[n, m])}{p(g[n, m])} \\ &= \left\{ \prod_{n=1}^N \prod_{m=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(1/2\sigma^2)(g[n, m]-f[n, m])^2} \times \frac{1}{Z} e^{-\beta s[n, m]} \right\} \\ &\quad \times \frac{1}{p(g[n, m])}, \end{aligned} \quad (9.170)$$

where  $\beta$  is a selectable (trial-and-error) parameter similar to the Tikhonov parameter  $\lambda$ , and  $s[n, m]$  is the dissimilarity index of pixel  $[n, m]$ , computed from the MLE segmented image.

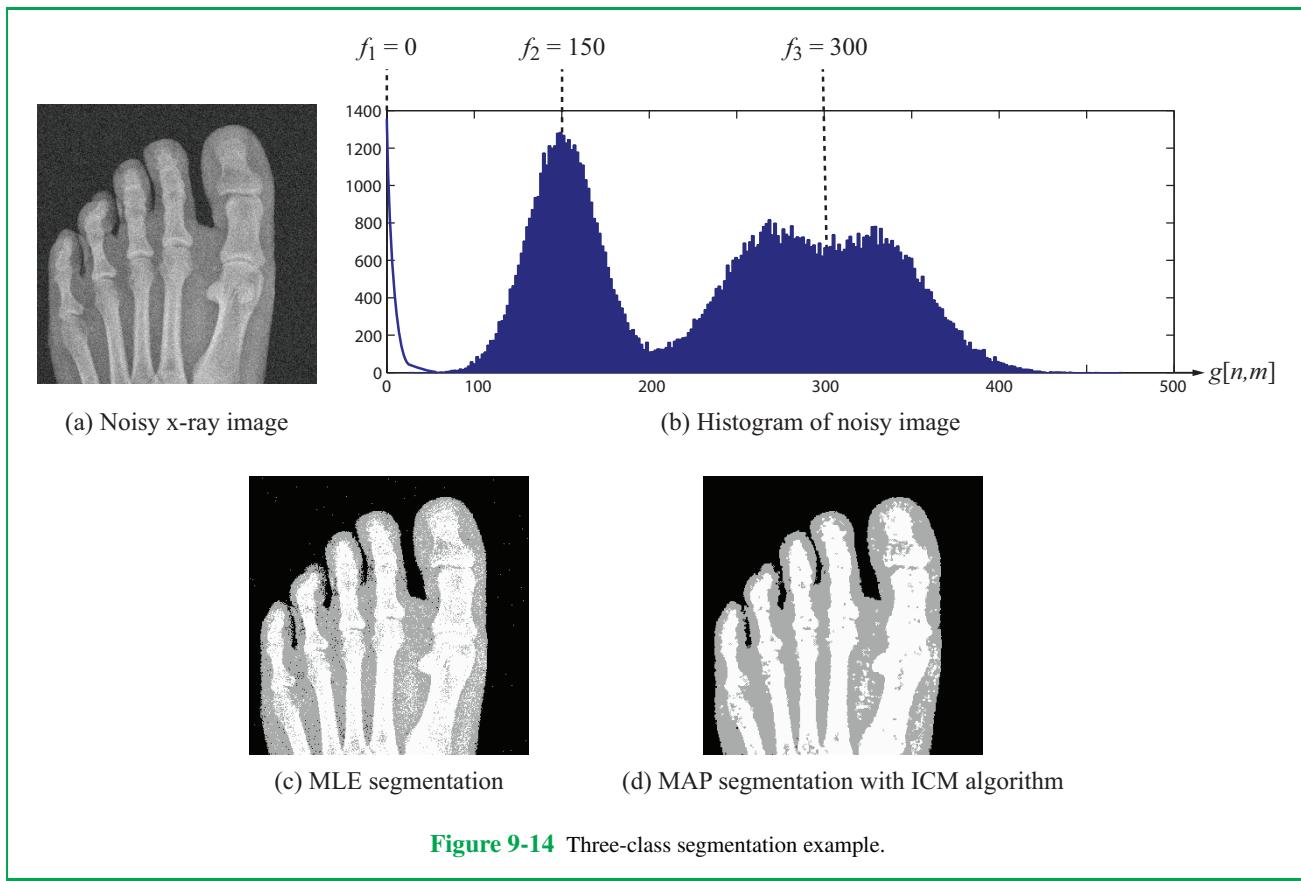
The natural log of the MAP likelihood function is

$$\begin{aligned} \ln(p(f[n, m]|g[n, m])) &= \left\{ -\frac{N^2}{2} \ln(2\pi\sigma^2) - \ln Z - \ln C \right\} \\ &+ \left\{ -\frac{1}{2\sigma^2} \sum_{n=1}^N \sum_{m=1}^N (g[n, m] - f[n, m])^2 - \beta s[n, m] \right\}, \end{aligned} \quad (9.171)$$

where we have replaced  $p(g[n, m])$  with constant  $C$  because  $p(g[n, m])$  has no influence on the maximization process. In Eq. (9.171), we have two groups of terms. The first group—consisting of three terms—has no role in the maximization process, whereas the second group does. Of the second group, let us consider the terms associated with pixel  $[n, m]$  and let us call their combination  $z[n, m]$ :

$$z[n, m] = -\frac{1}{2\sigma^2} ((g[n, m] - f[n, m])^2 - \beta s[n, m]). \quad (9.172)$$

We wish to assign pixel  $f[n, m]$  one of three values, namely  $f_1 = 0$ ,  $f_2 = 150$ , or  $f_3 = 300$ . The assignment also classifies the pixel as background, tissue, or bone. The assignment seeks to maximize the likelihood function, which (because of the minus signs) is accomplished by minimizing the value of  $z[n, m]$ . In the absence of the second term in Eq. (9.172), the process would collapse to the MLE segmentation of the previous subsection,



**Figure 9-14** Three-class segmentation example.

but the presence of the term  $\beta s[n,m]$  introduces the degree of similarity/dissimilarity of pixel  $[n,m]$  relative to its neighbors into the segmentation decision.

For each pixel  $[n,m]$ ,  $g[n,m]$  is the observed value of that pixel in the noisy image,  $\beta$  is a trial-and-error parameter,  $\sigma^2$  is the image variance (usually determined through calibration tests), and  $s[n,m]$  is the dissimilarity index obtained from the MLE segmentation image. By computing  $z[n,m]$  three times, once with  $f[n,m]$  in Eq. (9.172) set equal to  $f_1$ , another with  $f[n,m]$  set equal to  $f_2$ , and a third time with  $f[n,m]$  set equal to  $f_3$ , we obtain three values for  $z[n,m]$ . MAP segmentation selects the smallest of the three absolute values of  $z[n,m]$ , and assigns that pixel to the corresponding class. The outcome is shown in Fig. 9-14(d).

Computationally, a commonly used algorithm for realizing the MAP segmentation is the **Iterated Conditional Modes**

(**ICM**) algorithm. The algorithm repeats the segmentation process iteratively until it reaches a defined threshold.

### Example 9-3: Four-Square Image

Apply the ICM algorithm with  $\beta = 300$  to segment the  $64 \times 64$  noisy binary image shown in Fig. 9-15(a). The noise level is characterized by a variance  $\sigma^2 = 0.25$ .

**Solution:** We start by computing and displaying the image histogram shown in Fig. 9-15(b). Based on the histogram, we select  $f_1 = 1.5$  for the darker class and  $f_2 = 2.5$  for the brighter class.

Next, we apply MLE segmentation. We assign each pixel  $[n,m]$  the value  $f_1$  or  $f_2$  depending on which one of them is

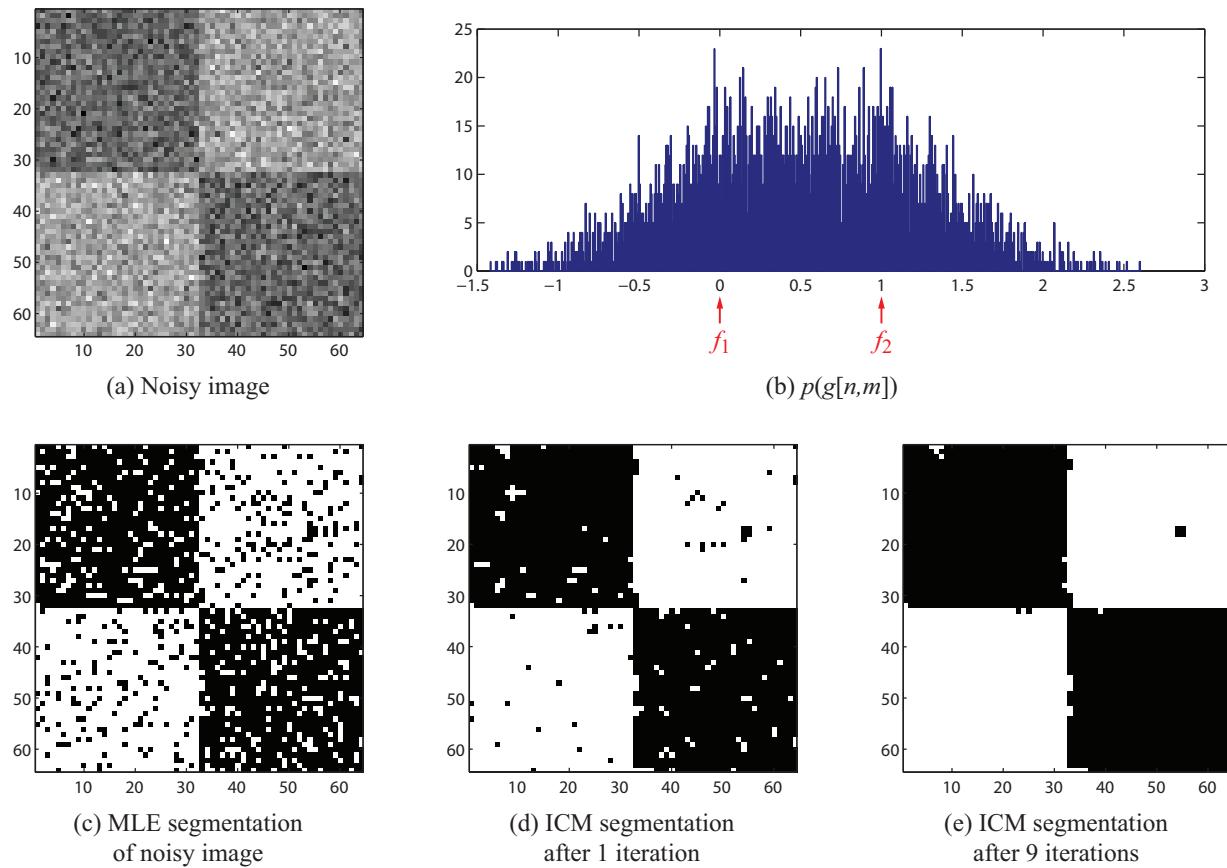


Figure 9-15 Example 9-2.

closest in value to the pixel value  $g[n, m]$ . The result is displayed in **Fig. 9-15(c)**.

Finally, we apply MAP segmentation using the ICM algorithm to obtain the image in **Fig. 9-15(d)**. The segmentation can be improved through iterative repetition of the process. The first iteration generates an MLE image, computes  $s[n, m]$  for each pixel, and then computes a MAP image. Each subsequent iteration computes a new set of values for  $s[n, m]$  and a new MAP image. The image in **Fig. 9-15(e)** is the result of 9 iterations. Except for a few misidentified pixels—mostly along the boundaries of the four squares, the MAP segmentation procedure provides very good discrimination between the white and black squares.

**Concept Question 9-11:** What application of Markov random fields was covered in Section 9-11?

## Summary

### Concepts

- The goal of estimation is to estimate an unknown  $x$  from observation  $y_{\text{obs}}$  of a random variable or process  $y$  using a conditional pdf  $p(y|x)$  which comes from a model, and possibly an *a priori* pdf  $p(x)$  for  $x$ .
- The maximum likelihood estimate  $\hat{x}_{\text{MLE}}(y_{\text{obs}})$  is the value of  $x$  that maximizes the likelihood function  $p(y_{\text{obs}}|x)$ , or equivalently its logarithm.
- The maximum *a posteriori* estimate  $\hat{x}_{\text{MAP}}(y_{\text{obs}})$  is the value of  $x$  that maximizes the *a posteriori* pdf  $p(x|y_{\text{obs}})$ , or equivalently its logarithm.  $p(x|y_{\text{obs}})$  is computed from likelihood function  $p(y_{\text{obs}}|x)$  and *a priori* pdf  $p(x)$  using Bayes's rule (see below).
- The least-squares estimate  $\hat{x}_{\text{LS}}(y_{\text{obs}})$  is the value of  $x$  that

minimizes the mean square error  $E[(x - \hat{x}_{\text{LS}}(y))^2]$ .

- The 2-D versions of these estimators are generalizations of the 1-D ones.
- Using a white power spectral density makes the stochastic Wiener filter reduce to the deterministic Wiener filter.
- A fractal model of power spectral density greatly improves performance.
- A Markov random field (MRF) models each image pixel as a random variable conditionally dependent on its neighboring pixels.
- A MRF model and the ICM algorithm can be used to segment an image.

### Mathematical Formulae

#### Bayes's rule

$$p(x|y) = p(y|x) \frac{p(x)}{p(y)}$$

#### MAP log-likelihood

$$\log p(x|y_{\text{obs}}) = \log p(y_{\text{obs}}|x) + \log p(x) - \log(p(y_{\text{obs}}))$$

#### Least-squares estimate

$$\hat{x}_{\text{LS}}(y_{\text{obs}}) = E[x | y = y_{\text{obs}}] = \frac{\int x' p(y_{\text{obs}}|x') p(x') dx'}{\int p(y_{\text{obs}}|x') p(x') dx'}$$

#### Gaussian least-squares estimate

$$\hat{x}_{\text{LS}}(y_{\text{obs}}) = \bar{x} + \mathbf{K}_{\mathbf{x}, \mathbf{y}} \mathbf{K}_{\mathbf{y}}^{-1} (\mathbf{y}_{\text{obs}} - \bar{\mathbf{y}})$$

#### Sample mean

$$\hat{x}_{\text{LS}}(y_{\text{obs}}) = \frac{1}{N} \sum_{n=1}^N y_{\text{obs}}[n]$$

#### Least-squares estimation orthogonality principle

$$E[(\mathbf{x} - \hat{\mathbf{x}}_{\text{LS}}(\mathbf{y})) \mathbf{y}^T] = \mathbf{0}$$

#### 1-D Linear least-squares estimator

$$\hat{x}_{\text{LLSE}}[n] = h[n] * y_{\text{obs}}[n]$$

#### 1-D Deterministic Wiener deconvolution filter

$$\hat{\mathbf{x}}(\Omega) = \mathbf{y}_{\text{obs}}(\Omega) \frac{\mathbf{H}_{\text{blur}}^*(\Omega)}{|\mathbf{H}_{\text{blur}}(\Omega)|^2 + \sigma_v^2}$$

#### 1-D Stochastic Wiener deconvolution filter

$$\hat{\mathbf{x}}(\Omega) = \mathbf{y}_{\text{obs}}(\Omega) \frac{\mathbf{H}_{\text{blur}}^*(\Omega) \mathbf{S}_x(\Omega)}{|\mathbf{H}_{\text{blur}}(\Omega)|^2 \mathbf{S}_x(\Omega) + \lambda^2}$$

#### Fractal model

$$S_x(f) = \frac{c}{|f|^a}, \text{ where } a = 1 \text{ or } 2$$

#### Gibbs distribution of a Markov random field

$$p(\{f[n, m]\}) = \frac{1}{Z} e^{-U[n, m]}$$

### Important Terms

Provide definitions or explain the meaning of the following terms:

Bayes's rule

Gibbs distribution

Markov random field

sample mean

deterministic Wiener filter

ICM algorithm

maximum likelihood

stochastic Wiener filter

fractal power spectral density

least-squares

maximum *a posteriori*

## PROBLEMS

### Section 9-1: Estimation Methods

**9.1** A coin with  $\mathbb{P}[\text{heads}] = x$  is flipped  $N$  times. The results of each flip are independent. We observe  $n_0$  = number of heads in  $N$  flips.

(a) If  $x$  is an unknown constant, compute the MLE estimate  $\hat{x}_{\text{MLE}}(n_0)$ .

(b) If  $x$  is a random variable with *a priori* pdf

$$p(x) = \begin{cases} 10e^{-10x} & \text{for } 0 \leq x \leq 1, \\ 0 & \text{otherwise,} \end{cases}$$

derive a quadratic equation for the MAP estimate  $\hat{x}_{\text{MAP}}(n_0)$ . Neglect  $\mathbb{P}[x > 1] = e^{-10} = 0.000045$ .

(c) If  $N = 92$  and  $n_0 = 20$ , compute  $\hat{x}_{\text{MAP}}(20)$ .

**9.2** A coin with  $\mathbb{P}[\text{heads}] = x$  is flipped  $N$  times. The results of flips are independent. We observe  $n_0$  = number of heads in  $N$  flips.

(a) If  $x$  is a random variable with *a priori* pdf

$$p(x) = \frac{1}{3} \delta\left(x - \frac{1}{4}\right) + \frac{2}{3} \delta\left(x - \frac{1}{2}\right)$$

so that  $\mathbb{P}[x = \frac{1}{4}] = \frac{1}{3}$  and  $\mathbb{P}[x = \frac{1}{2}] = \frac{2}{3}$ , compute an expression for the least-squares estimator  $\hat{x}_{\text{LS}}(n_0)$ .

(b) If  $N = 3$  and  $n_0 = 2$ , compute the least-squares estimate  $\hat{x}_{\text{LS}}(2)$ .

**9.3** An exponential random variable  $y$  has pdf

$$p(y|x) = \begin{cases} xe^{-xy} & \text{for } y > 0, \\ 0 & \text{for } y < 0. \end{cases}$$

We observe five independent values  $\mathbf{y}_0 = \{y_1, y_2, y_3, y_4, y_5\}$  of random variable  $y$ .  $x$  is an unknown constant. Compute the MLE  $\hat{x}_{\text{MLE}}(\{y_1, y_2, y_3, y_4, y_5\})$  of  $x$ .

**9.4** An exponential random variable  $y$  has pdf

$$p(y|x) = \begin{cases} xe^{-xy} & \text{for } y > 0, \\ 0 & \text{for } y < 0. \end{cases}$$

(a) If  $x$  is an unknown constant, compute the MLE estimate  $\hat{x}_{\text{MLE}}(y_0)$ .

(b) If  $x$  is a random variable with *a priori* pdf

$$p(x) = \begin{cases} be^{-bx} & \text{for } x > 0, \\ 0 & \text{for } x < 0, \end{cases}$$

where  $b$  is a known constant, compute the MAP estimate  $\hat{x}_{\text{MAP}}(y_0)$ .

(c) Explain the behavior of  $\hat{x}_{\text{MAP}}(y_0)$  when  $b \rightarrow \infty$  and when  $b \rightarrow 0$ .

**9.5** An exponential random variable  $y$  has pdf

$$p(y|x) = \begin{cases} xe^{-xy} & \text{for } y > 0, \\ 0 & \text{for } y < 0. \end{cases}$$

$x$  is a random variable with *a priori* pdf

$$p(x) = \begin{cases} be^{-bx} & \text{for } x > 0, \\ 0 & \text{for } x < 0, \end{cases}$$

where  $b$  is a known constant. Compute the least-squares estimate  $\hat{x}_{\text{LS}}(y_0)$ .

### Section 9-4: Least-Squares Estimation

**9.6**  $y(t)$  is a zero-mean WSS Gaussian random process with autocorrelation function  $R_y(\tau) = e^{-|\tau|}$ .

- (a) Let  $\mathbf{y} = \{y(1), y(2), y(3)\}$ . Determine the joint pdf  $p(\mathbf{y})$ .
- (b) Compute the least-squares estimate  $\hat{y}(3)_{\text{LS}}(y(2) = 6)$ .
- (c) Compute the least-squares estimate

$$\hat{y}(3)_{\text{LS}}(y(2) = 6, y(1) = 4).$$

**9.7**  $x(t)$  is a zero-mean WSS white Gaussian random process with autocorrelation  $R_x(\tau) = 4\delta(\tau)$ .  $x(t)$  is input into an LTI system with impulse response

$$h(t) = \begin{cases} 3e^{-2t} & \text{for } t > 0, \\ 0 & \text{for } t < 0. \end{cases}$$

- (a) Compute the autocorrelation  $R_y(\tau)$  of the output random process  $y(t)$ .
- (b) Let  $\mathbf{y} = \{y(3), y(7), y(9)\}$ . Determine the joint pdf  $p(\mathbf{y})$ .
- (c) Compute the least-squares estimate  $\hat{y}(7)_{\text{LS}}(y(5) = 6)$ .

(d) Compute the least-squares estimate

$$\hat{y}(7)_{\text{LS}}(y(5) = 6, y(3) = 4).$$

**9.8**  $x[n]$  is a zero-mean non-WSS random process with autocorrelation  $R_x[i, j] = \min[i, j]$ . Let  $i > j > k$ . Show that  $\hat{x}[i]_{\text{LS}}(x[j], x[k]) = \hat{x}[i]_{\text{LS}}(x[j])$ , so  $x[k]$  is irrelevant.

**9.9**  $x[n]$  is an IID Gaussian random process with

$$x[n] \sim \mathcal{N}(m, s),$$

where  $m = E[x[n]]$  and  $s = \sigma_{x[n]}^2$ . We are given observations

$$\mathbf{x}_0 = \{x_0[1], x_0[2], \dots, x_0[N]\}$$

of  $\{x[1], x[2], \dots, x[N]\}$ . The goal is to compute the MLE estimates  $\hat{m}(\mathbf{x}_0)$  and  $\hat{s}(\mathbf{x}_0)$  of mean  $m$  and variance  $s$ .

## Section 9-7: Spectral Estimation

**9.10** Section 9-7 showed discrete-space fractal images have power spectral densities

$$\mathbf{S}_f(\Omega_1, \Omega_2) = \frac{C}{(\Omega_1^2 + \Omega_2^2)},$$

$$0 < \Omega_{\min} < |\Omega_1|, \quad |\Omega_2| < \Omega_{\max} < \pi,$$

for some  $\Omega_{\min}$  and  $\Omega_{\max}$ . This problem suggests many real-world images also have such power spectral densities.

The following program uses a periodogram to estimate the power spectral density of the image in `????.mat` and fits a line to the log-log plot of  $S_f(\Omega_1, 0)$  versus  $\Omega_1$  for  $\Omega_{\min} = 0.1\pi$  and  $\Omega_{\max} = 0.9\pi$ . [Here “????” refers to the .mat files listed below.]

```
clear; load ????.mat; M=size(X, 1);
K=round(M/20);
FX=abs(fft2(X))/M; FX=log10(FX.*FX);
Omegal=log10(2*pi*[K:M/2-K]/M);
P=polyfit(Omegal, FX(1,K:M/2-K), 1);
Y=polyval(P, Omegal);
subplot(211), plot(Omegal, FX(1,K:M/2-K),
Omegal, Y, 'r'), axis tight, grid on
```

Run this program for the images contained in the following .mat files: (a) `clown.mat`; (b) `letters.mat`; (c) `sar.mat`; (d) `mri.mat`. What do these plots suggest about the power spectral densities of the images?

## Section 9-6: 2-D Estimation Problems

Deblurring due to an out-of-focus camera can be modeled crudely as 2-D convolution with a disk-shaped PSF

$$h[n, m] = \begin{cases} 1 & \text{for } m^2 + n^2 < R^2, \\ 0 & \text{for } m^2 + n^2 > R^2, \end{cases}$$

for some radius of  $R$  pixels. The program `srefocus.m` convolves with  $h[n, m]$  the image  $f[n, m]$  in the file `????.mat`, adds zero-mean white Gaussian noise with variance  $\sigma_v^2$  to the blurred image, and deconvolves the blurred image using each of the following two Wiener filters:

- the deterministic Wiener filter uses power spectral density  $S_f(\Omega_1, \Omega_2) = C$ ;
- the stochastic Wiener filter uses power spectral density  $S_f(\Omega_1, \Omega_2) = C/(\Omega_1^2 + \Omega_2^2)$ .

Both filters depend only on the reciprocal signal-to-noise-type ratio  $\sigma_v^2/C$ . In practice, neither  $C$  nor  $\sigma_v^2$  is known, so different values of  $\sigma_v^2/C$  would be tried. Use  $\sigma_v^2/C = 1$  here. The program `srefocus.m` will be used in the next eight problems. [Here “????” refers to the .mat files used in Problems 9.11 through 9.18.]

**9.11** Edit program `srefocus.m` to deconvolve the image in `clown.mat`. Use  $\sigma_v^2/C = 1$ .

**9.12** Edit program `srefocus.m` to deconvolve the image in `letters.mat`.  $\sigma_v^2/C = 1$ .

**9.13** Edit program `srefocus.m` to deconvolve the image in `xray.mat`. Use  $\sigma_v^2/C = 1$ .

**9.14** Edit program `srefocus.m` to deconvolve the image in `mri.mat`. Use  $\sigma_v^2/C = 1$ . Change the power spectral density to  $S_f(\Omega_1, \Omega_2) = C/((\Omega_1^2 + \Omega_2^2))^2$  (uncomment a line).

**9.15** Edit program `srefocus.m` so it merely denoises the noisy clown image. Use  $\sigma_v^2/C = 1$  and  $h[n, m] = \delta[n, m]$  (to make a denoising, not a deconvolution problem) and

- the deterministic Wiener filter uses the power spectral density  $S_f(\Omega_1, \Omega_2) = C$ ;
- the stochastic Wiener filter uses the power spectral density  $S_f(\Omega_1, \Omega_2) = C/(\Omega_1^2 + \Omega_2^2)$ .

**9.16** Edit program `srefocus.m` so it merely denoises the noisy letters image. Use  $\sigma_v^2/C = 1$  and  $h[n, m] = \delta[n, m]$  (to make a denoising, not a deconvolution problem) and

- the deterministic Wiener filter uses the power spectral density  $S_f(\Omega_1, \Omega_2) = C$ ;
- the stochastic Wiener filter uses the power spectral density  $S_f(\Omega_1, \Omega_2) = C / (\Omega_1^2 + \Omega_2^2)$ .

**9.17** Edit program `srefocus.m` so it merely denoises the noisy XRAY image. Use  $\sigma_v^2/C = 1$  and  $h[n,m] = \delta[n,m]$  (to make a denoising, not a deconvolution problem) and

- the deterministic Wiener filter uses power spectral density  $S_f(\Omega_1, \Omega_2) = C$ ;
- the stochastic Wiener filter uses power spectral density  $S_f(\Omega_1, \Omega_2) = C / (\Omega_1^2 + \Omega_2^2)$ .

**9.18** Edit program `srefocus.m` so it merely denoises the noisy MRI image. Use  $\sigma_v^2/C = 1$  and  $h[n,m] = \delta[n,m]$  (to make a denoising, not a deconvolution problem) and

- the deterministic Wiener filter uses the power spectral density  $S_f(\Omega_1, \Omega_2) = C$ ;
- the stochastic Wiener filter uses the power spectral density

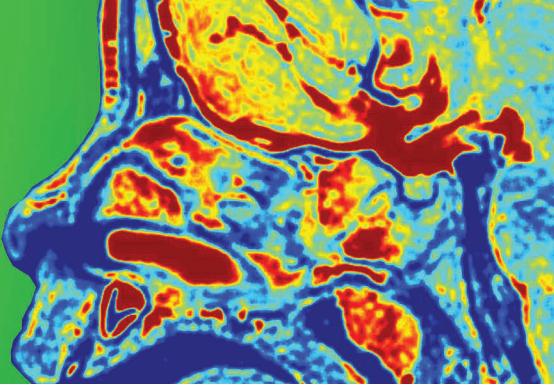
$$S_f(\Omega_1, \Omega_2) = \frac{C}{(\Omega_1^2 + \Omega_2^2)^2}.$$

## Section 9-10: Markov Random Fields

**9.19** Download the image in `field.mat` and the program `icm.m`.

- Display the image using  
`imagesc(X), colormap(gray).`
- Display its histogram for 200 bins using  
`hist(X(:, ), 200).`  
Eight pixel values occur more often than nearby values.  
Identify them. *Hint:* They are in arithmetic progression.
- Change the line `f = []` in `icm.m` to contain these values.
- Run `icm.m`. It uses 9 iterations,  $\sigma = 30$ , and  $\beta = 1$ .  
(Varying these values seems to make little difference to the final result.) Display the maximum-likelihood segmented and ICM-algorithm segmented images.

# CHAPTER 10



## 10

## Color Image Processing

### Contents

- Overview, 335
- 10-1** Color Systems, 335
- 10-2** Histogram Equalization and Edge Detection, 340
- 10-3** Color-Image Deblurring, 343
- 10-4** Denoising Color Images, 346
- Problems, 351

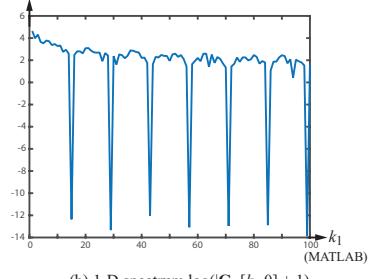
### Objectives

Learn to:

- Transform between the RGB and YIQ color systems.
- Apply image enhancement (histogram equalization and edge detection) to color images.
- Apply image restoration (deblurring and denoising) to color images.



(a) Blurred Christmas tree



(b) 1-D spectrum  $\log(|G_R[k_1, 0]| + 1)$



(c) Reconstructed Christmas tree

This chapter extends the image processing techniques developed in previous chapters for grayscale images to color images, which can be represented in either the RGB color system (suitable for image deblurring) or the YIQ color system (suitable for enhancement, so that colors are not distorted). Examples illustrate all of this.

## Overview

So far in this book, we have focused on the processing of **grayscale** (black-and-white) images, where the intensity of the image is described by a single 2-D function  $f(x,y)$  of continuous spatial variables  $(x,y)$  or  $f[n,m]$  of discrete spatial variables  $[n,m]$ . We considered several types of image processing applications including image denoising, deblurring, and segmentation. Now in this chapter, we consider **color** images and how to process them. It is important to note that by “color images” we mean **true-color** images, in which different colors are actually present in the image, not **false-color** images, in which the grayscale intensity of a black-and-white image is divided into ranges and then assigned to different colors. We use discrete-space images  $f[n,m]$ , although our treatment is equally applicable to continuous-space images  $f(x,y)$ .

A color image consists of three channels or components, which taken together contain the information present in the image. The actual content of the three channels depends on the **color system** used for the image. For example the **RGB** color system represents a color  $f[n,m]$  image as a triplet of **red**  $f_R[n,m]$ , **green**  $f_G[n,m]$ , and **blue**  $f_B[n,m]$  images:

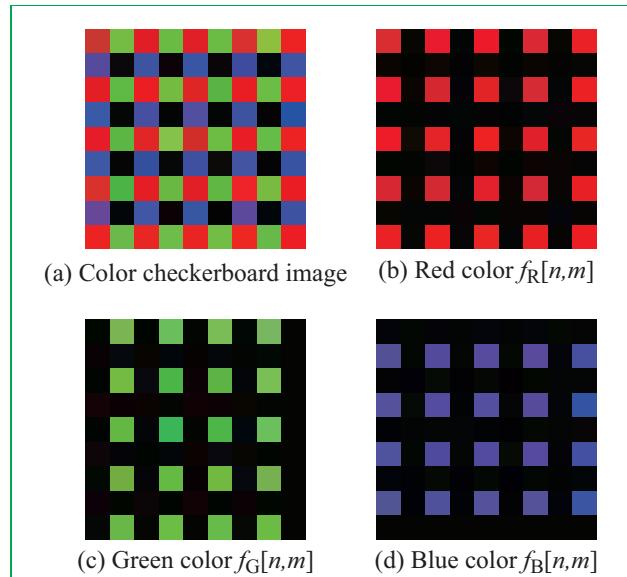
$$f[n,m] = \{f_R[n,m], f_G[n,m], f_B[n,m]\}. \quad (10.1)$$

It is important to note that it is *not* true that  $f[n,m] = f_R[n,m] + f_G[n,m] + f_B[n,m]$ —the three components are akin to elements of a vector. Instead, the color image consists of a simultaneous display of the 2-D functions  $f_R[n,m]$ ,  $f_G[n,m]$  and  $f_B[n,m]$ , as shown in **Fig. 10-1**. The color generation schemes used in displays and printers are discussed in Section 10-1.

Other color systems, specifically **CYK** (cyan, magenta, yellow, black) and **YIQ** (luminance, in-phase, quadrature), are presented in Section 10-1. Image enhancement (histogram equalization and edge detection) of color images is presented in Section 10-2, which uses YIQ, because use of RGB can result in distortion of the colors of  $f[n,m]$ . Deconvolution (deblurring) of color images is presented in Section 10-3, and denoising of color images is covered in Section 10-4. Both use RGB.

## 10-1 Color Systems

A color system is a representation of a color image using three channels or components. This section presents an overview of the three color systems most commonly used in displays and color printers.



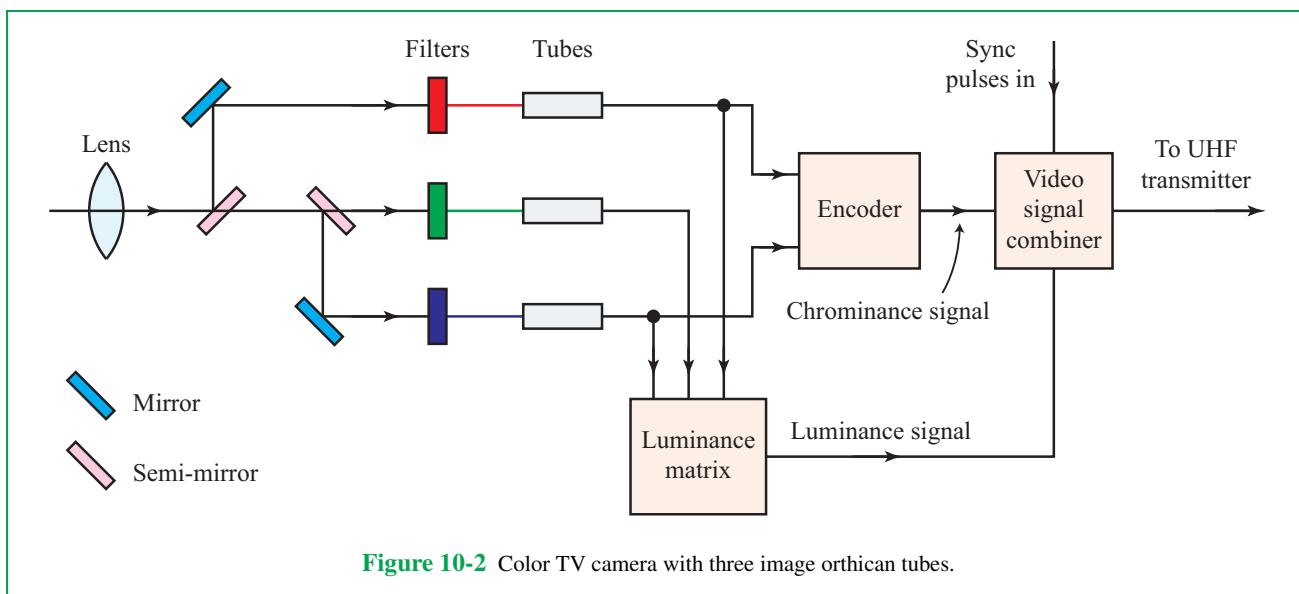
**Figure 10-1** The three-color checkerboard image comprises three single-color images, each comprising one-fourth of the total pixels of the checkerboard (because one-fourth of the pixels are black). In (b),  $f_R[n,m]$  is non-zero in only the non-black pixels, and a similar arrangement applies to  $f_G[n,m]$  and  $f_B[n,m]$ .

### 10-1.1 RGB (Additive) Color System

#### A. Acquiring Color Images

Color images are acquired using three sensors: red, green, and blue. Film images in color photography were acquired by splitting the image into three copies, and then filtering them using red, green, and blue filters to record red, green, and blue components of the image. TV images when TV color cameras were used were also acquired by splitting the image into three copies, and then filtering them using red, green, and blue filters before inputting them into three separate sensors (**Fig. 10-2**). More recently, three sets of **charge-coupled device (CCD)** sensors with red, green, and blue filters in front of them are used to acquire the red, green, and blue components of an image.

Hence, image processing of color images consists of processing three RGB components. Deconvolution of color images requires deconvolution of each RGB component separately, using



identical procedures (2-D DFT and Wiener filter) for each RGB component. Image enhancement, in particular edge detection and histogram equalization, is applied to the YIQ representation, rather than the RGB representation (the two representations are related by the linear transformation Eq. (10.4), in order to ensure that different colors are not enhanced differently. Denoising can use either RGB or YIQ representation. All of these tools are illustrated in the remainder of this chapter.

To acquire a color image, most digital cameras and scanners acquire red, green, and blue images separately. Scanners include a  $(M \times 3)$  array of CCD sensors with an  $(M \times 3)$  array of color filters in front of them, where  $M$  is the width of the scanner in pixels. One row of filters is red, another row of filters is green, and another row of filters is blue (see Fig. 1-6). The image is scanned line by line, and the  $m$ th line generates  $f_R[n, m]$ ,  $f_G[n, m]$  and  $f_B[n, m]$ .

## B. Displaying Color Images

As noted above, color photography was developed by filtering the lens image using red, green, and blue filters, and then developing in a photo lab three separate images in these colors. Color television with cathode-ray tubes (CRTs) uses three electron guns to “paint” a phosphor screen with red, green, and blue images. A mask behind the screen separates the outputs of the guns, so that the “red” gun illuminates phosphors that glow red

when irradiated with electrons, and similarly for green and blue. The phosphors are in groups of three, called pixels, one for each color.

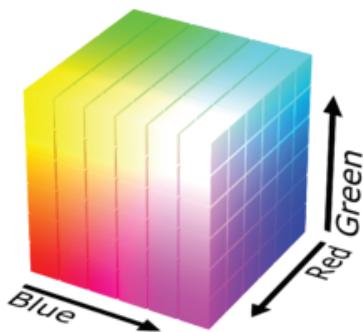
Color televisions and computer monitors with LED displays use three sets of LEDs to depict the red, green, and blue images. Each pixel in the display consists of three closely spaced LEDs, one for each color.

## C. RGB

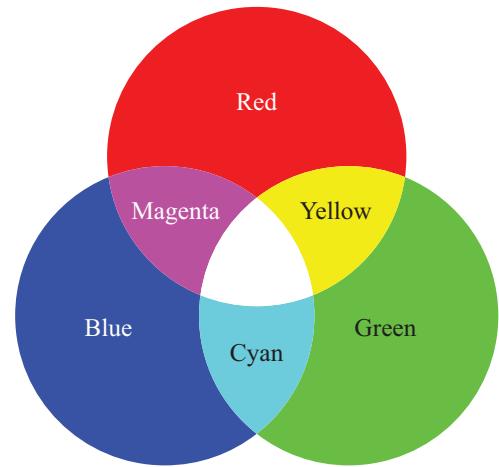
The RGB color system represents color images using Eq. (10.1). It is called an **additive** color system because non-RGB colors are created by taking a linear combination of  $\{f_R[n, m], f_G[n, m], f_B[n, m]\}$ . The additive process is illustrated by the **color cube** shown in Fig. 10-3. The coordinates are the weights applied to  $\{f_R[n, m], f_G[n, m], f_B[n, m]\}$  to produce the color shown in the cube. White, which has coordinates  $(1, 1, 1)$ , is obtained by combining red, green, and blue in equal strengths. The non-RGB colors are discussed in Subsection 10-1.2 where we discuss the CMYK color scheme.

Since TV screens and monitors are dark when they are not displaying an image, it is natural to create color images on them by displaying weighted sums (i.e., linear combinations) of  $\{f_R[n, m], f_G[n, m], f_B[n, m]\}$ . The lighting used in a dark theater for a play is another example of an additive color system.

Web browsers display web pages using an additive



**Figure 10-3** Color cube: various colors are linear combinations of red, green, and blue.



**Figure 10-4** Relation of CMY colors to RGB colors.

color scheme. The HTML command `<body bgcolor='`#RrGgBb'`>` changes the color of the background of a web page to a linear combination of red, green, and blue, where the weights are the hexadecimal (base-16) numbers  $(Rr)_{16}$  for red,  $(Gg)_{16}$  for green, and  $(Bb)_{16}$  for blue, with  $(f)_{16} = 15$ . So `<body bgcolor='`#fffff00'`>` changes the background color to yellow (red + green), since  $(Rr)_{16} = (Gg)_{16} = (ff)_{16}$  turns the red and green “lights” to their maximum values  $(ff)_{16} = 255$ . Each hexadecimal weight has  $16^2 = 256$  values so the number of possible colors is  $(256)^3 = 16,777,216$ . Still, not every conceivable color can be represented as a linear combination of the colors in  $\{f_R[n, m], f_G[n, m], f_B[n, m]\}$ .

So why is it that systems other than RGB are used?

## 10-1.2 CMYK (Subtractive) Color System

The C (**cyan**), M (**magenta**), Y (**yellow**), K (**black**) color system is used for printing color images on a printer. A CMYK color system was used to print drafts of this book, for example, on a color-laser printer. The toner for laser printers and the inks for color inkjet printers are in CMYK colors.

The colors cyan, magenta, and yellow are generated from red, green, and blue, as shown in **Fig. 10-4**. Yellow is the sum of red and green, or equivalently, yellow is white minus blue.

The CMY colors are also specified as the off-axis corners of the color cube of **Fig. 10-3**. In fact, on the color cube, cyan, magenta, and yellow are the complementary colors of red, green, and blue, respectively. For example, cyan

has the coordinates  $(0, 1, 1)$  on the color cube and red has coordinates  $(1, 0, 0)$ . The sum of all three additive colors is  $(1, 0, 0) + (0, 1, 0) + (0, 0, 1) = (1, 1, 1)$ , which is the coordinate for white. Similarly, yellow has coordinates  $(1, 1, 0)$  and blue has coordinates  $(0, 0, 1)$ . This is why the CMYK color system is called **subtractive**—it subtracts colors from white, instead of adding them to black the way RGB does.

## 10-1.3 Color Printing

### A. RGB Printers

Color printers do not use the RGB color system to print text and images, but let us pretend for the time being that they do. When the image of a **perfectly red** tomato is printed on white paper using an RGB inkjet printer, the printer deposits red ink onto the paper. A similar process occurs with the cartridge of a laser printer except that the material deposited into the paper is a fine red powder and its extent across the paper is defined by a magnetically charged toner. The printed tomato appears red because when illuminated by white light, the green and blue components of the white light are absorbed by the red ink or red powder, leaving only the red component of the incident light to be reflected by the printed image of the tomato. Hence, the **perfectly red** tomato appears red. However, if the tomato is yellow in color, which is equivalent to the sum of red and green in the RGB system (**Fig. 10-4**), an RGB inkjet printer would

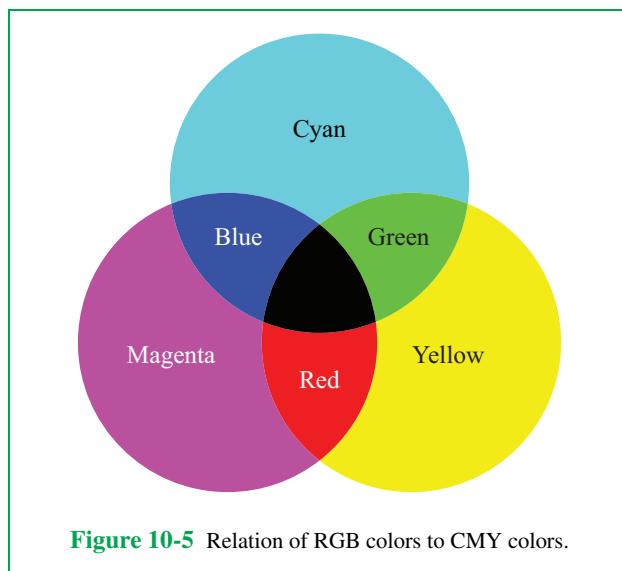
have to deposit both red and green ink onto the paper. When illuminated by white light, the red ink will absorb the green and blue components of the light and the green ink will absorb the red and blue components, leaving no light to be reflected by the printed page. Hence, the yellow tomato will appear black instead of yellow. Because the ink used in inkjet printers and the powder used in laser toners operate by subtracting their colors from white, they are more compatible with the subtractive CMYK color system than with the additive RGB system.

## B. CMYK Printers

In the subtractive CMYK color system (Fig. 10-5), red can be created by a mixture of yellow and magenta inks, green by a mixture of yellow and cyan inks, and so on. Using CMY ink colors solves the problem we encountered earlier with the yellow tomato. We can now print many colors, whether pure or not. Obviously, a yellow tomato can be printed using yellow ink, but what about a perfectly red tomato? Since yellow consists of red and green and magenta consists of red and blue, we can use the combination of yellow and magenta inks to print it since both colors include red. When illuminated by white light, the yellow ink absorbs blue and the magenta ink absorbs green, leaving only the red component of the white light to be reflected.

In CMYK, a color image is depicted as:

$$f[n,m] = \{f_C[n,m], f_M[n,m], f_Y[n,m], f_K[n,m]\}. \quad (10.2)$$



Black (K) is included in CMYK because while Fig. 10-4 shows that cyan + magenta + yellow = black, such an addition of inks requires much ink to produce black, which is a common color in images. Hence, black is included as a separate color in printing. Because CMYK consists of **three subtractive colors**—yellow, cyan, and magenta—and black, it is often referred to as a **four-color printing process**. In CMYK, silver is represented using the weights {0, 0, 0, 0.25}, and gold by {0, 0, 0.88, 0.15}.

The exact relation between RGB and CMYK color systems is more complicated than we have presented, since the inks used in printing are not exactly cyan-, magenta-, and yellow-colored.

We should note that CMYK is not used in image processing, only for image printing.

## 10-1.4 YIQ Color System

The Y (**luminance**), I (**In phase**), Q (**Quadrature**) color system was developed for television by the NTSC (National Television System Committee) in 1953. It differs from the RGB and CMYK color systems in that intensity is decoupled from color. The reason for decoupling intensity from color was so that images transmitted in color could easily be displayed on black-and-white televisions, which were still in common use throughout the 1950s. This feature also makes the YIQ color system useful for histogram equalization and edge detection because image intensity (luminance) can be varied without distorting colors, as we show later in Section 10-2.

The YIQ depiction of a color image  $f[n,m]$  is given by

$$f[n,m] = \{f_Y[n,m], f_I[n,m], f_Q[n,m]\}. \quad (10.3)$$

Even though the symbol  $f_Y[n,m]$  is used in both the YIQ and CMYK color systems, this is the standard notation used in color schemes. The meaning of  $f_Y[n,m]$  (yellow or luminance) should be obvious from the context.

The **luminance** component  $f_Y[n,m]$  is the intensity of  $f[n,m]$  at location  $[n,m]$ . The **chrominance** (color) components  $\{f_I[n,m], f_Q[n,m]\}$  depict the color of  $f[n,m]$  at location  $[n,m]$  using Fig. 10-6.

The horizontal axis (blue to orange) ranges over the colors to which the eye is most sensitive. The vertical axis (purple to green) ranges over colors to which the eye is less sensitive. Accordingly, for analog transmission of television signals, less bandwidth is needed to transmit the quadrature-modulated signal  $f_Q[n,m]$  than the in-phase (un-modulated) signal  $f_I[n,m]$ , with most of the transmission bandwidth reserved for the luminance signal  $f_Y[n,m]$ .

The chrominance components  $\{f_I[n,m], f_Q[n,m]\}$  can be regarded as rectangular color coordinates. In polar coordinates,

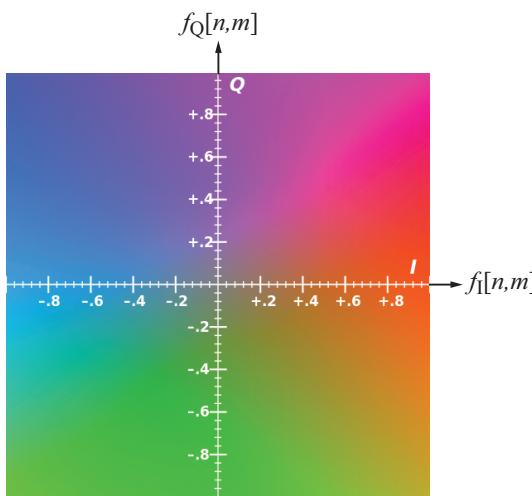


Figure 10-6 YIQ chrominance components.

these would be **hue** (tint) for the angular coordinate and **saturation** (vividness) for the radial coordinate. These are used in the **HSI** (Hue, Saturation, Intensity) color system noted below.

### A. Relating the RGB and YIQ Color Systems

The RGB (Eq. (10.1)) and YIQ (Eq. (10.3)) color system representations of a color image  $f[n, m]$  are related by the linear transformation

$$\begin{bmatrix} f_Y[n, m] \\ f_I[n, m] \\ f_Q[n, m] \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} f_R[n, m] \\ f_G[n, m] \\ f_B[n, m] \end{bmatrix}. \quad (10.4)$$

Note that the first row of this matrix sums to one, and each of the second and third rows sums to zero. This ensures that: (a)  $f_Y[n, m] \leq 1$  if  $f_R[n, m] \leq 1$ ,  $f_G[n, m] \leq 1$ , and  $f_B[n, m] \leq 1$ , and (b) the two color components are zero if all three colors are present in equal strengths, corresponding to a white image at location  $[n, m]$ , in which case  $f_Y[n, m]$  alone suffices to represent the image at  $[n, m]$ .

The first row of Eq. (10.4) is a formula for transforming the color image  $f[n, m]$  depicted using Eq. (10.1) to a grayscale

image. The formula for this is

$$f_{\text{gray}}[n, m] = 0.299f_R[n, m] + 0.587f_G[n, m] + 0.114f_B[n, m]. \quad (10.5)$$

The luminance image  $f_Y[n, m]$  is the image displayed on black-and-white televisions and monitors.

### B. Significance of the YIQ Color System

Of course, black-and-white monitors, let alone televisions, are rare today. The significance of the YIQ color system is that it decouples color from intensity, so we can lighten an image by applying histogram equalization to  $f_Y[n, m]$ , while leaving  $f_I[n, m]$  and  $f_Q[n, m]$  unaltered. Applying histogram equalization to each of  $f_R[n, m]$ ,  $f_G[n, m]$  and  $f_B[n, m]$  can alter the colors of the image (see Section 10-2). There is also the computational savings of applying image enhancement to a single image instead of to three images. Similar comments apply to edge detection in color images.

### 10-1.5 Other Color Systems

There are many other color systems, such as **HSI** (Hue, Saturation, Intensity) and **HSV** (Hue, Saturation, Value). For example, the HSI color system is related to the RGB system using

$$f_I[n, m] = \frac{1}{3}(f_R[n, m] + f_G[n, m] + f_B[n, m]), \quad (10.6a)$$

$$f_S[n, m] = 1 - \frac{3 \min[f_R[n, m], f_G[n, m], f_B[n, m]]}{f_R[n, m] + f_G[n, m] + f_B[n, m]}, \quad (10.6b)$$

$$f_H[n, m] = \cos^{-1} \left[ \frac{\frac{1}{2}((f_R - f_G) + (f_R - f_B))}{(f_R - f_G)^2 + (f_R - f_B)(f_G - f_B)} \right]. \quad (10.6c)$$

We do not discuss or use these color schemes in this book, nor do we discuss **color science**, which includes topics such as **photometry** (image color spectra) and **colorimetry** (color matching). These topics cover issues such as nonlinear response of sensors (including the eye), perception of colors by the eye and brain (psychophysics of color), imperfect ink colors for printing, and nonlinear response of inkjet printers (which print dots of different colors, just as monitors and televisions represent color images using clusters of dots of different colors). Gamma correction is an example of a nonlinear correction that arises in this context. For a good treatment of color science in image processing, see H. J. Trussel and M. J. Vrhel, *Fundamentals of Digital Imaging*, Cambridge University Press, 2008.

► In this book we will transform from the RGB color system to the YIQ color system using Eq. (10.4), then perform image enhancement using histogram equalization on  $f_Y[n, m]$ , and convert the result back to the RGB color system using the inverse of Eq. (10.4). ◀

The result can then be converted to another color system, such as HSI, if desired.

### 10-1.6 Reading Images to MATLAB

The MATLAB command `imread` is used to read images into 3-D MATLAB arrays. For example, `A=imread('f.jpg');` maps the  $M \times N$  JPEG image `f.jpg` into the  $M \times N \times 3$  3-D MATLAB array `A`, in which `A(:, :, 1)` is the  $M \times N$  array of  $f_R[m, n]$ , `A(:, :, 2)` is the  $M \times N$  array of  $f_G[m, n]$ , and `A(:, :, 3)` is the  $M \times N$  array of  $f_B[m, n]$ . For the checkerboard image in [Fig. 10-1\(a\)](#), `A(:, :, 1)` is shown in [Fig. 10-1\(b\)](#), `A(:, :, 2)` in [Fig. 10-1\(c\)](#), and `A(:, :, 3)` in [Fig. 10-1\(d\)](#). Many types of images, such as `.jpeg`, `.tiff` and `.bmp` can be handled using `imread`. `A` will be in `uint8` format, which should be converted into double-precision floating-point using `A=double(A);`. The values `A[i, j]` must be scaled to satisfy  $0 \leq A[i, j] \leq 1$ , using Eq. (5.1).

`imagesc(A)` depicts the  $M \times N \times 3$  3-D MATLAB array `A` as a color image, as in [Fig. 10-1](#).

**Concept Question 10-1:** Why do we use the YIQ color system in image processing? Why not just use the RGB color system exclusively?

**Exercise 10-1:** What should the output of `imagesc(ones(3, 3, 3))` be?

**Answer:** A  $3 \times 3$  all-white image (the combination of  $3 \times 3$  red, green, and blue images).

**Exercise 10-2:** An image has

$$f_R[n, m] = f_G[n, m] = f_B[n, m] = 1$$

for all  $[n, m]$ . What is its YIQ color system representation?

**Answer:** From Eq. (10.4),  $f_Y[n, m] = 1$  and

$$f_I[n, m] = f_Q[n, m] = 0$$

for all  $[n, m]$ , as discussed below Eq. (10.4).

## 10-2 Histogram Equalization and Edge Detection

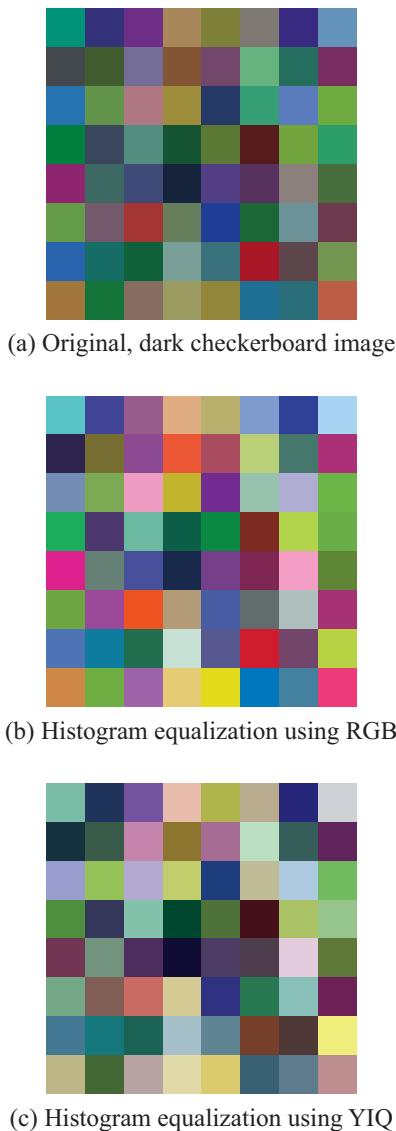
### 10-2.1 Histogram Equalization

Histogram equalization (HE) was covered in Chapter 5. It alters the pixel values of an image  $f[n, m]$  so that they are evenly distributed over the entire range of the output display, thereby brightening a dim image. In principle, histogram equalization can be applied to each of the three RGB components of the color image represented by Eq. (10.1). The drawback of such an approach is that since the colors of the image are mixtures of the red, green, and blue components of the image, disproportionate altering of the pixel values will alter the colors of the image. The image will appear brighter, but the colors may get distorted.

To avoid the color distortion, the RGB components can be mapped to YIQ components using Eq. (10.4), and then histogram equalization can be applied to only the luminance (brightness) component  $f_Y[n, m]$ . The process brightens the image but leaves the colors unaltered because the color of each pixel is determined by the two chrominance components  $f_I[n, m]$  and  $f_Q[n, m]$ , whose values are not altered. An inverse form of Eq. (10.4) is then used to transform the YIQ representation back to the RGB representation. This approach also has the computational advantage in that histogram equalization is applied to a single component,  $f_Y[n, m]$ , instead of to all three RGB components.

To compare the result of histogram equalization when applied (a) directly to an RGB image with (b) the result obtained by converting the RGB image to a YIQ image, followed by histogram equalization applied to the Y component and then converting the YIQ image back to RGB, we consider the dark checkerboard RGB image shown in [Fig. 10-7\(a\)](#).

Application of the histogram equalization method to the three RGB channels separately leads to the image in [Fig. 10-7\(b\)](#), whereas the application of histogram equalization to the Y channel leads to the image in [Fig. 10-7\(c\)](#). Both approaches generate brighter versions of the checkerboard image, but only the Y histogram equalization preserves the true colors of the original image.



**Figure 10-7** Application of histogram equalization to the dark checkerboard image in (a) results in the image in (b) when the equalization is applied to all three RGB channels and to the image in (c) when applied to the Y channel of the YIQ image.

### Example 10-1: Flag Image in RGB and YIQ Equalization

Use the image in [Fig. 10-8\(a\)](#) to compare the results of two histogram equalization experiments, one performed using RGB and another using YIQ.

**Solution:** Application of histogram equalization to each of the three RGB images separately, and then followed by combining the equalized images into a single color image results in the brighter, but color-distorted, image shown in [Fig. 10-8\(b\)](#). In contrast, application of histogram equalization to the Y channel of the YIQ image ([Fig. 10-8\(c\)](#)) preserves the color balance of the original image. Note the color of the field of stars.

## 10-2.2 Edge Detection in Color Images

The subject of edge detection as applied to grayscale images was covered in Section 5-4.2, wherein we demonstrated how the **Sobel edge detector** is used to convolve the grayscale image  $f[n, m]$  with two PSFs,  $h_H[n, m]$  and  $h_V[n, m]$ , to obtain the horizontal edge image  $d_H[n, m]$  and the vertical edge image  $d_V[n, m]$ :

$$d_H[n, m] = f[n, m] * * h_H[n, m] \quad (10.7a)$$

and

$$d_V[n, m] = f[n, m] * * h_V[n, m]. \quad (10.7b)$$

The two edge images are then combined to generate the gradient image  $g[n, m]$ :

$$g[n, m] = \sqrt{d_H[n, m]^2 + d_V[n, m]^2}. \quad (10.7c)$$

The Sobel edge-detector algorithm identifies a pixel  $[n, m]$  as an edge if  $g[n, m]$  exceeds a pre-specified threshold  $\Delta$ . Accordingly, the **edge image**  $z[n, m]$  is given by

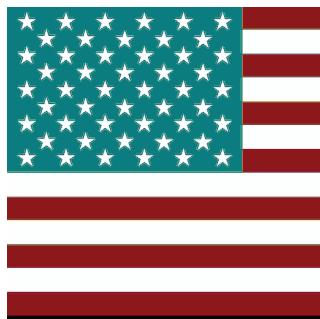
$$z[n, m] = \begin{cases} 1 & \text{for } g[n, m] > \Delta, \\ 0 & \text{for } g[n, m] < \Delta. \end{cases} \quad (10.8)$$

The value of  $\Delta$  is application-dependent.

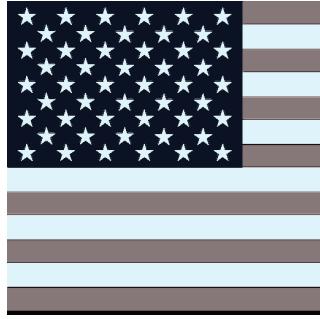
For color images, we can apply edge detection to each of  $f_R[n, m]$ ,  $f_G[n, m]$  and  $f_B[n, m]$  in an RGB image, or just to  $f_Y[n, m]$  in its YIQ equivalent. Aside from the computational savings of detecting edges in only one image instead of three, another reason for using just  $f_Y[n, m]$  is illustrated by the following example.



(a) Original dark flag image



(b) After histogram equalization of RGB image

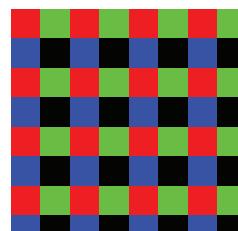


(c) After histogram equalization of YIQ image and conversion back to RGB format

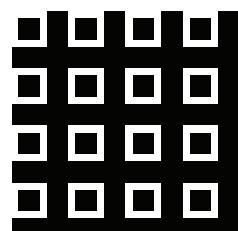
**Figure 10-8** The application of histogram equalization to each of the three channels of the RGB image results in color distortion, but that is not the case when applied to a YIQ image.

### Edge detection of checkerboard image

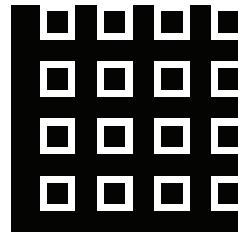
The  $(24 \times 24)$  checkerboard image in **Fig. 10-9(a)** is in RGB. Application of the Sobel edge detector algorithm with threshold  $\Delta = 0.1$  to each color channel separately leads to the images in parts (b) through (d) of the figure. The three colors generate different edge images. In contrast, **Fig. 10-9(e)** displays the edge image using the YIQ image, which correctly identifies all edges regardless of the pixel colors.



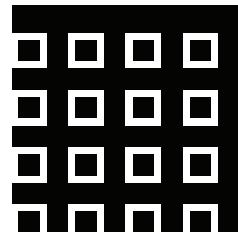
(a) Original checkerboard RGB image



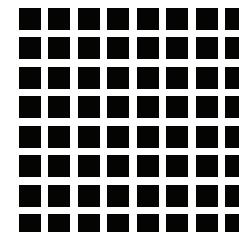
(b) Red color edge detection



(c) Green color edge detection



(d) Blue color edge detection



(e) Edge detection using YIQ

**Figure 10-9** Sobel edge detection: (a) original color image, (b)–(d): edge detection of each RGB color channel separately, and (e) edge detection using Y channel of YIQ image.

**Concept Question 10-2:** Why does applying histogram equalization using the RGB color system alter colors?

**Exercise 10-3:** Should gamma transformation be applied to images using the RGB or YIQ format?

**Answer:** YIQ format, because gamma transformation is nonlinear; applying it to an RGB image would distort its colors.

**Exercise 10-4:** Should linear transformation be applied to images using the RGB or YIQ color system?

**Answer:** RGB, since this is the color system used to display the transformed image. Applying linear transformation to YIQ images may result in  $g_R[n,m] \geq g_{\max}$ ,  $g_G[n,m] \geq g_{\max}$ , or  $g_B[n,m] \geq g_{\max}$  in Eq. (5.1).

## 10-3 Color-Image Deblurring

When a color scene/image is blurred as a result of convolution with the sensor's PSF  $h[n,m]$ , each of its RGB components is blurred. The blurred image consists of the three components

$$\begin{aligned} g_R[n,m] &= f_R[n,m] * * h[n,m], \\ g_G[n,m] &= f_G[n,m] * * h[n,m], \\ g_B[n,m] &= f_B[n,m] * * h[n,m]. \end{aligned} \quad (10.9)$$

For the YIQ image format, convolving Eq. (10.4) with  $h[n,m]$  leads to

$$\begin{bmatrix} g_Y[n,m] \\ g_I[n,m] \\ g_Q[n,m] \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} g_R[n,m] \\ g_G[n,m] \\ g_B[n,m] \end{bmatrix}. \quad (10.10)$$

Hence, all three of the YIQ components are also blurred, so there is no particular advantage to transforming the image from RGB to YIQ format. In practice deblurring (deconvolution) of a color image is performed separately on each of  $g_R[n,m]$ ,  $g_G[n,m]$  and  $g_B[n,m]$  to obtain the original image components  $f_R[n,m]$ ,  $f_G[n,m]$  and  $f_B[n,m]$ .

The deblurring process is illustrated through two examples.

### Example 10-2: Deblurring Checkerboard Image

Consider the  $8 \times 8$  color checkerboard image shown in Fig. 10-10(a), which we label  $f[n,m]$ , as it represents an original unblurred image of the checkerboard.

If the original scene was imaged by a sensor characterized by a circular PSF given by

$$h[n,m] = \begin{cases} 1 & \text{for } \sqrt{n^2 + m^2} \leq 4, \\ 0 & \text{for } \sqrt{n^2 + m^2} > 4. \end{cases} \quad (10.11)$$

(a) Generate the convolved image

$$g[n,m] = f[n,m] * * h[n,m], \quad (10.12)$$

and (b) apply deconvolution to reconstruct the original image.

**Solution:** (a) A 2-D display of the PSF, when centered at pixel  $(5,5)$ , is shown in Fig. 10-10(b). The PSF approximates a disk of radius 4. Included in the PSF are pixels  $[5,5]$ ,  $[5,5 \pm 4]$ , and  $[5 \pm 4, 5]$ . Hence, to include all its pixels, we approximate the PSF as a  $(9 \times 9)$  square. Application of the convolution operation given by Eq. (10.12) leads to the blurred image shown in Fig. 10-10(c). Per Section 5-2.5, if the size of image  $f[n,m]$  is  $(M \times M)$  and if  $h[n,m]$  is of size  $(L \times L)$ , then the size of the convolved image is  $(N \times N)$  with

$$N = M + L - 1 = 8 + 9 - 1 = 16.$$

Hence, the blurred image in Fig. 10-10(c) is  $(16 \times 16)$ .

(b) To deblur the image in Fig. 10-10(c) and recover the original image, we follow the recipe outlined in Section 6-4.1:

(1) Transform to the discrete frequency domain:

$$\mathbf{H}[k_1, k_2] = \text{DFT}\{[h[n,m]]\}, \quad (10.13a)$$

$$\mathbf{G}_R[k_1, k_2] = \text{DFT}\{[g_R[n,m]]\}, \quad (10.13b)$$

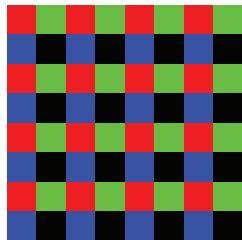
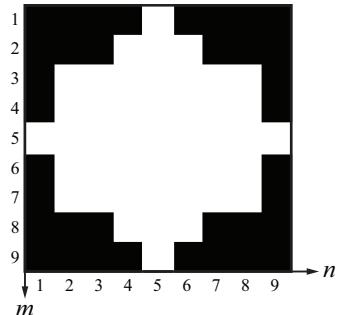
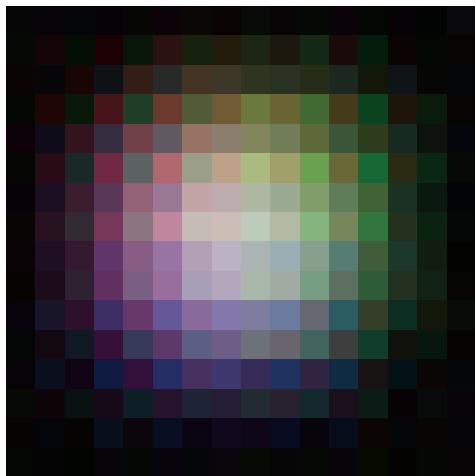
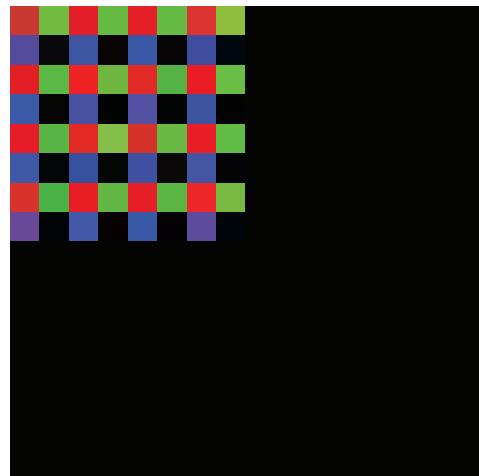
$$\mathbf{G}_G[k_1, k_2] = \text{DFT}\{[g_G[n,m]]\}, \quad (10.13c)$$

$$\mathbf{G}_B[k_1, k_2] = \text{DFT}\{[g_B[n,m]]\}.$$

(2) Since convolution in the spatial domain is equivalent to multiplication in the spatial frequency domain using  $(16 \times 16)$  2-D DFTs:

$$\mathbf{G}_R[k_1, k_2] = \mathbf{F}_R[k_1, k_2] \mathbf{H}[k_1, k_2], \quad (10.14)$$

and similar expressions apply to the green and blue images.

(a)  $(8 \times 8)$  original checkerboard RGB image(b) Extent of PSF when centered at pixel  $(5,5)$ (c)  $(16 \times 16)$  blurred checkerboard image(d)  $(16 \times 16)$  reconstructed checkerboard image

**Figure 10-10** Simulation of blurring and deblurring processes: (a) original image, (b) PSF  $h[n,m]$ , (c) original image blurred by system PSF, and (d) reconstructed image after removal of blurring.

Hence, for the red channel we use the computed results to obtain

$$\mathbf{F}_R[k_1, k_2] = \frac{\mathbf{G}_R[k_1, k_2]}{\mathbf{H}[k_1, k_2]}, \quad (10.15)$$

and similar expressions apply to the other two colors.

**(3)** Compute  $f_R[n, m]$  for all pixels:

$$f_R[n, m] = \text{DFT}^{-1}\{\mathbf{F}_R[k_1, k_2]\}, \quad (10.16)$$

and similar expressions apply to  $f_G[n, m]$  and  $f_B[n, m]$ . The de-

convolved image is shown in **Fig. 10-10(d)**. The blurring caused by the sensor has been completely removed. The reconstructed image is a  $(16 \times 16)$  zero-padded version of the original  $(8 \times 8)$  image.

**Concept Question 10-3:** Why didn't we need regularization in Example 10-2?

**Exercise 10-5:** Can the colors of a deconvolved image differ from those of the original image?

**Answer:** Yes. Regularization can alter the values of the reconstructed  $\{f_R[n,m], f_G[n,m], f_B[n,m]\}$  image, and this will affect the colors. However, the effect is usually too slight to be noticeable.

**Exercise 10-6:** If the PSFs for different colors are different, can we still deconvolve the image?

**Answer:** Yes. Use the proper PSF for each color.

### Example 10-3: Deblurring of Motion-Blurred Image

Horizontal motion by a camera relative to a stationary Christmas tree led to the  $(691 \times 1162)$  blurry image  $g[n,m]$  shown in **Fig. 10-11(a)**. Apply the motion-deblurring technique of Section 6-6.2 to each of the three RGB images to obtain a deblurred image  $f[n,m]$ . Use  $\lambda = 0.01$  (varying the value of  $\lambda$  does not appear to impact the results) and  $T = 1$  s.

#### Solution:

(1) Image  $g[n,m]$  consists of three images, namely  $g_R[n,m]$ ,  $g_G[n,m]$ , and  $g_B[n,m]$ . Application of the 2-D DSFT, as defined by Eq. (3.73a), to each of the three images leads to three spectral images  $\mathbf{G}_R(\Omega_1, \Omega_2)$ ,  $\mathbf{G}_G(\Omega_1, \Omega_2)$ , and  $\mathbf{G}_B(\Omega_1, \Omega_2)$ .

(2) For a blur PSF of horizontal extent  $(N+1)$  and an image of horizontal extent  $M$ , the blurred image has a horizontal extent of  $M + (N+1) - 1 = M + N$ . According to Eq. (6.49), motion blur in the horizontal direction causing an image to increase in width by  $N$  pixels is mathematically equivalent to a filter with a spatial frequency response  $\mathbf{H}(\Omega_1, \Omega_2)$  given by

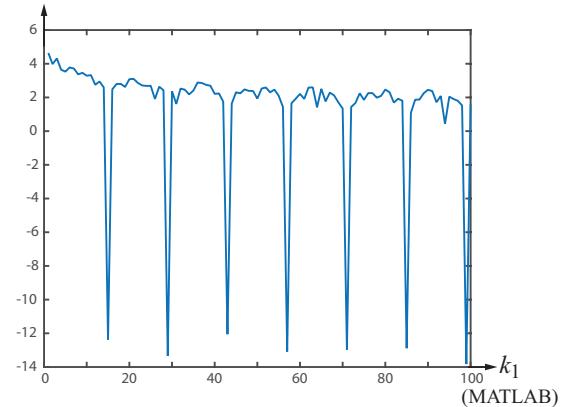
$$\mathbf{H}(\Omega_1, \Omega_2) = \frac{T}{N} \frac{\sin[\Omega_1 (\frac{N+1}{2})]}{\sin(\Omega_1/2)} e^{-j\Omega_1 N/2}. \quad (10.17)$$

Here,  $T$  is the total recording time and  $N$  is the total number of time shifts that occur during time  $T$ . Because the exact value of  $T$  is irrelevant to the deblurring procedure, we set  $T = 1$  s. In contrast, the value of  $N$  is crucial, so we need to extract it from the spectrum of one of the three color channels. Using the spectrum  $\mathbf{G}_R(\Omega_1, \Omega_2)$  of the red channel, we sample it at

$$\Omega_1 = \frac{2\pi k_1}{1162} \quad (10.18a)$$



(a) Blurred Christmas tree



(b) 1-D spectrum  $\log(|\mathbf{G}_R[k_1, 0]| + 1)$



(c) Reconstructed Christmas tree

**Figure 10-11** Deblurring of motion-blurred image of a Christmas tree.

and

$$\Omega_2 = \frac{2\pi k_2}{691}, \quad (10.18b)$$

to obtain  $\mathbf{G}_R[k_1, k_2]$ . Next, we plot the horizontal profile of  $|\mathbf{G}_R[k_1, k_2]|$  as a function of  $k_1$  at  $k_2 = 0$ . For display purposes, we offset  $|\mathbf{G}_R[k_1, k_2]|$  by 1 and plot the logarithm of the sum:

$$\log(|\mathbf{G}_R[k_1, 0]| + 1). \quad (10.19)$$

The MATLAB-generated plot of Eq. (10.19) is shown in [Fig. 10-11\(b\)](#). It exhibits a periodic sequence of sharp nulls corresponding to the zeros of  $\sin[\Omega_1(N+1)/2]$  in the spatial frequency response  $\mathbf{H}(\Omega_1, \Omega_2)$  of the PSF that caused the blurring of the image. The first null is at MATLAB index = 15, which corresponds to

$$k_1 = 15 - 1 = 14.$$

Since the first null of the sine function occurs when its argument is  $\pm\pi$ , it follows that

$$\pi = \frac{\Omega_1(N+1)}{2} = \frac{2\pi k_1}{1162} \frac{N+1}{2} = \frac{2\pi \times 14}{1162} \left( \frac{N+1}{2} \right),$$

which leads to  $N = 82$ .

(3) With  $\mathbf{H}(\Omega_1, \Omega_2)$  fully specified, we now apply the Wiener filter of Eq. (6.50) to the red-color spectrum to obtain the deblurred version

$$\begin{aligned} \mathbf{F}_R(\Omega_1, \Omega_2) &= \frac{\mathbf{G}_R(\Omega_1, \Omega_2) \mathbf{H}^*(\Omega_1, \Omega_2)}{|\mathbf{H}(\Omega_1, \Omega_2)|^2 + \lambda^2} \\ &= \frac{\mathbf{G}_R(\Omega_1, \Omega_2) \frac{T}{N} \frac{\sin[\Omega_1(\frac{N+1}{2})]}{\sin(\Omega_1/2)} e^{j\Omega_1 N/2}}{\left( \frac{T}{N} \right)^2 \left| \frac{\sin[\Omega_1(\frac{N+1}{2})]}{\sin(\Omega_1/2)} \right|^2 + \lambda^2}. \end{aligned} \quad (10.20)$$

Similar procedures are applied to the two other channels.

(4) Application of the inverse DFT to each of the three color channels yields  $f_R[n, m]$ ,  $f_G[n, m]$ , and  $f_B[n, m]$ , the combination of which yields the deblurred color image in [Fig. 10-11\(c\)](#). The motion-caused blurring has essentially been removed. The only artifact is the black band at the right-hand side of the image, which is due to the zero-padding used in the DFT (compare with [Fig. 10-10\(d\)](#)).

**Exercise 10-7:** If the length of the blur were  $N+1 = 6$ , at what  $\Omega_1$  values would  $\mathbf{H}(\Omega_1, \Omega_2) = 0$ ?

**Answer:** The numerator of Eq. (10.17) is zero when  $\sin(\Omega_1(N+1)/2) = 0$ , which occurs when  $\Omega_1(N+1)/2$  is a multiple of  $\pi$ , or equivalently when  $\Omega_1 = \pm k\pi/3$  for integers  $k$ . This is independent of the size of the image.

## 10-4 Denoising Color Images

In Chapter 7, we demonstrated how the discrete wavelet transform can be used to denoise an image by applying the combination of thresholding and shrinkage to the noisy image. The same approach is equally applicable to color images. The recipe outlined in Section 7-8.2 can be applied to each of the three RGB channels separately, and then the three modified images are combined to form the denoised color image. Since noise had been added to each RGB component separately, the denoising procedure also is applied to each RGB component separately. If preserving color is paramount, denoising can be applied to the Y component of YIQ, instead of to the RGB components.

We illustrate the effectiveness of the denoising technique through the following two examples.

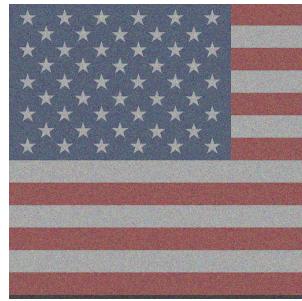
### Example 10-4: Denoising American Flag Image

To the original image shown in [Fig. 10-12\(a\)](#), a zero-mean white Gaussian noise random field with  $\sigma_v = 0.5$  was added to each of the RGB components. The resultant noisy image is shown in [Fig. 10-12\(b\)](#), and the associated signal-to-noise ratios are 3.32 dB for the red channel, 1.71 dB for the green, and 2.36 dB for the blue. Application of the 2-D Haar transform method outlined in Section 7-8 with a threshold/shrinkage factor  $\lambda = 3$  led to the denoised image shown in [Fig. 10-12\(c\)](#). Much of the noise has been removed.

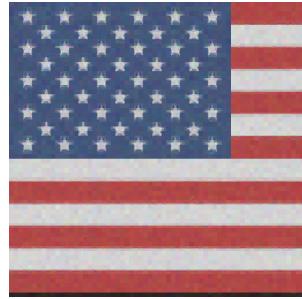
### Example 10-5: Brightening Toucan Image



(a) Original flag image



(b) Noisy image



(c) Denoised image

**Figure 10-12** Denoising American flag image using 2-D Haar transform method generates a less noisy image, but at the expense of high-resolution information.

The objective of this example is to brighten the toucan image shown in [Fig. 10-13\(a\)](#) using two different methods, namely the RGB and YIQ approaches, and then to compare and contrast the results. The plots in parts (b) to (d) display the histograms of the R, G, and B channels in the original image.

#### Method 1: RGB-Equalized Image

Application of the histogram-equalization method of Section 5-3 to the three RGB channels individually led to the image in [Fig. 10-14\(a\)](#). The underlying histograms are shown in parts (b)–(d) of the figure. The toucan image is indeed brighter than that of the original, but because the histogram equalization was performed separately and independently for the three RGB channels, the color balance in the RGB-equalized image is not the same as in the original image. Consequently, some of the colors got distorted.

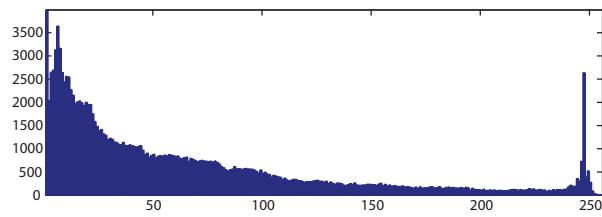
#### Method 2: YIQ-Equalized Image

After (a) converting the original RGB image to YIQ format, (b) applying histogram-equalization on only the Y channel, and then (c) converting the YIQ-equalized image to RGB, we end up with the image and associated histograms shown in [Fig. 10-15](#). The image in [Fig. 10-15\(a\)](#) is not as bright as the RGB-equalized image in [Fig. 10-14\(a\)](#), but the color balance is preserved in the YIQ-equalization method.

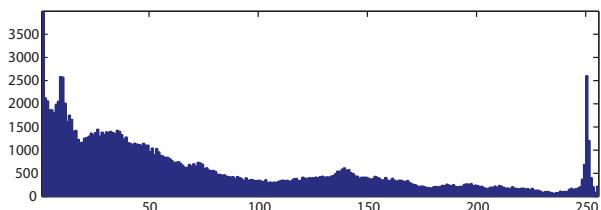
**Concept Question 10-4:** Why should wavelet-based denoising be applied to images in YIQ format and not RGB format?



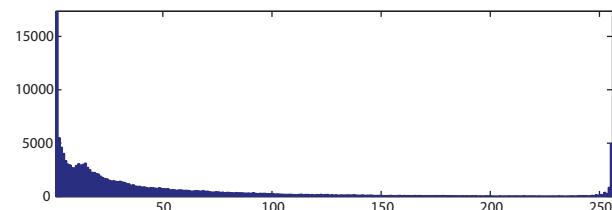
(a) Original toucan image



(b) Histogram of red (R) channel in original image



(c) Histogram of green (G) channel in original image

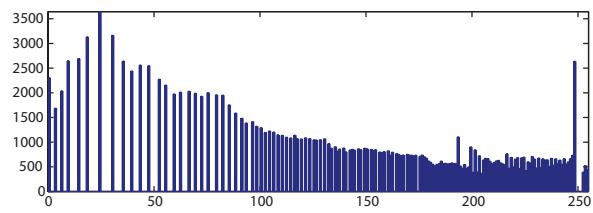


(d) Histogram of blue (B) channel in original image

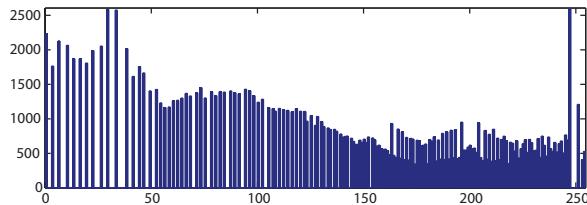
**Figure 10-13** Original image and associated histograms of its R, G, and B channels.



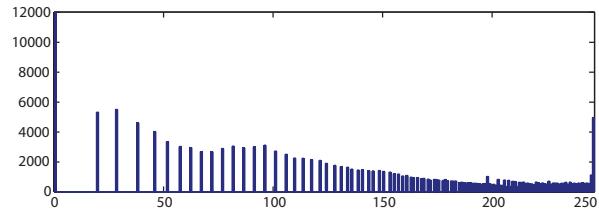
(a) RGB-equalized image



(b) Histogram of red (R) channel in RGB-equalized image



(c) Histogram of green (G) channel in RGB-equalized image



(d) Histogram of blue (B) channel in RGB-equalized image

**Figure 10-14** The RGB-equalized method generates a brighter image, but does not preserve color balance.

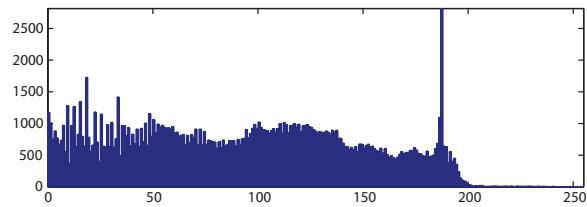
## Summary

### Concepts

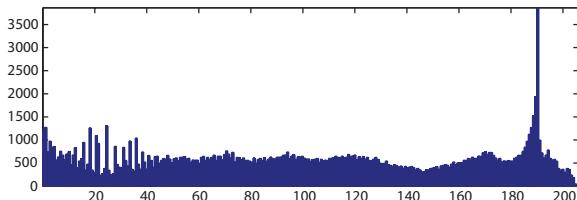
- Color images have three components: red, green, and blue, each of which is a separate 2-D function. This RGB color system represents how images are acquired and displayed.
- Other color systems include the CMYK system, used for color printing, and the YIQ system, used for image enhancement, which is applied only to the Y (luminance) component.
- Image restoration, such as denoising and deconvolution, uses the RGB color system, since each component is blurred or has noise added to it directly. Image enhancement, such as histogram equalization and edge detection, uses the YIQ system, since use of the RGB system can result in distortion of colors.
- Other colors can be represented as a linear combination of red, green, and blue, as depicted in the color cube.



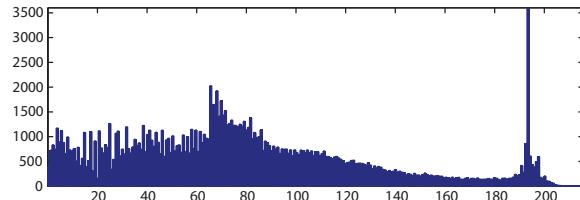
(a) YIQ-equalized image



(b) Histogram of red (R) channel in YIQ-equalized image



(c) Histogram of green (G) channel in YIQ-equalized image



(d) Histogram of blue (B) channel in YIQ-equalized image

**Figure 10-15** The YIQ-equalized method generates a partially brighter image, but does preserve color balance.

## Mathematical Formulae

### Components of a color image

$$f[n, m] = \{f_R[n, m], f_G[n, m], f_B[n, m]\}$$

### Relation between components of RGB and YIQ color systems

$$\begin{bmatrix} f_Y[n, m] \\ f_I[n, m] \\ f_Q[n, m] \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} f_R[n, m] \\ f_G[n, m] \\ f_B[n, m] \end{bmatrix}$$

### RGB to grayscale

$$f_{\text{gray}}[n, m] = 0.299f_R[n, m] + 0.587f_G[n, m] + 0.114f_B[n, m]$$

### MATLAB commands for color images

`imread`, `imagesc`

## Important Terms

Provide definitions or explain the meaning of the following terms:

additive color scheme  
chrominance

CMYK color scheme  
color cube

luminance  
RGB color scheme

subtractive color scheme  
YIQ color scheme

## PROBLEMS

### Section 10-1: Color Systems

**10.1** *Hang a checkerboard ornament on a Christmas tree.* The goal of this problem is to hang a color checkerboard ornament in the middle of the Christmas tree in MATLAB file `xmastree.mat`.

- (a) Create in MATLAB a  $(64 \times 64)$  checkerboard image, consisting of an  $(8 \times 8)$  array of alternating red and green  $(8 \times 8)$  blocks. Zero-pad this to  $(691 \times 1080)$  (the size of the image in `xmastree.mat`).
- (b) Add the zero-padded checkerboard image to the Christmas tree image. Display the resulting image. Why did this procedure produce a poor result?
- (c) Set the pixel values in the Christmas tree image where the checkerboard image is nonzero to zero. Now add the zero-padded checkerboard image to this image. Display the resulting two images. Why did this procedure produce a much better result?

**10.2** *Put the clown at the top of a Christmas tree.* The goal of this problem is to put the clown image in `clown.mat` at the top of the Christmas tree in MATLAB file `xmastree.mat`.

- (a) Convert the black-and-white clown image in `clown.mat` from YIQ to RGB format. ( $f_I[n,m]$  and  $f_Q[n,m]$  are both zero;  $f_Y[n,m]$  is the black-and-white image.) Zero-pad this to  $(691 \times 1080)$  (the size of the image in `xmastree.mat`).
- (b) Add the zero-padded checkerboard image to the Christmas tree image. Display the resulting image. Why didn't this work?
- (c) Set the pixel values in the Christmas tree image where the zero-padded clown image is nonzero to zero. Now add the zero-padded clown image to this image. Display the resulting two images. Why did this work?

### Section 10-2: Histogram Equalization and Edge Detection

**10.3** Apply histogram equalization (HE) to the Christmas tree image in `xmastree.mat` using the program `histcolor.m`. Display results for HE of RGB and YIQ formats.

**10.4** Apply histogram equalization (HE) to the Christmas tree image in `xmastreedark.mat` using the program `histcolor.m`. Display results for HE of RGB and YIQ formats.

**10.5** Use `sobelc.m` to apply Sobel edge detection to the toucan image in `toucan.mat`. Display the original image and the result of edge detection applied to the Y image component. Use a threshold of 0.5.

**10.6** Use `sobelc.m` to apply Sobel edge detection to the plumage image in `plumage.mat`. Display the original image and the result of edge detection applied to the Y image component. Use a threshold of 0.5.

### Section 10-3: Color Image Deblurring

**10.7** *Deblur motion-blurred Christmas tree.* Download the  $(686 \times 399)$  motion-blurred Christmas tree image in `xmasdarkmotion.mat`.

- (a) Display the motion-blurred Christmas tree image.
- (b) Display the log of the magnitude of the red component of `xmasdarkmotion.mat`. Zero-pad  $(686 \times 399)$  to  $(686 \times 400)$  to speed up the 2-D FFT. Use the 2-D spectrum to find the length  $N + 1$  of the PSF of the motion blur.
- (c) Use  $\lambda = 0.01$  and  $T = 1$  in `motiondeblurcolor.m` to deblur and display the tree.

**10.8** *Deblur motion-blurred Christmas tree.* Download the motion-blurred Christmas tree image in `xasmotion.mat`.

- (a) Display the motion-blurred Christmas tree image.
- (b) Use  $\lambda = 0.01$ ,  $T = 1$ , and  $N = 15$  in `motiondeblurcolor.m` to deblur the tree.

**10.9** *Refocus an out-of-focus image.* An out-of-focus image can be modeled as the image convolved with a disk-shaped PSF. The program `refocuscolor.m` convolves an image with a disk PSF and then deconvolves. Use `refocuscolor.m` to blur and deblur the  $(691 \times 1080)$  image in `xmastree.mat`. Uncomment a line in `refocuscolor.m` and use  $\lambda = 0.01$  and disk radius  $R = 20$ .

**10.10** *Refocus an out-of-focus image.* An out-of-focus image can be modeled as the image convolved with a disk-shaped PSF. The program `refocuscolor.m` convolves an image with a disk PSF and then deconvolves. Use `refocuscolor.m` to blur and then deblur the  $(340 \times 453)$  image in `toucan.mat`. Uncomment a line in `refocuscolor.m` and use  $\lambda = 0.01$  and disk radius  $R = 11$ .

**10.11** *Refocus an out-of-focus image.* An out-of-focus image can be modeled as the image convolved with a disk-shaped PSF. The program `refocuscolor.m` convolves an image with a disk PSF and then deconvolves. Use `refocuscolor.m` to

blur and then deblur the  $(512 \times 512)$  image in `flag.mat`. Uncomment a line in `refocuscolor.m` and use  $\lambda = 0.01$  and disk radius  $R = 29$ .

**10.12 Refocus an out-of-focus image.** An out-of-focus image can be modeled as the image convolved with a disk-shaped PSF. The program `refocuscolor.m` convolves an image with a disk PSF and then deconvolves. Use `refocuscolor.m` to blur and then deblur the plumage image in `plumage.mat`. Uncomment a line in `refocuscolor.m` and use  $\lambda = 0.01$  and disk radius  $R = 18$ .

## Section 10-4: Denoising Color Images

**10.13** Use Haar transform threshold and shrinkage program `haardenoiseimage.m` to denoise each RGB component of the  $(8 \times 8)$  checkerboard image in `checker.mat`. Zero-mean white Gaussian noise with variance  $\sigma^2$  is added to each RGB component. Uncomment a line in `haardenoiseimage.m` and use  $\lambda = 1$  and  $\sigma = 0.2$ .

**10.14** Use Haar transform threshold and shrinkage program `haardenoiseimage.m` to denoise each RGB component of the Christmas tree image in `xmastree.mat`. Zero-mean white Gaussian noise with variance  $\sigma^2$  is added to each RGB component. The image is clipped to  $(704 \times 704)$  since `haardenoiseimage.m` requires square images. Uncomment a line in `haardenoiseimage.m` and use  $\lambda = 0.2$  and  $\sigma = 0.1$ .

**10.15** Use Haar transform threshold and shrinkage program `haardenoiseimage.m` to denoise each RGB component of the toucan image in `toucan.mat`. Zero-mean white Gaussian noise with variance  $\sigma^2$  is added to each RGB component. Image is zero-padded to  $(480 \times 480)$  since `haardenoiseimage.m` requires square images. Uncomment a line in `haardenoiseimage.m` and use  $\lambda = 0.1$  and  $\sigma = 0.1$ .

**10.16** Use Haar transform threshold and shrinkage program `haardenoiseimage.m` to denoise each RGB component of the plumage image in `plumage.mat`. Zero-mean white Gaussian noise with variance  $\sigma^2$  is added to each RGB component. Image is zero-padded to  $(512 \times 512)$  since `haardenoiseimage.m` requires square images. Uncomment a line in `haardenoiseimage.m` and use  $\lambda = 0.2$  and  $\sigma = 0.1$ .

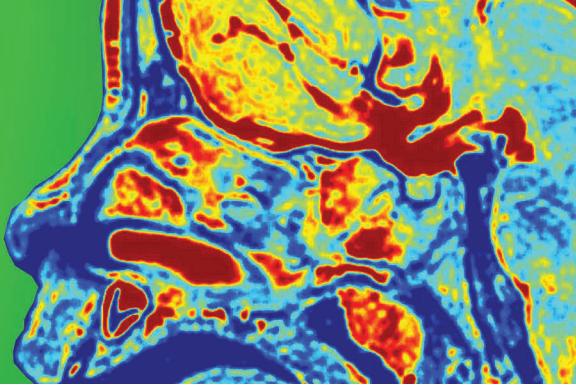
**10.17** Use db3 transform threshold and shrinkage program `daubdenoiseimage.m` to denoise each RGB component of the Christmas tree image in `xmastree.mat`. Zero-mean

white Gaussian noise with variance  $\sigma^2$  is added to each RGB component. The image is clipped to  $(704 \times 704)$  since `daubdenoiseimage.m` requires square images. Uncomment a line in `daubdenoiseimage.m` and use  $\lambda = 0.05$  and  $\sigma = 0.05$ .

**10.18** Use db3 transform threshold and shrinkage program `daubdenoiseimage.m` to denoise each RGB component of the toucan image in `toucan.mat`. Zero-mean white Gaussian noise with variance  $\sigma^2$  is added to each RGB component. Image is zero-padded to  $(480 \times 480)$  since `daubdenoiseimage.m` requires square images. Uncomment a line in `daubdenoiseimage.m` and use  $\lambda = 0.05$  and  $\sigma = 0.05$ .

**10.19** Use db3 transform threshold and shrinkage program `daubdenoiseimage.m` to denoise each RGB component of the plumage image in `plumage.mat`. Zero-mean white Gaussian noise with variance  $\sigma^2$  is added to each RGB component. Image is zero-padded to  $(512 \times 512)$  since `daubdenoiseimage.m` requires square images. Uncomment a line in `daubdenoiseimage.m` and use  $\lambda = 0.05$  and  $\sigma = 0.05$ .

# CHAPTER 11



## 11 Image Recognition

### Contents

- Overview, 354
  - 11-1** Image Classification by Correlation, 354
  - 11-2** Classification by MLE, 357
  - 11-3** Classification by MAP, 358
  - 11-4** Classification of Spatially Shifted Images, 360
  - 11-5** Classification of Spatially Scaled Images, 361
  - 11-6** Classification of Rotated Images, 366
  - 11-7** Color Image Classification, 367
  - 11-8** Unsupervised Learning and Classification, 373
  - 11-9** Unsupervised Learning Examples, 377
  - 11-10** K-Means Clustering Algorithm, 380
- Problems, 384

### Objectives

Learn to:

- Use correlation, MLE, and MAP to classify an image.
- Use cross-correlation to classify a shifted image.
- Use coordinate transformation (logarithmic or polar) to classify a spatially scaled or rotated image.
- Classify color images.
- Use the SVD to perform unsupervised learning.

Now is the time for all good men and women to come to the aid of their party.

(a) Image of text



(b) Thresholded cross-correlation  $\rho_5(n_0, m_0)$

Now is the time for all good men and women to come to the aid of their party.

(c) Image of text (green) and thresholded cross-correlation (red)

Image recognition amounts to classification of a given image as one of several given possible candidate images. Classification can be regarded as a discrete estimation problem in which the candidate image that maximizes a likelihood function is chosen. If there are no candidate images, unsupervised learning can be used to determine image classes from a set of training images.

## Overview

Along the very bottom section of the check shown in [Fig. 11-1](#) are three sequences of numbers. The first sequence, labeled the “Routing Number,” specifies the identity of the bank that had issued the check; the second sequence is the “Account Number” and it identifies the owner of the account; and the last sequence is the “Check Number” itself. When the check is inserted inside a check reader, an optical sensor scans across the rectangular space containing the three sequences of numbers and then displays the sequences on an electronic screen. [How does the check reader recognize the individual digits comprising the three sequences?](#) The answers to this and other [image recognition](#) questions are at the heart of the present and succeeding chapter.

The objective of an image recognition algorithm is to correctly classify an observed image  $g_{obs}[n,m]$  into one of  $\mathcal{L}$  possible classes on the basis of stored, true (reference) images  $f_k[n,m]$  of the  $\mathcal{L}$  classes, where  $\{k = 1, \dots, \mathcal{L}\}$ . If the classes are digits printed in a specific font, then the objective is to identify the correct digit from among the 10 possible digits between 0 and 9. In text recognition, not only is the number of classes (letters in the alphabet) much larger, but the letters may be printed in lower or upper case and may include more than one type of font. Other image recognition applications may involve classifying objects on the basis of their shapes and colors.

In some cases, the image recognition algorithm requires some form of prior [training](#) or [learning](#) using sample images of the types of objects under consideration. Otherwise, for  $\mathcal{L}$  classes of objects with known true (reference) images  $f_k[n,m]$ —with  $\{k = 1, 2, \dots, \mathcal{L}\}$ —an algorithm [measures the degree of](#)

[similarity](#) between  $g_{obs}[n,m]$  and each stored, reference image  $f_k[n,m]$  and then selects [the class index](#)  $k$  that maximizes the similarity. Some algorithms use the [correlation](#) between  $g_{obs}[n,m]$  and each  $f_k[n,m]$  as the parameter with which to measure the degree of similarity, while other algorithms may use measurement parameters that incorporate other image properties as well. Often the observed image  $g_{obs}[n,m]$  contains random noise, and the shape of the imaged object may be distorted due to some angular rotation or size modification. A robust classification algorithm needs to be adept at dealing with these types of deviations. These and other aspects of image recognition are addressed in this chapter.

Image recognition algorithms rely on one of three approaches:

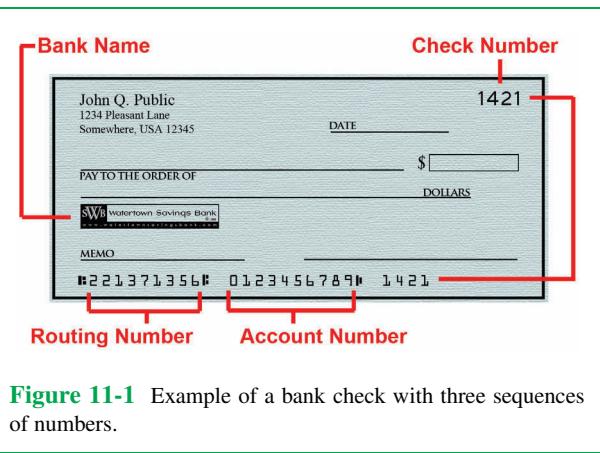
- (1) [Correlation](#) between the observed image and each of the stored reference images, in conjunction with *a priori* probability information about the frequency of occurrence of the unknown objects.
- (2) [Unsupervised training](#) using training images with *unknown* identities.
- (3) [Supervised training](#) using training images with *known* identities.

The present chapter covers the first two approaches, and the third approach is covered in Chapter 12.

### 11-1 Image Classification by Correlation

Consider the 10 numerals shown in [Fig. 11-2\(a\)](#). The numerals are printed in white, using a standard bank font, against a black background. In practice, check numbers are read by a reading head (optical scanner) whose size in pixels is  $(9 \times 1)$ . As the head moves horizontally across each  $(9 \times 7)$  numeral, a 1-D signal of duration 7 samples is generated as the output “signature” of that numeral. Upon comparing the measured signal with the 10 possible signals stored in the system, the system selects the numeral (from among 1, 2, ..., 9, 0) that provides the best match.

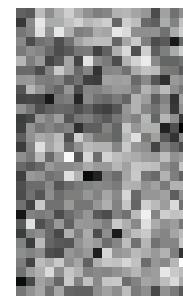
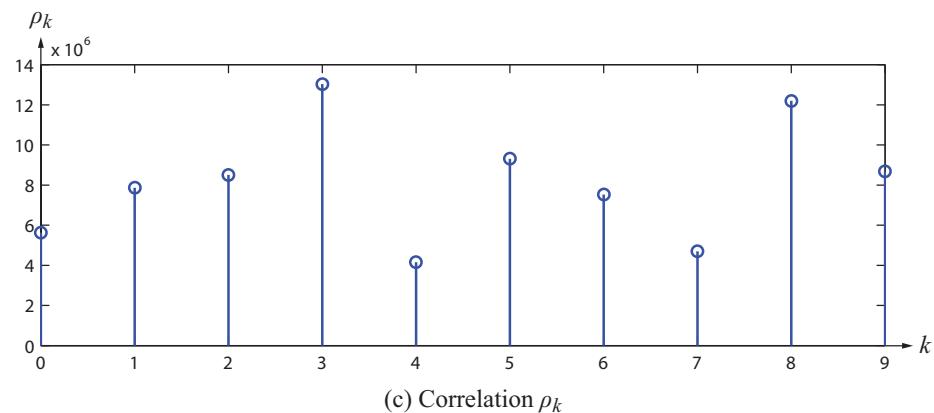
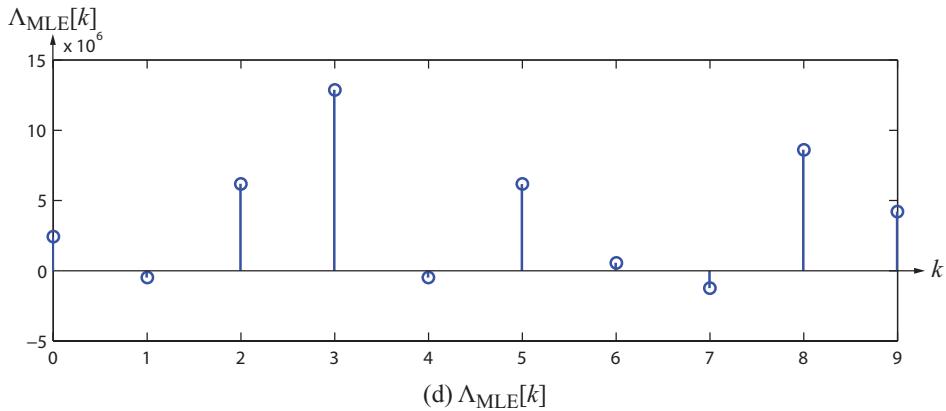
An alternative, and more reliable, approach to performing the numeral recognition problem is to use a 2-D image instead of a 1-D signal. Let us assume that each numeral in the check account number is allocated a rectangular space of  $M \times N$  pixels, comprising  $N$  horizontal pixels and  $M$  vertical rows of pixels. When the optical scanner sweeps across the rectangle containing the bank account number, it images the rectangle and then partitions it into individual images each  $(M \times N)$  pixels in size. An expanded view of one such image is shown in [Fig. 11-2\(b\)](#).



**Figure 11-1** Example of a bank check with three sequences of numbers.



(a) Numerals 1,2,...,9,0 in Bank font

(b) Noisy image  $g_{\text{obs}}[n,m]$  of numeral 3(c) Correlation  $\rho_k$ (d)  $\Lambda_{\text{MLE}}[k]$ 

**Figure 11-2** (a) Numerals 1–9 and 0 in bank font, (b) noisy ( $30 \times 18$ ) image  $g_{\text{obs}}[n,m]$  of numeral 3 with SRN =  $-1.832$  dB, (c) correlation  $\rho_k$ , and (d) MLE criterion  $\Lambda_2[k]$ .

It contains a single numeral, but it is difficult to recognize the numeral's identity because the image is quite noisy (the signal-to-noise ratio is  $-1.832$  dB, corresponding to a signal energy only 66% of that of the noise). The image size is  $(30 \times 18)$  and the true identity of the numeral is 3, but we assume at this stage of the classification example that we do not yet know the true identity of the numeral in **Fig. 11-2(b)**.

Let us designate the observed image of the unknown numeral as  $g_{\text{obs}}[n, m]$ , which consists of the sum of two images, namely images  $f_k[n, m]$  and  $v[n, m]$ :

$$g_{\text{obs}}[n, m] = f_k[n, m] + v[n, m], \quad (11.1) \\ (0 \leq n \leq N-1, \quad 0 \leq m \leq M-1),$$

where  $f_k[n, m]$  is the true image of the unknown (but we do not yet know the value of index  $k$ ) and  $v[n, m]$  is a random-noise image. Index  $k$  denotes the *class* of image  $f_k[n, m]$ : for the present example,  $k$  extends over the range  $k = 1$  for class 1 to  $k = 10$  for class 10, with class 1 representing numeral 1, class 9 representing numeral 9, and class 10 representing numeral 0. The goal of the classification algorithm is to determine the **true value of index  $k$** , from the observed image  $g_{\text{obs}}[n, m]$ , given the stored reference images  $f_1[n, m]$  through  $f_{10}[n, m]$ .

A common approach to establishing the identity of an unknown class is to compute the **correlation**  $\rho_k$  between the observed image  $g_{\text{obs}}[n, m]$  and the reference image  $f_k[n, m]$  for all  $\mathcal{L}$  classes, and the select the value of  $k$  associated with the highest correlation. The correlation  $\rho_k$  is given by

$$\rho_k = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g_{\text{obs}}[n, m] f_k[n, m]. \quad (11.2)$$

Since  $g_{\text{obs}}[n, m]$  and  $f_k[n, m]$  are both  $(M \times N)$ , computation of  $\rho_k$  requires  $MN$  multiplications and additions for each value of  $k$ . If  $g_{\text{obs}}[n, m]$  and  $f_k[n, m]$  have sparse wavelet transforms, we can take advantage of Parseval's theorem for the wavelet transform (Eq. (7.14b)) to compute  $\rho_k$ , because multiplications by zero do not count as computations. The Haar transform requires only subtractions and divisions by  $\sqrt{2}$ .

To determine the identity of the numeral in the noisy image shown in **Fig. 11-2(b)**,  $\rho_k$  was computed using Eq. (11.2) for all 10 values of  $k$ , and the results are displayed in **Fig. 11-2(c)**. Among the 10 correlations,  $\rho_3$  of class 3 (corresponding to numeral 3) exhibits the largest value, but it exceeds  $\rho_8$  of class 8 by only a small amount. The correct numeral in image  $g_{\text{obs}}[n, m]$  is indeed 3, so the correlation method is successful in classifying it correctly, but the margin of error is rather slim. As we shall see shortly, the MLE classification method outlined in the next

## Notation for Training and Classification

### Single-Color Images

$f_k[n, m]$  = reference image of class  $k$ ,  $\{k = 1, \dots, \mathcal{L}\}$

$f_k^{(i)}[n, m]$  =  $i$ th training image of class  $k$

$\mathcal{L}$  = number of classes

$g_{\text{obs}}[n, m]$  = observed image of unknown class

$M \times N$  = image size

$\rho_k$  = correlation between  $f_k[n, m]$  and  $g_{\text{obs}}[n, m]$

$p[k]$  = *a priori* probability of occurrence of class  $k$

$E_{f_k}$  = energy of image  $f_k[n, m]$

$\Lambda_{\text{MLE}}$  = MLE maximization parameter

$\hat{k}_{\text{MLE}}$  = value of  $k$  that maximizes  $\Lambda_{\text{MLE}}$

$\Lambda_{\text{MAP}}$  = MAP maximization parameter

$\hat{k}_{\text{MAP}}$  = value of  $k$  that maximizes  $\Lambda_{\text{MAP}}$

### Color Images

$\mathbf{f}_k^{(i)}[n, m]$  =  $i$ th training image **vector** for class  $k$   
 $= [f_{k,R}^{(i)}[n, m], f_{k,G}^{(i)}[n, m], f_{k,B}^{(i)}[n, m]]$

$g_{\text{obs}}[n, m]$  = observed image **vector**

$\mathbf{K}_k[n, m]$  = class-specific covariance matrix

$\mathbf{K}_{k0}$  = joint covariance matrix

section also classifies the observed image correctly, but it does so with a superior margin of error.

**Exercise 11-1:** We have two classes of  $(1 \times 1)$  images  $f_1[n, m] = 1$  and  $f_2[n, m] = 4$ . Use correlation to classify  $g_{\text{obs}}[n, m] = 2$ .

**Answer:**  $\rho_1 = (2)(1) = 2$  and  $\rho_2 = (2)(4) = 8$ . Since  $\rho_2 > \rho_1$ , classify  $g_{\text{obs}}[n, m]$  as  $k = 2$ . This is counterintuitive (2 is closer to 1 than to 4), so see Exercise 11-2.

## 11-2 Classification by MLE

For  $\mathcal{L}$  classes, the observed  $(M \times N)$  image  $g_{\text{obs}}[n, m]$  is one of:

$$g_{\text{obs}}[n, m] = \begin{cases} f_1[n, m] + v[n, m] & \text{class } \#1, \\ f_2[n, m] + v[n, m] & \text{class } \#2, \\ \vdots & \vdots \\ f_{\mathcal{L}}[n, m] + v[n, m] & \text{class } \mathcal{L}, \end{cases} \quad (11.3)$$

where  $v[n, m]$  is a  $(M \times N)$  zero-mean white Gaussian noise random field with variance  $\sigma_v^2$ .

### 11-2.1 Marginal pdfs

Accordingly, for each location  $[n, m]$ , exactly one of the following **marginal pdfs** is the pdf of  $g_{\text{obs}}[n, m]$ :

$$g_{\text{obs}}[n, m] \sim \begin{cases} \mathcal{N}(f_1[n, m], \sigma_v^2) & \text{class } \#1, \\ \mathcal{N}(f_2[n, m], \sigma_v^2) & \text{class } \#2, \\ \vdots & \vdots \\ \mathcal{N}(f_{\mathcal{L}}[n, m], \sigma_v^2) & \text{class } \mathcal{L}, \end{cases} \quad (11.4)$$

where  $\mathcal{N}(f_k[n, m], \sigma_v^2)$  is a short-hand notation analogous to that in Eq. (8.86), and it states that the pdf of  $g_{\text{obs}}[n, m]$  for class  $k$  has the form of the Gaussian pdf given by Eq. (8.38a):

$$p(g_{\text{obs}}[n, m]) = \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{-(g_{\text{obs}}[n, m] - f_k[n, m])^2 / (2\sigma_v^2)}. \quad (11.5)$$

The form of the marginal pdf (i.e., the pdf specific to pixel  $[n, m]$ ) given by Eq. (11.5) is a consequence of the fact that  $v[n, m]$  is a zero-mean random field, as a result of which the mean value of  $g_{\text{obs}}[n, m]$  is simply  $f_k[n, m]$ .

Notationally,  $p(g_{\text{obs}}[n, m])$  is the **marginal pdf** of pixel  $[n, m]$  and  $p(\{g_{\text{obs}}[n, m]\})$  is the **joint pdf** of all pixels in image  $g_{\text{obs}}[n, m]$ . Since for class  $k$ , image  $g_{\text{obs}}[n, m]$  is the sum of  $f_k[n, m]$  and white random noise  $v[n, m]$ , and since the amount of noise  $v[n, m]$  added to pixel  $[n, m]$  is independent of the amount of noise added to other pixels, image values  $\{g_{\text{obs}}[n, m]\}$  can be regarded as independent if the amount of noise added to  $\{f_k[n, m]\}$  is significant. This is certainly the case for the image shown in **Fig. 11-2(b)**. [If the noise is insignificant in comparison with the signal, the classification task becomes rather trivial; the correlation method of the preceding subsection should be able to correctly classify the observed image with no error.] Hence, the joint pdf  $p(\{g_{\text{obs}}[n, m]\})$  is equal to the product over

all locations  $[n, m]$  of the marginal pdfs  $p(g_{\text{obs}}[n, m])$  given by Eq. (11.5):

$$\begin{aligned} p(\{g_{\text{obs}}[n, m]\}) &= \prod_{n=0}^{N-1} \prod_{m=0}^{M-1} p(g[n, m]) \\ &= \frac{1}{(2\pi\sigma_v^2)^{NM/2}} \prod_{n=0}^{N-1} \prod_{m=0}^{M-1} e^{-(g_{\text{obs}}[n, m] - f_k[n, m])^2 / (2\sigma_v^2)} \\ &= \frac{1}{(2\pi\sigma_v^2)^{NM/2}} e^{-\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_k[n, m])^2 / (2\sigma_v^2)}. \end{aligned} \quad (11.6)$$

Following Section 9-3, the likelihood function  $p(\{g_{\text{obs}}[n, m]\})$  is one of the following joint pdfs:

$$\begin{aligned} p(\{g_{\text{obs}}[n, m]\}) &= \frac{1}{(2\pi\sigma_v^2)^{NM/2}} \\ &\times \begin{cases} e^{-(1/2\sigma_v^2) \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_1[n, m])^2} & \text{class } \#1, \\ e^{-(1/2\sigma_v^2) \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_2[n, m])^2} & \text{class } \#2, \\ \vdots & \vdots \\ e^{-(1/2\sigma_v^2) \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_{\mathcal{L}}[n, m])^2} & \text{class } \#\mathcal{L}, \end{cases} \end{aligned} \quad (11.7)$$

and the natural log-likelihood  $\ln[p(\{g_{\text{obs}}[n, m]\})]$  is

$$\begin{aligned} \ln[p(\{g_{\text{obs}}[n, m]\})] &= -\frac{NM}{2} \ln(2\pi\sigma_v^2) - \frac{1}{2\sigma_v^2} \\ &\times \begin{cases} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_1[n, m])^2 & \text{class } \#1, \\ \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_2[n, m])^2 & \text{class } \#2, \\ \vdots & \vdots \\ \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_{\mathcal{L}}[n, m])^2 & \text{class } \#\mathcal{L}. \end{cases} \end{aligned} \quad (11.8)$$

### 11-2.2 MLE of Index $k$

The **maximum likelihood estimate**  $\hat{k}_{\text{MLE}}$  is the value of  $k$  that maximizes the log-likelihood function given by Eq. (11.8). Since  $-\frac{NM}{2} \log(2\pi\sigma_v^2)$  is added to all terms, and  $-1/(2\sigma_v^2)$  multiplies all terms,  $\hat{k}_{\text{MLE}}$  is the value of  $k$  that maximizes  $\Lambda_1[k]$  defined as

$$\Lambda_1[k] = -\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_k[n, m])^2. \quad (11.9)$$

This expression has an evident interpretation: Choose the value of  $k$  such that  $f_k[n, m]$  is closest to  $g_{\text{obs}}[n, m]$  in the  $\ell_2$  (sum of

squares) norm defined in Eq. (7.117b).

Computation of  $\hat{k}_{\text{MLE}}$  can be simplified further by expanding Eq. (11.9), since

$$\begin{aligned}\Lambda_1[k] &= - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (g_{\text{obs}}[n, m] - f_k[n, m])^2 \\ &= - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g_{\text{obs}}[n, m]^2 - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f_k[n, m]^2 \\ &\quad + 2 \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g_{\text{obs}}[n, m] f_k[n, m].\end{aligned}\quad (11.10)$$

The first term is the energy of  $g_{\text{obs}}[n, m]$ , which is independent of  $k$ , so we can ignore it. The second term is the energy  $E_{f_k}$  of  $f_k[n, m]$ . The recipe for computing  $\hat{k}_{\text{MLE}}$  simplifies to choosing the value of  $k$  that maximizes the **MLE** criterion  $\Lambda_{\text{MLE}}[k]$ :

$$\Lambda_{\text{MLE}}[k] = 2 \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g_{\text{obs}}[n, m] f_k[n, m] - E_{f_k}. \quad (11.11)$$

In view of Eq. (11.2),  $\Lambda_{\text{MLE}}[k]$  is related to the correlation  $\rho_k$  by

$$\Lambda_{\text{MLE}}[k] = 2\rho_k - E_{f_k}. \quad (11.12)$$

Returning to the image of **Fig. 11-2(b)**, computation of the MLE parameter  $\Lambda_{\text{MLE}}[k]$  leads to the stem plot in **Fig. 11-2(d)**, which correctly selects numeral 3 as the numeral in the noisy image, with a significant margin between the values of  $\Lambda_{\text{MLE}}[3]$  and  $\Lambda_{\text{MLE}}[8]$ .

	Classified as → 0 1 2 3 4 5 6 7 8 9									
True numeral ↓										
0	78	4	1	5	1	3	2	2	0	4
1	0	77	4	0	6	2	7	3	0	1
2	5	9	69	1	7	1	2	5	1	0
3	2	2	0	85	2	3	4	0	1	1
4	0	3	4	0	70	5	5	3	8	2
5	1	4	2	4	5	76	0	4	1	3
6	0	6	1	0	3	1	83	3	2	1
7	0	3	11	2	5	4	1	69	4	1
8	0	0	1	3	7	1	3	6	75	4
9	0	3	1	0	3	4	7	2	4	76

The first row pertains to numeral 0. Of the 100 noisy images containing an image of numeral 0, 78 were classified correctly, 4 were misclassified as numeral 1, 1 was misclassified as numeral 2, etc. Each row sums to 100, and similar interpretations apply to the other numerals. A perfect classifier would be a  $(10 \times 10)$  diagonal matrix of 100s. For the noisy images used in this trial, the classification accuracy varied between a low of 69% for numeral 7 and a high of 85% for numeral 3.

**Concept Question 11-1:** Why is the maximum likelihood (MLE) classifier different from correlation?

**Exercise 11-2:** We have two classes of  $(1 \times 1)$  images  $f_1[n, m] = 1$  and  $f_2[n, m] = 4$ . Use the MLE classifier to classify  $g_{\text{obs}}[n, m] = 2$ .

**Answer:**  $\rho_1 = (2)(1) = 2$  and  $\rho_2 = (2)(4) = 8$ .  $\Lambda_{\text{MLE}}[1] = 2(2) - 1^2 = 3$  and  $\Lambda_{\text{MLE}}[2] = 2(8) - 4^2 = 0$ . Since  $\Lambda_{\text{MLE}}[1] > \Lambda_{\text{MLE}}[2]$ , classify  $g_{\text{obs}}[n, m]$  as  $k = 1$ . Including energies of the images resulted in a different classification.

### 11-2.3 MLE Bank Numeral Classification

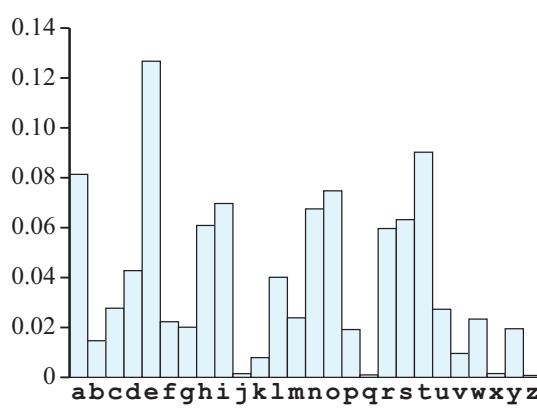
We now extend the numeral classification example by performing 1000 trials comprised of 100 noisy images of each of the 10 numerals 0 to 9. By selecting  $\sigma_v^2 = 10^6$ , we decrease the signal-to-noise ratio from  $-1.832$  dB, an image example of which is shown in **Fig. 11-1(b)**, down to an average of  $-16.7$  dB among the various trials and numerals. At this ratio, the average signal energy is only 2% of the noise energy!

The results of the trials using the MLE criterion  $\Lambda_{\text{MLE}}[k]$  to classify each of the 1000 trials are summarized in the form of the following **confusion matrix**:

### 11-3 Classification by MAP

Sometimes *a priori* probabilities  $p[k]$  are available for the various  $f_k[n, m]$ . This is not the case for zip codes or bank account numbers (all ten digits are equally likely), but it is the case for letters of the alphabet. **Figure 11-3** depicts the frequencies of appearances of a letter in typical text. These frequencies function as *a priori* probabilities  $p[k]$  for each letter in **letter recognition** in an image of text.

The *a priori* probabilities  $p[k]$  can be incorporated into the image recognition problem by using an MAP formulation instead of an MLE formulation, by simply multiplying the likelihood



**Figure 11-3** Frequencies of appearances of letters, which can be regarded as *a priori* probabilities  $p[k]$ .

function in Eq. (11.7) by  $p[k]$ . The modification adds  $\ln[p[k]]$  to the log-likelihood function in Eq. (11.8). Repeating the above derivation,  $\hat{k}_{\text{MAP}}$  is computed by choosing the  $k$  that maximizes  $\Lambda_{\text{MAP}}[k]$ :

$$\Lambda_{\text{MAP}}[k] = 2\rho_k - E_{f_k} + 2\sigma_v^2 \ln p[k]. \quad (11.13)$$

The MAP (maximum *a posteriori*) classifier is also known as the **minimum error probability (MEP)** classifier since it minimizes the probability of an incorrect choice of  $k$ . Note that if  $p[k] = 1/\mathcal{L}$ , so that each class is equally likely, the MAP classifier reduces to the MLE classifier. Also note that  $\sigma_v^2$  functions as a trade-off parameter between the *a priori* information  $p[k]$  and the *a posteriori* information  $\rho_k$ : the noisier the observations, the larger is  $\sigma_v^2$ , and the greater the weight given to *a priori* information  $p[k]$ . The smaller the noise, the heavier is the weight given to *a posteriori* (from the noisy data) information  $\rho_k$ .

### Example 11-1: $(2 \times 2)$ Image Classification Example

Classify noisy image

$$g_{\text{obs}}[n, m] = \begin{bmatrix} g_{0,0} & g_{1,0} \\ g_{0,1} & g_{1,1} \end{bmatrix}$$

into one of the two classes defined by the image  $\begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix}$  and  $\begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix}$ . Obtain a classification rule in terms of the values of the four elements of  $g_{\text{obs}}[n, m]$  using (a) correlation and (b) MLE.

#### Solution:

**(a) Classification by correlation:** In terms of the notation introduced earlier,

$$f_1[n, m] = \begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix} \quad (11.14a)$$

and

$$f_2[n, m] = \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix}. \quad (11.14b)$$

The energies of  $f_1[n, m]$  and  $f_2[n, m]$  are

$$E_{f_1} = 2^2 + 1^2 + 5^2 + 7^2 = 79, \quad (11.15a)$$

$$E_{f_2} = 8^2 + 6^2 + 4^2 + 3^2 = 125. \quad (11.15b)$$

The correlations between  $g_{\text{obs}}[n, m]$  and each of  $f_1[n, m]$  and  $f_2[n, m]$  are

$$\rho_1 = 2g_{0,0} + g_{1,0} + 5g_{0,1} + 7g_{1,1}, \quad (11.16a)$$

$$\rho_2 = 8g_{0,0} + 6g_{1,0} + 4g_{0,1} + 3g_{1,1}. \quad (11.16b)$$

For a given image  $g[n, m]$ , we classify it as

$$f_1[n, m], \quad \text{if } \rho_1 > \rho_2 \quad (11.17a)$$

or as

$$f_2[n, m], \quad \text{if } \rho_1 < \rho_2. \quad (11.17b)$$

**(b) Classification by MLE:** Using Eq. (11.11), the MLE parameter  $\Lambda_2[k]$  is given by

$$\Lambda_2[1] = 4g_{0,0} + 2g_{1,0} + 10g_{0,1} + 14g_{1,1} - 79, \quad (11.18a)$$

$$\Lambda_2[2] = 16g_{0,0} + 12g_{1,0} + 8g_{0,1} + 6g_{1,1} - 125. \quad (11.18b)$$

Upon subtracting the expression for  $\Lambda_2[1]$  from the expression for  $\Lambda_2[2]$ , we obtain the following classification rule:

Image of  $g_{\text{obs}}[n, m]$  is

$$f_1[n, m], \quad \text{if } (12g_{0,0} + 10g_{1,0} - 2g_{0,1} - 8g_{1,1} - 46) < 0 \quad (11.19a)$$

and

$$f_2[n, m], \quad \text{if } (12g_{0,0} + 10g_{1,0} - 2g_{0,1} - 8g_{1,1} - 46) > 0. \quad (11.19b)$$

**Concept Question 11-2:** Why is the maximum *a posteriori* probability (MAP) classifier different from the maximum likelihood (MLE) classifier?

**Exercise 11-3:** We have two classes of  $(1 \times 1)$  images  $f_1[n, m] = 1$  and  $f_2[n, m] = 4$ , with respective *a priori* probabilities  $p[k=1] = 0.1$  and  $p[k=2] = 0.9$ . The additive noise has  $\sigma_v^2 = 3$ . Use the MAP classifier to classify  $g_{\text{obs}}[n, m] = 2$ .

**Answer:**  $\rho_1 = (2)(1) = 2$  and  $\rho_2 = (2)(4) = 8$ .  $\Lambda_{\text{MAP}}[1] = 2(2) - 1^2 + 2(3)\log(0.1) = -10.8$  and  $\Lambda_{\text{MAP}}[2] = 2(8) - 4^2 + 2(3)\log(0.9) = -0.63$ . Since  $\Lambda_{\text{MAP}}[2] > \Lambda_{\text{MAP}}[1]$ , classify  $g_{\text{obs}}[n, m]$  as  $k = 2$ . The *a priori* probabilities biased the classification towards  $k = 2$ .

## 11-4 Classification of Spatially Shifted Images

Thus far, we have assumed that the location, size, and orientation of the observed image  $g_{\text{obs}}[n, m]$  is the same as those of the reference images  $\{f_k[n, m], k = 1, 2, \dots, \mathcal{L}\}$  for all  $\mathcal{L}$  classes of numerals, letters, objects, etc. Now, in this and the next two sections, we examine how to perform image classification successfully even when the reference images are shifted in location, of different size, or different orientation relative to the observed image.

### 11-4.1 Unknown Relative Shift

In the absence of a spatial shift between the observation and reference images, all images are arranged so that their origins  $[0, 0]$  coincide. In some applications, however, the observed image  $g_{\text{obs}}[n, m]$  or the reference images  $f_k[n, m]$  may be shifted by an unknown amount  $[n_0, m_0]$  relative to each other. One such application is the search for occurrence of a specific letter in a block of text, an example of which is presented later in the form of Example 11-4.

Since it is the relative—rather than the absolute—spatial shift that matters, the image classification problem can be formulated as either:

$g_{\text{obs}}[n, m]$ -Reference Format:

$$g_{\text{obs}}[n, m] = \begin{cases} f_1[n - n_0, m - m_0] & \text{class \#1,} \\ f_2[n - n_0, m - m_0] & \text{class \#2,} \\ \vdots & \vdots \\ f_{\mathcal{L}}[n - n_0, m - m_0] & \text{class \#\mathcal{L},} \end{cases} \quad (11.20)$$

or as

$f_k[n, m]$ -Reference Format:

$$g_{\text{obs}}[n - n_0, m - m_0] = \begin{cases} f_1[n, m] & \text{class \#1,} \\ f_2[n, m] & \text{class \#2,} \\ \vdots & \vdots \\ f_{\mathcal{L}}[n, m] & \text{class \#\mathcal{L}.} \end{cases} \quad (11.21)$$

### 11-4.2 Classification by Cross-Correlation

In both cases, the objective is to select the correct class through some appropriate comparison between  $g_{\text{obs}}$  and  $f_k$  for all possible values of  $k$ . One such measure of comparison is the cross-correlation  $\rho_k[n_0, m_0]$  between  $g_{\text{obs}}$  and  $f_k$  when one of them is shifted by  $[n_0, m_0]$  relative to the other. Of the two formats, the observation-reference format given by Eq. (11.20) is more applicable to search algorithms, so we adopt it in here for computing the cross-correlation:

$$\rho_k[n_0, m_0] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g_{\text{obs}}[n, m] f_k[-(n_0 - n), -(m_0 - m)]. \quad (11.22)$$

Both images are of size  $(M \times N)$  pixels.

By rearranging the order of the indices of  $f_k$ , we can cast the summations in the form of a convolution:

$$\begin{aligned} \rho_k[n_0, m_0] &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g_{\text{obs}}[n, m] f_k[-(n_0 - n), -(m_0 - m)] \\ &= g_{\text{obs}}[n_0, m_0] * * f_k[-n_0, -m_0], \end{aligned} \quad (11.23)$$

The image representing  $\rho_k[n_0, m_0]$  is called the **cross-correlation** between  $g_{\text{obs}}[n, m]$  and  $f_k[n, m]$ . Using properties #4 and #5 in **Table 3-3**,  $\rho_k[n, m]$  can be computed readily using the zero-padded 2-D DFT:

$$\mathbf{G}_{\text{obs}}[k_1, k_2] = (2M \times 2N) \text{DFT}\{g_{\text{obs}}[n, m]\}, \quad (11.24a)$$

$$\mathbf{F}_k[k_1, k_2] = (2M \times 2N) \text{DFT}\{f_k[n, m]\}, \quad (11.24b)$$

$$\rho_k[n, m] = \text{DFT}^{-1}\{\mathbf{G}_{\text{obs}}[k_1, k_2] \mathbf{F}_k[2N - k_1, 2M - k_2]\}. \quad (11.24c)$$

### 11-4.3 Classification Recipe for Discrete-Space Images

The recipe for classification when an unknown relative shift  $[n_0, m_0]$  may exist between  $g_{\text{obs}}[n, m]$  and  $\{f_k[n, m], k = 1, 2, \dots, \mathcal{L}\}$  consists of the following steps:

1. Compute  $\rho_k[n, m]$  for all values of  $k: \{k = 1, 2, \dots, \mathcal{L}\}$  using Eq. (11.24c).
2. Among the total of  $(M \times N)$  pixels  $\times \mathcal{L}$  classes  $= MN\mathcal{L}$ , identify the combination of pixel location  $[n, m]$  and class  $k$  that yields the largest value of  $\rho_k[n, m]$ .
3. Label that specific pixel location as  $[n, m] = [n_0, m_0]$  and label the identified value of  $k$  as the unknown class  $K$ .

#### Example 11-2: Identifying Letter “e” in Text Image

**Figure 11-4(a)** displays a  $(33 \times 256)$  image of two lines of text. Use cross-correlation to identify all the locations of the letter “e” within the text image. In the printed text, each lower-case letter is allocated  $(9 \times 7)$  pixels and reference images  $f_k[n, m]$  are provided for all lower-case letters.

**Solution:** Application of the recipe outlined in Section 11-5.3 leads to the identification of 7 pixels with cross-correlation values that were much larger than those of other pixels. After examining the distribution of values, all values of the cross-correlation smaller than 800,000 were set to zero, thereby highlighting the high-value pixels shown in **Fig. 11-4(b)**. With the text printed in green in **Fig. 11-4(c)**, the identified pixels in part (b) are now printed in red. Examination of the image confirms that the cross-correlation algorithm has successfully identified all occurrences of the letter “e” in the text image.

### 11-4.4 Classification Recipe for Continuous-Space Images

In analogy with Eq. (11.20), for continuous-space images the classification problem is formulated as

$$g_{\text{obs}}(x, y) = \begin{cases} f_1(x - x_0, y - y_0) & \text{class } \#1, \\ f_2(x - x_0, y - y_0) & \text{class } \#2, \\ \vdots & \vdots \\ f_{\mathcal{L}}(x - x_0, y - y_0) & \text{class } \#\mathcal{L}, \end{cases} \quad (11.25)$$

where the true class is class  $\#K$  and variables  $K, x_0$  and  $y_0$  are all unknown. The cross-correlation for the continuous-space case is given by

$$\rho_k(x_0, y_0) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g_{\text{obs}}(\xi, \eta) f_k(\xi - x_0, \eta - y_0) d\xi d\eta, \quad (11.26)$$

and the classification recipe proceeds as follows:

1. Given observation image  $g_{\text{obs}}(x, y)$  and reference images  $f_k(x, y)$ , compute the cross-correlation  $\rho_k(x_0, y_0)$  for all values of  $k: \{k = 1, 2, \dots, \mathcal{L}\}$ , and for all spatial shifts  $(x_0, y_0)$  that offer nonzero overlap between  $g_{\text{obs}}(x, y)$  and  $f_k(x - x_0, y - y_0)$ .
2. Identify the combination of shift  $(x_0, y_0)$  and class  $k$  that exhibits the largest value of  $\rho_k(x_0, y_0)$ .

## 11-5 Classification of Spatially Scaled Images

We now consider the problem of image classification in which the observed image  $g_{\text{obs}}(x, y)$  is a *spatially scaled* version of the reference images  $f_k(x, y)$ , or vice versa. Our presentation is specific to *continuous-space* observation and reference images,  $g_{\text{obs}}(x, y)$  and  $f_k(x, y)$ . In a later part of this section, we address how to apply the classification algorithm developed for continuous-space images to discrete-space images. Additionally, to keep the presentation simple and manageable, *we assume that the observed images are noise-free*.

The topic of spatial scaling of images was presented earlier in the book, in Section 3-2.2A. If  $g_{\text{obs}}(x, y)$  is a spatially scaled version of  $f_k(x, y)$ , then

$$g_{\text{obs}}(a_x x, a_y y) = f_k(x, y) \quad (11.27a)$$

or

Now is the time for all good men and  
women to come to the aid of their party.

(a) Image of text

(b) Thresholded cross-correlation  $\rho_5(n_0, m_0)$ 

Now is the time for all good men and  
women to come to the aid of their party.

(c) Image of text (green) and thresholded cross-correlation (red)

**Figure 11-4** Identifying the letter “e” in Example 11-4.

$$g_{\text{obs}}(x, y) = f_k \left( \frac{x}{a_x}, \frac{y}{a_y} \right). \quad (11.27b)$$

The positive-value constants  $a_x$  and  $a_y$  are *unknown spatial scaling factors*. If  $a_x > 1$ ,  $g_{\text{obs}}(x, y)$  is  $f_k(x, y)$ , but magnified in the  $x$  direction by  $a_x$ . Conversely, if  $a_x < 1$ ,  $g(x, y)$  is  $f_k(x, y)$  but shrunk in the  $x$  direction by  $a_x$ . Similar variations apply to  $a_y$  along the  $y$  direction. An obvious example of image classification in the presence of spatial scaling is when the relative sizes of letters and numerals in the images to be classified are different from those in the reference images  $f_k(x, y)$ .

The spatial-scaling classification problem can be formulated as

$$g_{\text{obs}}(x, y) = \begin{cases} f_1 \left( \frac{x}{a_x}, \frac{y}{a_y} \right) & \text{class } \#1, \\ f_2 \left( \frac{x}{a_x}, \frac{y}{a_y} \right) & \text{class } \#2, \\ \vdots & \vdots \\ f_{\mathcal{L}} \left( \frac{x}{a_x}, \frac{y}{a_y} \right) & \text{class } \#\mathcal{L}, \end{cases} \quad (11.28)$$

where the true class is class  $\#K$  and all images are nonzero only for  $x, y > 0$ . Class  $K$  and variables  $a_x$ , and  $a_y$  are all unknown.

### 11-5.1 Logarithmic Spatial Transformation

Clearly, because  $g_{\text{obs}}$  and  $f_k$  have different spatial scales, the image classification problem cannot be solved using the same cross-correlation approach presented in earlier sections.

However, we can still solve the problem—and use cross-correlation—by performing *logarithmic warping* of spatial coordinates  $x$  and  $y$ . To that end, we change variables from  $(x, y)$  to  $(x', y')$ :

$$x' = \ln(x) \quad \longleftrightarrow \quad x = e^{x'}, \quad (11.29a)$$

$$y' = \ln(y) \quad \longleftrightarrow \quad y = e^{y'}. \quad (11.29b)$$

Next, we define *spatially warped images*  $g'_{\text{obs}}(x', y')$  and  $f'_k(x', y')$ :

$$g'_{\text{obs}}(x', y') = g_{\text{obs}}(e^{x'}, e^{y'}), \quad (11.30a)$$

$$f'_k(x', y') = f_k(e^{x'}, e^{y'}), \quad (11.30b)$$

and **logarithmically transformed scale factors** ( $a'_x, a'_y$ ):

$$a'_x = \ln(a_x) \iff a_x = e^{a'_x}, \quad (11.31a)$$

$$a'_y = \ln(a_y) \iff a_y = e^{a'_y}. \quad (11.31b)$$

Using these transformations, the spatially warped observation image  $g'_{\text{obs}}(x', y')$  can be related to the spatially warped reference image  $f'_k(x', y')$  as follows:

$$\begin{aligned} g'_{\text{obs}}(x', y') &= g_{\text{obs}}(e^{x'}, e^{y'}) = f_k \left( \frac{e^{x'}}{a_x}, \frac{e^{y'}}{a_y} \right) \\ &= f_k \left( \frac{e^{x'}}{e^{a'_x}}, \frac{e^{y'}}{e^{a'_y}} \right) \\ &= f_k(e^{(x' - a'_x)}, e^{(y' - a'_y)}) \\ &= f'_k(x' - a'_x, y' - a'_y). \end{aligned} \quad (11.32)$$

Hence, in the logarithmically transformed spatial variables  $(x', y')$ , spatial scaling by  $(a_x, a_y)$  becomes a spatial shift by  $(a'_x, a'_y)$ . An example is illustrated in **Fig. 11-5**.

The problem statement defined by Eq. (11.28) can now be reformulated as

$$g'_{\text{obs}}(x', y') = \begin{cases} f'_1(x' - a'_x, y' - a'_y) & \text{class } \#1, \\ f'_2(x' - a'_x, y' - a'_y) & \text{class } \#2, \\ \vdots & \vdots \\ f'_{\mathcal{L}}(x' - a'_x, y' - a'_y) & \text{class } \#\mathcal{L}, \end{cases} \quad (11.33)$$

where now the unknown true class is still class  $k = K$ , but the other unknowns are  $a'_x$  and  $a'_y$ . The formulation defined by Eq. (11.33) is the same as that of the spatial-shift classification problem given by Eq. (11.25) in the previous section. The cross-correlation between  $g'_{\text{obs}}(x', y')$  and  $f'_k(x', y')$  when the latter is spatially shifted by  $(x_0, y_0)$  is given by the convolution

$$\begin{aligned} \rho_k(x_0, y_0) &= \iint g'_{\text{obs}}(x', y') f'(x' - x_0, y' - y_0) dx' dy' \\ &= g'_{\text{obs}}(x_0, y_0) * f'(-x_0, -y_0). \end{aligned} \quad (11.34)$$

## 11-5.2 Classification Recipe

The following recipe is for continuous-space images. If the images are in discrete-space form, interpolation can be applied to convert them into continuous-space form using the methods described in Chapter 4.

The recipe for classifying observation image  $g_{\text{obs}}(x, y)$ , given

that its coordinates may have an unknown spatial scaling relative to those of reference images  $f_k(x, y)$ , consists of the following steps:

1. Use Eqs. (11.29) and (11.30) to transform all images  $g(x, y)$  and  $f_k(x, y)$  to logarithmic format  $g'(x', y')$  and  $f'_k(x', y')$ .
2. Use Eq. (11.34) to compute  $\rho_k(x_0, y_0)$  for each value of  $k$ :  $\{k = 1, 2, \dots, \mathcal{L}\}$ , and all spatial shifts  $(x_0, y_0)$  that offer nonzero overlap between  $g'(x', y')$  and  $f'_k(x' - x_0, y' - y_0)$ , using the 2-D CSFT.
3. Identify the combination of shift  $(x_0, y_0)$  and class  $k$  that yields the largest value of  $\rho_k(x_0, y_0)$ . Label that combination as  $(a'_x, a'_y)$  and  $k = \text{class } K$ .
4. With  $(a'_x, a'_y)$  known, the scaling factors  $(a_x, a_y)$  can then be determined using Eq. (11.31).

### Example 11-3: Scaled-Image Classification

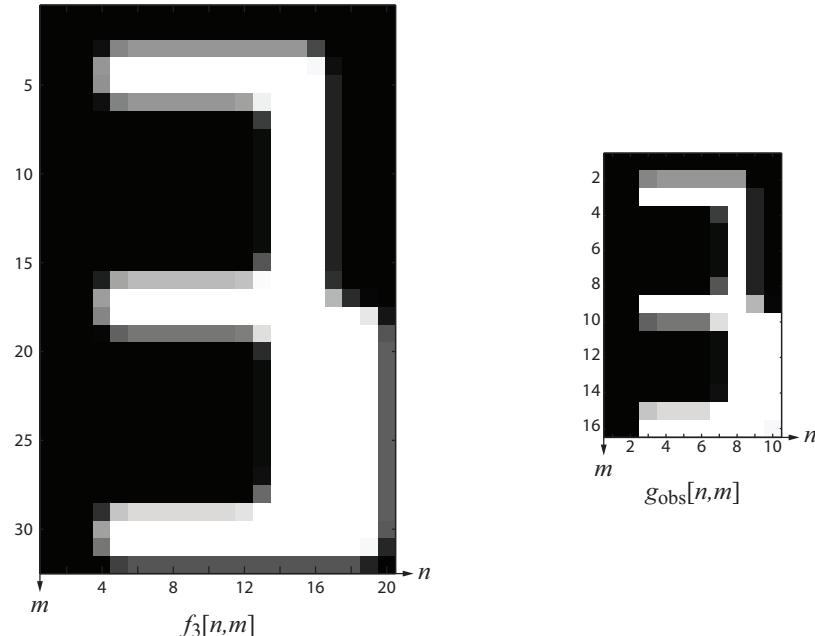
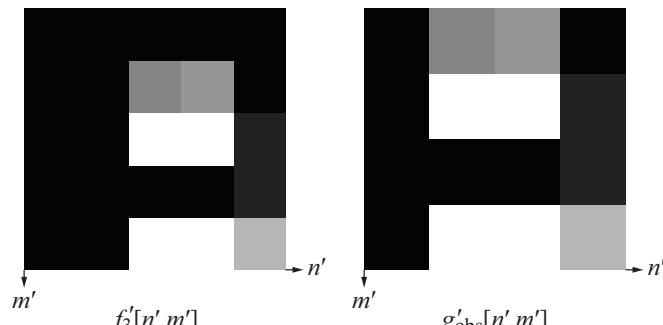
In **Fig. 11-6(a)**, we are given displays of 10 reference images of bank-font digits  $f_k[n, m]$ , with  $k = 1$  to 9 representing the numerals 1 to 9, respectively, and  $k = 10$  representing 0. Each image is  $(32 \times 20)$  in size, even though the original size was only  $(30 \times 18)$ ; the original image was zero-padded by inserting two rows and two columns of zeros at the top and left of the image. The zero-padding was used to ensure that the logarithmically transformed image had sufficient space to be shifted.

We also are given an observed image  $g_{\text{obs}}[n, m]$  shown in **Fig. 11-6(b)**, which we know to be a downsized scaled version of one of the 10 digits in **Fig. 11-6(a)**, with a scaling factor of 2 or 4, but we do not know which particular digit it represents, nor do we know the exact value of the scaling factor. [Actually,  $g_{\text{obs}}[n, m]$  is digit “3” at half-size, but that information is yet to be established by the image classifier.] Classify the unclassified image  $g_{\text{obs}}[n, m]$ .

**Solution:** In Section 11-4.1, we used a natural-log transformation to convert a **spatially scaled** continuous-space image  $g_{\text{obs}}(x, y)$  into a **spatially shifted** continuous-space image  $g'_{\text{obs}}(x', y')$ . For a discrete-space image  $g_{\text{obs}}[n, m]$ , we use base 2 logarithms:

$$n' = \log_2 n \iff n = 2^{n'}, \quad (11.35a)$$

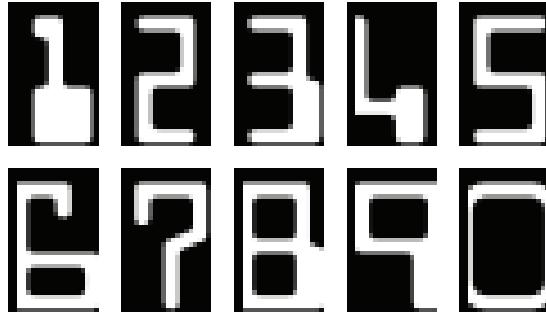
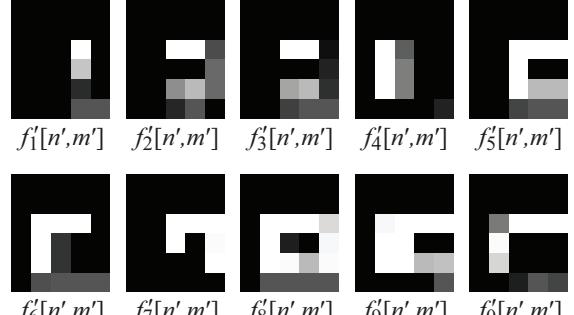
$$m' = \log_2 m \iff m = 2^{m'}. \quad (11.35b)$$

(a) Images in linear coordinates  $[n, m]$ (b) Images in logarithmic coordinates  $[n', m']$ 

$n$ and $m$	1	2	4	8	16	32
$n'$ and $m'$	0	1	2	3	4	5

(c) Relationship between  $[n, m]$  and  $[n', m']$ 

**Figure 11-5** (a)  $(32 \times 20)$  image  $f_3[n, m]$  and  $(16 \times 10)$  image  $g_{\text{obs}}[n, m]$ , (b) the same images in logarithmic format in base 2 (instead of base  $e$ ), also note that  $g'_{\text{obs}}[n', m'] = f'_3[n' + 1, m' + 1]$ , and (c) the relation between  $[n, m]$  and  $[n', m']$ . [Note that 32 pixels in  $[n, m]$  space transform into 6 (not 5) pixels in  $[n', m']$  space.]

(a) Reference images  $f_k[n,m]$ (b) Observed image  $g_{\text{obs}}[n,m]$ **Figure 11-6** (32 × 20) reference images and (unclassified) observed image of Example 11-3.(a) Reference images  $f'_k[n',m']$  in logarithmic format(b) Observed image  $g'_{\text{obs}}[n',m']$  in logarithmic format**Figure 11-7** (6 × 5) reference images and (5 × 4) (unknown) observation image in logarithmic format.

The spatially logarithmically transformed image  $g'_{\text{obs}}[n',m']$  in  $[n',m']$  space is

$$g'_{\text{obs}}[n',m'] = g_{\text{obs}}[2^{n'}, 2^{m'}]. \quad (11.36)$$

Based on the given information,

$$g_{\text{obs}} = f_k \left[ \frac{n}{a}, \frac{m}{a} \right] \quad \text{for } 1 \leq \frac{n}{a} \leq 20, \quad 1 \leq \frac{m}{a} \leq 32, \quad (11.37)$$

with the scaling factor  $a = 1/2$  or  $1/4$ . For convenience, we introduce the inverse scaling factor

$$b = \frac{1}{a} = 2^i, \quad \text{with } i = \begin{cases} 1 & \text{for } a = 1/2, \\ 2 & \text{for } a = 1/4. \end{cases} \quad (11.38)$$

Upon using Eq. (11.38) in Eq. (11.37) and then incorporating the scaling relationship into Eq. (11.36), we have

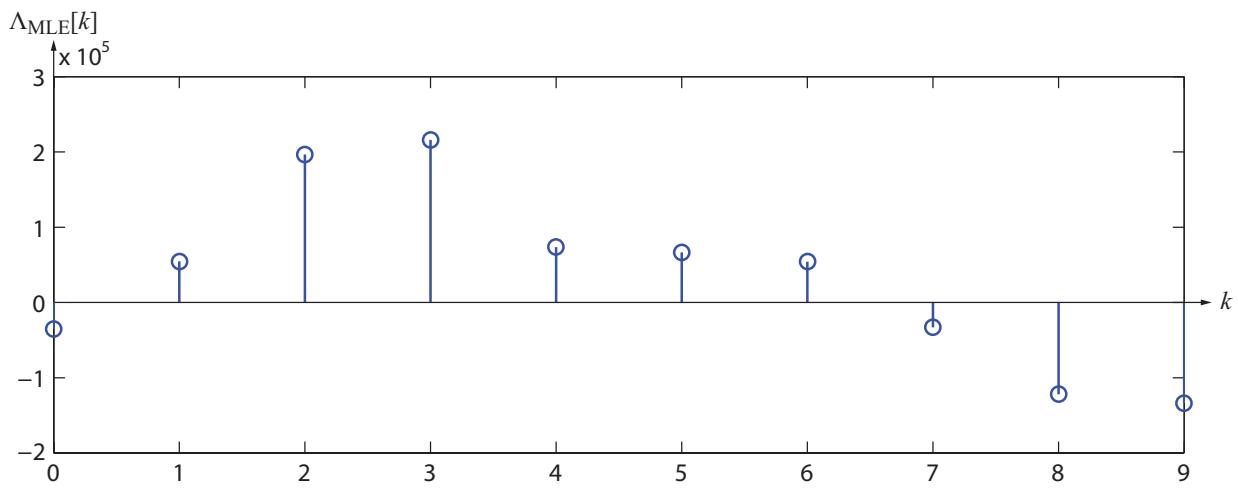
$$\begin{aligned} g'_{\text{obs}}[n',m'] &= f_k[2^{i n'}, 2^{i m'}] \\ &= f_k[2^{n'+i}, 2^{m'+i}] \\ &= f'_k[n'+i, m'+i], \end{aligned} \quad (11.39)$$

where in the last step we used the definition given by Eq. (11.36) to convert  $f_k$  into its logarithmic equivalent  $f'_k$ . The result encapsulated by Eq. (11.39) states that in  $[n',m']$  space, the observed image  $g'_{\text{obs}}[n',m']$  is a spatially shifted version of  $f'_k[n',m']$  and the shift is  $i$  pixels along each direction. Hence, application of the spatial-shift recipe outlined in Section 11-3 should lead to correctly identifying the values of both  $k$  and  $i$ . The validity of this statement is verified by Figs. 11-7 and 11-8. In Fig. 11-7, we show the reference and observed images in logarithmic format. Visual inspection reveals the close similarity between  $f'_3[n',m']$  and  $g_{\text{obs}}[n',m']$ , and that the spatial shift between them is  $i = 1$ , corresponding to  $a = 1/2$ .

Computational classification is realized by computing the MLE criterion  $\Lambda_{\text{MLE}}[k]$  as defined by Eq. (11.12):

$$\Lambda_{\text{MLE}}[k] = 2\rho'_k - E_{f'_k}, \quad (11.40)$$

where  $\rho'_k$  is the cross-correlation between  $g'_{\text{obs}}[n',m']$  and  $f'_k[n',m']$  and  $E_{f'_k}$  is the energy of image  $f'_k[n',m']$ . Because the energies of the 10 digits in  $[n',m']$  space vary widely (as evidenced by the wide range in the number of white pixels among the 10 digits in Fig. 11-7(a)), use of the MLE criterion



**Figure 11-8**  $\Lambda_{\text{MLE}}[k]$  based on correlation between noisy digit  $g'_{\text{obs}}[n, m]$  and each of the ten digits  $f'_k[n, m]$  for  $\{k = 0, 1, \dots, 10\}$ .

$\Lambda_{\text{MLE}}[k]$  provides a better classifier than  $\rho'_k$  alone. The computed values of  $\Lambda_{\text{MLE}}[k]$ , plotted in Fig. 11-8, show that  $\Lambda_{\text{MLE}}$  is largest for digit “3.” Note that because images  $f'_2[n, m]$  and  $f'_3[n, m]$  in Fig. 11-7(a) look similar, their computed MLE criteria (Fig. 11-8) are close in values.

**Concept Question 11-3:** In classification of spatially-shifted images, why not just correlate with all possible shifts of the image?

by

$$\begin{bmatrix} x = r \cos \theta \\ y = r \sin \theta \end{bmatrix} \leftrightarrow \begin{bmatrix} r = \sqrt{x^2 + y^2} \\ \theta = \tan^{-1}(y/x) \end{bmatrix}. \quad (11.41)$$

The image classification problem with unknown relative rotation  $\theta_0$  can then be reformulated as

$$\tilde{g}_{\text{obs}}(r, \theta) = \begin{cases} \tilde{f}_1(r, \theta - \theta_0) & \text{class \#1,} \\ \tilde{f}_2(r, \theta - \theta_0) & \text{class \#2,} \\ \vdots & \vdots \\ \tilde{f}_{\mathcal{L}}(r, \theta - \theta_0) & \text{class \#\mathcal{L},} \end{cases} \quad (11.42)$$

where  $\tilde{g}_{\text{obs}}(\cdot, \theta)$  and  $\tilde{f}_k(r, \theta)$  are  $g_{\text{obs}}(x, y)$  and  $f_k(x, y)$  in polar coordinates, respectively.

The form of Eq. (11.42) is the same as that of Eq. (11.20) for the spatial-shift problem, except that now the variables are  $(r, \theta)$  instead of  $(x, y)$ . In fact, the problem is relatively simpler because we only have one unknown spatial variable, namely  $\theta_0$ , rather than two. Using the cross-correlation concept that served us well in earlier sections, we define the **rotation cross-correlation**  $\tilde{\rho}_k(r, \theta)$  as

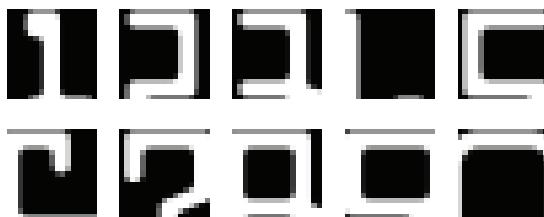
$$\tilde{\rho}_k(r, \theta) = \int \tilde{g}_{\text{obs}}(r, \theta') \tilde{f}_k(r, \theta' - \theta) d\theta'. \quad (11.43)$$

## 11-6 Classification of Rotated Images

We now consider the continuous-space and noiseless version of the image classification problem in which the observation  $g_{\text{obs}}(x, y)$  is a **rotated** (by an unknown angle  $\theta_0$ ) version of  $f_k(x, y)$ , or vice versa. As before,  $K$  is the unknown true value of  $k$ .

### 11-6.1 Polar-Coordinate Transformation

To incorporate the unknown angular rotation into the image classification problem, we reformulate the problem using polar coordinates  $(r, \theta)$  instead of Cartesian coordinates  $(x, y)$ . Per Fig. 3-11(a) and Eq. (3.44), the two pairs of variables are related

(a)  $(17 \times 17)$  clipped reference images(b) Upper half of numeral "3" rotated by  $90^\circ$ **Figure 11-9** Reference images and rotated observed image of Example 11-4.

## 11-6.2 Classification Recipe

1. Transform all images  $g_{\text{obs}}(x,y)$  and  $g_f(x,y)$  to polar coordinates format  $\tilde{g}_{\text{obs}}(r,\theta)$  and  $\tilde{f}_k(r,\theta)$ .
2. Use Eq. (11.43) to compute  $\tilde{\rho}_k(r,\theta)$  for every possible combination of  $k$  (among  $\mathcal{L}$  classes) and  $\theta$ , using the 2-D CSFT.
3. Identify the combination that yields the largest value of  $\tilde{\rho}_k(r,\theta)$ . Label  $k = K$  and  $\theta = \theta_0$ .

### Example 11-4: Rotated Image Classification

The  $(32 \times 20)$  reference images shown in **Fig. 11-6(a)** have been clipped to  $(17 \times 17)$ , so only the upper half of each image is shown in **Fig. 11-9(a)**. The  $90^\circ$ -rotated image of one of the numerals, namely numeral "3," is shown in **Fig. 11-9(b)**. Classify the rotated image, presuming that its identity is unknown, as is its degree of rotation.

**Solution:** The recipe in Subsection 11-5.2 relies on computing the cross-correlation  $\tilde{\rho}_k(r,\theta)$  defined by Eq. (11.43). Doing so requires two prerequisite steps, namely using interpolation to convert the reference images and the observation image from discrete-space format to continuous-space format, and then

applying Eq. (11.41) to convert the coordinates of the 11 images from Cartesian  $(x,y)$  to polar  $(r,\theta)$ . The transformed images are displayed in **Fig. 11-10**, with the horizontal axis representing  $\theta$  and the vertical axis representing  $r$ . Close examination of the images reveals that the observed image is identical with image "3" of the reference images, except for a  $90^\circ$  shift. Hence, it is not surprising that application of Eq. (11.43) against all 10 reference images and at many values of  $\theta$  between  $0^\circ$  and  $360^\circ$  produces the highest value for the cross-correlation when  $k = 3$  and  $\theta = 90^\circ$ .

**Concept Question 11-4:** When is interpolation unnecessary for classification of spatially scaled images?

## 11-7 Color Image Classification

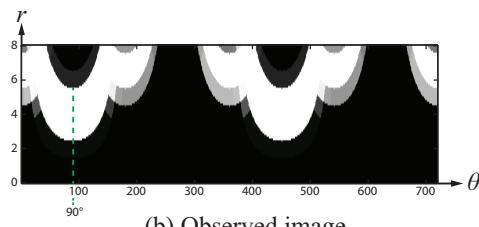
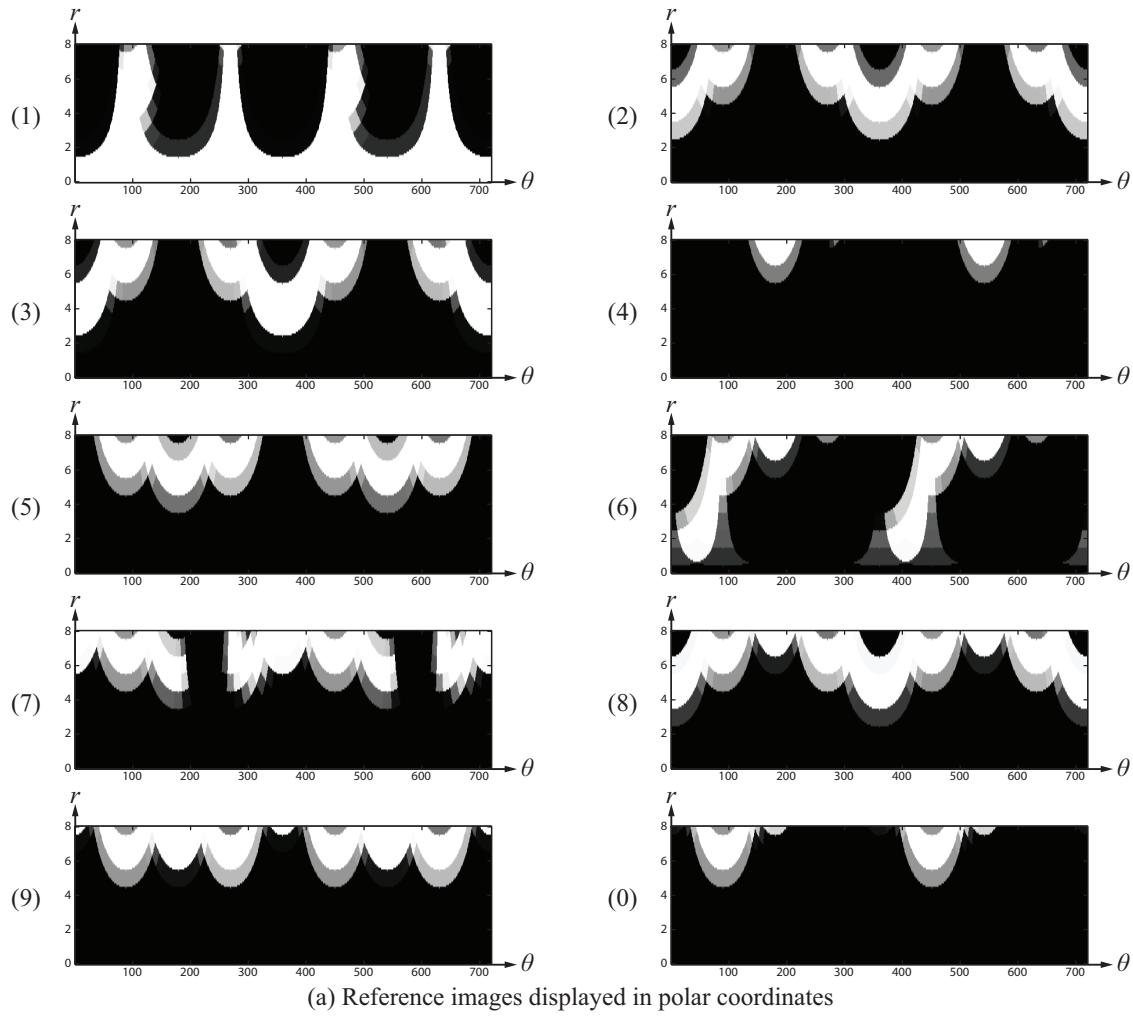
### 11-7.1 Notation for Color Images

We now extend the classification methods of the preceding section to color images consisting of red (R), green (G), and blue (B) channels. The treatment is equally applicable to non-optical sensors, such as imaging radars operating at three different wavelengths, and is extendable to multispectral images comprised of more than three channels.

In Sections 11-1 through 11-3, we considered the scenario where we were given a set of  $\mathcal{L}$  possible images  $\{f_k[n,m], k = 1, 2, \dots, \mathcal{L}\}$  and the goal was to classify a noisy image  $g_{\text{obs}}[n,m]$  into one of the  $\mathcal{L}$  classes. Now we consider the case where the  $\{f_k[n,m]\}$  images must themselves be estimated from a set of  $I$  **training images**  $\{f_k^{(i)}[n,m], k = 1, \dots, \mathcal{L}; i = 1, \dots, I\}$ . We will assume that all random variables are jointly Gaussian, so it will be necessary for us to estimate mean vectors and covariance matrices for each class  $k$  from these training images.

In the numeral classification example of Section 11-1, we had  $\mathcal{L} = 10$  classes of numerals, one observed image  $g_{\text{obs}}[n,m]$  and one known reference image  $f_k[n,m]$  for each value of  $k$ , with  $k$  extending between 1 and 10. With color images, the observed image consists of three channels, and so do all of the training images. Another consideration that we should now address is the fact that in many classification problems the identity of a given class is not unique. If we are dealing with numerals that are always printed using a single type of font, then we only need a single reference image per numeral (1 to 9, plus 0), but if we need to classify numerals that may have been printed using multiple types of fonts, then we would need a much larger set of reference images.

Similar **within-class** variations occur among text letters



**Figure 11-10** In polar coordinates, the observed image in (b) matches the reference image of “3”, shifted by  $90^\circ$ .

printed in different fonts, as well as in many object recognition applications.

With these two considerations (3-color channels and within-class variability) in mind, we introduce the following notation:

**Number of classes:**  $\mathcal{L}$ , with class index  $k = 1, 2, \dots, \mathcal{L}$ .

**Image size:**  $M \times N$

**Number of training images per class:**  $I$ , with training image index  $i = 1, 2, \dots, I$ .

**Red channel  $i$ th training image for class  $k$ :**  $f_{k,R}^{(i)}[n,m]$

**$i$ th training image vector of 2-D functions of  $[n,m]$  for class  $k$ :**

$$\mathbf{f}_k^{(i)}[n,m] = [f_{k,R}^{(i)}[n,m], f_{k,G}^{(i)}[n,m], f_{k,B}^{(i)}[n,m]]^T \quad (11.44)$$

**Mean red channel training image for class  $k$ :**

$$\bar{f}_{k,R}[n,m] = \frac{1}{I} \sum_{i=1}^I f_{k,R}^{(i)}[n,m] \quad (11.45)$$

**Mean training image vector for class  $k$ :**

$$\bar{\mathbf{f}}_k[n,m] = [\bar{f}_{k,R}[n,m], \bar{f}_{k,G}[n,m], \bar{f}_{k,B}[n,m]]^T \quad (11.46)$$

**Covariance matrix estimate for class  $k$  at location  $[n,m]$ :**

$$\begin{aligned} \mathbf{K}_k[n,m] &= \frac{1}{I} \sum_{i=1}^I (\mathbf{f}_k^{(i)}[n,m] - \bar{\mathbf{f}}_k[n,m])(\mathbf{f}_k^{(i)}[n,m] - \bar{\mathbf{f}}_k[n,m])^T \\ &= \frac{1}{I} \sum_{i=1}^I [\mathbf{f}_k^{(i)}[n,m](\mathbf{f}_k^{(i)}[n,m])^T - \bar{\mathbf{f}}_k[n,m](\bar{\mathbf{f}}_k[n,m])^T]. \end{aligned} \quad (11.47)$$

The estimated covariance matrix  $\mathbf{K}_k[n,m]$  accounts for the variability at location  $[n,m]$  among the  $I$  training images, relative to their mean  $\bar{\mathbf{f}}_k[n,m]$ , and correlation between different colors in an image.

**Location-independent covariance matrix for class  $k$ :**

$$\bar{\mathbf{K}}_k = \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \mathbf{K}_k[n,m] \quad (11.48)$$

The **location-independent covariance matrix**  $\bar{\mathbf{K}}_k$  is the sample mean over  $[n,m]$  of  $\mathbf{K}_k[n,m]$ . It is used to represent the in-class variability when  $\mathbf{K}_k[n,m]$  is statistically independent of location  $[n,m]$ , which is a valid assumption in many applications.

### Observed-image model for class $k$ :

$$\mathbf{g}_{\text{obs}}[n,m] = \mathbf{f}_k[n,m] + \mathbf{v}[n,m], \quad (11.49)$$

where  $\mathbf{v}[n,m]$  is a vector of length 3 (one for each color), modeled as a zero-mean white Gaussian noise random field with variance  $\sigma_v^2$  for each color component.

**Gaussian model for  $\mathbf{g}_{\text{obs}}[n,m]$ :**

$$\begin{aligned} \mathbf{g}_{\text{obs}}[n,m] &= [g_{k,R}[n,m], g_{k,G}[n,m], g_{k,B}[n,m]]^T \\ &\sim \mathcal{N}([\bar{\mathbf{f}}_k[n,m]], \mathbf{K}_{k0}), \end{aligned} \quad (11.50)$$

where  $\mathbf{K}_{k0}$  is a  $(3 \times 3)$  **joint covariance matrix** that accounts for both  $\bar{\mathbf{K}}_k$ , the in-class variability of the noise-free training images, and the added noise represented by  $\sigma_v^2$ :

$$\mathbf{K}_{k0} = \bar{\mathbf{K}}_k + \sigma_v^2 \mathbf{I}_3, \quad (11.51)$$

where  $\mathbf{I}_3$  is the  $(3 \times 3)$  identity matrix.

## 11-7.2 Likelihood Functions

Even though for class  $k$ , vector  $\bar{\mathbf{f}}_k[n,m]$  and covariance matrix  $\bar{\mathbf{K}}_k$  are both estimated from the vectors of the  $I$  training images  $\{\mathbf{f}_k^{(i)}[n,m]\}$ , we use them as the “true” values, not just estimates, in the classification algorithms that follow. The goal of the classification algorithm is to determine the value of the class index  $k$  from among the  $\mathcal{L}$  possible classes.

Given an observation image vector  $\mathbf{g}_{\text{obs}}[n,m]$  and an average reference image vector  $\bar{\mathbf{f}}_k[n,m]$  for each class index  $k$ , we define the difference image vector  $\Delta_k[n,m]$  as

$$\begin{aligned} \Delta_k[n,m] &= \mathbf{g}_{\text{obs}}[n,m] - \bar{\mathbf{f}}_k[n,m] \\ &= [g_{k,R}[n,m], g_{k,G}[n,m], g_{k,B}[n,m]]^T \\ &\quad - [\bar{f}_{k,R}[n,m], \bar{f}_{k,G}[n,m], \bar{f}_{k,B}[n,m]]^T. \end{aligned} \quad (11.52)$$

By treating  $\bar{\mathbf{f}}_k[n,m]$  as the mean value of  $\mathbf{g}_{\text{obs}}[n,m]$ , and in view of the model given by Eq. (11.50), the location-specific **marginal pdf**  $p(\mathbf{g}_{\text{obs}}[n,m])$  of length-3 Gaussian random vector  $\mathbf{g}_{\text{obs}}[n,m]$  for class  $k$  is given by

$$\begin{aligned} p(\mathbf{g}_{\text{obs}}[n,m]) &= \frac{1}{(2\pi)^{3/2} \sqrt{\det(\mathbf{K}_{k0})}} \\ &\quad \times e^{-\frac{1}{2} ((\Delta_k[n,m])^T \mathbf{K}_{k0}^{-1} \Delta_k[n,m])}. \end{aligned} \quad (11.53)$$

The  $\{\mathbf{g}_{\text{obs}}[n,m]\}$  values at different locations  $[n,m]$  are independent random vectors because noise vectors  $\mathbf{v}[n,m]$  are location-

independent. Accordingly, their joint pdf is

$$\begin{aligned} p(\{\mathbf{g}_{\text{obs}}[n, m]\}) &= \prod_{n=0}^{N-1} \prod_{m=0}^{M-1} p(\mathbf{g}_{\text{obs}}[n, m]) \\ &= \frac{1}{(2\pi)^{3NM/2}} \frac{1}{(\det(\mathbf{K}_{k0}))^{NM/2}} \\ &\quad \times \prod_{n=0}^{N-1} \prod_{m=0}^{M-1} e^{-\frac{1}{2}((\Delta_k[n, m])^T \mathbf{K}_{k0}^{-1} \Delta_k[n, m])} \\ &= \frac{1}{(2\pi)^{3NM/2}} \frac{1}{(\det(\mathbf{K}_{k0}))^{NM/2}} \\ &\quad \times e^{-\frac{1}{2}(\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (\Delta_k[n, m])^T \mathbf{K}_{k0}^{-1} \Delta_k[n, m])}. \end{aligned} \quad (11.54)$$

The joint pdf given by Eq. (11.54) is the **likelihood function** of the observed image vector  $\mathbf{g}_{\text{obs}}[n, m]$ . An expanded version for the individual classes is given by

$$\begin{aligned} p(\{\mathbf{g}_{\text{obs}}[n, m]\}) &= \frac{1}{(2\pi)^{3NM/2}} \\ &\times \left\{ \begin{array}{ll} \frac{1}{(\det(\mathbf{K}_{10}))^{NM/2}} \prod_{n=0}^{N-1} \prod_{m=0}^{M-1} e^{-\frac{1}{2}((\Delta_1[n, m])^T \mathbf{K}_{10}^{-1} \Delta_1[n, m])} & \text{class 1,} \\ \frac{1}{(\det(\mathbf{K}_{20}))^{NM/2}} \prod_{n=0}^{N-1} \prod_{m=0}^{M-1} e^{-\frac{1}{2}((\Delta_2[n, m])^T \mathbf{K}_{20}^{-1} \Delta_2[n, m])} & \text{class 2,} \\ \vdots & \\ \frac{1}{(\det(\mathbf{K}_{L0}))^{NM/2}} \prod_{n=0}^{N-1} \prod_{m=0}^{M-1} e^{-\frac{1}{2}((\Delta_L[n, m])^T \mathbf{K}_{L0}^{-1} \Delta_L[n, m])} & \text{class } L. \end{array} \right. \end{aligned} \quad (11.55)$$

The corresponding natural log-likelihood function is given by

$$\begin{aligned} \ln(p(\{\mathbf{g}_{\text{obs}}[n, m]\})) &= -\frac{3NM}{2} \ln(2\pi) \\ &\quad - \frac{NM}{2} \ln(\det(\mathbf{K}_{10})) \\ &\quad - \frac{1}{2} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} ((\Delta_1[n, m])^T \mathbf{K}_{10}^{-1} \Delta_1[n, m]) \quad \text{class 1,} \\ &\quad - \frac{NM}{2} \ln(\det(\mathbf{K}_{20})) \\ &\quad - \frac{1}{2} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} ((\Delta_2[n, m])^T \mathbf{K}_{20}^{-1} \Delta_2[n, m]) \quad \text{class 2,} \\ &\quad \vdots \\ &\quad - \frac{NM}{2} \ln(\det(\mathbf{K}_{L0})) \\ &\quad - \frac{1}{2} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} ((\Delta_L[n, m])^T \mathbf{K}_{L0}^{-1} \Delta_L[n, m]) \quad \text{class } L. \end{aligned} \quad (11.56)$$

### 11-7.3 Classification by MLE

The maximum likelihood estimate  $\hat{k}_{\text{MLE}}$  of class  $k$  is the value of  $k$  that maximizes the log-likelihood function given by Eq. (11.56). Since the first term  $[-(3NM/2)\ln(2\pi)]$  is common to all terms, it has no impact on the choice of  $k$ . Hence,  $\hat{k}_{\text{MLE}}$  is the value of  $k$  that maximizes the **MLE criterion**  $\Lambda_1[k]$  given by

$$\Lambda_1[k] = -NM \ln(\det(\mathbf{K}_{k0})) - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} ((\Delta_k[n, m])^T \mathbf{K}_{k0}^{-1} \Delta_k[n, m]). \quad (11.57)$$

According to Eq. (11.51), the joint covariance matrix  $\mathbf{K}_{k0}$  incorporates two statistical variations,  $\mathbf{K}_k$  due to variations among training images for class  $k$ , and  $\sigma_v^2$  due to the added noise. If the classification application is such that  $\mathbf{K}_k$  does not vary with class  $k$  (or the variation from class to class is relatively minor), we then can treat  $\mathbf{K}_k$  as a class-independent covariance matrix  $\mathbf{K}$ , with a corresponding joint covariance matrix  $\mathbf{K}_0 = \mathbf{K} + \sigma_v^2 \mathbf{I}_3$ . As a consequence, the first term in Eq. (11.57) becomes class-independent and can be removed from the maximization process, which leads us to adjust the definition of the MLE criterion to

$$\Lambda_1[k] = - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} ((\Delta_k[n, m])^T \mathbf{K}_0^{-1} \Delta_k[n, m]). \quad (11.58)$$

Upon replacing  $\Delta_k[n, m]$  with its defining expression given by Eq. (11.52), Eq. (11.58) becomes

$$\begin{aligned}\Lambda_1[k] &= - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} ((\Delta_k[n, m])^T \mathbf{K}_0^{-1} \Delta_k[n, m]) \\ &= - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (\mathbf{g}_{\text{obs}}[n, m] - \bar{\mathbf{f}}_k[n, m])^T \mathbf{K}_0^{-1} \\ &\quad \times (\mathbf{g}_{\text{obs}}[n, m] - \bar{\mathbf{f}}_k[n, m]) \\ &= - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \mathbf{g}_{\text{obs}}[[n, m]] \mathbf{K}_0^{-1} \mathbf{g}_{\text{obs}}[n, m] \\ &\quad - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \bar{\mathbf{f}}_k[n, m] \mathbf{K}_0^{-1} \bar{\mathbf{f}}_k[n, m] \\ &\quad + 2 \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} ((\mathbf{g}_{\text{obs}}[n, m])^T \mathbf{K}_0^{-1} \bar{\mathbf{f}}_k[n, m]). \quad (11.59)\end{aligned}$$

The last term comes from noting that a scalar equals its own transpose, which is useful in matrix algebra.

The first term in Eq. (11.59) is independent of  $k$ . Hence, computation of  $\hat{k}_{\text{MLE}}$  simplifies to choosing the value of  $k$  that maximizes the **modified MLE criterion**

$$\begin{aligned}\Lambda_{\text{MLE}}[k] &= 2 \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} ((\mathbf{g}_{\text{obs}}[n, m])^T \mathbf{K}_0^{-1} \bar{\mathbf{f}}_k[n, m]) \\ &\quad - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \bar{\mathbf{f}}_k[n, m] \mathbf{K}_0^{-1} \bar{\mathbf{f}}_k[n, m]. \quad (11.60)\end{aligned}$$

The expression given by Eq. (11.60) is the vector version of the expression given earlier in Eq. (11.11) for the scalar case.

additional term:

$$\begin{aligned}\Lambda_{\text{MAP}}[k] &= 2 \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (\mathbf{g}_{\text{obs}}[n, m])^T \mathbf{K}_0^{-1} \bar{\mathbf{f}}_k[n, m] \\ &\quad - \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \bar{\mathbf{f}}_k[n, m] \mathbf{K}_0^{-1} \bar{\mathbf{f}}_k[n, m] + 2 \ln(p[k]). \quad (11.61)\end{aligned}$$

The MAP classifier (estimator of  $k$ ) selects the value of  $k$  that maximizes  $\Lambda_{\text{MAP}}[k]$ .

### Example 11-5: Color Image Classification

Develop a classification rule for a two-class,  $(1 \times 1)$  color image classifier for a  $(1 \times 1)$  image  $[g_R, g_G, g_B]$  with no additive noise and equal class probabilities ( $p[k = 1] = p[k = 2]$ ), given the following 4 training images per class:

Class  $k = 1$ :

$$\mathbf{f}_1^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{f}_1^{(2)} = \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix}, \mathbf{f}_1^{(3)} = \begin{bmatrix} 4 \\ 4 \\ 0 \end{bmatrix}, \mathbf{f}_1^{(4)} = \begin{bmatrix} 4 \\ 0 \\ 4 \end{bmatrix}, \quad (11.62)$$

Class  $k = 2$ :

$$\mathbf{f}_2^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix}, \mathbf{f}_2^{(2)} = \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix}, \mathbf{f}_2^{(3)} = \begin{bmatrix} 0 \\ 4 \\ 4 \end{bmatrix}, \mathbf{f}_2^{(4)} = \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix}. \quad (11.63)$$

**Solution:** The sample means of the four training sets are

$$\bar{\mathbf{f}}_1 = \frac{1}{4} \left( \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \\ 0 \end{bmatrix} + \begin{bmatrix} 4 \\ 0 \\ 4 \end{bmatrix} \right) = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \quad (11.64)$$

and

$$\bar{\mathbf{f}}_2 = \frac{1}{4} \left( \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix} + \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix}. \quad (11.65)$$

### 11-7.4 Classification by MAP

An noted earlier in connection with Fig. 11-3, the probability of occurrence  $p[k]$  of the letters of the alphabet varies widely among the 26 letters of the English language. The same may be true for other classification applications. The maximum *a priori* (MAP) classifier takes advantage of this *a priori* information by multiplying the likelihood function given by Eq. (11.54) by  $p[k]$ . This modification leads to a MAP classification criterion  $\Lambda_{\text{MAP}}[k]$  given by the same expression for  $\Lambda_{\text{MLE}}[k]$ , but with an

The sample covariance for  $k = 1$  is

$$\begin{aligned}\bar{\mathbf{K}}_1 &= \frac{1}{4} \left( \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} [0, 0, 0] + \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix} [4, 0, 0] \right) \\ &\quad + \frac{1}{4} \left( \begin{bmatrix} 4 \\ 4 \\ 0 \end{bmatrix} [4, 4, 0] + \begin{bmatrix} 4 \\ 0 \\ 4 \end{bmatrix} [4, 0, 4] \right) \\ &\quad - \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} [3, 1, 1] = \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix},\end{aligned}\quad (11.66)$$

and the sample covariance for  $k = 2$  is

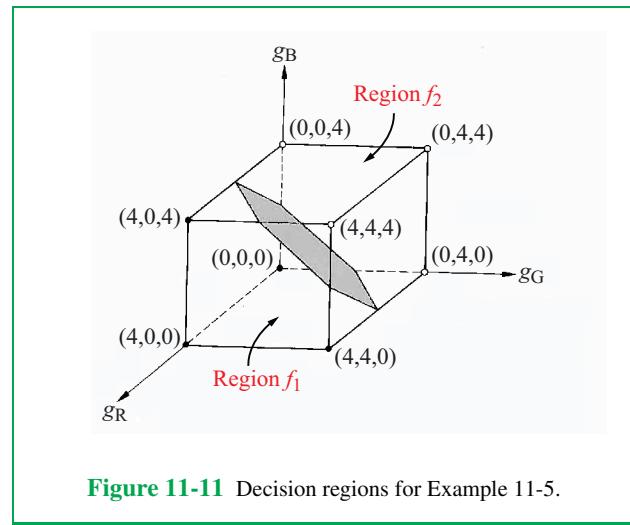
$$\begin{aligned}\bar{\mathbf{K}}_2 &= \frac{1}{4} \left( \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix} [0, 0, 4] + \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix} [0, 4, 0] \right) \\ &\quad + \frac{1}{4} \left( \begin{bmatrix} 0 \\ 4 \\ 4 \end{bmatrix} [0, 4, 4] + \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix} [4, 4, 4] \right) \\ &\quad - \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix} [1, 3, 3] = \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix}.\end{aligned}\quad (11.67)$$

Because  $\bar{\mathbf{K}}_1 = \bar{\mathbf{K}}_2$ , Eq. (11.60) applies. The classification rule is: Choose the value of  $k$  that maximizes  $\Lambda_{MLE}[k]$  in Eq. (11.60). Setting  $\bar{\mathbf{K}}_1 = \bar{\mathbf{K}}_2 = \bar{\mathbf{K}}$ , the inverse of  $\bar{\mathbf{K}}$  is

$$\bar{\mathbf{K}}^{-1} = \frac{1}{4} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix}. \quad (11.68)$$

For  $k = 1$ ,  $\Lambda_{MLE}[1]$  of Eq. (11.60) becomes

$$\begin{aligned}\Lambda_{MLE}[1] &= 2[g_R, g_G, g_B] \frac{1}{4} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \\ &\quad - [3, 1, 1] \frac{1}{4} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \\ &= 2g_R - 3,\end{aligned}\quad (11.69)$$



**Figure 11-11** Decision regions for Example 11-5.

and for  $k = 2$ ,

$$\begin{aligned}\Lambda_{MLE}[2] &= 2[g_R, g_G, g_B] \frac{1}{4} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix} \\ &\quad - [1, 3, 3] \frac{1}{4} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix} \\ &= 4g_G + 4g_B - 2g_R - 11.\end{aligned}\quad (11.70)$$

The classification rule (choose the larger of  $\Lambda_{MLE}[1]$  and  $\Lambda_{MLE}[2]$ ) simplifies to the following simple sign test:

- Choose  $f_1$  if:  $-4g_R + 4g_G + 4g_B - 8 < 0$ ,
- Choose  $f_2$  if:  $-4g_R + 4g_G + 4g_B - 8 > 0$ .

In 3-D space, with axes  $\{g_R, g_G, g_B\}$ , the boundary is a plane that separates the regions in which  $\{g_R, g_G, g_B\}$  is assigned to  $f_1$  and in which it is assigned to  $f_2$ . The regions are shown in **Fig. 11-11**.

#### Example 11-6: $(2 \times 2)$ Image Classifier

We are given two  $(2 \times 2)$  color image classes:

$$f_{1,R} = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}, \quad f_{1,G} = \begin{bmatrix} 4 & 5 \\ 6 & 0 \end{bmatrix}, \quad f_{1,B} = \begin{bmatrix} 7 & 8 \\ 9 & 0 \end{bmatrix},$$

$$f_{2,R} = \begin{bmatrix} 3 & 2 \\ 1 & 0 \end{bmatrix}, \quad f_{2,G} = \begin{bmatrix} 6 & 5 \\ 4 & 0 \end{bmatrix}, \quad f_{2,B} = \begin{bmatrix} 9 & 8 \\ 7 & 0 \end{bmatrix}. \quad (11.71)$$

Find the rule for classifying an observed  $(2 \times 2)$  color image

$$\begin{aligned} g_R &= \begin{bmatrix} g_{0,0}^R & g_{0,1}^R \\ g_{1,0}^R & g_{1,1}^R \end{bmatrix}, \\ g_G &= \begin{bmatrix} g_{0,0}^G & g_{0,1}^G \\ g_{1,0}^G & g_{1,1}^G \end{bmatrix}, \\ g_B &= \begin{bmatrix} g_{0,0}^B & g_{0,1}^B \\ g_{1,0}^B & g_{1,1}^B \end{bmatrix}. \end{aligned} \quad (11.72)$$

Assume  $\bar{\mathbf{K}}_1 = \bar{\mathbf{K}}_2 = \mathbf{I}$  and equal *a priori* probabilities.

**Solution:** The second term in Eq. (11.60) represents the energy  $E_{\tilde{\mathbf{f}}_k}$  of class  $k$ . By inspection  $E_{\tilde{\mathbf{f}}_1} = E_{\tilde{\mathbf{f}}_2}$  ( $\tilde{\mathbf{f}}_1$  and  $\tilde{\mathbf{f}}_2$  have the same numbers). Hence the second term in Eq. (11.60) is the same for  $k = 1$  and 2, so we can ignore it. Consequently, Eq. (11.60) simplifies to

$$\begin{aligned} \Lambda_{MLE}[1] &= 1g_{0,0}^R + 2g_{0,1}^R + 3g_{1,0}^R \\ &\quad + 4g_{0,0}^G + 5g_{0,1}^G + 6g_{1,0}^G \\ &\quad + 7g_{0,0}^B + 8g_{0,1}^B + 9g_{1,0}^B, \end{aligned} \quad (11.73)$$

$$\begin{aligned} \Lambda_{MLE}[2] &= 3g_{0,0}^R + 2g_{0,1}^R + 1g_{1,0}^R \\ &\quad + 6g_{0,0}^G + 5g_{0,1}^G + 4g_{1,0}^G \\ &\quad + 9g_{0,0}^B + 8g_{0,1}^B + 7g_{1,0}^B. \end{aligned} \quad (11.74)$$

**Classification Rule:** Choose the larger of  $\Lambda_{MLE}[1]$  and  $\Lambda_{MLE}[2]$ . Note that  $g_{1,1}$  is not used for any color, as expected.

**Concept Question 11-5:** How do we estimate image vectors  $\mathbf{f}_k[n, m]$  from the training image vectors  $\mathbf{f}_k^{(i)}[n, m]$ ?

## 11-8 Unsupervised Learning and Classification

In *unsupervised learning* we are given a set of  $\mathcal{L}$  **training images**. We know nothing about what the images are supposed to represent, or to what classes they belong, or even what the classes are. The goals of unsupervised learning and classification include:

- (1) To identify classes from the training images.
- (2) To classify each of the training images.

- (3) To provide a simple rule for classifying any new image into one of the classes identified by the training images.

### 11-8.1 Singular Value Decomposition

Before diving into the mechanics of unsupervised learning and classification, we provide an overview of an important matrix tool known as **singular value decomposition (SVD)**.

The SVD is used to factorize an  $(M \times N)$  matrix  $\mathbf{F}$  into the product of two **orthogonal** matrices  $\mathbf{U}$  and  $\mathbf{V}$ , and one diagonal matrix  $\mathbf{S}$ :

$$\mathbf{F} = \mathbf{U}\mathbf{S}\mathbf{V}^T. \quad (11.75)$$

The four matrices have the following attributes:

- (1)  $\mathbf{F}$  is any  $(M \times N)$  matrix.
- (2)  $\mathbf{U}$  is an  $(M \times M)$  orthogonal matrix, which means

$$\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I}_M,$$

where  $\mathbf{I}_M$  is an  $(M \times M)$  identity matrix.

- (3)  $\mathbf{V}$  is an  $(N \times N)$  orthogonal matrix:

$$\mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T = \mathbf{I}_N.$$

- (4)  $\mathbf{S}$  is an  $(M \times N)$  diagonal matrix, of the following form:

- (a) If  $M \geq N$ :

$$\mathbf{S} = \begin{bmatrix} \text{diag}[\sigma_j] \\ \mathbf{0}_{M-N, N} \end{bmatrix},$$

where  $\text{diag}[\sigma_j]$  is an  $M \times M$  diagonal matrix of  $\{\sigma_j\}$ , and in this case  $\mathbf{F}$  and  $\mathbf{S}$  are called **tall matrices**.

- (b) If  $M \leq N$ :

$$\mathbf{S} = \begin{bmatrix} \text{diag}[\sigma_j] & \mathbf{0}_{M, N-M} \end{bmatrix},$$

where  $\text{diag}[\sigma_j]$  is an  $M \times M$  diagonal matrix of  $\{\sigma_j\}$ , and in this case  $\mathbf{F}$  and  $\mathbf{S}$  are called **reclining matrices**.

Note that  $\mathbf{0}_{m,n}$  is an  $(m \times n)$  matrix of zeros.

The  $\{\sigma_j\}$  are called **singular values**. The number of singular values is the smaller of  $M$  and  $N$ . Note that  $\mathbf{F}$  and  $\mathbf{S}$  both have the same size  $(M \times N)$ . More information on  $\{\sigma_j\}$  is presented shortly.

### 11-8.2 SVD Computation by Eigendecomposition

In this book, we do not prove the existence of the SVD, nor review algorithms for computing it. However, we offer the

following simple approach to compute the SVD of  $\mathbf{F}$  by noting that

$$\begin{aligned}\mathbf{FF}^T &= (\mathbf{USV}^T)(\mathbf{VS}^T\mathbf{U}^T) \\ &= (\mathbf{US})(\mathbf{V}^T\mathbf{V})(\mathbf{S}^T\mathbf{U}) = \mathbf{U} \operatorname{diag}[\sigma_j^2] \mathbf{U}^T\end{aligned}\quad (11.76a)$$

and

$$\begin{aligned}\mathbf{F}^T\mathbf{F} &= (\mathbf{VS}^T\mathbf{U}^T)(\mathbf{USV}^T) \\ &= (\mathbf{VS}^T)(\mathbf{U}^T\mathbf{U})(\mathbf{SV}^T) = \mathbf{V} \operatorname{diag}[\sigma_j^2] \mathbf{V}^T.\end{aligned}\quad (11.76b)$$

These relationships rely on the fact that  $\mathbf{U}$  and  $\mathbf{V}$  are orthogonal matrices:  $\mathbf{VV}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$ .

As demonstrated shortly in the SVD example, the eigenvectors of  $\mathbf{FF}^T$  constitute the columns of  $\mathbf{U}$ , the eigenvectors of  $\mathbf{F}^T\mathbf{F}$  constitute the columns of  $\mathbf{V}$ , and the  $\{\sigma_j\}$  are the nonzero eigenvalues of  $\mathbf{FF}^T$  and of  $\mathbf{F}^T\mathbf{F}$ . Given  $\mathbf{V}$  and  $\{\sigma_j^2\}$ , or  $\mathbf{U}$  and  $\{\sigma_j^2\}$ , we can compute  $\mathbf{U}$  or  $\mathbf{V}$  as follows:

$$\mathbf{U} = \mathbf{FV} \operatorname{diag}[\sigma_j^{-1}], \quad (11.77a)$$

$$\mathbf{V}^T = \operatorname{diag}[\sigma_j^{-1}] \mathbf{U}^T \mathbf{F}, \quad (11.77b)$$

both of which follow directly from  $\mathbf{F} = \mathbf{USV}^T$ .

► The SVD of a matrix  $\mathbf{A}$  can be computed using MATLAB by the command  $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \operatorname{svd}(\mathbf{A})$ ; ◀

### Example 11-7: Compute SVD

Matrix  $\mathbf{F}$  is given by

$$\mathbf{F} = \begin{bmatrix} 96 & 39 & -40 \\ -72 & 52 & 30 \end{bmatrix}. \quad (11.78a)$$

Compute the SVD of  $\mathbf{F}$  (a) by MATLAB and (b) by eigendecomposition of  $\mathbf{FF}^T$ .

**Solution:** (a) The MATLAB command  $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \operatorname{svd}(\mathbf{F})$  yields

$$\mathbf{U} = \begin{bmatrix} 4/5 & 3/5 \\ -3/5 & 4/5 \end{bmatrix}, \quad (11.78b)$$

$$\mathbf{S} = \begin{bmatrix} 130 & 0 & 0 \\ 0 & 65 & 0 \end{bmatrix}, \quad (11.78c)$$

$$\mathbf{V} = \begin{bmatrix} 12/13 & 0 & 5/13 \\ 0 & 1 & 0 \\ -5/13 & 0 & 12/13 \end{bmatrix}. \quad (11.78d)$$

The two singular values are  $\sigma_1 = 130$  and  $\sigma_2 = 65$ .

The singular values are usually put in decreasing order  $\sigma_1 > \sigma_2 > \dots$  by reordering the columns of  $\mathbf{U}$  and the rows of  $\mathbf{V}^T$ .

(b) Using Eq. (11.78a), we compute

$$\begin{aligned}\mathbf{FF}^T &= \begin{bmatrix} 96 & 39 & -40 \\ -72 & 52 & 30 \end{bmatrix} \begin{bmatrix} 96 & -72 \\ 39 & 52 \\ -40 & 30 \end{bmatrix} \\ &= \begin{bmatrix} 12337 & -6084 \\ -6084 & 8788 \end{bmatrix}.\end{aligned}$$

Postmultiplying  $\mathbf{FF}^T$  in Eq. (11.76a) by  $\mathbf{U}$  and using  $\mathbf{U}^T\mathbf{U} = \mathbf{I}$  gives

$$(\mathbf{FF}^T)\mathbf{U} = \mathbf{U} \operatorname{diag}[\sigma_j^2]. \quad (11.78e)$$

Next, let  $\mathbf{u}_j$  be the  $j$ th column of  $(M \times M)$  matrix  $\mathbf{U}$ . Then the  $j$ th column of Eq. (11.78e) is

$$(\mathbf{FF}^T)\mathbf{u}_j = \sigma_j^2 \mathbf{u}_j, \quad (11.78f)$$

This is because postmultiplying  $\mathbf{U}$  by  $\operatorname{diag}[\sigma_j^2]$  multiplies the  $j$ th column of  $\mathbf{U}$  by  $\sigma_j^2$ , and  $\mathbf{U}$  is an orthogonal matrix characterized by  $\mathbf{u}_i^T \mathbf{u}_j = \delta[i-j]$ . Equation (11.78f) states that  $\mathbf{u}_j$  is an eigenvector of  $\mathbf{FF}^T$  with associated eigenvalue  $\sigma_j^2$ , which means that  $(\mathbf{FF}^T - \sigma_j^2 \mathbf{I})$  is singular and therefore its determinant is zero:

$$\det(\mathbf{FF}^T - \sigma_j^2 \mathbf{I}) = 0. \quad (11.78g)$$

The roots of this quadratic polynomial are  $\sigma_1^2$  and  $\sigma_2^2$ , which can be computed by inserting the matrix  $\mathbf{FF}^T$  computed earlier into Eq. (11.78g):

$$\det \left( \begin{bmatrix} 12337 & -6084 \\ -6084 & 8788 \end{bmatrix} - \sigma_j^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) = 0,$$

which leads to

$$(12337 - \sigma_j^2)(8788 - \sigma_j^2) - (-6084)^2 = 0.$$

The solution of the quadratic equation gives

$$\sigma_1 = 130,$$

$$\sigma_2 = 65.$$

To find  $\mathbf{V}$ , premultiply  $\mathbf{F} = \mathbf{USV}^T$  first by  $\mathbf{U}^T$  and then by  $\text{diag}[\sigma_j^{-2}]$ . Using  $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ , the process gives

$$\tilde{\mathbf{V}}^T = \text{diag}[\sigma_i^{-2}]\mathbf{U}^T\mathbf{F}, \quad (11.78h)$$

where  $\tilde{\mathbf{V}}^T$  is the first  $M$  rows of  $\mathbf{V}^T$ . This is as expected:  $\mathbf{A} = \mathbf{USV}^T$  uses only the first  $M$  rows of  $\mathbf{V}^T$ . Since  $\mathbf{V}^T$  must be orthogonal, the remaining rows of  $\mathbf{V}^T$  must be chosen to be orthogonal to its first  $M$  rows. The remaining rows of  $\mathbf{V}^T$  can be computed from its first  $M$  rows using, say, Gram-Schmidt orthonormalization.

The SVD of  $\mathbf{F}$  can therefore be computed as follows:

1. Compute eigenvalues and eigenvectors of  $\mathbf{FF}^T$ .
2. The eigenvalues are  $\sigma_i^2$  and the eigenvectors are  $\mathbf{u}_i$ .
3. Compute the first  $M$  rows  $\tilde{\mathbf{V}}^T$  of  $\mathbf{V}^T$  using

$$\tilde{\mathbf{V}}^T = \text{diag}[\sigma_i^{-2}]\mathbf{U}^T\mathbf{F}.$$

4. Compute the remaining rows of  $\mathbf{V}^T$  using Gram-Schmidt orthonormalization.

Another way to compute the SVD is from

$$\begin{aligned} \mathbf{F}^T\mathbf{F} &= (\mathbf{VS}^T\mathbf{U}^T)(\mathbf{USV}^T) \\ &= (\mathbf{VS}^T)(\mathbf{U}^T\mathbf{U})(\mathbf{SV}^T) \\ &= \mathbf{V}\text{diag}[\sigma_j^2, \underbrace{0}_{N-M}] \mathbf{V}^T. \end{aligned} \quad (11.78i)$$

Repeating the earlier argument,  $\mathbf{V}$  is the matrix of eigenvectors, and  $\{\sigma_j^2\}$  are the nonzero eigenvalues, of  $(N \times N)$  matrix  $\mathbf{F}^T\mathbf{F}$ . Note that  $\mathbf{F}^T\mathbf{F}$  has  $N - M$  zero eigenvalues, since the rank of  $\mathbf{F}^T\mathbf{F}$  is  $M$ .

Inserting  $\sigma_1$  into Eq. (11.78i) gives

$$\begin{bmatrix} 12337 - 16900 & -6084 \\ -6084 & 8788 - 16900 \end{bmatrix} \mathbf{u}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

which has the solution (after normalization so that  $\mathbf{u}_1^T\mathbf{u}_1 = 1$ ):  $\mathbf{u}_1 = [4/5, -3/5]^T$ .

Similarly, for  $\sigma_2$ ,

$$\begin{bmatrix} 12337 - 4225 & -6084 \\ -6084 & 8788 - 4225 \end{bmatrix} \mathbf{u}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

has the solution (after normalization so that  $\mathbf{u}_2^T\mathbf{u}_2 = 1$ ):  $\mathbf{u}_2 = [3/5, 4/5]^T$ .

Using Eq. (11.78h),  $\tilde{\mathbf{V}}^T$  is computed to be

$$\tilde{\mathbf{V}}^T = \begin{bmatrix} 12/13 & 0 & -5/13 \\ 0 & 1 & 0 \end{bmatrix}.$$

The third row of  $\mathbf{V}^T$  is computed to be orthogonal to the two rows of  $\tilde{\mathbf{V}}^T$  using Gram-Schmidt and is

$$\mathbf{v}_3^T = [5/13 \ 0 \ 12/13].$$

### 11-8.3 Interpretations of SVD

#### A. Rotations, Reflections and Scalings

An orthogonal matrix can be interpreted as a rotation and/or a reflection of coordinate axes. A  $(2 \times 2)$  orthogonal matrix can often be put into the form of  $R_\theta$  defined in Eq. (3.38), which we repeat here as

$$R_\theta = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}. \quad (11.79)$$

For example, by comparing the entries of  $\mathbf{U}$  in Eq. (11.78b) with the entries of  $R_\theta$ , we ascertain that  $\mathbf{U}$  is a rotation matrix with  $\theta = 36.87^\circ$ . Similarly,  $\mathbf{V}$  in Eq. (11.78d) represents a rotation matrix with  $\theta = 22.62^\circ$ .

Hence, the SVD of a matrix  $\mathbf{F}$  can be interpreted as: (1) a rotation and/or reflection of axes, followed by (2) scaling of the rotated axes, followed by (3) another rotation and/or reflection of axes.

#### B. Expansion in Orthonormal Vectors

Now we introduce another interpretation of the SVD, which will prove particularly useful in unsupervised learning. In the sequel, we assume that  $M < N$ , so  $\mathbf{F}$  is a **reclining matrix**. Let us define:

- (1)  $\mathbf{f}_i$  as the  $i$ th column of  $\mathbf{F}$ , for  $i = 1, 2, \dots, N$ :

$$\mathbf{f}_i = \begin{bmatrix} f_{i,1} \\ f_{i,2} \\ \vdots \\ f_{i,M} \end{bmatrix}, \quad (11.80)$$

- (2)  $\mathbf{u}_i$  as the  $i$ th column of  $\mathbf{U}$ , for  $i = 1, 2, \dots, M$ :

$$\mathbf{u}_i = \begin{bmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,M} \end{bmatrix}, \quad (11.81)$$

and

- (3)  $v_{i,j}$  as the  $(i,j)$ th element of  $\mathbf{V}$ . It follows that  $v_{j,i}$  is the  $(i,j)$ th element of matrix  $\mathbf{V}^T$ .

With these definitions, the  $n$ th column of the equation  $\mathbf{F} = \mathbf{USV}^T$  can be rewritten as

$$\begin{aligned}\mathbf{f}_i &= \sum_{j=1}^M \mathbf{u}_j (\sigma_j v_{i,j}) \\ &= c_{i,1} \mathbf{u}_1 + c_{i,2} \mathbf{u}_2 + \cdots + c_{i,M} \mathbf{u}_M, \quad \text{for } i = 1, \dots, N,\end{aligned}\quad (11.82)$$

where  $c_{i,j} = \sigma_j v_{i,j}$ . The coefficients  $\{c_{i,j}\}$  are called **coordinates** of vector  $\mathbf{f}_i$  in the basis  $\{\mathbf{u}_j\}$  of the  $M$ -dimensional space  $R^M$ .

For example, applying Eq. (11.82) to the SVD example of Eq. (11.78) gives

$$\begin{aligned}\mathbf{f}_3 &= \begin{bmatrix} -40 \\ 30 \end{bmatrix} = \mathbf{u}_1(\sigma_1 v_{3,1}) + \mathbf{u}_2(\sigma_2 v_{3,2}) \\ &= \begin{bmatrix} 4/5 \\ -3/5 \end{bmatrix} \underbrace{130(-5/13)}_{-50} + \begin{bmatrix} 3/5 \\ 4/5 \end{bmatrix} \underbrace{65(0)}_0.\end{aligned}\quad (11.83)$$

According to Eq. (11.82) each column of  $\mathbf{F}$  can be written as a linear combination of the  $M$  columns  $\{\mathbf{u}_i\}$  of the orthogonal matrix  $\mathbf{U}$ , with the coefficients in the linear combination being  $\{\sigma_j v_{i,j}\}$ . Note that the final  $N-M$  rows of  $\mathbf{V}^T$  are not used to compute  $\mathbf{F}$ , because they multiply the zero part of the reclining matrix  $\mathbf{S}$ .

The coefficients  $\sigma_j v_{i,j}$  can be computed using

$$\sigma_j v_{i,j} = (\mathbf{U}^T \mathbf{f}_i)_j, \quad j = 1, \dots, M; \quad i = 1, \dots, N, \quad (11.84)$$

where  $(\mathbf{U}^T \mathbf{f}_i)_j$  is the  $j^{th}$  element of the column vector  $\mathbf{U}^T \mathbf{f}_i$ . Equation (11.84) provides the  $(j,i)$ th element of either  $\mathbf{U}^T \mathbf{F}$  or  $\mathbf{S} \mathbf{V}^T$ .

For example, for  $i = 3$  and  $j = 1$ ,

$$\sigma_1 v_{3,1} = [4/5 \quad -3/5] \begin{bmatrix} -40 \\ 30 \end{bmatrix} = -50;$$

and for  $i = 3$  and  $j = 2$ ,

$$\sigma_2 v_{3,2} = [3/5 \quad 4/5] \begin{bmatrix} -40 \\ 30 \end{bmatrix} = 0.$$

### C. Comparison with Orthonormal Functions

The column vectors  $\mathbf{u}_i$  of the orthogonal matrix  $\mathbf{U}$  are themselves orthonormal vectors. This is because  $\mathbf{U}^T \mathbf{U} = \mathbf{I}_M$  is equivalent to stating

$$\mathbf{u}_i^T \mathbf{u}_j = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \quad (11.85)$$

Equation (11.82) is the same as the expansion given in Eq. (7.4) for a function  $x[i]$ , for all  $i$ , in orthonormal functions  $\{\phi_i(t)\}$ . Furthermore, Eq. (11.84) is the linear algebraic equivalent of the formula given by Eq. (7.6) for computing the coefficients  $\{\mathbf{x}_i\}$  in the orthonormal expansion in Eq. (7.4). The orthonormality of the vectors  $\{\mathbf{u}_i\}$  in  $\mathbf{U}$  is the linear algebraic equivalent of orthonormality of the basis functions  $\{\phi_i[n]\}$  in Eq. (7.5).

For convenience, we repeat Eqs. (7.4)–(7.6), modified to finite number  $N$  of real-valued basis functions  $\phi_i[n]$ , coefficients  $x_i$  and times  $n$ , and  $C = 1$ :

$$x[n] = \sum_{i=1}^N x_i \phi_i[n], \quad (11.86a)$$

$$\delta[i-j] = \sum_{n=1}^N \phi_i[n] \phi_j[n], \quad (11.86b)$$

and

$$x_i = \sum_{n=1}^N x[n] \phi_i[n]. \quad (11.86c)$$

### 11-8.4 Dimensionality Reduction

Let us arrange the singular values  $\{\sigma_i\}$  to be in decreasing order

$$\sigma_1 > \sigma_2 > \cdots > \sigma_M,$$

which can be accomplished by reordering the columns of  $\mathbf{U}$  and the rows of  $\mathbf{V}^T$ .

Often there is a **threshold** singular value  $\sigma_T$  with the following properties:

- (1)  $\sigma_1 > \sigma_2 > \cdots > \sigma_{T+1} > \sigma_T$  and
- (2)  $\sigma_T \gg \sigma_{T-1} > \sigma_{T-2} > \cdots > \sigma_M$ .

Thus, all singular values with index larger than  $T$  are much smaller than  $\sigma_T$ , and therefore they can be ignored in

Eq. (11.82). The **truncated SVD** becomes

$$\mathbf{f}_i \approx \sum_{j=1}^T \mathbf{u}_j (\sigma_j v_{i,j}). \quad (11.87)$$

The summation in Eq. (11.87) provides a reasonable approximation so long as  $\sigma_T$  is much larger than singular values with higher indices. In terms of orthogonal functions, this is analogous to truncating a Fourier series expansion to  $T$  terms. The resulting sum is often a good approximation to the function.

The significance of Eq. (11.87) is that each column  $\mathbf{f}_i$  of  $\mathbf{F}$  can now be approximated well by only  $T$  of the  $M$  terms in the orthonormal expansion given by Eq. (11.82). Each  $\mathbf{f}_i$  is represented using  $T$  coefficients:

$$\mathbf{f}_i \approx c_{i,1}\mathbf{u}_1 + c_{i,2}\mathbf{u}_2 + \cdots + c_{i,T}\mathbf{u}_T, \quad (11.88)$$

which contains only the first  $T$  terms in Eq. (11.82).

Replacing the expansion in Eq. (11.82) with the truncated expansion in Eq. (11.87) is called **dimensionality reduction**. This not only reduces the amount of computation, but also allows visualization of the image classes in a T-D subspace instead of M-D  $\mathcal{R}^M$ . The dimensionality reduction process is illustrated through forthcoming examples.

**Concept Question 11-6:** Why do we set small singular values to zero?

## 11-9 Unsupervised Learning Examples

In unsupervised learning, we are given a set of  $I$  **training images**  $\{f^{(i)}[n,m], i = 1, \dots, I\}$ . Even though we have no information about the contents of the  $I$  images, the goal is to partition the images into  $\mathcal{L}$  classes  $\{f_k[n,m], k = 1, \dots, \mathcal{L}\}$ , as was done previously in Sections 11-1 to 11-4, but now determined by the images themselves. If we are then given an additional image  $g_{obs}[n,m]$ , we must provide a simple rule for classifying it into one of the  $\mathcal{L}$  classes.

We provide two simple examples to illustrate how SVD is used in unsupervised learning and classification. In the first example, the images are intentionally chosen to be small in size ( $2 \times 2$ ) to allow us to carry out the computations manually. A total of 5 training images and 1 observation image are involved. The experience gained from the first example then allows us to consider a second example comprised of 24 training images, each ( $3 \times 3$ ) in size, and 1 observation image of the same size.

### Notation for Unsupervised Training and Classification

$f^{(i)}[n,m] = (M \times N)$   $i$ th training image, with class unknown,  $\{i = 1, \dots, I\}$

$I$  = number of training images

$\mathcal{L}$  = number of classes

$g_{obs}[n,m]$  = observed image of unknown class

$N' = MN$  = number of pixels per image

$\mathbf{f}^{(i)}$  =  $i$ th training image vector, generated by column unwrapping  $f^{(i)}[n,m]$  into  
 $[f^{(i)}[0,0], \dots, f^{(i)}[0,M-1], \dots, f^{(i)}[1,0], \dots,$   
 $f^{(i)}[1,M-1], \dots, f^{(i)}[N-1,0], \dots,$   
 $f^{(i)}[N-1, M-1]]^T$

$\mathbf{F} = [\mathbf{f}_1 \quad \mathbf{f}_2 \quad \dots \quad \mathbf{f}_I]^T$  = matrix composed of the columns of the  $I$  vectors representing images  $f^{(i)}[n,m]$ ,  $\{i = 1, 2, \dots, I\}$ . Size of  $\mathbf{F}$  is  $N' \times I$ , where  $N' = MN$

$\{\sigma_j\}$  = singular values of SVD representation

$\mathbf{U}$  and  $\mathbf{V}$  = SVD orthogonal matrices

$\mathbf{S}$  = SVD diagonal matrix

### 11-9.1 Unsupervised Learning Example with 5 Training Images

In this simple example, we are given  $I = 5$  training images,  $\{f^{(i)}[n,m]\}$ , identified by superscript  $i$  with  $i = 1, 2, \dots, 5$ . All 5 images are  $(2 \times 2)$  in size. We also are given an observation image  $g_{obs}[n,m]$ . Our goal is to use the training images to determine the distinct classes contained in the training images, using unsupervised learning, and then to classify  $g_{obs}[n,m]$  into one of those classes.

The six  $(2 \times 2)$  images are given by

$$f^{(1)}[n,m] = \begin{bmatrix} 1.1 & 0.1 \\ 0.1 & 1.0 \end{bmatrix}, \quad (11.89a)$$

$$f^{(2)}[n,m] = \begin{bmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{bmatrix}, \quad (11.89b)$$

$$f^{(3)}[n,m] = \begin{bmatrix} 1.0 & 0.1 \\ 0.0 & 0.9 \end{bmatrix}, \quad (11.89c)$$

$$f^{(4)}[n,m] = \begin{bmatrix} 0.0 & 0.9 \\ 1.1 & 0.1 \end{bmatrix}, \quad (11.89d)$$

$$f^{(5)}[n,m] = \begin{bmatrix} 1.1 & 1.0 \\ 0.9 & 1.0 \end{bmatrix}, \quad (11.89e)$$

$$g_{\text{obs}}[n,m] = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}. \quad (11.89f)$$

## A. Training Matrix

We unwrap the  $f^{(i)}[n,m]$  by columns as in MATLAB's `F(:)` and assemble the unwrapped images into a  $(4 \times 5)$  **training matrix**  $\mathbf{F}$ . The  $i$ th column of the training matrix  $\mathbf{F}$  is the unwrapped vector image  $\mathbf{f}^{(i)}[n,m]$ :

$$\begin{aligned} \mathbf{F} &= [\mathbf{f}_1 \quad \mathbf{f}_2 \quad \mathbf{f}_3 \quad \mathbf{f}_4 \quad \mathbf{f}_5] \\ &= \begin{bmatrix} 1.1 & 0.1 & 1.0 & 0.0 & 1.1 \\ 0.1 & 1.0 & 0.0 & 1.1 & 0.9 \\ 0.1 & 1.0 & 0.1 & 0.9 & 1.0 \\ 1.0 & 0.1 & 0.9 & 0.1 & 1.0 \end{bmatrix}. \end{aligned} \quad (11.90)$$

Using the recipe given in Section 11-8.1(B) or the MATLAB command `[U, S, V] = svd(F)`, we obtain the following matrices:

$$\mathbf{U} = \begin{bmatrix} 0.54 & -0.52 & -0.20 & -0.63 \\ 0.47 & 0.56 & 0.63 & -0.26 \\ 0.49 & 0.48 & -0.69 & 0.25 \\ 0.50 & -0.44 & 0.29 & 0.69 \end{bmatrix}, \quad (11.91a)$$

$$\mathbf{S} = \begin{bmatrix} 2.94 & 0 & 0 & 0 & 0 \\ 0 & 1.86 & 0 & 0 & 0 \\ 0 & 0 & 0.14 & 0 & 0 \\ 0 & 0 & 0 & 0.02 & 0 \end{bmatrix}, \quad (11.91b)$$

$$\mathbf{V} = \begin{bmatrix} 0.40 & -0.49 & 0.45 & -0.63 & 0.01 \\ 0.36 & 0.51 & -0.38 & -0.44 & -0.53 \\ 0.35 & -0.46 & -0.06 & 0.54 & -0.61 \\ 0.34 & 0.54 & 0.70 & 0.31 & -0.01 \\ 0.68 & -0.01 & -0.39 & 0.17 & 0.59 \end{bmatrix}. \quad (11.91c)$$

It is apparent from matrix  $\mathbf{S}$  that both  $\sigma_1 = 2.94$  and  $\sigma_2 = 1.86$  are much larger than  $\sigma_3 = 0.14$  and  $\sigma_4 = 0.02$ . Hence, we can truncate the SVD to  $T = 2$  in Eq. (11.88), which gives

$$\begin{aligned} \mathbf{f}^{(1)} &\approx \mathbf{u}_1(\sigma_1 v_{1,1}) + \mathbf{u}_2(\sigma_2 v_{1,2}) \\ &= (2.94 \times 0.40)\mathbf{u}_1 + (1.86 \times (-0.49))\mathbf{u}_2 \\ &= 1.19\mathbf{u}_1 - 0.90\mathbf{u}_2, \end{aligned} \quad (11.92a)$$

where  $\mathbf{u}_1$  and  $\mathbf{u}_2$  are the first and second columns of  $\mathbf{U}$ . Similarly, application of Eq. (11.88) for  $i = 2, 3, 4$ , and 5 yields

$$\begin{aligned} \mathbf{f}^{(2)} &\approx (\sigma_1 v_{2,1})\mathbf{u}_1 + (\sigma_2 v_{2,2})\mathbf{u}_2 \\ &= 1.06\mathbf{u}_1 + 0.94\mathbf{u}_2, \end{aligned} \quad (11.92b)$$

$$\begin{aligned} \mathbf{f}^{(3)} &\approx (\sigma_1 v_{3,1})\mathbf{u}_1 + (\sigma_2 v_{3,2})\mathbf{u}_2 \\ &= 1.04\mathbf{u}_1 - 0.86\mathbf{u}_2, \end{aligned} \quad (11.92c)$$

$$\begin{aligned} \mathbf{f}^{(4)} &\approx (\sigma_1 v_{4,1})\mathbf{u}_1 + (\sigma_2 v_{4,2})\mathbf{u}_2 \\ &= 1.01\mathbf{u}_1 + 1.00\mathbf{u}_2, \end{aligned} \quad (11.92d)$$

$$\begin{aligned} \mathbf{f}^{(5)} &\approx (\sigma_1 v_{5,1})\mathbf{u}_1 + (\sigma_2 v_{5,2})\mathbf{u}_2 \\ &= 2.00\mathbf{u}_1 - 0.23\mathbf{u}_2. \end{aligned} \quad (11.92e)$$

All five images assume the form of Eq. (11.88):

$$\mathbf{f}^{(i)} \approx c_{i,1}\mathbf{u}_1 + c_{i,2}\mathbf{u}_2.$$

With  $\mathbf{u}_1$  and  $\mathbf{u}_2$  as orthogonal dimensions,  $c_{i,1}$  and  $c_{i,2}$  are the coordinates of image  $\mathbf{f}^{(1)}$  in  $(\mathbf{u}_1, \mathbf{u}_2)$  space.

## B. Subspace Representation

If we regard  $\mathbf{u}_1$  and  $\mathbf{u}_2$  in Eq. (11.92) as orthogonal axes, then  $\mathbf{f}^{(1)}$ , the dimensionally reduced representation of training image 1, has coordinates  $(1.19, -0.90)$ , as depicted in Fig. 11-12. Similar assignments are made to the other four training images. The five symbols appear to be clustered into three image classes centered approximately at coordinates  $\{(1,1), (1,-1), (2,0)\}$ . We assign:

- Class #1: to cluster centered at  $(1,1)$ ,
- Class #2: to cluster centered at  $(1,-1)$ ,
- Class #3: to cluster centered at  $(2,0)$ .

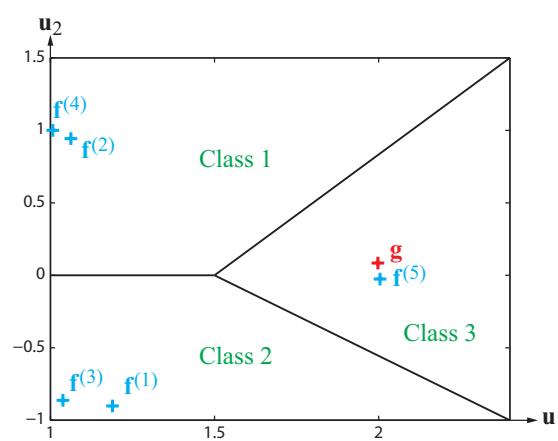
Given these three clusters, we divide the  $(\mathbf{u}_1, \mathbf{u}_2)$  domain into the three regions shown in Fig. 11-12.

## C. Classification of Observation Image

For the observation image defined by Eq. (11.89f), we need to determine its equivalent coordinates  $(g_1, g_2)$  in  $(\mathbf{u}_1, \mathbf{u}_2)$  space. We do so by applying the following recipe:

1. Unwrap  $g_{\text{obs}}[n,m]$  by columns to form vector  $\mathbf{g}$ , which in the present case gives

$$\mathbf{g} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}. \quad (11.93)$$



**Figure 11-12** Depiction of 2-D subspace spanned by  $\mathbf{u}_1$  and  $\mathbf{u}_2$ . The blue + symbols represent each column of the training matrix (i.e., each training image  $f^{(i)}[n, m]$ ). They cluster into 3 classes. The red + represents the observation image  $g_{\text{obs}}[n, m]$ .

2. Use the form of Eq. (11.84)—with  $\mathbf{f}_i$  replaced with  $\mathbf{g}$  and  $c_{i,j} = \sigma_j v_{i,j}$  replaced with  $g_i$ —to compute coordinates  $g_1$  to  $g_4$ :

$$\begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix} = \mathbf{U}^T \mathbf{g} = \begin{bmatrix} 0.54 & 0.47 & 0.49 & 0.50 \\ -0.52 & 0.56 & 0.48 & -0.44 \\ -0.20 & 0.63 & -0.69 & 0.29 \\ -0.63 & -0.26 & 0.25 & 0.69 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2.0 \\ 0.08 \\ 0.03 \\ 0.05 \end{bmatrix}. \quad (11.94)$$

3. From the result given by Eq. (11.94), we deduce that the reduced-coordinate representation of  $g_{\text{obs}}[n, m]$  in  $(\mathbf{u}_1, \mathbf{u}_2)$  space is  $(g_1, g_2) = (2.0, 0.08)$ .

4. These coordinates place  $\mathbf{g}$  in Class 3 (Fig. 11-12).

For new observations with  $(g_1, g_2)$  coordinates, the class regions in Fig. 11-12 correspond to the following classification scheme:

- (a) Class #1 if  $g_2 > 0$  and  $g_2 - g_1 > -1.5$ ,
- (b) Class #2 if  $g_2 < 0$  and  $g_2 + g_1 < 1.5$ ,
- (c) Class #3 if  $g_1 > 1.5$  and  $g_1 - 1.5 > g_2 > -(g_1 - 1.5)$ .

## 11-9.2 Unsupervised Learning Example with 24 Training Images

Now that we understand how to use SVD to truncate and classify simple  $(2 \times 2)$  images, let us consider a slightly more elaborate example involving 24 training images, each  $(3 \times 3)$  in size. The images are displayed in Fig. 11-13(a) and the goal is to develop a classification scheme on the basis of unsupervised learning.

The procedure is summarized as follows:

1. Each of the 24 images is unwrapped by columns into a  $(9 \times 1)$  vector.

2. The 24 vectors are assembled into a  $(9 \times 24)$  training matrix  $\mathbf{F}$ .

3. MATLAB is used to compute matrices  $\mathbf{U}$ ,  $\mathbf{S}$ , and  $\mathbf{V}$  of the SVD of  $\mathbf{F}$ , namely  $\mathbf{F} = \mathbf{USV}^T$ .

4. From  $\mathbf{S}$ , its diagonal elements, corresponding to the singular values  $\sigma_1$  through  $\sigma_9$ , are read off and then plotted to determine how many dimensions the images can be reduced to within a reasonable degree of approximation. The plotted values of  $\sigma_i$  for the images in Fig. 11-13(a) are shown in Fig. 11-13(b). Because  $\sigma_1$  and  $\sigma_2$  are much larger than the remaining singular values, we can approximate the images by reducing the dimensionality from  $M = 9$  down to  $T = 2$ .

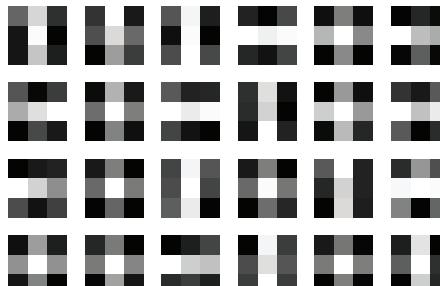
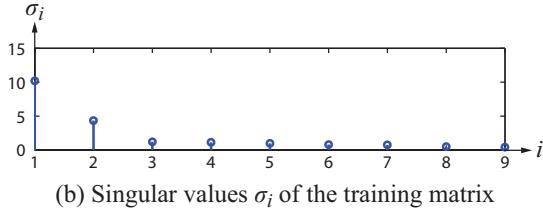
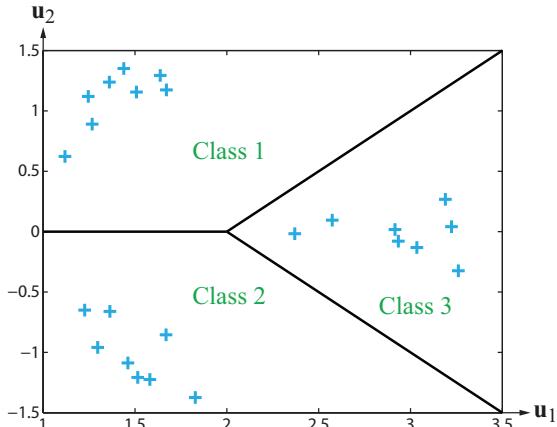
5. The reduced images can be represented in terms of coefficients similar to the form in Eq. (11.92). These coefficients become coordinates, leading to the 24 “+” symbols in Fig. 11-13(c).

6. Based on the cluster of points in Fig. 11-13(c), the  $(\mathbf{u}_1, \mathbf{u}_2)$  space is divided into three classes. For an observation image with an observation vector  $\mathbf{g}$ ,

- (a) Class #1 if  $g_2 > 0$  and  $g_2 - g_1 > -2$ ,
- (b) Class #2 if  $g_2 < 0$  and  $g_2 + g_1 < 2$ ,
- (c) Class #3 if  $g_1 > 2$  and  $g_1 - 2 > g_2 > -(g_1 - 2)$ .

While the two examples presented in this subsection have served to illustrate the general approach to unsupervised learning and classification, the image scenarios were limited to small-size images, the number of reduced dimensions  $T$  was only 2 in both cases, and the clustering of points was fairly obvious, making the development of the classification schemes rather obvious and convenient. How should we approach the general case where the training images are larger than  $(3 \times 3)$ , the number of reduced dimensions  $T$  is greater than 2, and the points are not clustered into clearly separate groups? The answer is:

(a) Neither the number of available training images nor their size has any effect on the SVD procedure outlined earlier in this subsection. We used small-size images simply for the convenience of writing their matrices on paper.

(a) 24 training images, each  $(3 \times 3)$  in size(b) Singular values  $\sigma_i$  of the training matrix(c) Depiction of 2-D subspace spanned by  $\mathbf{u}_1$  and  $\mathbf{u}_2$ . The blue + symbols represent individual columns of the training matrix. They cluster into 3 classes.**Figure 11-13** (a) 24 training images, (b) plot of singular values, and (c) clusters of training images in  $(\mathbf{u}_1, \mathbf{u}_2)$  space.

**(b)** For a reduced number of dimensions  $T = 2$ , it is easy to generate 2-D clusters. For  $T = 3$ , we can generate and display 3-D clusters onto a 2-D page, but the scales get distorted. For

$T \geq 4$ , we can no longer rely on our visual perspective to identify clusters and define classification regions. We therefore need mathematical tools to help us cluster the training images into any specified number of classes and in any number of dimensions, and to do so in accordance with an objective criterion. One such tool is the **K-Means Clustering Algorithm** and its associated **Voronoi sets**.

## 11-10 K-Means Clustering Algorithm

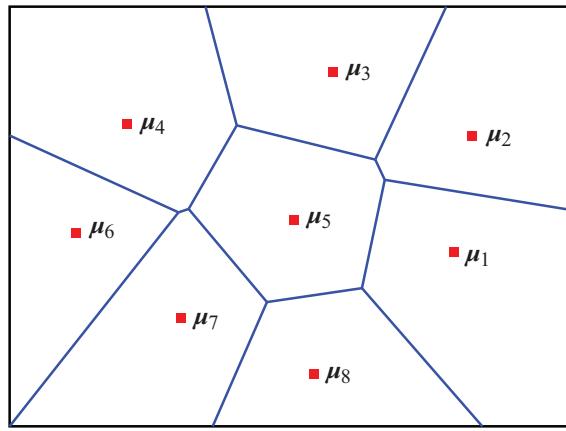
### 11-10.1 Voronoi Sets

Let us start with a review and update of our notation:

- $I$ : Number of training images
- $(M \times N)$ : Size of each image, in pixels
- $f^{(i)}[n, m]$ :  $i$ th image
- $\mathbf{f}^{(i)}$ :  $(N' \times 1)$  column vector generated by column unwrapping image  $f^{(i)}[n, n]$  into a column of length  $N' = MN$
- $\mathcal{L}$ : Specified number of classes into which the training images are to be clustered
- $N' = MN$ : Number of dimensions of the  $R^{N'}$  space in which SVD clustering has to occur, assuming no dimensionality reduction is used
- $T$ : Number of dimensions if dimensionality reduction is justified

For many realistic images,  $M$  and  $N$  may each be on the order of 1000, and therefore  $N' = MN$  may be on the order of  $10^6$ . Even if the number of dimensions is reduced to a smaller number  $T$ , it remains much larger than the 3-D space we can manage visually. The material that follows is applicable for clustering images using SVD in  $N'$ -dimensional space, as well as in  $T$ -dimensional space if dimensionality reduction had been applied.

Given  $I$  images, defined in terms of their unwrapped column vectors  $\{\mathbf{f}^{(i)}, i = 1, 2, \dots, I\}$ , we wish to cluster them in  $R^{N'}$  space into  $\mathcal{L}$  classes pre-specified by **centroids** at locations  $\{\boldsymbol{\mu}_p, p = 1, 2, \dots, \mathcal{L}\}$ . With each image represented by a point in  $R^{N'}$  space, the sets of points obtained by partitioning  $R^{N'}$  into regions such that each region consists of the set of points closest to the specified centroids are called **Voronoi sets** (or regions), named for Georgy Voronov. **Figure 11-14** shows a 2-D space divided into 8 Voronoi regions, each identified by a centroid. The boundaries of each region are defined such that every point location within that region is closer to the centroid of that region than to any of the other centroids. The distance between two



**Figure 11-14** Voronoi regions surrounding centroids  $\mu_1$  through  $\mu_8$ .

points, whether in 2-D or  $N'$ -D space, is the Euclidean distance defined in accordance with Eq. (7.117b). In the present context, for a test image  $\mathbf{f}_i$  and a centroid  $\boldsymbol{\mu}_p$  of region  $p$ , defined by

$$\mathbf{f}^{(i)} = \begin{bmatrix} f_{i,1} \\ f_{i,2} \\ \vdots \\ f_{i,N'} \end{bmatrix} \quad (11.95a)$$

and

$$\boldsymbol{\mu}_p = \begin{bmatrix} \mu_{p,1} \\ \mu_{p,2} \\ \vdots \\ \mu_{p,N'} \end{bmatrix}, \quad (11.95b)$$

the Euclidean distance between them is

$$\ell_{i,p} = \left[ \sum_{j=1}^{N'} |f_{i,j} - \mu_{p,j}|^2 \right]^{1/2}. \quad (11.96)$$

After computing  $\ell_{i,p}$  for all  $\mathcal{L}$  regions, image  $\mathbf{f}^{(i)}$  is assigned to the region for which  $\ell_{i,p}$  is the smallest. The boundaries in Fig. 11-14 are established by exercising such an assignment to every point in the space  $R^{N'}$ .

## 11-10.2 K-Means Algorithm

The boundaries of the Voronoi regions, such as those in Fig. 11-14, are established by the locations of the pre-specified centroids  $\{\boldsymbol{\mu}_p, p = 1, 2, \dots, \mathcal{L}\}$ . In unsupervised learning and classification, no information is available on the locations of those centroids, so we need an algorithm to create them. The **K-Means Algorithm** is a procedure for clustering  $I$  points—i.e.,  $I$  images defined in terms of their vectors  $\{\mathbf{f}^{(i)}, i = 1, 2, \dots, I\}$ —into  $\mathcal{L}$  clusters of points in  $R^{N'}$ , where  $N'$  is the column length of the image vectors. We still have to specify the number of classes  $\mathcal{L}$ , but not the locations of their centroids. The procedure is iterative and consists of the following steps:

1. Choose an initial set of  $\mathcal{L}$  clustering centroids  $\{\boldsymbol{\mu}_p^{(0)}, p = 1, 2, \dots, \mathcal{L}\}$  at random. The superscript of each  $\boldsymbol{\mu}_p^{(0)}$  denotes that it is the initial choice (zeroth iteration).
2. Compute the  $\mathcal{L}$  Voronoi sets (regions) using the  $I$  available image vectors, by measuring the distance between each image vector  $\mathbf{f}^{(i)}$  and each centroid  $\boldsymbol{\mu}_p^{(0)}$  and then assigning  $\mathbf{f}^{(i)}$  to the nearest centroid.
3. For each Voronoi set  $p$  use the locations of the image vectors assigned to it to compute their combined centroid and label it  $\boldsymbol{\mu}_p^{(1)}$  (first iteration). The centroid of a Voronoi set is the mean (average) location of the image vectors contained within that set. For purposes of illustration, if Voronoi set 3 contains  $(4 \times 1)$  image vectors  $\mathbf{f}^{(2)}, \mathbf{f}^{(5)}$ , and  $\mathbf{f}^{(8)}$ , and they are given in SVD format (Eq. (11.82)) by

$$\begin{aligned} \mathbf{f}^{(2)} &= 4\mathbf{u}_1 + 3\mathbf{u}_2 + 6\mathbf{u}_3 + (0)\mathbf{u}_4, \\ \mathbf{f}^{(5)} &= 2\mathbf{u}_1 + (0)\mathbf{u}_2 + 3\mathbf{u}_3 + 2\mathbf{u}_4, \\ \mathbf{f}^{(8)} &= (0)\mathbf{u}_1 + (0)\mathbf{u}_2 + 3\mathbf{u}_3 + 7\mathbf{u}_4, \end{aligned}$$

where  $\mathbf{u}_1$  to  $\mathbf{u}_4$  are the column vectors of matrix  $\mathbf{U}$ , as defined in Section 11-8.3, then the new centroid  $\boldsymbol{\mu}_3^{(1)}$  is given by

$$\begin{aligned} \boldsymbol{\mu}_3^{(1)} &= \frac{1}{3}[(4+2+0)\mathbf{u}_1 + (3+0+0)\mathbf{u}_2 \\ &\quad + (6+3+3)\mathbf{u}_3 + (0+2+7)\mathbf{u}_4] \\ &= 2\mathbf{u}_1 + \mathbf{u}_2 + 4\mathbf{u}_3 + 3\mathbf{u}_4. \end{aligned}$$

Here,  $\boldsymbol{\mu}_3^{(1)}$  defines the location of the **first iteration** of centroid 3.

4. Repeat steps 2 and 3 after incrementing each superscript on  $\boldsymbol{\mu}_p$  by 1.

5. Continue the iterative process until the algorithm has converged per a defined criterion regarding the average distance between centroids of successive iterations.

**Example 11-8: K-Means Clustering Algorithm**

The K-Means algorithm was applied to the  $N$  data points shown in [Fig. 11-15\(a\)](#), which were generated using two sets of random points. The algorithm was initialized using the K-Means++ algorithm, which works as follows:

- Choose one data point at random. Call this point  $\mathbf{x}_0$ , the centroid for the first cluster.
- Choose another data point  $\mathbf{x}_n$  at random. The probability that the data point  $\mathbf{x}_n$  is chosen is proportional to the square distance between  $\mathbf{x}_n$  and  $\mathbf{x}_0$ . Specifically,

$$\mathcal{P}[\mathbf{x}_n] = \frac{||\mathbf{x}_n - \mathbf{x}_0||_2^2}{\sum_{i=1}^N ||\mathbf{x}_i - \mathbf{x}_0||_2^2}.$$

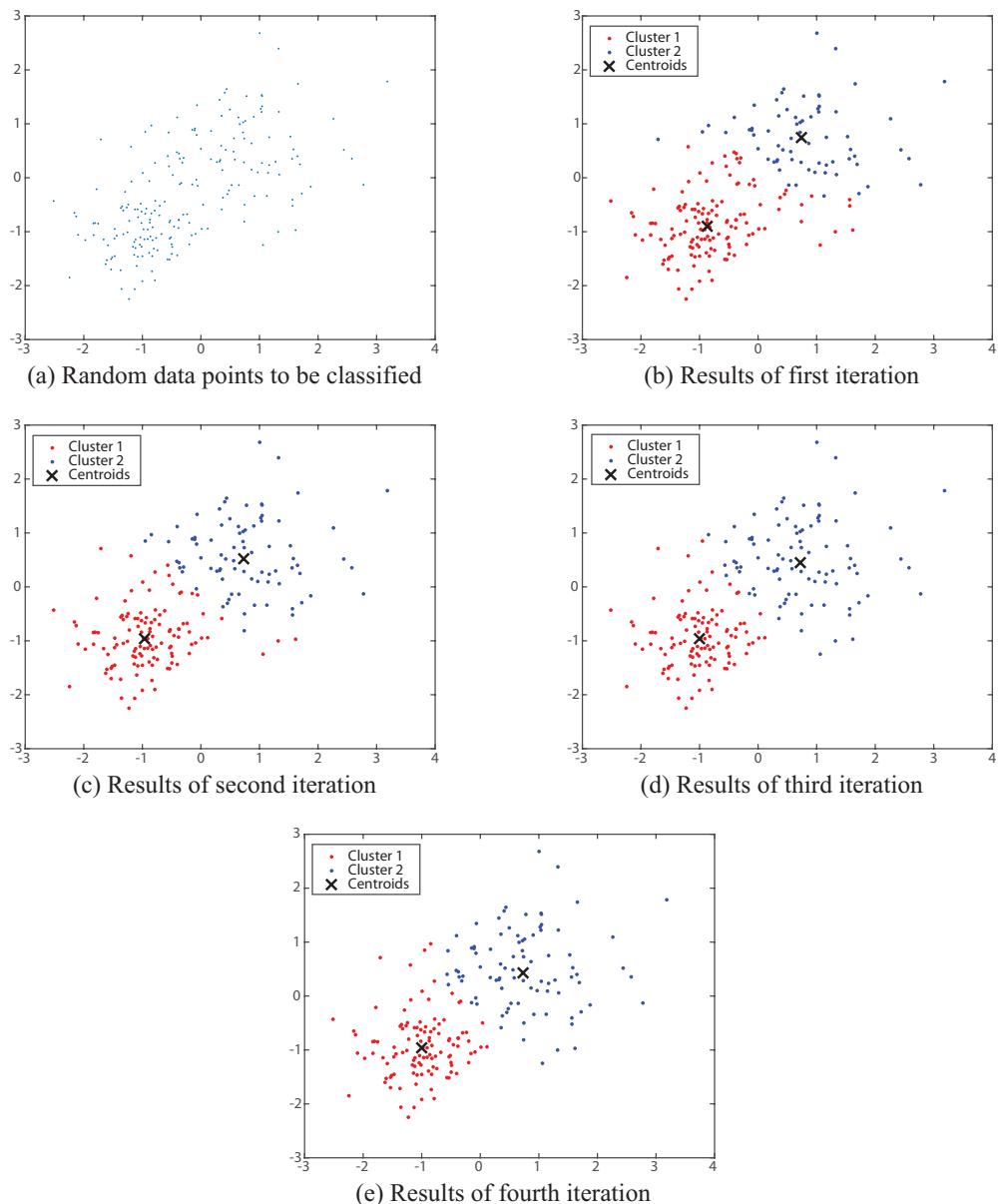
Thus, points far away from the first centroid  $\mathbf{x}_0$  are more likely to be chosen. Point  $\mathbf{x}_n$  is assigned as the centroid for the second cluster.

Results of different iterations of the K-Means algorithm are shown in parts (b) through (e) of [Fig. 11-15](#), after which the algorithm seems to have converged.

The K-Means algorithm was run using the MATLAB command `idx=kmeans(X, k)` available in MATLAB's Machine Learning toolbox. Here  $X$  is the  $(N \times 2)$  array of data point coordinates,  $k$  is the number of clusters (which must be set ahead of time; here  $k=2$ ), and `idx` is the  $(N \times 1)$  column vector of indices of clusters to which the data points whose coordinates are in each row of  $X$  are assigned. Each element of `idx` is an integer between 1 and  $k$ . For example, the data point whose coordinates are  $[X(3,1), X(3,2)]$  is assigned to cluster #`idx`(3).

`kmeans` has many options. For information, use `help kmeans`.

**Concept Question 11-7:** What is the K-Means Algorithm for?



**Figure 11-15** (a) Random data points and results of K-means clustering after (b) 1 iteration, (c) 2 iterations, (d) 3 iterations, and (e) 4 iterations.

## Summary

### Concepts

- Classification is often performed by choosing the image with the largest correlation with the observed image.
- MLE classification is performed by subtracting each image energy from double the correlation with the observed image.
- MAP classification is performed by subtracting each image energy from double the correlation with the observed image, and adding the logarithm of the *a priori* probability of the image times double the noise variance.
- Classification of an image with an unknown shift can be performed using cross-correlation instead of correlation. Classification of an image with an unknown scaling can be performed by cross-correlation on logarithmic spatial scaling.

- Jointly Gaussian color images can be classified using labelled training images to estimate the mean and covariance matrix of each image class, and using these estimates in vector versions of the MLE and MAP classifiers.
- Unsupervised learning can be performed using unlabeled training images, by unwrapping each image into a column vector, assembling these into a training matrix, computing the SVD of the training matrix, setting small singular values to zero, and regarding the orthonormal vectors associated with the remaining singular values as subspace bases. Each training image is a point in this subspace. The points can be clustered using the K-Means algorithm, if necessary.

### Mathematical Formulae

#### Correlation

$$\rho_k = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g_{\text{obs}}[n, m] f_k[n, m]$$

#### MAP classifier

$$\Lambda_{\text{MAP}}[k] = 2\rho_k - E_{f_k} + 2\sigma_v^2 \log p[k]$$

#### Cross-correlation for unknown shift

$$\rho_k[n, m] = g_{\text{obs}}[n, m] * * f_k[-n, -m]$$

#### Singular value decomposition (SVD)

$$\mathbf{F} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

#### Expansion in orthonormal vectors

$$\mathbf{f}_i = \sum_{j=1}^T \mathbf{u}_j (\sigma_j v_{i,j})$$

#### Coefficients of orthonormal vectors

$$c_{i,j} = \sigma_j v_{i,j} = (\mathbf{U}^T \mathbf{f}_i)_j$$

### Important Terms

Provide definitions or explain the meaning of the following terms:

correlation  
cross-correlation

dimensionality reduction  
K-Means algorithm

orthonormal vectors  
recognition

subspace  
SVD

training matrix  
unsupervised learning

## PROBLEMS

### Section 11-1: Image Classification by Correlation

**11.1** A problem with using correlation  $\rho_k$  (defined in Eq. (11.2)) to compare an observed image  $g_{\text{obs}}[n, m]$  with each one of a set of reference images  $\{f_k[n, m]\}$  is that the  $f_k[n, m]$  with the largest pixel values tends to have the largest  $\rho_k$ . We can avoid this problem as follows. Define the correlation coefficient

$$\tilde{\rho}_k = \sum \sum \tilde{g}_{\text{obs}}[n, m] \tilde{f}_k[n, m], \text{ where}$$

$$\tilde{g}_{\text{obs}}[n, m] = \frac{g_{\text{obs}}[n, m]}{\sqrt{\sum \sum g_{\text{obs}}^2[n, m]}}$$

and

$$\tilde{f}_k[n, m] = \frac{f_k[n, m]}{\sqrt{\sum \sum f_k^2[n, m]}}.$$

- (a) Expand  $0 \leq \sum \sum (\tilde{g}_{\text{obs}}[n,m] \pm \tilde{f}_k[n,m])^2$  and show that  $|\tilde{\rho}_k| \leq 1$ .  
 (b) If  $g_{\text{obs}}[n,m] = af_k[n,m]$  for any constant  $a > 0$ , show that  $\tilde{\rho}_k = 1$ .

**11.2** In Section 11-2, let the reference images  $\{f_k[n,m]\}$  all be energy-normalized to

$$\tilde{f}_k[n,m] = \frac{f_k[n,m]}{\sqrt{\sum \sum f_k^2[n,m]}},$$

and the observed image be  $g[n,m] = \tilde{f}_k[n,m] + v[n,m]$ , where  $v[n,m]$  is a zero-mean white Gaussian noise random field with variance  $\sigma_v^2$ . This is the problem considered in Section 11-2, except with energy-normalized reference images. Show that the MLE classifier is now the value of  $k$  that maximizes the correlation coefficient  $\tilde{\rho}_k = \sum \sum \tilde{g}_{\text{obs}}[n,m] \tilde{f}_k[n,m]$  between  $g[n,m]$  and  $f_k[n,m]$ .

### Section 11-3: Classification by MAP

**11.3** Determine a rule for classifying the image

$$g_{\text{obs}}[n,m] = \begin{bmatrix} g_{0,0} & g_{1,0} \\ g_{0,1} & g_{1,1} \end{bmatrix}$$

into one of the three classes

$$f_1[n,m] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},$$

$$f_2[n,m] = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix},$$

$$f_3[n,m] = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix},$$

using

- (a) MLE classification;  
 (b) MAP classification with *a priori* probabilities  $p[1] = 0.1$ ,  $p[2] = 0.3$ ,  $p[3] = 0.6$ . The noise variance is  $\sigma_v^2 = 10$ .

**11.4** Determine a rule for classifying the image

$$g_{\text{obs}}[n,m] = \begin{bmatrix} g_{0,0} & g_{1,0} \\ g_{0,1} & g_{1,1} \end{bmatrix}$$

into one of the three classes

$$f_1[n,m] = \begin{bmatrix} 7 & 1 \\ 5 & 2 \end{bmatrix},$$

$$f_2[n,m] = \begin{bmatrix} 2 & 5 \\ 1 & 7 \end{bmatrix},$$

$$f_3[n,m] = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix},$$

using

- (a) MLE classification;

- (b) MAP classification with *a priori* probabilities  $p[1] = 0.1$ ,  $p[2] = 0.3$ ,  $p[3] = 0.6$ . The noise variance is  $\sigma_v^2 = 10$ .

### Section 11-4: Classification of Spatially Shifted Images

**11.5** Download `text.jpg` and `text.m` from the book website. These were used in Example 11-2 to find all occurrences of “e” in `text.jpg`. Modify `text.m` to find all occurrences of “a” in `text.jpg`. You will need to (i) find a subset of the image containing “a” instead of “e” and (ii) change the threshold using trial-and-error. Produce a figure like [Fig. 11-11\(c\)](#).

**11.6** Download `text.jpg` and `text.m` from the book website. These were used in Example 11-2 to find all occurrences of “e” in `text.jpg`. Modify `text.m` to find all occurrences of “m” in `text.jpg`. You will need to (i) find a subset of the image containing “m” instead of “e” and (ii) change the threshold using trial and error. Produce a figure like [Fig. 11-11\(c\)](#).

**11.7** To machine-read numerals, such as the routing and account numbers on the check in [Fig. 11-1](#), it is necessary to segment (chop up) the number into individual numerals, and then use one of the methods of Sections 11-1 through 11-3 to identify each individual numeral. Segmentation is often performed by the hardware (camera) used to read the number.

The program `bank1.m` reads the image of the bank font numerals in [Fig. 11-2\(a\)](#), segments the image into  $(30 \times 18)$  images of each numeral, reassembles the images of numerals back into the sequence of numerals in [Fig. 11-2\(a\)](#), and machine-reads the result, using correlation, into a sequence of numbers that represent the numerals in [Fig. 11-2\(a\)](#).

Modify `bank1.m` so that it machine reads the account number in [Fig. 11-1](#). This requires assembling the images of numerals into the sequence of numerals in the account number in [Fig. 11-1](#). Plot the computed numerals in a stem plot. [Figure 11-2\(a\)](#) is stored in `bank.png`.

**11.8** To machine-read numerals, such as the routing and account numbers on the check in [Fig. 11-1](#), it is necessary to segment (chop up) the number into individual numerals, and then use one of the methods of Sections 11-1 through

11-3 to identify each individual numeral. Segmentation is often performed by the hardware (camera) used to read the number.

The program `bank1.m` reads the image of the bank font numerals in [Fig. 11-2\(a\)](#), segments the image into  $(30 \times 18)$  images of each numeral, reassembles the images of numerals back into the sequence of numerals in [Fig. 11-2\(a\)](#), and machine-reads the result, using correlation, into a sequence of numbers that represent the numerals in [Fig. 11-2\(a\)](#).

Modify `bank1.m` so that it machine-reads the routing number in [Fig. 11-1](#). This requires assembling the images of numerals into the sequence of numerals in the routing number in [Fig. 11-1](#). Plot the computed numerals in a stem plot. [Figure 11-2\(a\)](#) is stored in `bank.png`.

### Section 11-5: Classification of Spatially Scaled Images

**11.9** To machine-read numerals, such as the routing and account numbers on the check in [Fig. 11-1](#), it is necessary to segment (chop up) the number into individual numerals, and then use one of the methods of Sections 11-1 through 11-3 to identify each individual numeral. Segmentation is often performed by the hardware (camera) used to read the number. But the numbers may have a different size from the stored numerals, in which case the numbers are spatially scaled.

The program `scale1.m` reads the image of the bank font numerals in [Fig. 11-2\(a\)](#), segments the image into  $(30 \times 18)$  images of each numeral, decimates each numeral by  $(2 \times 2)$ , reassembles the images of decimated numerals into the sequence of numerals in [Fig. 11-2\(a\)](#), uses logarithmic spatial scaling to transform the problem into a shifted images problem, and machine-reads the result, using correlation, into a sequence of numbers that represent the numerals in [Fig. 11-2\(a\)](#).

Modify `scale1.m` so that it machine-reads the account number in [Fig. 11-1](#). This requires assembling the images of numerals into the sequence of numerals in the account number in [Fig. 11-1](#). Plot the computed numerals in a stem plot. [Figure 11-2\(a\)](#) is stored in `bank.png`.

**11.10** To machine-read numerals, such as the routing and account numbers on the check in [Fig. 11-1](#), it is necessary to segment (chop up) the number into individual numerals, and then use one of the methods of Sections 11-1 through 11-3 to identify each individual numeral. Segmentation is often performed by the hardware (camera) used to read the number. But the numbers may have a different size from the stored numerals, in which case the numbers are spatially scaled.

The program `scale1.m` reads the image of the bank font numerals in [Fig. 11-2\(a\)](#), segments the image into  $(30 \times 18)$

images of each numeral, decimates each numeral by  $(2 \times 2)$ , reassembles the images of decimated numerals into the sequence of numerals in [Fig. 11-2\(a\)](#), uses logarithmic spatial scaling to transform the problem into a shifted images problem, and machine-reads the result, using correlation, into a sequence of numbers that represent the numerals in [Fig. 11-2\(a\)](#).

Modify `scale1.m` so that it machine-reads the routing number in [Fig. 11-1](#). This requires assembling the images of numerals into the sequence of numerals in the routing number in [Fig. 11-1](#). Plot the computed numerals in a stem plot. [Figure 11-2\(a\)](#) is stored in `bank.png`.

### Section 11-7: Classification of Color Images

**11.11** Show that if there is no randomness in image color components, i.e.,  $\mathbf{K}_k = \mathbf{0}$ , that Eq. (11.61) for MAP classification of color images reduces to Eq. (11.13) for MAP classification of grayscale images summed over all three color components.

**11.12** Determine a rule for classifying the  $(2 \times 2)$  color image

$$g_{\text{obs},R} = \begin{bmatrix} g_{0,0,R} & g_{0,1,R} \\ g_{1,0,R} & g_{1,1,R} \end{bmatrix},$$

$$g_{\text{obs},G} = \begin{bmatrix} g_{0,0,G} & g_{0,1,G} \\ g_{1,0,G} & g_{1,1,G} \end{bmatrix},$$

$$g_{\text{obs},B} = \begin{bmatrix} g_{0,0,B} & g_{0,1,B} \\ g_{1,0,B} & g_{1,1,B} \end{bmatrix},$$

into one of the two  $(2 \times 2)$  color image classes

$$f_{1,R} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad f_{1,G} = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, \quad f_{1,B} = \begin{bmatrix} 9 & 1 \\ 2 & 3 \end{bmatrix},$$

$$f_{2,R} = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}, \quad f_{2,G} = \begin{bmatrix} 8 & 7 \\ 6 & 5 \end{bmatrix}, \quad f_{2,B} = \begin{bmatrix} 3 & 2 \\ 1 & 9 \end{bmatrix},$$

using (assume  $\mathbf{K}_1 = \mathbf{K}_2 = \mathbf{I}$  throughout)

**(a)** MLE classification;

**(b)** MAP classification with *a priori* probabilities  $p[1] = 0.2$ ,  $p[2] = 0.8$ , and  $\sigma_v^2 = 0$ .

**11.13** Determine a rule for classifying the  $(2 \times 2)$  color image

$$g_{\text{obs},R} = \begin{bmatrix} g_{0,0,R} & g_{0,1,R} \\ g_{1,0,R} & g_{1,1,R} \end{bmatrix},$$

$$g_{\text{obs},G} = \begin{bmatrix} g_{0,0,G} & g_{0,1,G} \\ g_{1,0,G} & g_{1,1,G} \end{bmatrix},$$

$$\mathbf{g}_{\text{obs},B} = \begin{bmatrix} g_{0,0,B} & g_{0,1,B} \\ g_{1,0,B} & g_{1,1,B} \end{bmatrix},$$

into one of the two  $(2 \times 2)$  color image classes

$$f_{1,R} = \begin{bmatrix} 7 & 1 \\ 2 & 4 \end{bmatrix}, \quad f_{1,G} = \begin{bmatrix} 8 & 6 \\ 4 & 2 \end{bmatrix}, \quad f_{1,B} = \begin{bmatrix} 9 & 7 \\ 5 & 3 \end{bmatrix},$$

$$f_{2,R} = \begin{bmatrix} 4 & 2 \\ 1 & 7 \end{bmatrix}, \quad f_{2,G} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}, \quad f_{2,B} = \begin{bmatrix} 3 & 5 \\ 7 & 9 \end{bmatrix},$$

using (assume  $\mathbf{K}_1 = \mathbf{K}_2 = \mathbf{I}$  throughout)

(a) MLE classification;

(b) MAP classification with *a priori* probabilities  $p[1] = 0.2$ ,  $p[2] = 0.8$ , and  $\sigma_v^2 = 0$ .

**11.14** Following Example 11-5, determine an MLE rule for classifying a  $(1 \times 1)$  color image  $[g_R, g_G, g_B]^T$  into two classes, determined from the following training images:

For class #1:

$$\mathbf{f}_1^1 = \begin{bmatrix} 0 \\ 0 \\ 8 \end{bmatrix}, \quad \mathbf{f}_1^2 = \begin{bmatrix} 0 \\ 8 \\ 8 \end{bmatrix}, \quad \mathbf{f}_1^3 = \begin{bmatrix} 8 \\ 8 \\ 0 \end{bmatrix}, \quad \mathbf{f}_1^4 = \begin{bmatrix} 8 \\ 8 \\ 8 \end{bmatrix}.$$

For class #2:

$$\mathbf{f}_2^1 = \begin{bmatrix} 4 \\ 4 \\ 12 \end{bmatrix}, \quad \mathbf{f}_2^2 = \begin{bmatrix} 4 \\ 12 \\ 12 \end{bmatrix}, \quad \mathbf{f}_2^3 = \begin{bmatrix} 12 \\ 12 \\ 4 \end{bmatrix}, \quad \mathbf{f}_2^4 = \begin{bmatrix} 12 \\ 12 \\ 12 \end{bmatrix}.$$

Assume no additive noise and equal *a priori* probabilities  $p[1] = p[2] = 0.5$ .

**11.15** Following Example 11-5, determine an MLE rule for classifying a  $(1 \times 1)$  color image  $[g_R, g_G, g_B]^T$  into two classes, determined from the following training images:

For class #1:

$$\mathbf{f}_1^1 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}, \quad \mathbf{f}_1^2 = \begin{bmatrix} 5 \\ 3 \\ 1 \end{bmatrix}, \quad \mathbf{f}_1^3 = \begin{bmatrix} 3 \\ 5 \\ 7 \end{bmatrix}, \quad \mathbf{f}_1^4 = \begin{bmatrix} 3 \\ 1 \\ 5 \end{bmatrix}.$$

For class #2:

$$\mathbf{f}_2^1 = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}, \quad \mathbf{f}_2^2 = \begin{bmatrix} 6 \\ 4 \\ 2 \end{bmatrix}, \quad \mathbf{f}_2^3 = \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix}, \quad \mathbf{f}_2^4 = \begin{bmatrix} 4 \\ 2 \\ 6 \end{bmatrix}.$$

Assume no additive noise and equal *a priori* probabilities  $p[1] = p[2] = 0.5$ .

## Section 11-8: Unsupervised Learning and Classification

**11.16** Show that the approximations in the truncated expansions of Eq. (11.92)(a-e) do a good job of approximating the columns of the training matrix  $\mathbf{F}$  in Eq. (11.90).

## Section 11-9: Unsupervised Learning Examples

**11.17** We are given the eight  $(2 \times 2)$  training images

$$\mathbf{f}^{(1)} = \begin{bmatrix} 1.1 & 0.1 \\ 0.1 & 1.1 \end{bmatrix}, \quad \mathbf{f}^{(2)} = \begin{bmatrix} 0.9 & 0.0 \\ 0.1 & 0.9 \end{bmatrix},$$

$$\mathbf{f}^{(3)} = \begin{bmatrix} 1.1 & 0.1 \\ 0.0 & 0.9 \end{bmatrix}, \quad \mathbf{f}^{(4)} = \begin{bmatrix} 0.9 & 0.0 \\ 0.0 & 0.1 \end{bmatrix},$$

$$\mathbf{f}^{(5)} = \begin{bmatrix} 0.1 & 1.1 \\ 1.1 & 0.1 \end{bmatrix}, \quad \mathbf{f}^{(6)} = \begin{bmatrix} 0.0 & 0.9 \\ 0.9 & 0.1 \end{bmatrix},$$

$$\mathbf{f}^{(7)} = \begin{bmatrix} 0.1 & 1.1 \\ 0.9 & 0.0 \end{bmatrix}, \quad \mathbf{f}^{(8)} = \begin{bmatrix} 0.0 & 0.9 \\ 1.1 & 0.0 \end{bmatrix}.$$

Use unsupervised learning to determine classes for these eight training images, and a rule for classifying the observed image

$$\mathbf{g}_{\text{obs}} = \begin{bmatrix} 0.0 & 1.1 \\ 1.1 & 0.0 \end{bmatrix}.$$

**11.18** We are given the eight  $(2 \times 2)$  training images

$$\mathbf{f}^{(1)} = \begin{bmatrix} 0.9 & 0.1 \\ 0.9 & 0.1 \end{bmatrix}, \quad \mathbf{f}^{(2)} = \begin{bmatrix} 1.1 & 0.0 \\ 1.1 & 0.1 \end{bmatrix},$$

$$\mathbf{f}^{(3)} = \begin{bmatrix} 0.9 & 0.1 \\ 1.1 & 0.0 \end{bmatrix}, \quad \mathbf{f}^{(4)} = \begin{bmatrix} 1.1 & 0.0 \\ 0.9 & 0.0 \end{bmatrix},$$

$$\mathbf{f}^{(5)} = \begin{bmatrix} 0.1 & 0.9 \\ 0.1 & 0.9 \end{bmatrix}, \quad \mathbf{f}^{(6)} = \begin{bmatrix} 0.0 & 1.1 \\ 0.1 & 1.1 \end{bmatrix},$$

$$\mathbf{f}^{(7)} = \begin{bmatrix} 0.1 & 0.9 \\ 0.0 & 1.1 \end{bmatrix}, \quad \mathbf{f}^{(8)} = \begin{bmatrix} 0.0 & 1.1 \\ 0.0 & 0.9 \end{bmatrix}.$$

Use unsupervised learning to determine classes for these eight training images, and a rule for classifying the observed image

$$\mathbf{g}_{\text{obs}} = \begin{bmatrix} 0.0 & 1.1 \\ 0.0 & 1.1 \end{bmatrix}.$$

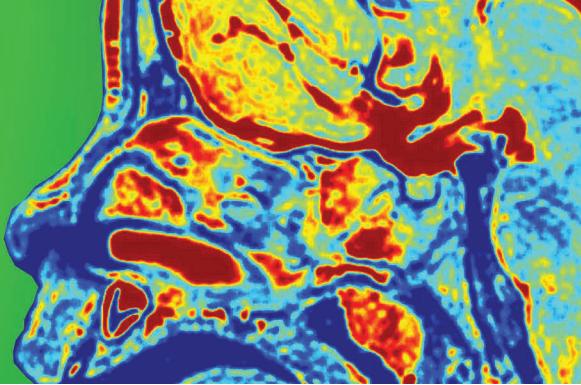
### Section 11-10: Clustering

**11.19** We are given a set of  $N$  points  $\{(f_1^{(i)}, f_2^{(i)}), i = 1, \dots, N\}$  in the plane  $\mathcal{R}^2$ . Prove that the point  $(f_1, f_2)$  minimizing

$$\sum_{i=1}^N \|(f_1^{(i)}, f_2^{(i)}) - (f_1, f_2)\|_2^2$$

is the centroid of the  $N$  points  $\{(f_1^{(i)}, f_2^{(i)}), i = 1, \dots, N\}$ . This is how centroids are determined in the K-Means algorithm. Note that the quantity to be minimized is the sum of squared distances from the points to the centroid, not the sum of distances.

# CHAPTER 12



## 12 Supervised Learning and Classification

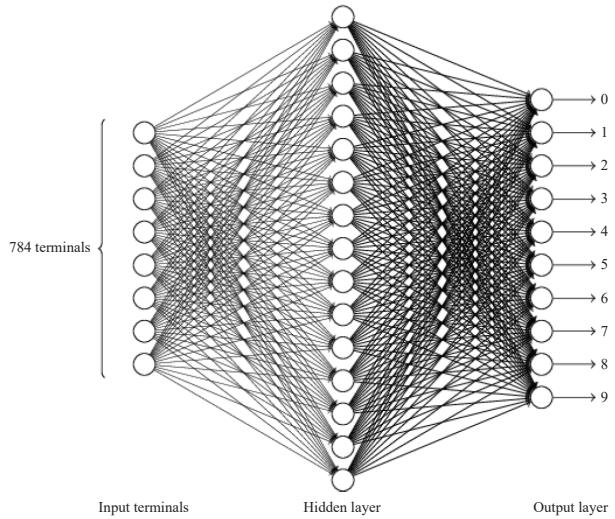
### Contents

- Overview, 390
- 12-1** Overview of Neural Networks, 390
- 12-2** Training Neural Networks, 396
- 12-3** Derivation of Backpropagation, 403
- 12-4** Neural Network Training Examples, 404
- Problems, 408

### Objectives

Learn to:

- Recognize the difference between unsupervised and supervised learning.
- Use a perceptron to implement classification with a single separating hyperplane.
- Use a hidden layer of perceptrons to implement more complicated classification.
- Train a neural network using backpropagation.



Neural networks and machine learning have become important tools in signal and image processing and computer science. No book on image or signal processing is complete without some coverage of these, which have revolutionized society. A neural network can be trained, using sets of labeled training images, to recognize digits or faces. This chapter presents a very brief introduction to neural networks for image classification.

## Overview

Having covered image classification algorithms based on *unsupervised training* in the preceding chapter, we now switch our attention to algorithms that rely on supervised training instead. To clearly distinguish between the structures of the two approaches, we begin this overview with a short summary of the former and an overview of the latter.

### A. Summary of Unsupervised Learning

In *unsupervised learning* and classification, we are given a set of  $I$  training images, each containing an image of one class (such as a numeral or a letter) from among  $\mathcal{L}$  possible classes, but the identities of the image classes are not known to us. *The goal is to cluster the  $I$  images into  $\mathcal{L}$  domains and to develop a classification scheme for assigning new observation images to the correct class.*

Each  $(M \times N)$  training image is unwrapped by columns to form a column vector of length  $N' = MN$ . The column vectors, denoted  $\mathbf{f}^{(i)}$  with  $\{i = 1, 2, \dots, I\}$ , are stacked alongside each other to form the  $(N' \times I)$  training matrix  $\mathbf{F}$ . After computing the SVD of matrix  $\mathbf{F}$ , the dimensionality of vectors  $\mathbf{f}^{(i)}$  is reduced from length  $N'$  to a truncated length  $T$  by setting small-valued singular values  $\sigma_i$  to zero. The reduced-dimensionality image vectors  $\mathbf{f}^{(i)}$  are each expressed as the sum of unit vectors  $\{\mathbf{u}_j, i = 1, 2, \dots, T\}$  with coefficients (coordinates)  $c_{i,j}$  as expressed by Eq. (11.92). The coordinates define the location of  $\mathbf{f}^{(i)}$  in  $R^T$  space. Next, Voronoi domains are established by clustering the  $I$  images into  $\mathcal{L}$  classes using the K-Means algorithm. Finally, boundaries are established between the Voronoi domains, thereby creating a scheme for classifying new observation images.

### B. Overview of Supervised Learning

In *supervised learning*, we are given a set of  $I$  training images, just as in unsupervised learning, except that now we *do know* the class of each of the training images. Each training image  $\mathbf{f}_k^{(i)}$  is said to be *labeled* with its proper image class  $k$ , so there is no need to determine the image classes, which we needed to do with unsupervised learning. Labeled training images of object classes, such as faces, are readily available from data collected from the internet. *The goal of supervised training is to use the labeled training images to develop an algorithm for classifying new observation images.*

The classification algorithm is implemented using a networked (connected) set of simple processors called *perceptrons*

or *neurons*. The network of processors is called a *neural network*. The act of using the labeled training images to compute the weights (gains) in the neural network is called *training* the neural network. The trained neural network accepts as input the pixel values of a new image to be classified, and outputs the class to which the new image belongs.

A good example of a supervised learning problem is optical character recognition of handwritten zip codes. A zip code is a sequence of five or nine digits used to identify letter and package destinations for the U.S. Postal System (USPS). But handwritten zip codes seldom resemble specific images of digits, because people write digits in different ways and styles. Supervised learning can be used to recognize handwritten zip codes by using as training images a labeled database of handwritten digits maintained by the U.S. National Institute of Standards and Technology (NIST) and labeled by the actual digit each handwritten digit represents, as identified by a human judge. Supervised learning has been applied to optical character recognition of handwritten Chinese symbols (with a training image size of 3 GB). We discuss the zip code application further in Section 12-4.

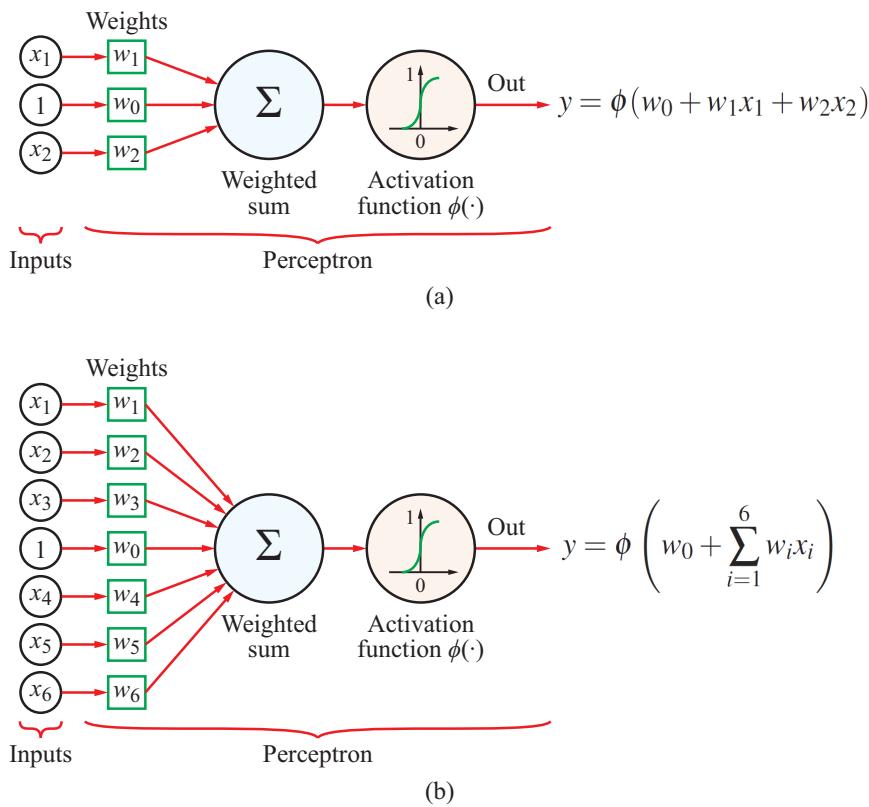
**Concept Question 12-1:** What is the difference between supervised and unsupervised learning?

## 12-1 Overview of Neural Networks

A neural network is a network of simple processors called neurons, as they are perceived to mimic the action of neurons in a brain. Neural networks are also called *multilayered perceptrons*.

The idea behind neural networks, when they were originally developed in the 1950s, was that a neural network mimics the communication structure in the brain, which consists of about 100 billion neurons, each connected to about one thousand of its neighbors in an elaborate network. It was thought that such a network of processors might mimic a brain in its ability to identify images and objects on the basis of previous exposure to similar images and objects (*learning*). The analogy to brain function is now viewed as “tenuous.”

The major advantage of using trained neural networks for image classification is that no modeling or understanding of the image classification problem is necessary. The way in which the trained neural network works need not be (and seldom is) understandable to humans. In the remainder of the present section, we introduce how perceptrons work and how multiple perceptrons are connected together to form a neural network.



**Figure 12-1** Perceptrons with (a) 2 inputs and (b) 6 inputs.

Then, in Section 12-2, we discuss how neural networks are trained, using the labeled training images. An example follows in Section 12-4.

### 12-1.1 Perceptrons

A perceptron is a simple processor that accepts a set of  $N$  numbers  $\{x_n, n = 1, \dots, N\}$  as inputs, multiplies each input  $x_n$  by a weight  $w_n$ , adds a constant  $w_0$  to the weighted sum of inputs, feeds the result into an **activation function**  $\phi(x)$  that mimics a step function, and outputs a number  $y$  that is close to either 0 or 1. The process mimics a biological neuron, which “fires” only if the weighted sum of its inputs exceeds a certain threshold. Synapses in the brain (which connect neurons) are biological analogues of the weights in a perceptron.

#### A. Components of a Perceptron

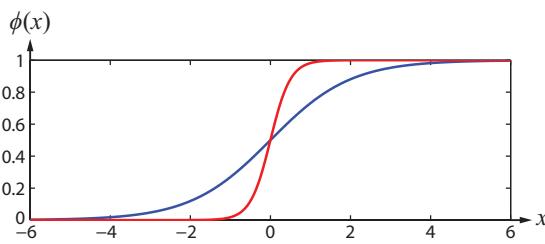
**Figure 12-1(a)** shows a perceptron with  $N = 2$  inputs. The added constant  $w_0$  usually is viewed as another weight multiplying a constant input of 1.

In another example, a perceptron with six inputs is shown in **Fig. 12-1(b)**.

A perceptron with  $N$  inputs  $\{x_n, n = 1, \dots, N\}$  is implemented by the formula

$$y = \phi(w_0 + w_1 x_1 + w_2 x_2 + \dots + w_N x_N). \quad (12.1)$$

The **activation function**  $\phi(x)$  is commonly chosen to be (for



**Figure 12-2** Activation function  $\phi(x)$  for  $a = 1$  (blue) and  $a = 4$  (red). The larger the value of  $a$ , the more closely  $\phi(x)$  resembles a step function.

some **activation constant**  $a > 0$ ) the **sigmoid** function

$$\phi(x) = \frac{1}{1 + e^{-ax}} \approx \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x < 0. \end{cases} \quad (12.2)$$

This choice of activation function is plotted in **Fig. 12-2** for  $a = 1$  and  $a = 4$ . The similarity to a step function is evident.

The rationale for choosing this particular mathematical representation for the activation function is supported by two attributes: (1) its resemblance to a step function, and (2) its derivative has the simple form

$$\frac{d\phi}{dx} = a \phi(x) (1 - \phi(x)). \quad (12.3)$$

The derivative of the activation function is used in Section 12-2 for training the neural network. Note that to implement a simple linear combination of the perceptron's inputs, the activation function should be chosen such that  $\phi(x) = x$ .

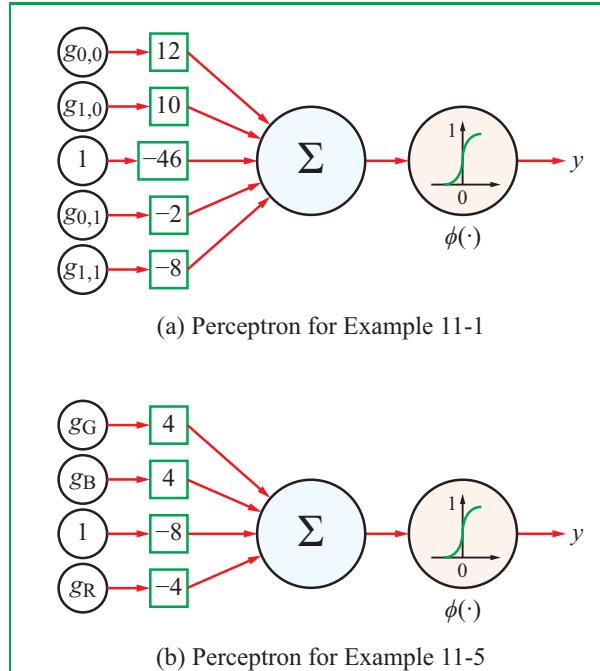
## B. Classification Using a Single Perceptron

Another reason for using perceptrons is that single-stage classification algorithms can each be implemented using a single perceptron.

By way of example, let us consider the classification rules we derived earlier in Examples 11-1 and 11-5. The classification rule given by Eq. (11.19) is:

$$\begin{aligned} \text{Choose } f_1: & \text{ if } 12g_{0,0} + 10g_{1,0} - 2g_{0,1} - 8g_{1,1} - 46 < 0, \\ \text{Choose } f_2: & \text{ if } 12g_{0,0} + 10g_{1,0} - 2g_{0,1} - 8g_{1,1} - 46 > 0. \end{aligned}$$

This rule can be implemented using the perceptron in **Fig. 12-3(a)** with  $\phi(x)$  set equal to the step function  $u(x)$ . If



**Figure 12-3** Perceptrons for implementing the classification rules of Examples 11-1 and 11-5.

output  $y = 1$ , choose  $f_2$ , and if  $y = 0$ , choose  $f_1$ .

Similarly, consider the classification rule given in Example 11-5, which stated:

$$\begin{aligned} \text{Choose } f_1: & \text{ if } 4g_G + 4g_B - 4g_R - 8 < 0, \\ \text{Choose } f_2: & \text{ if } 4g_G + 4g_B - 4g_R - 8 > 0. \end{aligned}$$

The rule can be implemented using the perceptron in **Fig. 12-3(b)**. If output  $y = 1$ , choose  $f_2$ , and if  $y = 0$ , choose  $f_1$ .

**Exercise 12-1:** Use a perceptron to implement a digital OR logic gate ( $y = x_1 + x_2$ , except  $1 + 1 = 1$ ).

**Answer:** Use **Fig. 12-1(a)** with  $\phi(x) \approx u(x)$ , weights  $w_1 = w_2 = 1$ , and any  $w_0$  such that  $-1 < w_0 < 0$ .

**Exercise 12-2:** Use a perceptron to implement a digital AND logic gate ( $y = x_1 x_2$ ).

**Answer:** Use **Fig. 12-1(a)** with  $\phi(x) \approx u(x)$ , weights  $w_1 = w_2 = 1$ , and any  $w_0$  such that  $-2 < w_0 < -1$ .

## 12-1.2 Networks of Perceptrons

### A. Background

Networks of perceptrons, in which the output of each perceptron is fanned out and serves as an input into many other perceptrons, are called **neural networks**, an example of which is shown in Fig. 12-4.

The leftmost vertical stack of squares is called the **input layer** (of **terminals**). Each square is a terminal that accepts one number as an input and fans it out to the perceptrons in the hidden layer to its immediate right. *The number of terminals in the input layer equals the number of pixels in the image.* Terminals are also called **nodes**, and they often are depicted using circles, making them easy to confuse with perceptrons (which they are not). To distinguish the terminals in the input layer from the perceptrons in succeeding layers, we depict the terminals in Fig. 12-4 as squares instead of circles.

The rightmost vertical stack of perceptrons is called the **output layer**. Each circle is now a perceptron, which includes weights and an activation function, which are not shown explicitly in Fig. 12-4. *The number of perceptrons in the output*

*layer is usually the number of image classes.* The output of the neural network is usually close to 1 for one of the perceptrons in the output layer, and close to zero for all other perceptrons in the output layer. The output-layer perceptron with an output of 1 identifies the classification of the image. In terms of the formulation in Section 11-1, the  $K$ th perceptron in the output layer outputting 1 is analogous to  $K$  maximizing the log-likelihood function. There are also other possibilities; the output layer may output ones and zeros that constitute bits in a binary representation of integer  $K$ .

The vertical stacks of perceptrons between the input and output layers are called **hidden layers** (of perceptrons). The hidden layers greatly increase the ability of the neural network to perform classification, partly because there are more weights and partly because the neural network has a more complicated structure, including more activation functions. The activation functions allow classification regions that are more complicated than those we have seen so far.

An underlying rationale for the validity of neural networks is the **Universal Approximation Theorem**, which states that any “reasonable” nonlinear function  $f(x_1, x_2, \dots, x_N)$  can be approximated as

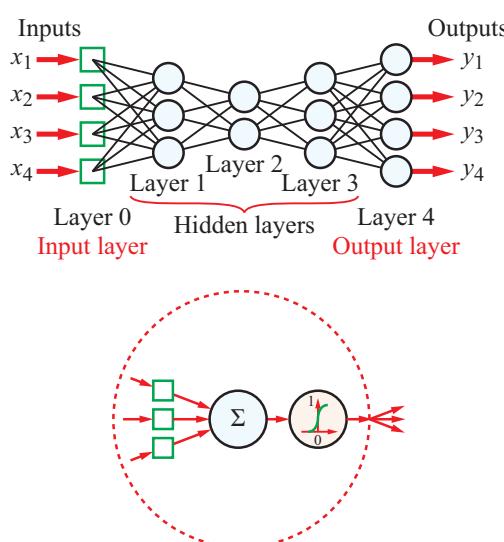
$$f(x_1, x_2, \dots, x_N) \approx \sum_{i=1}^M c_i \phi \left( \sum_{j=1}^N w_{i,j} x_j \right), \quad a < x_n < b, \quad (12.4)$$

for some constants  $N, M, c_i, w_{i,j}, a, b$  and some nonlinear function  $\phi(\cdot)$ . There are no specific rules for the number of hidden layers or number of perceptrons in each hidden layer. A very rough rule of thumb is that the total number of perceptrons in hidden layers should be one-fourth the sum of the numbers of inputs and outputs.

### B. Deep-Learning Neural Networks

**Deep-learning** neural networks may have dozens of hidden layers, with each layer consisting of hundreds of perceptrons, and requiring teraflops of computation. A trained deep-learning neural network may be able to perform surprising feats of classification, such as spam filtering and face recognition. Much of the research on neural networks is focused on the use of huge neural networks for deep learning.

Deep-learning neural networks became practical around 2009 when it became possible to construct them using **graphical processing units (GPUs)**. A GPU is a parallel-processing chip capable of performing simple computations (like those in a perceptron) in parallel in thousands of parallel computing cores. They were developed for creating images for computer games



**Figure 12-4** A neural network with 4 inputs and 4 outputs. Each blue circle in the hidden and output layers is a perceptron with associated weights and an activation function (see enlarged circle).

and similar applications that required computations for rotation of vertices in 3-D images. An example of a GPU is the NVIDIA GeForce GTX1080 video card. The applicability of GPUs to neural networks should be evident.

### C. Examples of Tiny Neural Networks

For practical applications, even the smallest neural networks have thousands of perceptrons. For example, a neural network for reading handwritten zip codes reads each digit using a camera that generates a  $(28 \times 28)$  image of each digit. Hence, the input layer of the neural network has  $N' = 28^2 = 784$  terminals, which is too large to display as a figure. The output layer has  $\mathcal{L} = 10$  perceptrons (one for each possible output  $\{0, 1, \dots, 9\}$ ). There is usually one hidden layer with about 15 perceptrons. Hence, in the forthcoming examples, we limit our discussion to tiny networks designed to demonstrate how neural networks operate, even though their sizes are very small.

### D. Classification with a Single Separating Hyperplane

- A single perceptron can be used to classify images in binary ( $\mathcal{L} = 2$  classes) classification problems in which the regions are segregated by a *separating hyperplane*. ◀

A *hyperplane* in  $\mathbb{R}^N$  is a surface of the form

$$a_1x_1 + a_2x_2 + \dots + a_Nx_N = b, \quad (12.5)$$

where  $(x_1, x_2, \dots, x_N)$  are the coordinates of a point in  $\mathbb{R}^N$  and  $\{a_1, a_2, \dots, a_N\}$  and  $b$  are constants. For  $N = 2$ , Eq. (12.5) becomes

$$x_2 = \frac{b}{a_2} - \frac{a_1}{a_2}x_1,$$

which is a line with slope  $-a_1/a_2$ . For  $N = 3$ , Eq. (12.5) becomes a plane like the one in [Fig. 11-11](#). A separating hyperplane divides  $\mathbb{R}^N$  into two classification regions, one for each image class.

Examples 11-1 and 11-5 are examples of binary classification problems with a separating hyperplane. Classification was accomplished using the single perceptrons in [Figs. 12-3\(a\)](#) and [\(b\)](#), respectively. The AND and OR gates in Exercises 12-1 and 12-2 are two other examples of binary classification.

Many classification problems do not have separating hyperplanes separating classification regions. The classification

regions are Voronoi sets separated by multiple hyperplanes, or more complicated curved boundaries.

### E. Classification Using Multiple Hyperplanes

Let us consider the XOR (exclusive OR) logic gate defined by

$x_1$	0	0	1	1
$x_2$	0	1	0	1
$y$	0	1	1	0

The classification regions for the XOR gate are shown in [Fig. 12-5](#), which include two hyperplanes.

The  $y = 1$  region is not contiguous, so a single perceptron will not be able to classify the input pair  $(x_1, x_2)$  correctly. Instead, we use three perceptrons, connected as shown in [Fig. 12-5\(b\)](#).

With the activation functions  $\phi(\cdot)$  of all three perceptrons set to act like step functions  $u(x)$ , the lower perceptron in the hidden layer of [Fig. 12-5\(b\)](#) computes  $\phi(x_2 - x_1 - 1)$ , whose value is 1 in the region above the line  $x_2 - x_1 = 1$  (hyperplane 1 in  $\mathbb{R}^2$ ).

Similarly, the upper perceptron in the hidden layer computes  $\phi(x_1 - x_2 - 1)$ , whose value is 1 in the lower right region below the line  $x_1 - x_2 = 1$  (hyperplane 2 in  $\mathbb{R}^2$ ).

The rightmost perceptron (in the output layer) implements an OR logic gate operation. The output is  $y = 1$  if either of its inputs is 1, and 0 if neither of its inputs is 1. The  $-0.5$  could be replaced by any number  $b$  such that  $-1 < b < 0$ .

#### Example 12-1: Neural Network Classifier

Implement a neural network classifier for the classification rule developed in Section 11-9.1.

**Solution:** In Section 11-9.1, we illustrated how to develop an unsupervised classification rule for an unknown  $(2 \times 2)$  image

$$g[n, m] = \begin{bmatrix} g[0, 0] & g[1, 0] \\ g[0, 1] & g[1, 1] \end{bmatrix}, \quad (12.6)$$

given five  $(2 \times 2)$  images  $\{f^{(i)}[n, m], i = 1, 2, \dots, 5\}$  with unknown classes. The procedure led to coordinates  $g_1$  and  $g_2$  given by

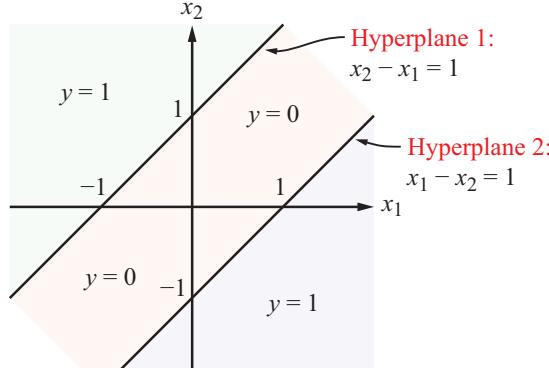
$$g_1 = (\mathbf{U}^T \mathbf{g})_1 \quad (12.7a)$$

and

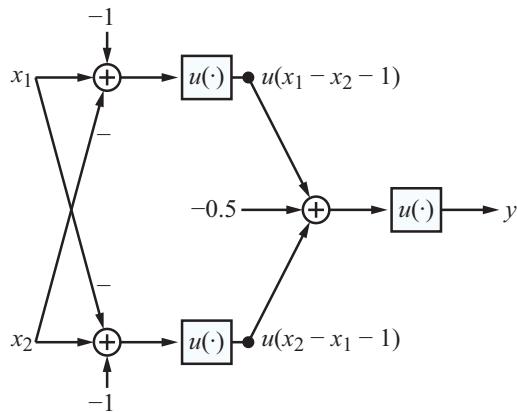
$$g_2 = (\mathbf{U}^T \mathbf{g})_2, \quad (12.7b)$$

where  $\mathbf{g}$  is the vector equivalent of unwrapped  $g[n, m]$ , namely

$$\mathbf{g} = [g[0, 0], g[0, 1], g[1, 0], g[1, 1]]^T, \quad (12.8)$$



(a) Classification regions for XOR gate



(b) Neural network implementation of XOR gate

**Figure 12-5** Classification regions and neural network for XOR gate. Here,  $\phi(\cdot) \approx u(\cdot)$  is a step function.

and  $\mathbf{U}$  is a  $(4 \times 4)$  matrix computed from the five given images using the SVD method outlined in Section 11-9.1. In Eq. (12.7),  $(\mathbf{U}^T \mathbf{g})_i$  is the  $i$ th element of the column vector  $(\mathbf{U}^T \mathbf{g})$ . From Eq. (11.91a), the transpose of  $\mathbf{U}$  is given by

$$\mathbf{U}^T = \begin{bmatrix} 0.54 & 0.47 & 0.49 & 0.50 \\ -0.52 & 0.56 & 0.48 & -0.44 \\ -0.20 & 0.63 & -0.69 & 0.29 \\ -0.63 & -0.26 & 0.25 & 0.69 \end{bmatrix}. \quad (12.9)$$

Using Eqs. (12.8) and (12.9) in Eq. (12.7) gives

$$\begin{aligned} g_1 &= 0.54 g[0,0] + 0.47 g[0,1] \\ &\quad + 0.49 g[1,0] + 0.50 g[1,1], \end{aligned} \quad (12.10a)$$

$$\begin{aligned} g_2 &= -0.52 g[0,0] + 0.56 g[0,1] \\ &\quad + 0.48 g[1,0] - 0.44 g[1,1]. \end{aligned} \quad (12.10b)$$

The solution in Section 11-9.1 led to the existence of three classes with the following rule:

$$(1) \text{ Class } \#1 \text{ if: } g_2 > 0 \text{ and } g_2 - g_1 > -1.5, \quad (12.11a)$$

$$(2) \text{ Class } \#2 \text{ if: } g_2 < 0 \text{ and } g_2 + g_1 < 1.5, \quad (12.11b)$$

$$(3) \text{ Class } \#3 \text{ if: } g_1 > 1.5 \text{ and } g_1 - 1.5 > g_2 > -(g_1 - 1.5). \quad (12.11c)$$

To construct a neural network with the appropriate perceptrons, it is useful to cast the classification rules in the form of AND or OR statements. To that end, we introduce variables  $a_1$ ,  $a_2$ , and  $a_3$ , and we use activation functions that mimic step functions  $u(\cdot)$ :

$$a_1 = \phi(g_2 - g_1 + 1.5)$$

$$\approx u(g_2 - g_1 + 1.5) = \begin{cases} 1 & \text{if } (g_2 - g_1 + 1.5) > 0, \\ 0 & \text{if } (g_2 - g_1 + 1.5) < 0, \end{cases} \quad (12.12a)$$

$$a_2 = \phi(g_2) \approx \begin{cases} 1 & \text{if } g_2 > 0, \\ 0 & \text{if } g_2 < 0, \end{cases} \quad (12.12b)$$

and

$$a_3 = \phi(g_2 + g_1 - 1.5) \approx \begin{cases} 1 & \text{if } g_2 + g_1 - 1.5 > 0, \\ 0 & \text{if } g_2 + g_1 - 1.5 < 0. \end{cases} \quad (12.12c)$$

The steps required for realizing  $a_1$  to  $a_3$  are shown in **Fig. 12-6**.

The classification rule for Class #1, as given by Eq. (12.11a), requires that both  $a_1$  AND  $a_2$  be 1, which can be realized by combining them to obtain

$$a_4 = a_1 + a_2 - 1.5, \quad (12.13a)$$

and then subjecting  $a_4$  to a final activation-function operation  $\phi(a_4)$ :

$$y_1 = \phi(a_4) = \phi(a_1 + a_2 - 1.5). \quad (12.13b)$$

The AND table shown in **Fig. 12-6** confirms that output  $y_1 = 1$  only when both  $a_1$  AND  $a_2$  are 1.

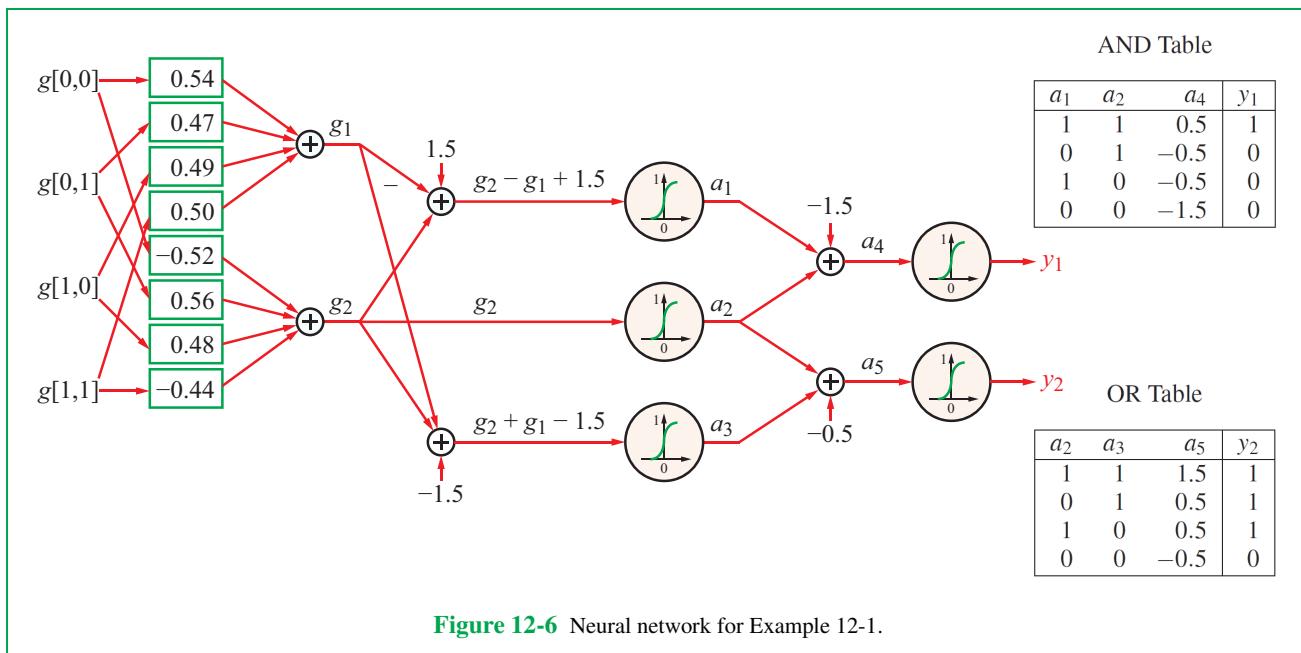


Figure 12-6 Neural network for Example 12-1.

► For Class #1, the classification rule is:

If  $y_1 = 1$ , choose Class #1.

Next, we consider the rule for Class #2, as defined by Eq. (12.11b), which also is an AND statement, as it requires that two simultaneous conditions be satisfied, namely that  $a_2$  and  $a_3$  are both zero. Instead of implementing an AND operation requiring both  $a_2$  and  $a_3$  to be zero, we can equivalently implement an OR operation requiring both  $a_2$  and  $a_3$  to be 1. This is made possible by the bottom branch in Fig. 12-6:

$$y_2 = \phi(a_5) = \phi(a_2 + a_3 - 0.5). \quad (12.14)$$

► For Class #2, the classification rule is:

If  $y_2 = 0$ , choose Class #2.

For class #3, the classification rule is straightforward:

If  $y_1 = 0$  and  $y_2 = 1$ , then  $g[n, m]$  does not belong to either Class #1 or Class #2, and therefore it has to belong to Class #3.

**Concept Question 12-2:** What is the difference between a perceptron and a neural network?

**Concept Question 12-3:** A perceptron is a nonlinear function of a linear combination of several inputs. Why does it have this form?

**Concept Question 12-4:** If the input to a neural network is to be classified into one of two classes separated by a separating hyperplane, what is the minimum number of neurons that must be in the neural network?

## 12-2 Training Neural Networks

So far, we have presented miniature-size neural networks for which the weights and structure could be determined rather easily. Neural networks for real-world problems are far more complicated. The weights are determined by using labeled training images. The procedure for determining the weights is called **training** the neural network. We now present this procedure.

There are actually three different procedures for training a

neural network. At the heart of all three procedures is an algorithm called **backpropagation**. This term comes from the fact that the neural network is first run forward (in increasing layer number, or left to right), using a labeled training image, and the gradient of the mean-square error with respect to the weights is then propagated backwards (in decreasing layer number, or right to left) using the results of the forward run.

The gradient of the mean-square error, computed using backpropagation, is then used in a **steepest-descent** (or **gradient**) algorithm, in one of three ways.

We explain these techniques in the next three subsections. First, we review steepest descent algorithms. Next, we derive the backpropagation algorithm, and finally, we present three procedures for computing the weights from labeled training images.

## 12-2.1 Gradient (Steepest Descent) (SD) Minimization Algorithm

A **steepest-descent** (**SD**) algorithm, also known as a **gradient** or (misleadingly) a **gradient-descent** algorithm (the gradient does not descend) is an iterative algorithm for finding the minimum value of a differentiable function  $f(x_1, x_2, \dots, x_N)$  of  $N$  spatial variables  $\mathbf{x} = [x_1, x_2, \dots, x_N]^T$ . The function to be minimized must be real-valued and scalar (not vector-valued), although SD can also be applied to minimize scalar functions of vector-valued or complex-valued functions, such as  $\|f(\mathbf{x})\|_2$ . To maximize  $f(\mathbf{x})$ , we apply SD to  $-f(\mathbf{x})$ . The minimum occurs at  $\mathbf{x} = \mathbf{x}^*$ , where  $*$  denotes the minimizing value, not complex conjugate (this is standard notation for optimization).

SD relies on the mathematical fact that the **gradient**  $\nabla f(\mathbf{x})$  specifies the (vector) direction in which  $f(\mathbf{x})$  increases fastest. Recall that, even though  $f$  is a scalar,  $\nabla f$  is a vector defined as

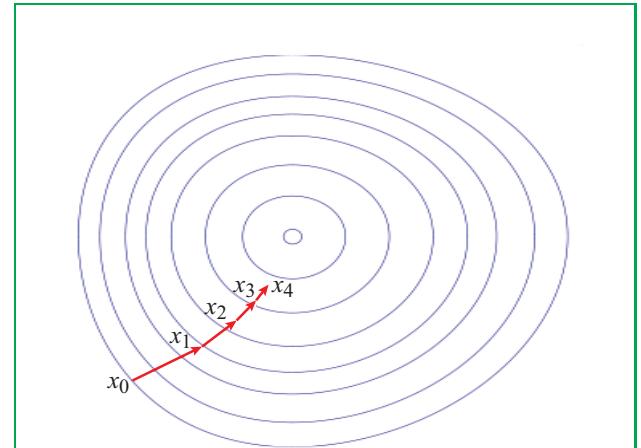
$$\nabla f = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_N} \right]^T. \quad (12.15)$$

The basic iteration of an SD algorithm is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mu \nabla f(\mathbf{x}^{(k)}), \quad (12.16)$$

where  $\mathbf{x}^{(k)}$  is the estimate of the minimizing  $\mathbf{x}$  at the  $k$ th iteration, and  $\mu$  is the step-size, a small discretization length. Vector  $\mathbf{x}^{(k)}$  is perturbed by a distance  $\mu$  in the direction that decreases  $f(\mathbf{x}^{(k)})$  the fastest.

The iterative process stops when  $\mathbf{x}^{(k+1)} \approx \mathbf{x}^{(k)}$ , corresponding to  $\nabla f(\mathbf{x}) \approx 0$ . This may be the location  $\mathbf{x}^*$  of the **global minimum** of  $f(\mathbf{x})$ , or it may be only a **local minimum** of  $f(\mathbf{x})$ .



**Figure 12-7** Contours of equal values of  $f(\mathbf{x})$  and the path (in red) taken by an SD algorithm.

These are both illustrated in Example 12-2 below.

A useful interpretation of SD is to regard  $f(\mathbf{x})$  as the elevation at a location  $\mathbf{x}$  in a bowl-shaped surface. The minimizing value  $\mathbf{x}^*$  is at the bottom of the bowl. By taking a short step in the direction in which  $f(\mathbf{x})$  is decreasing, we presumably get closer to the bottom  $\mathbf{x}^*$  of the bowl. However, we may get caught in a **local minimum**, at which  $\nabla f(\mathbf{x}) \geq 0$ , so taking another step in any direction would increase  $f(\mathbf{x})$ .

**Figure 12-7** illustrates how taking short steps in the direction in which  $f(\mathbf{x})$  is decreasing may eventually take us to the minimizing value  $\mathbf{x}^*$ . The curves are contours (as in a topographical map) of equal “elevations” of  $f(\mathbf{x})$  values. This is merely illustrative; backpropagation uses a 1-D SD algorithm.

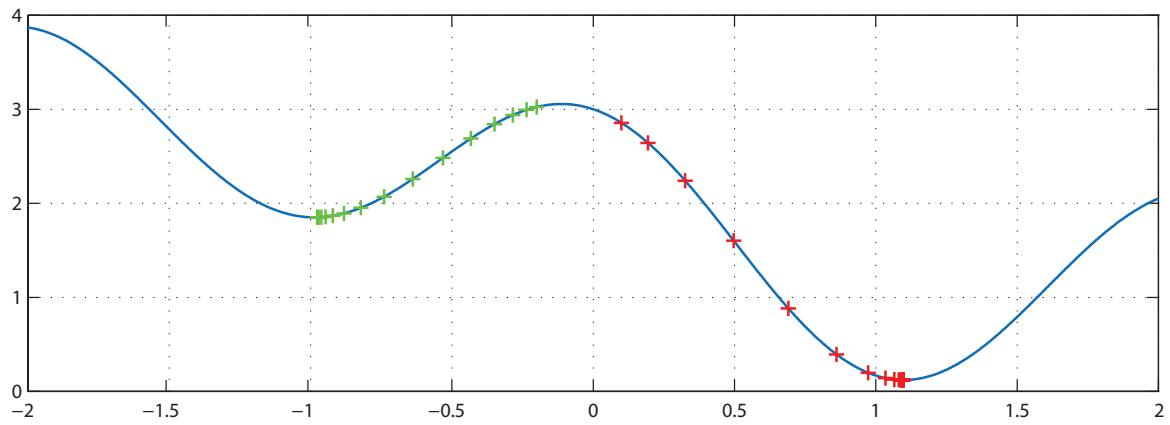
### Example 12-2: 1-D SD Algorithm

The 1-D version of Eq. (12.16) for minimizing a function  $f(x)$  is

$$x^{(k+1)} = x^{(k)} - \mu \frac{df}{dx}(x^{(k)}), \quad (12.17)$$

where  $k$  is the iteration index. Apply the algorithm given by Eq. (12.17) to compute the minimizing value  $x^*$  for

$$f(x) = \cos(3x) - \sin(x) + 2, \quad -2 < x < 2. \quad (12.18)$$



**Figure 12-8**  $f(x)$  (in blue) and  $\{(x^{(k)}, f(x^{(k)})), k = 1, \dots, 15\}$ , in red when initialized using  $x^{(0)} = 0.1$ , and in green when initialized using  $x^{(0)} = -0.2$ .

**Solution:** The function  $f(x)$  is plotted in blue in [Fig. 12-8](#). Over the range  $-2 < x < 2$ ,  $f(x)$  has its global minimum  $x^*$  near  $x = 1$ , as well as a local minimum near  $x = -1$ . Since

$$f'(x) = \frac{df}{dx} = -3 \sin(3x) - \cos(x), \quad |x| < 2,$$

the SD algorithm of Eq. (12.17) becomes

$$x^{(k+1)} = x^{(k)} + \mu (3 \sin(3x^{(k)}) + \cos(x^{(k)})). \quad (12.19)$$

When initialized using  $x^{(0)} = 0.1$  and  $\mu = 0.05$ , the SD algorithm converged to the (desired) global minimum at  $x \approx 1$  after 15 iterations. Ordered pairs  $\{(x^{(k)}, f(x^{(k)})), k = 1, \dots, 15\}$  are plotted using red “+” symbols in [Fig. 12-8](#).

But when initialized using  $x^{(0)} = -0.2$ , the SD algorithm converged to the (undesired) local minimum at  $x \approx -1$ . The ordered pairs  $\{(x^{(k)}, f(x^{(k)})), k = 1, \dots, 15\}$  are plotted using green “+” symbols in [Fig. 12-8](#). In both cases the algorithm had smaller updates at locations  $x$  where  $df/dx$  was small, as expected from Eq. (12.17). At locations where  $df/dx = 0$ , the algorithm stopped; these were the global and local minima.

## 12-2.2 Supervised-Training Scenario: Classifying Numerals 0 to 9

Let us consider how a neural network should perform to correctly classify images of numerals 0 to 9. In [Fig. 12-9](#), the

input layer consists of 784 terminals, corresponding to the intensity values of  $(28 \times 28)$  image pixels. The true identity of the test input image is the numeral 4. The output layer consists of 10 terminals designed to correspond to the numerals 0 to 9. Once the neural network has been properly trained using the backpropagation algorithm described later, the fifth output terminal (corresponding to numeral 4) should report an output of 1, and all of the other output terminals should report outputs of zero. If the input image is replaced with an image of a different numeral, the output terminals should change values accordingly.

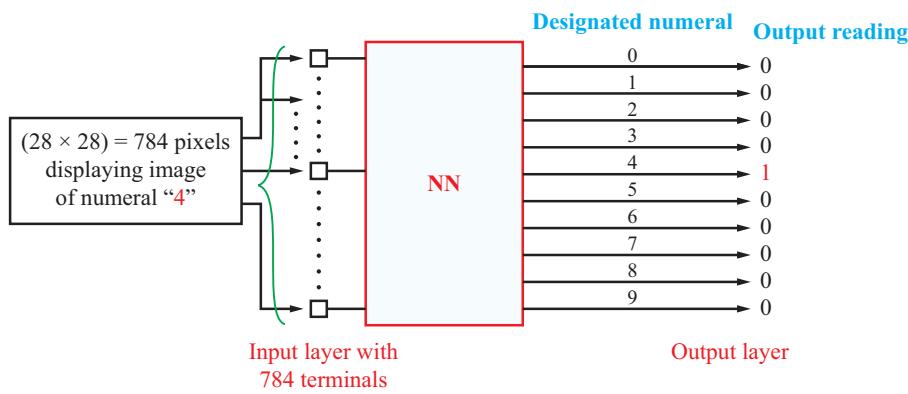
## 12-2.3 Backpropagation

### A. Overview of Backpropagation

**Backpropagation** is an algorithm for “training” (computing the weights) of a neural network. To perform backpropagation, we are given a set of  $I$  labeled training images  $\{\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_I\}$ , where each  $\mathbf{f}_i$  is an unwrapped training image. That is, the original 2-D  $(M \times N)$   $i$ th image  $f_i[n, m]$  is unwrapped by column into a vector  $\mathbf{f}_i$  of length  $N' = MN$ :

$$\mathbf{f}_i = [f_1[i], f_2[i], \dots, f_j[i], \dots, f_{N'}[i]]^T. \quad (12.20)$$

Element  $f_j[i]$  is the value of the  $j$ th pixel of the unwrapped training image  $i$ . The neural network has an input vector and an output vector. Image  $\mathbf{f}_i$  has  $N'$  elements, so the input layer of



**Figure 12-9** When the image of numeral 4 is introduced at the input of a properly trained neural network, the output terminals should all be zeros, except for the terminal corresponding to numeral 4.

the neural network should have  $N'$  terminals. The output of the last layer consists of  $\mathcal{L}$  neurons, corresponding to the number of image classes, and the combination of the  $\mathcal{L}$  values constitutes an output vector. The desired variable  $d_i$  is the number denoting perfect classification of image  $i$  with input vector  $\mathbf{f}_i$ , and it is called the *label* of  $\mathbf{f}_i$ . For the zip-code example in which each training image is  $(28 \times 28)$  in size, the input vector is of length  $N' = 28 \times 28 = 784$ , but the output vector is only of length  $\mathcal{L} = 10$ , corresponding to the digits 0 through 9.

The goal of backpropagation is to determine the weights in the neural network that minimize the *mean-square error*  $\mathcal{E}$  over all  $I$  training images.

## B. Notation

The neural-network notation can be both complicated and confusing, so we devote this subsection to nomenclature, supported by the configuration shown in **Fig. 12-10**.

With this notation in hand, we now define the *mean-square error*  $\mathcal{E}$  over all  $I$  training images as

$$\mathcal{E} = \frac{1}{I} \sum_{i=1}^I \mathcal{E}[i], \quad (12.22)$$

where

$$\mathcal{E}[i] = \frac{1}{2} \sum_{p=1}^{\mathcal{L}} (e_{L,p}[i])^2 = \frac{1}{2} \sum_{p=1}^{\mathcal{L}} (d_p[i] - y_{L,p}[i])^2. \quad (12.23)$$

The quantity inside the summation is the difference between the desired output of neuron  $p$  of the output layer and the actual output of that neuron. The summation is over all  $\mathcal{L}$  neurons in the output layer.

## 12-2.4 Backpropagation Training Procedure

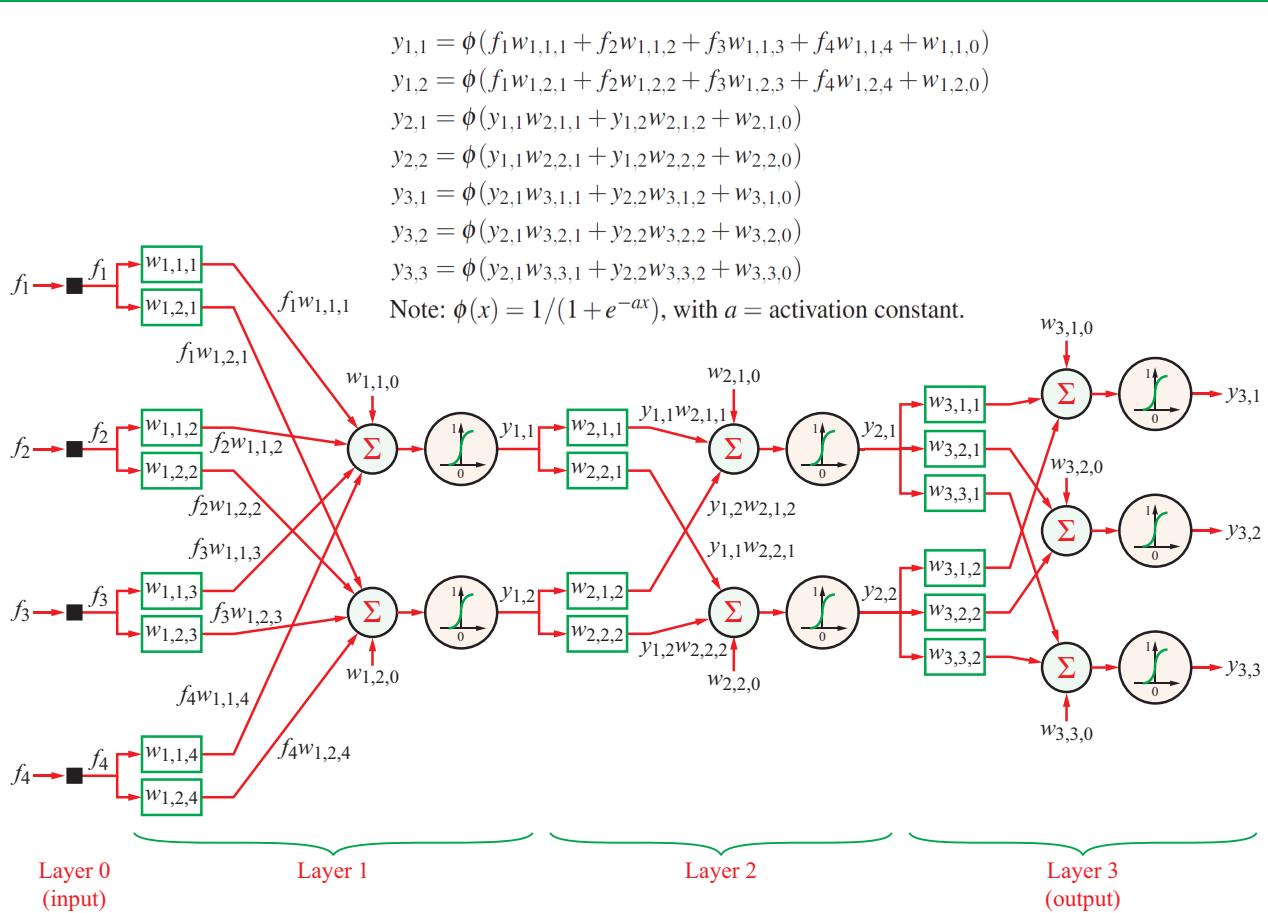
The classification accuracy of the neural network depends on the relationship between the  $N'$  inputs associated with the input image and the  $\mathcal{L}$  outputs associated with the  $\mathcal{L}$  classes. That relationship depends, in turn, on the assignment of weights  $w_{\ell,p,q}$ , representing the gain between the output of the  $q$ th neuron in layer  $(\ell - 1)$  and the input to the  $p$ th neuron in layer  $\ell$  (**Fig. 12-11**). Backpropagation is the tool used to determine those weights.

The process involves the use of  $I$  training images and one or more iterations. We will outline three backpropagation procedures with different approaches. The procedures utilize relationships extracted from the derivation given later in Section 12-3.

### A. Procedure 1: Iterate over $k$ , One Image at a Time

**Image  $i = 1$ , Iteration  $k = 0$**

**1. Initialize:** For test image  $i = 1$  (chosen at random and labeled image 1), initialize the weights  $w_{\ell,p,q}^{(0)}[1]$  with randomly chosen



**Figure 12-10** Neural network with 4 terminals at input layer, corresponding to image vector  $\mathbf{f} = [f_1 \ f_2 \ f_3 \ f_4]^T$ , 2 hidden layers of 2 neurons each, and an output layer with three terminals, corresponding to three image classes.

values. The superscript (0) denotes that  $w_{\ell,p,q}^{(0)}[1]$  are the initial assignments.

### Weight Initialization

The initial values of the weights  $w$  in the neural network should be small enough such that the linear combination of inputs is between 0 and 1. In addition, even when the activation function (sigmoid function) has a steep slope, the neuron output will be roughly proportional to the linear combination (instead of just being 1 or 0) for a wide range of input values.

However, if a weight  $w_{\ell,p,q}$  is too small, the output  $y_{\ell-1,q}$  of the  $q$ th neuron in layer  $(\ell-1)$  will have little effect on the input  $w_{\ell,p,q}y_{\ell-1,q}$  to the  $p$ th neuron in layer  $\ell$ . Also, if the weight is too large, it will dominate all of the other inputs to the  $p$ th neuron in layer  $\ell$ . To prevent these extreme situations from occurring, the initial weights should be chosen appropriately: the more inputs into a neuron, the smaller the initial weights should be, and in all cases, the initial weights should be randomly distributed.

A common rule of thumb for initializing the weights is to choose a random number between 0 and 1 if there are 50 perceptrons in the neural network, between 0 and 0.5 if there

### Symbols and Indices

- $i$ : Training-image index label,  $i = 1, 2, \dots, I$
- $I$ : Total number of training images
- $\mathbf{f}_i$ : Vector representing unwrapped image  $f_i[n, m]$
- $f_j[i]$ : Value of  $j$ th pixel of vector  $\mathbf{f}_i$ ,  $j = 1, 2, \dots, N'$
- $\mathcal{L}$ : Total number of classes, also total number of outputs
- $N'$ : ( $= MN$ ): Length of training image vectors  $\mathbf{f}_i$ , which also is the number of input terminals at layer 0
- $k$ : Iteration index, as in  $(\cdot)^{(k)}$
- $\ell$ : Layer index in neural network,  $\ell = 0, \dots, L$ , with  $\ell = 0$  representing the input layer and  $\ell = L$  representing the output layer
- $L$ : Total number of layers, excluding the input layer (which contains terminals, not perceptrons)
- $\ell, p$ :  $p$ th neuron in  $\ell$ th layer,  $p = 1, 2, \dots, n_\ell$
- $n_\ell$ : Total number of neurons in  $\ell$ th layer

### Inputs, Outputs, and Differences

- $n_{\text{in}} = N'$ : Number of inputs
- $n_{\text{out}} = \mathcal{L}$ : Number of outputs
- $w_{\ell, p, q}[i]$ : Weight (gain) between output of  $q$ th neuron in layer  $(\ell - 1)$  and input to  $p$ th neuron in layer  $\ell$  for image  $i$

$v_{\ell, p}[i]$  = Activity of  $p$ th neuron in  $\ell$ th layer for image  $i$

$$v_{\ell, p}[i] = \sum_{q=0}^{n_{\ell-1}-1} w_{\ell, p, q}[i] y_{\ell-1, q}[i], \quad (12.21a)$$

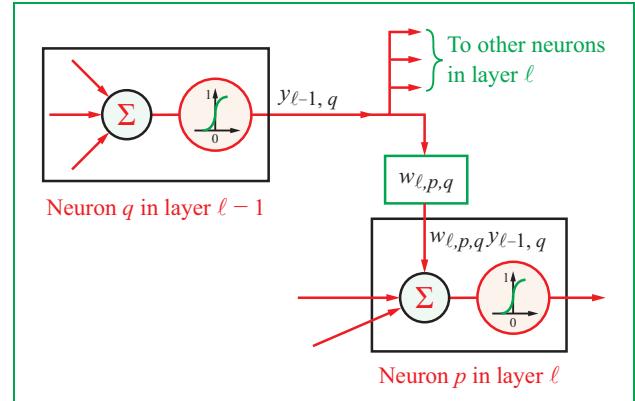
with  $y_{\ell-1, 0}[i] = 1$  for the constant input for each perceptron in layer  $\ell$ .

$$\begin{aligned} y_{\ell, p}[i] &= \text{output of } p\text{th neuron in layer } \ell \text{ for image } i \\ &= \phi(v_{\ell, p}[i]) = \frac{1}{1 + \exp(-av_{\ell, p}[i])}, \end{aligned} \quad (12.21b)$$

with  $a$  = activation constant.

$d_p[i]$  = Desired output of  $p$ th neuron in output layer  $\ell = L$  for image  $i$

$$\begin{aligned} e_{L, p}[i] &= \text{Error for } p\text{th neuron in output layer } \ell = L \\ &= d_p[i] - y_{L, p}[i]. \end{aligned} \quad (12.21c)$$



**Figure 12-11** Neuron  $q$  in layer  $(\ell - 1)$  generates output  $y_{\ell-1, q}$ . The contribution of that neuron to neuron  $p$  in layer  $\ell$  is  $w_{\ell, p, q} y_{\ell-1, q}$ , where  $w_{\ell, p, q}$  is an iteratively adjusted weighting coefficient.

are between 50 and 200 perceptrons in the neural network, and between 0 and 0.2 if there are more than 200 perceptrons in the neural network.

**2. Feed forward:** Run the neural network forward (in increasing layer index  $\ell$ ).

**3. Measure outputs of all layers:** Measure all of the outputs  $\{y_{\ell, p}^{(0)}[1], \ell = 1, 2, \dots, L\}$  of all perceptrons in all layers of the neural network, not just the output layer.

**4. Initialize output gradients:** Set  $\delta_{L, p}^{(0)}[1]$  for all perceptrons in the final layer ( $\ell = L$ ) using

$$\delta_{L, p}^{(0)}[1] = -a(d_p[1] - y_{L, p}^{(0)}[1]) y_{L, p}^{(0)}[1] (1 - y_{L, p}^{(0)}[1]), \quad (12.24)$$

where  $y_{L, p}^{(0)}[1]$  is the output of perceptron  $p$  of the last layer, generated in response to test image 1,  $d_p[1]$  is the desired output of the  $p$ th neuron in the output layer—which is either 1 or 0, depending on whether image 1 is class  $p$  or not, respectively—and  $a$  is a selectable activation constant (see Eq. (12.2)). If image 1 belongs to class 5 from among 8 classes, for example, then the desired output  $d_p[1]$  is 1 for  $p = 5$  and zero for the other 7 outputs ( $p = 1, 2, 3, 4, 6, 7$ , and 8).

**5. Backpropagate:** in decreasing layer number, generate values for the local gradients  $\delta_{\ell, p}^{(0)}[1]$ , starting at  $\ell = L$  and ending at

$\ell = 1$ , using the recipe

$$\delta_{\ell,p}^{(0)}[1] = a y_{\ell,p}^{(0)}[1] \times (1 - y_{\ell,p}^{(0)}[1]) \times \sum_{q=1}^{n_{\ell+1}} \delta_{\ell+1,q}^{(0)}[1] w_{\ell+1,q,p}^{(0)}[1]. \quad (12.25)$$

#### 6. Compute the weight iteration correction:

$$\Delta_{\ell,p,q}^{(0)}[1] = \delta_{\ell,p}^{(0)}[1] y_{\ell-1,q}^{(0)}[1] \quad (12.26)$$

for all layers and neurons in those layers. The relationship given by Eq. (12.25) is extracted from the complete derivation of backpropagation given in Section 12-3.

#### Image 1, Iteration $k = 1$

7. **Update all weights:** from those used in iteration  $k = 0$  to a new set for iteration  $k = 1$  using

$$w_{\ell,p,q}^{(1)}[1] = w_{\ell,p,q}^{(0)}[1] - \mu \Delta_{\ell,p,q}^{(0)}[1], \quad (12.27)$$

where  $\mu$  is a small step size.

8. Repeat steps 2 to 6.

#### Image 1, Iterations $k = 2$ to $K$

Increment  $k$  and repeat steps 2–6 until the process converges to a predefined set of output thresholds (weights stop changing). The final iteration is identified as  $k = K$ .

#### Images 2 to $I$

Repeat the process in steps 1–8 for all remaining images  $i = 2$  to  $I$ . For each image, initialize the neural network with the last iteration weights of the first training image. For any iteration  $k$  and image  $i$ , the relevant relationships are:

$$\delta_{L,p}^{(k)}[i] = -a(d_p[i] - y_{L,p}^{(k)}[i]) y_{L,p}^{(k)}[i] (1 - y_{L,p}^{(k)}[i]), \quad (12.28a)$$

$$\begin{aligned} \delta_{\ell,p}^{(k)}[i] &= a y_{\ell,p}^{(k)}[i] (1 - y_{\ell,p}^{(k)}[i]) \\ &\times \sum_{q=1}^{n_{\ell+1}} \delta_{\ell+1,q}^{(k)}[i] w_{\ell+1,q,p}^{(k)}[i], \end{aligned} \quad (12.28b)$$

$$\Delta_{\ell,p,q}^{(k)}[i] = \delta_{\ell,p}^{(k)}[i] y_{\ell-1,q}^{(k)}[i], \quad (12.28c)$$

and

$$w_{\ell,p,q}^{(k+1)}[i] = w_{\ell,p,q}^{(k)}[i] - \mu \Delta_{\ell,p,q}^{(k)}[i]. \quad (12.28d)$$

► Procedure 1 computes a local (and hopefully also a global) minimum of  $\mathcal{E}[i]$ . The drawback of this procedure is that it requires many runs of the neural network and backpropagation to make use of all of the training images. Hence, the procedure is seldom used in practice. ◀

#### B. Procedure 2: Iterate over Images, but Backpropagate Only Once for Each Image

1. Perform steps 1–7 of Procedure 1.
2. Repeat steps 2–7 of Procedure 1, but with a different training image. Use the weights  $w_{\ell,p,q}^{(1)}$  computed for the previous training image.
3. Repeat the preceding step for all remaining training images.

► Procedure 2 differs from Procedure 1 in that the steepest descent (SD) algorithm is run for only a *single iteration* ( $k = 0$  to  $k = 1$ ) for each training image. Procedure 2, called *incremental learning*, is often described as a *gradient descent* algorithm, but *this is an incorrect description* because the SD algorithm is run for only a *single iteration*. The computed weights  $\{w_{\ell,p,q}^{(1)}[i]\}$  do not minimize  $\mathcal{E}[i]$ —they only make it slightly smaller than the randomly chosen initial weights  $\{w_{\ell,p,q}^{(0)}[i]\}$  do. However, Procedure 2 does get all of the training images involved more quickly than Procedure 1, and is more commonly used in practice. ◀

#### C. Procedure 3: Stochastic Approach

This procedure is similar to Procedure 2 except that it uses a *stochastic gradient descent* algorithm. Given  $I$  training images, the images are partitioned into  $I_2$  **batches** of  $I_1$  images each, selected randomly. Hence,  $I = I_1 I_2$ . Using index  $i_1$  for the images within a batch, with  $\{i_1 = 1, 2, \dots, I_1\}$ , and index  $i_2$  for the batch of images, with  $\{i_2 = 1, 2, \dots, I_2\}$ , the summation defining the total mean-square error (MSE) in Eq. (12.22) can be rewritten as a sum of the error over the  $I_2$  batches:

$$\mathcal{E} = \frac{1}{I_2} \sum_{i_2=1}^{I_2} \mathcal{E}[i_2]. \quad (12.29a)$$

Here,  $\mathcal{E}[i_2]$  is the MSE of batch  $i_2$ , which is given by

$$\mathcal{E}[i_2] = \frac{1}{I_1} \sum_{i_2=1}^{I_1} \mathcal{E}[i_1, i_2], \quad (12.29b)$$

where  $\mathcal{E}[i_1, i_2]$  is the MSE for image  $i_1$  in batch  $i_2$ . In view of Eq. (12.23), for a neural network whose output layer  $L$  is composed of  $\mathcal{L}$  neurons,  $\mathcal{E}[i_1, i_2]$  is given by

$$\mathcal{E}[i_1, i_2] = \frac{1}{2} \sum_{p=1}^{\mathcal{L}} (d_p[i_1, i_2] - y_{L,p}[i_1, i_2])^2. \quad (12.29c)$$

In Procedures 1 and 2, the weights in the neural network are updated from one iteration to the next, using the expressions given by Eq. (11.48) for the local gradients  $\delta_{\ell,p}$ . A similar process is used in the current procedure, except that now the weights for each particular neuron are updated using the average weight for that neuron among  $I_1$  training images in one of the  $I_2$  batches (in practice, it does not much matter which particular batch is selected).

**Concept Question 12-5:** What is backpropagation and what is it used for?

**Concept Question 12-6:** What does it mean to train a neural network?

**Concept Question 12-7:** Does a gradient (steepest descent) algorithm always find the global minimum?

## 12-3 Derivation of Backpropagation

In the preceding section we outlined *how* to apply the backpropagation algorithm in the form of three slightly different iterative procedures. A critical ingredient is the relationship given by the combination of Eqs. (12.28c and d), namely

$$w_{\ell,p,q}^{(k+1)}[i] = w_{\ell,p,q}^{(k)}[i] - \mu \Delta_{\ell,p,q}[i] = w_{\ell,p,q}^{(k)} - \mu \delta_{\ell,p}^{(k)}[i] y_{\ell-1,q}^{(k)}[i]. \quad (12.30)$$

This relationship, which is used to update the weight  $w_{\ell,p,q}^{(k)}[i]$  of iteration  $k$  to weight  $w_{\ell,p,q}^{(k+1)}[i]$  in iteration  $(k+1)$ , is based on the **steepest descent (SD)** algorithm, which uses the iterative form

$$w_{\ell,p,q}^{(k+1)}[i] = w_{\ell,p,q}^{(k)}[i] - \mu \frac{\partial \mathcal{E}[i]}{\partial w_{\ell,p,q}^{(k)}[i]}. \quad (12.31)$$

The second term involves the differential of the mean-square error  $\mathcal{E}[i]$ . The derivation that follows will show that the second term in Eq. (12.31) is indeed equal to the second term in Eq. (12.30).

We start by defining the **local gradient**  $\delta_{\ell,p}[i]$  as

$$\delta_{\ell,p}[i] = \frac{\partial \mathcal{E}[i]}{\partial v_{\ell,p}[i]}, \quad (12.32)$$

where  $v_{\ell,p}[i]$  is the activity of neuron  $p$  in layer  $\ell$  defined in Eq. (12.21a). Using the chain rule gives

$$\frac{\partial \mathcal{E}[i]}{\partial w_{\ell,p,q}^{(k)}[i]} = \frac{\partial \mathcal{E}[i]}{\partial v_{\ell,p}[i]} \frac{\partial v_{\ell,p}[i]}{\partial w_{\ell,p,q}^{(k)}[i]} = \delta_{\ell,p}[i] \frac{\partial v_{\ell,p}[i]}{\partial w_{\ell,p,q}^{(k)}[i]}. \quad (12.33)$$

Use of the definition given by Eq. (12.21a) leads to

$$\frac{\partial v_{\ell,p}[i]}{\partial w_{\ell,p,q}^{(k)}[i]} = y_{\ell-1,q}[i], \quad (12.34)$$

and Eq. (12.33) simplifies to

$$\frac{\partial \mathcal{E}[i]}{\partial w_{\ell,p,q}^{(k)}[i]} = \delta_{\ell,p} y_{\ell-1,q}[i]. \quad (12.35)$$

A second use of the chain rule gives

$$\begin{aligned} \delta_{\ell,p}[i] &= \frac{\partial \mathcal{E}[i]}{\partial v_{\ell,p}[i]} = \sum_{s=1}^{n_{\ell+1}} \frac{\partial \mathcal{E}[i]}{\partial v_{\ell+1,s}[i]} \frac{\partial v_{\ell+1,s}[i]}{\partial v_{\ell,p}[i]} \\ &= \sum_{s=1}^{n_{\ell+1}} \delta_{\ell+1,s}[i] \frac{\partial v_{\ell+1,s}[i]}{\partial v_{\ell,p}[i]}. \end{aligned} \quad (12.36)$$

Next, change  $\ell$  to  $\ell+1$  in Eq. (12.21a) and also use Eq. (12.21b) to obtain:

$$v_{\ell+1,s}[i] = \sum_{q=0}^{n_{\ell}} w_{\ell+1,s,q}^{(k)}[i] \phi(v_{\ell,p}[i]). \quad (12.37)$$

When taking a partial derivative with respect to  $v_{\ell,p}[i]$ , only the  $q=p$  term is nonzero, and it is

$$\frac{\partial v_{\ell+1,s}[i]}{\partial v_{\ell,p}[i]} = w_{\ell+1,s,p}^{(k)}[i] \phi'(v_{\ell,p}[i]). \quad (12.38)$$

Upon factoring out  $\phi'(v_{\ell,p}[i])$ , Eq. (12.36) becomes

$$\delta_{\ell,p}[i] = \phi'(v_{\ell,p}[i]) \sum_{s=1}^{m_{\ell+1}} \delta_{\ell+1,s}[i] w_{\ell+1,s,p}^{(k)}[i]. \quad (12.39)$$

Using Eq. (12.3) for the derivative of the sigmoid function  $\phi(x)$  defined in Eq. (12.2), and also in Eq. (12.21b), gives

$$\begin{aligned} \phi'(v_{\ell,p}[i]) &= a(1 - \phi(v_{\ell,p}[i])) \phi(v_{\ell,p}[i]) \\ &= a(1 - y_{\ell,p}[i]) y_{\ell,p}[i]. \end{aligned} \quad (12.40)$$

Hence, we now can compute  $\delta_{\ell,p}[i]$  recursively in decreasing layer number  $\ell$ , starting at output layer  $\ell = L$ , using

$$\delta_{\ell,p}[i] = a(1 - y_{\ell,p}[i]) y_{\ell,p}[i] \sum_{s=1}^{m_{\ell+1}} \delta_{\ell+1,s}[i] w_{\ell+1,s,p}^{(k)}[i]. \quad (12.41)$$

We still have to initialize this recursion at  $\ell = L$ . To do that, we apply the chain rule (again!) to the definition given by Eq. (12.32) at  $\ell = L$ :

$$\delta_{L,p}[i] = \frac{\partial \mathcal{E}[i]}{\partial v_{L,p}[i]} = \frac{\partial \mathcal{E}[i]}{\partial e_{L,p}[i]} \frac{\partial e_{L,p}[i]}{\partial y_{L,p}[i]} \frac{\partial y_{L,p}[i]}{\partial v_{L,p}[i]}. \quad (12.42)$$

Each of these partial derivatives can be simplified easily. From the definition given by Eq. (12.23) for  $\mathcal{E}[i]$ , we have

$$\frac{\partial \mathcal{E}[i]}{\partial e_{L,p}[i]} = e_{L,p}[i]. \quad (12.43)$$

From the definition for  $e_{L,p}[i]$  given by Eq. (12.21c),

$$\frac{\partial e_{L,p}[i]}{\partial y_{L,p}[i]} = -1. \quad (12.44)$$

From Eqs. (12.2) and (12.3),

$$\begin{aligned} \frac{\partial y_{L,p}[i]}{\partial v_{L,p}[i]} &= \phi'(v_{L,p}[i]) \\ &= a(1 - \phi(v_{L,p}[i])) \phi(v_{L,p}[i]) \\ &= a(1 - y_{L,p}) y_{L,p}. \end{aligned} \quad (12.45)$$

Substituting these results in Eq. (12.42) gives the initialization  $\delta_{L,p}[i]$ :

$$\delta_{L,p}[i] = -a(d_p[i] - y_{L,p}[i])(1 - y_{L,p})y_{L,p}. \quad (12.46)$$

## 12-4 Neural Network Training Examples

### 12-4.1 Training Image Selection

It is customary to use only some members of the training set to train the neural network, and then use the remaining members to test the performance of the neural network. If only half of the training set members are used to train, and the other half to test, we can repeat training by exchanging these two classes. This is called ***cross-validation***.

Members of the training set should be selected at random, so that not all training set images are labeled with the same image class.

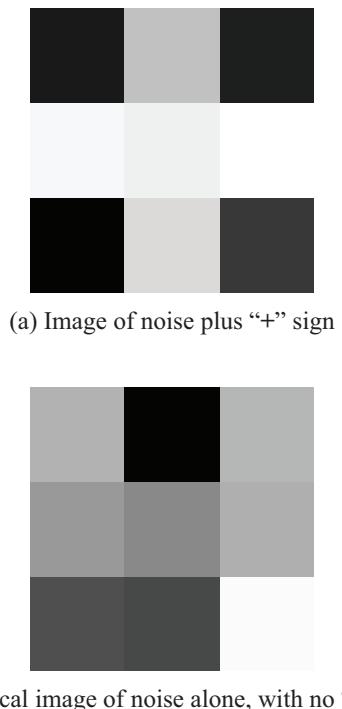
### 12-4.2 Identifying “+” Symbol

In this simple example, we train a neural network to determine the presence or absence of a “+” in a noisy  $(3 \times 3)$  image, from a set of  $I$  labeled training images, all  $(3 \times 3)$  in size, half of which contain the “+” symbol—in addition to zero-mean Gaussian noise—and half contain only noise. Typical examples of the training images are shown in Fig. 12-12.

This problem can be solved using the procedure MLE described in Section 11-2 of the preceding chapter, but the derivation requires the use of likelihood functions and relies heavily on the zero-mean white Gaussian nature of the noise. By using a neural network, we do not need to use the mathematical analyses associated with the likelihood functions, but we should evaluate the classification accuracy of the neural network to make sure it is acceptable.

The neural network we use in this example has nine terminals (nodes) in its input layer (Fig. 12-13), for the nine pixel values in a  $(3 \times 3)$  image, nine perceptrons in a single hidden layer, and a single perceptron in the output layer, which (ideally) outputs 1 if the “+” is present and zero if the “+” is not present. The total number of weights is  $9^2 = 81$  connecting the input layer to the hidden layer, and an additional 9 weights connecting the hidden layer to the output layer. These do not include the 10 constant inputs in the 10 perceptrons. Hence, the total is  $81 + 9 + 10 = 100$  weights.

The neural network was trained using 100,000 training images, half of which included the “+” and the other half of which did not. Each training image included a realization of a zero-mean Gaussian noise field with  $\sigma = 0.1$ . Typical image examples are shown in Fig. 12-12. The weights were all initialized with random numbers distributed with a zero-mean Gaussian distributions with  $\sigma = 1$ . The step size for steepest descent (SD)

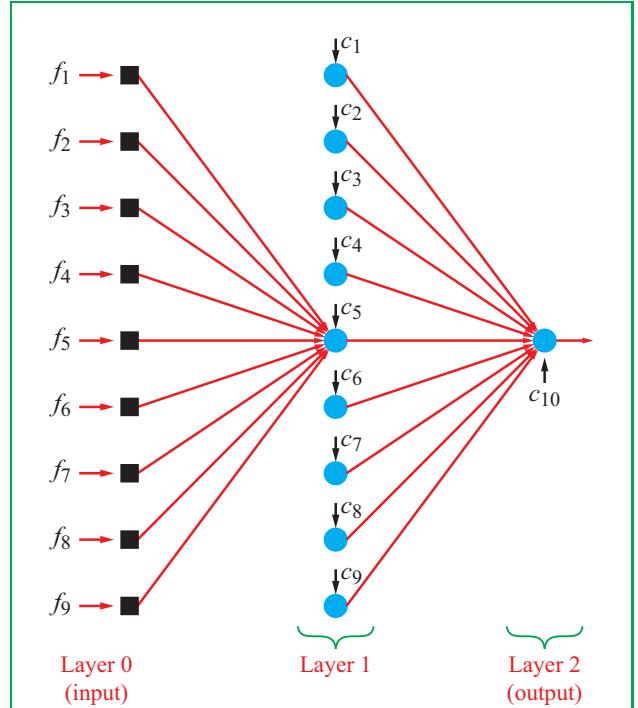


**Figure 12-12** Typical examples of noisy ( $3 \times 3$ ) images (a) with and (b) without “+” symbol.

was set at  $\mu = 0.1$ , and for the activation function  $\phi(\cdot)$ , the activation parameter was set at  $a = 7$ . With such a large value for  $a$ ,  $\phi(\cdot)$  resembles a step function. Procedure 2 of Section 12-3 was used.

The neural network was tested using 100 additional training images. The results are shown in Fig. 12-14. The correct classification is shown in blue and the neural network output classification is shown in red. The neural network correctly classified the “+” symbol in 98% of the 100 test images.

The performance of the neural network depended heavily on the initial values of the weights  $w_{\ell,p,q}$ . For a few initializations, the resulting neural network performed rather poorly, but for most initializations the resulting neural networks correctly classified all test images. A typical neural network correctly classified 98% of the test images.



**Figure 12-13** Basic structure of neural network. The input image is represented by  $\mathbf{f} = [f_1, f_2, \dots, f_q]^T$ . Each blue circle represents a perceptron that weighs the 9 inputs, adds them up, adds a constant ( $c_0$  through  $c_9$ ), and goes through an activation function  $\phi(\cdot)$ .

### 12-4.3 Zip-Code Reader

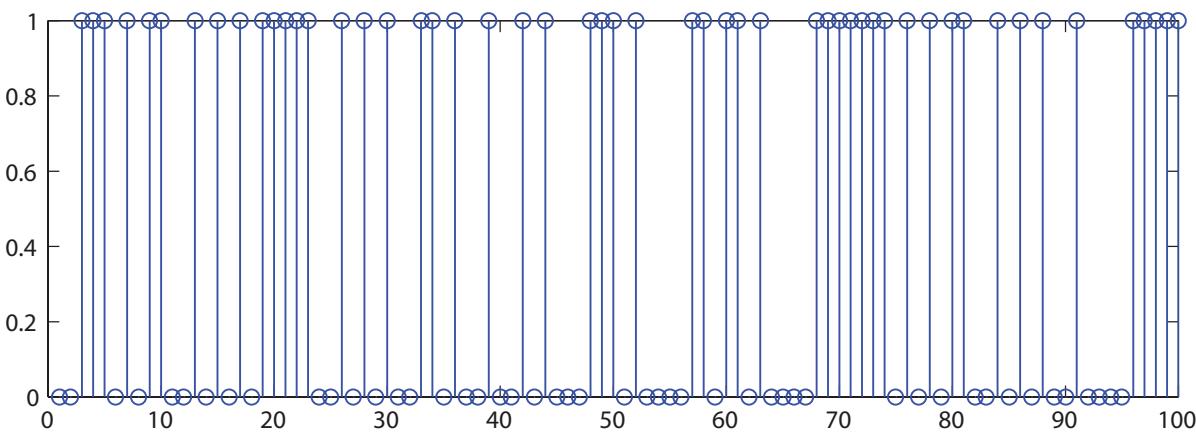
For a zip-code reader, each input  $\mathbf{f}[i]$  is a vector of length 784. Each element of  $\mathbf{f}[i]$  is a pixel value read by a camera that reads each digit as a  $(28 \times 28)$  image. The number of pixels is then  $28^2 = 784$ . Each training image is labeled with a digit (one of  $\{0, 1, \dots, 9\}$ ) selected by a human judge. Training image  $\mathbf{f}[i]$ , the corresponding output  $\mathbf{y}[i]$ , and the desired output  $\mathbf{d}[i]$  are given by

$$\mathbf{f}[i] = [f_1[i], f_2[i], \dots, f_{784}[i]]^T, \quad \{i = 1, 2, \dots, I\}, \quad (12.47a)$$

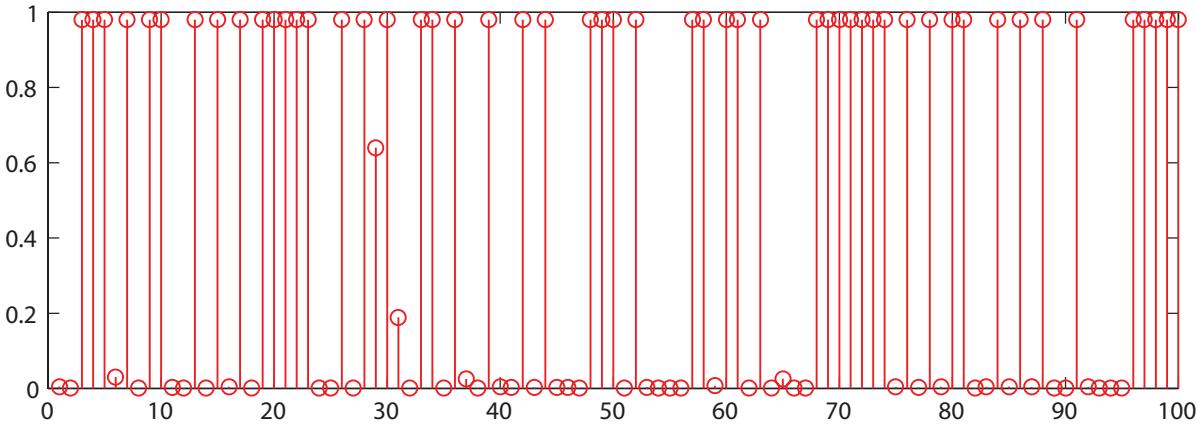
$$\mathbf{y}[i] = [y_1[i], y_2[i], \dots, y_{10}[i]]^T, \quad (12.47b)$$

and

$$\mathbf{d}[i] = [d_1[i], d_2[i], \dots, d_{10}[i]]^T, \quad (12.47c)$$



**Figure 12-14** Performance of neural network for Detection of “+.” Correct in blue, output in red.

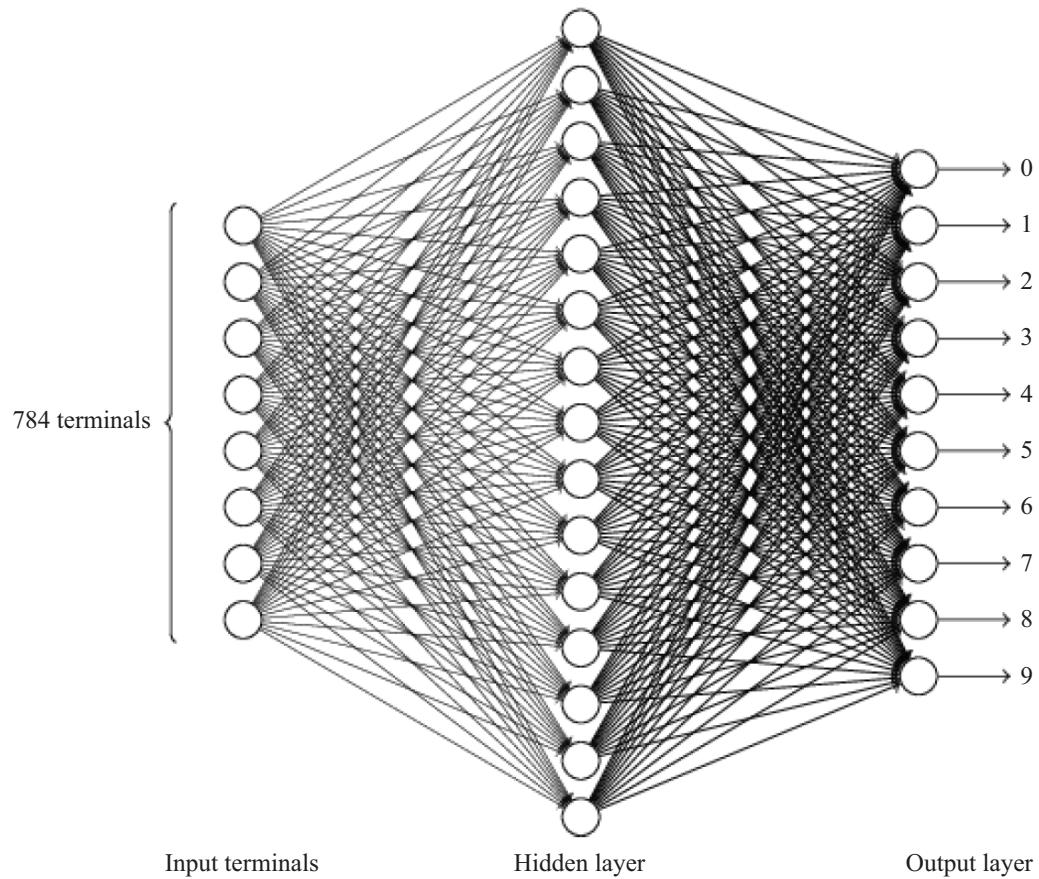


**Fig. 12-15.**

with

$$d_j[i] = \begin{cases} 1 & \text{for } j = \text{correct digit,} \\ 0 & \text{for } j = \text{incorrect digit.} \end{cases} \quad (12.47d)$$

A training set  $I = 60,000$  images of handwritten digits is available at the U.S. National Institute of Standards and Technology (NIST). The zip code digits were handwritten by 250 different people. The neural network, comprised of 784 input terminals, 10 output terminals, and many hidden layers is illustrated in



**Figure 12-15** The basic structure of a zip-code reader.

## Summary

### Concepts

- The output of a perceptron is the application of an activation function to a weighted sum of its inputs. A perceptron mimics the biological action of a neuron, and perceptrons are often called neurons.
- A common choice for activation function is the sigmoid function (below).
- Classification rules defined by a separating hyperplane can be implemented using a single perceptron.
- A neural network is a network of perceptrons, connected in a way that mimics the connections of neurons in the

- brain.
- The weights in a neural network are computed using an algorithm called backpropagation, using a set of labelled training images. Backpropagation uses one iteration of a gradient or steepest descent algorithm.
- Computing the weights in a neural network by applying backpropagation using a set of labelled training images is called training the neural network. There are three different ways of performing training.

### Mathematical Formulae

#### Perceptron

$$y = \phi(w_0 + w_1x_1 + \dots + w_Nx_N)$$

#### Sigmoid function

$$\phi(x) = \frac{1}{1 + e^{-ax}}$$

### Important Terms

Provide definitions or explain the meaning of the following terms:

activation function  
backpropagation  
gradient

hidden layers  
input layer  
neural network

neuron  
output layer  
perceptron

sigmoid function  
steepest descent  
supervised learning

training

## PROBLEMS

### Section 12-1: Overview of Neural Networks

**12.1** In Exercise 12-1, you had to determine by inspection the weights of a perceptron so that it implemented an OR gate. Write out a set of nonlinear equations whose solution is the weights in Fig. 12-1(a).

**12.2** In Exercise 12-2, you had to determine by inspection the weights of a perceptron so that it implemented an AND gate. Write out a set of nonlinear equations whose solution is the weights in Fig. 12-1(a).

**12.3** An image  $\begin{bmatrix} g_{0,0} & g_{1,0} \\ g_{1,0} & g_{1,1} \end{bmatrix}$  is to be classified as either  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  or  $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ . Write a set of equations whose solution is the weights replacing those in Fig. 12-3(a).

**12.4** An image  $\begin{bmatrix} g_{0,0} & g_{1,0} \\ g_{1,0} & g_{1,1} \end{bmatrix}$  is to be classified as either  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  or  $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ . Specify the weights in a perceptron like Fig. 12-3(a) that classifies the image.

**12.5** A binary adder implements binary addition (with carry). It has the truth table

$x_1$	0	0	1	1
$x_2$	0	1	0	1
sum	0	1	1	0
carry	0	0	0	1

where  $\text{sum} = x_1 + x_2 \pmod{2}$  and carry is the carry ( $1 + 1 = 10$  base 2). Modify the neural network in Fig. 12-5(b) to implement a binary adder.

**12.6** A binary-to-decimal converter accepts as input a 2-bit binary number  $(x_1x_2)_2$  and converts it to a decimal number 0,

1, 2 or 3. Design a neural network that accepts as inputs  $\{x_1, x_2\}$  and outputs  $\{y_0, y_1, y_2, y_3\}$  where if  $(x_1 x_2)_2 = K$ , then  $y_K = 1$  and  $\{y_k, k \neq K\}$  are all zero.

**12.7** (This problem may require review of Chapter 2.) A 1-D signal  $x(t)$  of duration 1 s is sampled at 44100 samples/s, resulting in discrete-time signal  $x[n] = x(n/44100)$  of duration 44100 s. Design a neural network for determining the presence or absence of a trumpet playing note A (fundamental frequency 440 Hz) by examining the first four harmonics of the trumpet signal. Assume the real parts of their DFT are positive.

**12.8** (This problem may require review of Chapter 5.) Design a neural network for edge detection on a 1-D signal  $\{x[n]\}$  of duration  $N$ . An edge is at  $n = n_0$  if  $|x[n_0] - x[n_0 - 1]| > T$  for some threshold  $T$ . Let  $x[-1] = x[0]$ . The extension to 2-D edge detection of horizontal or vertical edges is straightforward.

## Section 12-2: Training Neural Networks

**12.9** The program `neuron.m` trains a neuron using labeled training vectors. Run the program to train (determine 2 weights of) a neuron to implement an OR gate. This is the same problem as Problem 12.1. The neuron has the form of [Fig. 12-1\(a\)](#). Use 1000 iterations with step size  $\mu = 0.01$ ,  $a = 7$ , and initialize all weights with 0.1. Compare the neuron outputs  $y[i]$  with the desired output  $d[j]$  for  $j = 1, 2, 3, 4$  in a table.

**12.10** The program `neuron.m` trains a neuron using labelled training vectors. Run the program to train (determine 2 weights of) a neuron to implement an AND gate. This is the same problem as Problem 12.2. The neuron has the form of [Fig. 12-1\(a\)](#). Use 1000 iterations with step size  $\mu = 0.01$ ,  $a = 7$ , and initialize all weights with 0.1. Compare the neuron outputs  $y[i]$  with the desired output  $d[j]$  for  $j = 1, 2, 3, 4$  in a table.

**12.11** The program `neuron.m` trains a neuron using labelled training vectors. Run the program to train a neuron to classify a  $(2 \times 2)$  image as  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  or  $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ . This is the same problem as Problem 12.3. The neuron has the form of [Fig. 12-1\(b\)](#). Use 1000 iterations with step size  $\mu = 0.01$ ,  $a = 7$ , and initialize all weights with 0.1. Compare the neuron outputs  $y[i]$  with the desired output  $d[j]$  for  $j = 1, 2$  in a table.

## Section 12-3: Derivation of Backpropagation

**12.12** We clarify the derivation of backpropagation by applying it to a single neuron. Single neurons are discussed in Section 12-1.1 and illustrated in [Fig. 12-1](#). We are given  $I$

training input  $M$ -vectors  $\mathbf{x}[i] = [x_1[i], x_2[i], \dots, x_M[i]]^T$ , where  $i = 1 \dots I$  and  $I$  labels  $\{d[i], i = 1, \dots, I\}$ , where  $d[i]$  is the desired output for training vector  $\mathbf{x}[i]$ . The goal is to compute weights  $\{w_j, j = 0, \dots, M\}$  that minimize  $\mathcal{E}[i] = \frac{1}{2} (d[i] - y[i])^2$ , where  $y[i] = \phi(\sum_{j=0}^M w_j x_j[i])$  and  $x_0 = 1$  implements the single neuron.

- (a)** Derive a steepest descent algorithm to compute  $\{w_j, j = 0, \dots, M\}$  minimizing  $\mathcal{E}[i]$ .
- (b)** Show that this is the output layer  $\ell = \mathcal{L}$  in the backpropagation derivation.

## Section 12-4: Neural Network Training Examples

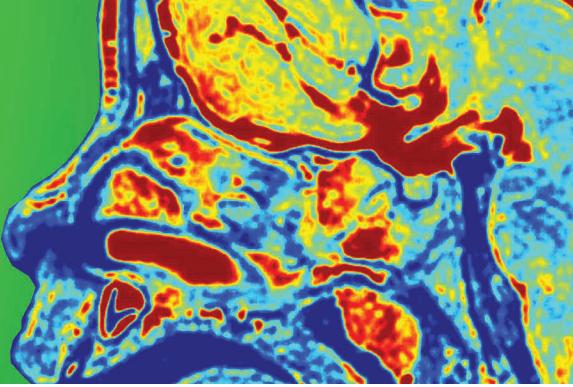
**12.13** Program `P1213.m` creates 100 random points in the square  $-10 < x_1, x_2 < 10$  and labels them as being inside or outside a circle of radius 8 centered at the origin (so the areas inside and outside the circle are roughly equal:  $\pi(8)^2 = 201 \approx 200 = \frac{1}{2}(20)^2$ ). It then trains a neural network with 2 inputs ( $x_1$  and  $x_2$  coordinates of each point), 1 output neuron (for inside or outside the circle), and a hidden layer of 10 neurons. It uses 1000 iterations, each running over 100 training points (2 coordinates each),  $\mu = 0.01$ , and  $a = 7$ . Run `P1213.m` using different initializations until it successfully assigns each training point as being in category #1 (inside) or #2 (outside) the circle. Of course, the neural network doesn't "know" circles; it "learns" this from training.

**12.14** Program `P1214.m` creates 100 random points in the square  $-10 < x_1, x_2 < 10$  and labels them as being inside or outside a parabola  $x_2 = x_1^2/10$ . It then trains a neural network with 2 inputs ( $x_1$  and  $x_2$  coordinates of each point), 1 output neuron (for inside or outside the parabola), and a hidden layer of 10 neurons. It uses 1000 iterations, each running over 100 training points (2 coordinates each),  $\mu = 0.01$ , and  $a = 7$ . Run `P1214.m` using different initializations until it successfully assigns each training point as being in category #1 (inside) or #2 (outside) the parabola. Of course, the neural network doesn't "know" parabolas; it "learns" this from training.

**12.15** Program `P1215.m` creates 100 random points in the square  $-10 < x_1, x_2 < 10$  and labels them as being inside or outside 4 quarter circles centered on the corners. The areas inside and outside the circle are roughly equal:  $\pi(8)^2 = 201 \approx 200 = \frac{1}{2}(20)^2$ . It then trains a neural network with 2 inputs ( $x_1$  and  $x_2$  coordinates of each point), 1 output neuron (for inside or outside the parabola), and a hidden layer of 10 neurons. It uses 1000 iterations, each running over 100 training points (2 coordinates each),  $\mu = 0.01$ , and  $a = 7$ . Run `P1215.m` using different initializations until it successfully

assigns each training point as being in category #1 (inside) or #2 (outside) the circles. Of course, the neural network doesn't "know" circles; it "learns" this from training.

# APPENDIX A



## A

## Review of Complex Numbers

A **complex number**  $\mathbf{z}$  may be written in the **rectangular form**

$$\mathbf{z} = x + jy, \quad (\text{A.1})$$

where  $x$  and  $y$  are the **real** ( $\Re(\mathbf{z})$ ) and **imaginary** ( $\Im(\mathbf{z})$ ) parts of  $\mathbf{z}$ , respectively, and  $j = \sqrt{-1}$ . That is,

$$x = \Re(\mathbf{z}), \quad y = \Im(\mathbf{z}). \quad (\text{A.2})$$

Note that  $\Im(3 + j4) = 4$ , not  $j4$ .

Alternatively,  $\mathbf{z}$  may be written in **polar form** as

$$\mathbf{z} = |\mathbf{z}|e^{j\theta} = |\mathbf{z}|/\theta \quad (\text{A.3})$$

where  $|\mathbf{z}|$  is the magnitude of  $\mathbf{z}$ ,  $\theta$  is its phase angle, and the form  $/\theta$  is a useful shorthand representation commonly used in numerical calculations. By applying **Euler's identity**,

$$e^{j\theta} = \cos \theta + j \sin \theta, \quad (\text{A.4})$$

we can convert  $\mathbf{z}$  from polar form, as in Eq. (A.3), into rectangular form, as in Eq. (A.1),

$$\mathbf{z} = |\mathbf{z}|e^{j\theta} = |\mathbf{z}| \cos \theta + j |\mathbf{z}| \sin \theta, \quad (\text{A.5})$$

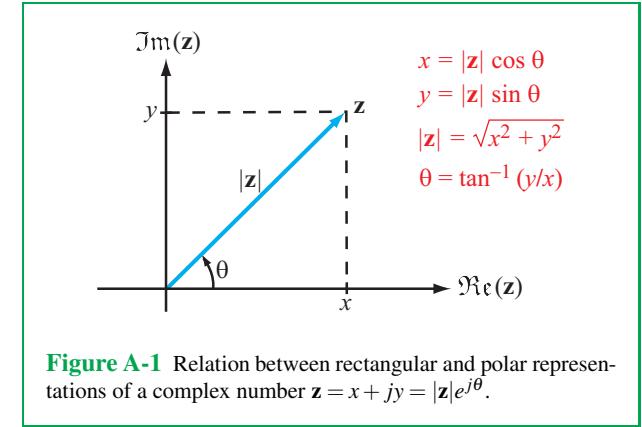
which leads to the relations

$$x = |\mathbf{z}| \cos \theta, \quad y = |\mathbf{z}| \sin \theta, \quad (\text{A.6})$$

$$|\mathbf{z}| = \sqrt{x^2 + y^2}, \quad \theta = \tan^{-1}(y/x). \quad (\text{A.7})$$

The two forms of  $\mathbf{z}$  are illustrated graphically in **Fig. A-1**. Because in the complex plane, a complex number assumes the form of a vector, it is represented by a bold letter.

When using Eq. (A.7), care should be taken to ensure that  $\theta$  is in the proper quadrant by noting the signs of  $x$  and  $y$  individually.



**Figure A-1** Relation between rectangular and polar representations of a complex number  $\mathbf{z} = x + jy = |\mathbf{z}|e^{j\theta}$ .

as illustrated in **Fig. A-2**. Specifically,

$$\theta = \begin{cases} \tan^{-1}(y/x) & \text{if } x > 0, \\ \tan^{-1}(y/x) \pm \pi & \text{if } x < 0, \\ \pi/2 & \text{if } x = 0 \text{ and } y > 0, \\ -\pi/2 & \text{if } x = 0 \text{ and } y < 0. \end{cases}$$

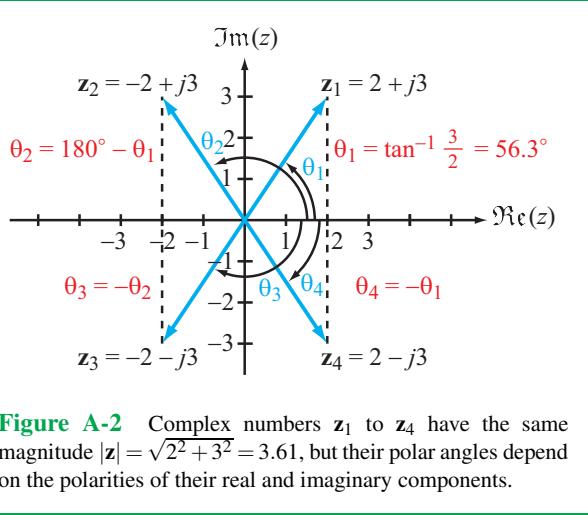
Complex numbers  $\mathbf{z}_2$  and  $\mathbf{z}_4$  point in opposite directions and their phase angles  $\theta_2$  and  $\theta_4$  differ by  $180^\circ$ , despite the fact that  $(y/x)$  has the same value in both cases.

The **complex conjugate** of  $\mathbf{z}$ , denoted with a star superscript (or asterisk), is obtained by replacing  $j$  (wherever it appears) with  $-j$ , so that

$$\mathbf{z}^* = (x + jy)^* = x - jy = |\mathbf{z}|e^{-j\theta} = |\mathbf{z}|/\theta. \quad (\text{A.8})$$

The magnitude  $|\mathbf{z}|$  is equal to the positive square root of the product of  $\mathbf{z}$  and its complex conjugate:

$$|\mathbf{z}| = \sqrt{\mathbf{z}\mathbf{z}^*}. \quad (\text{A.9})$$



We now highlight some of the salient properties of complex algebra.

**Equality:** If two complex numbers  $\mathbf{z}_1$  and  $\mathbf{z}_2$  are given by

$$\mathbf{z}_1 = x_1 + jy_1 = |\mathbf{z}_1|e^{j\theta_1}, \quad (\text{A.10a})$$

$$\mathbf{z}_2 = x_2 + jy_2 = |\mathbf{z}_2|e^{j\theta_2}, \quad (\text{A.10b})$$

then  $\mathbf{z}_1 = \mathbf{z}_2$  if and only if (*iff*)  $x_1 = x_2$  and  $y_1 = y_2$  or, equivalently,  $|\mathbf{z}_1| = |\mathbf{z}_2|$  and  $\theta_1 = \theta_2$ .

### Addition:

$$\mathbf{z}_1 + \mathbf{z}_2 = (x_1 + x_2) + j(y_1 + y_2). \quad (\text{A.11})$$

### Multiplication:

$$\begin{aligned} \mathbf{z}_1 \mathbf{z}_2 &= (x_1 + jy_1)(x_2 + jy_2) \\ &= (x_1 x_2 - y_1 y_2) + j(x_1 y_2 + x_2 y_1), \end{aligned} \quad (\text{A.12a})$$

or

$$\begin{aligned} \mathbf{z}_1 \mathbf{z}_2 &= |\mathbf{z}_1|e^{j\theta_1} \cdot |\mathbf{z}_2|e^{j\theta_2} \\ &= |\mathbf{z}_1||\mathbf{z}_2|e^{j(\theta_1 + \theta_2)} \\ &= |\mathbf{z}_1||\mathbf{z}_2|[\cos(\theta_1 + \theta_2) + j\sin(\theta_1 + \theta_2)]. \end{aligned} \quad (\text{A.12b})$$

**Division:** For  $\mathbf{z}_2 \neq 0$ ,

$$\begin{aligned} \frac{\mathbf{z}_1}{\mathbf{z}_2} &= \frac{x_1 + jy_1}{x_2 + jy_2} \\ &= \frac{(x_1 + jy_1)}{(x_2 + jy_2)} \cdot \frac{(x_2 - jy_2)}{(x_2 - jy_2)} \\ &= \frac{(x_1 x_2 + y_1 y_2) + j(x_2 y_1 - x_1 y_2)}{x_2^2 + y_2^2}, \end{aligned} \quad (\text{A.13a})$$

or

$$\begin{aligned} \frac{\mathbf{z}_1}{\mathbf{z}_2} &= \frac{|\mathbf{z}_1|e^{j\theta_1}}{|\mathbf{z}_2|e^{j\theta_2}} \\ &= \frac{|\mathbf{z}_1|}{|\mathbf{z}_2|} e^{j(\theta_1 - \theta_2)} \\ &= \frac{|\mathbf{z}_1|}{|\mathbf{z}_2|} [\cos(\theta_1 - \theta_2) + j\sin(\theta_1 - \theta_2)]. \end{aligned} \quad (\text{A.13b})$$

**Powers:** For any positive integer  $n$ ,

$$\begin{aligned} \mathbf{z}^n &= (|\mathbf{z}|e^{j\theta})^n \\ &= |\mathbf{z}|^n e^{jn\theta} = |\mathbf{z}|^n (\cos n\theta + j\sin n\theta), \end{aligned} \quad (\text{A.14})$$

$$\begin{aligned} \mathbf{z}^{1/2} &= \pm |\mathbf{z}|^{1/2} e^{j\theta/2} \\ &= \pm |\mathbf{z}|^{1/2} [\cos(\theta/2) + j\sin(\theta/2)]. \end{aligned} \quad (\text{A.15})$$

### Useful relations:

$$-1 = e^{j\pi} = e^{-j\pi} = 1/\underline{180^\circ}, \quad (\text{A.16a})$$

$$j = e^{j\pi/2} = 1/\underline{90^\circ}, \quad (\text{A.16b})$$

$$-j = -e^{j\pi/2} = e^{-j\pi/2} = 1/\underline{-90^\circ}, \quad (\text{A.16c})$$

$$\sqrt{j} = (e^{j\pi/2})^{1/2} = \pm e^{j\pi/4} = \frac{\pm(1+j)}{\sqrt{2}}, \quad (\text{A.16d})$$

$$\sqrt{-j} = \pm e^{-j\pi/4} = \frac{\pm(1-j)}{\sqrt{2}}. \quad (\text{A.16e})$$

For quick reference, the preceding properties of complex numbers are summarized in Table A-1. Note that if a complex number is given by  $(a + jb)$  and  $b = 1$ , it can be written either as  $(a + j1)$  or simply as  $(a + j)$ . Thus,  $j$  is synonymous with  $j1$ .

**Table A-1** Properties of complex numbers.

<b>Euler's Identity:</b> $e^{j\theta} = \cos \theta + j \sin \theta$	
$\sin \theta = \frac{e^{j\theta} - e^{-j\theta}}{2j}$	$\cos \theta = \frac{e^{j\theta} + e^{-j\theta}}{2}$
$\mathbf{z} = x + jy =  \mathbf{z} e^{j\theta}$	$\mathbf{z}^* = x - jy =  \mathbf{z} e^{-j\theta}$
$x = \Re(\mathbf{z}) =  \mathbf{z}  \cos \theta$	$ \mathbf{z}  = \sqrt{\mathbf{z}\mathbf{z}^*} = \sqrt{x^2 + y^2}$
$y = \Im(\mathbf{z}) =  \mathbf{z}  \sin \theta$	$\theta = \tan^{-1}(y/x)$
$\mathbf{z}^n =  \mathbf{z} ^n e^{jn\theta}$	$\mathbf{z}^{1/2} = \pm  \mathbf{z} ^{1/2} e^{j\theta/2}$
$\mathbf{z}_1 = x_1 + jy_1$	$\mathbf{z}_2 = x_2 + jy_2$
$\mathbf{z}_1 = \mathbf{z}_2$ iff $x_1 = x_2$ and $y_1 = y_2$	$\mathbf{z}_1 + \mathbf{z}_2 = (x_1 + x_2) + j(y_1 + y_2)$
$\mathbf{z}_1 \mathbf{z}_2 =  \mathbf{z}_1   \mathbf{z}_2  e^{j(\theta_1 + \theta_2)}$	$\frac{\mathbf{z}_1}{\mathbf{z}_2} = \frac{ \mathbf{z}_1 }{ \mathbf{z}_2 } e^{j(\theta_1 - \theta_2)}$
$-1 = e^{j\pi} = e^{-j\pi} = 1/\pm 180^\circ$	
$j = e^{j\pi/2} = 1/\pm 90^\circ$	$-j = e^{-j\pi/2} = 1/\pm 90^\circ$
$\sqrt{j} = \pm e^{j\pi/4} = \pm \frac{(1+j)}{\sqrt{2}}$	$\sqrt{-j} = \pm e^{-j\pi/4} = \pm \frac{(1-j)}{\sqrt{2}}$

**Example A-1: Working with Complex Numbers**

Given two complex numbers

$$\begin{aligned}\mathbf{V} &= 3 - j4, \\ \mathbf{I} &= -(2 + j3),\end{aligned}$$

- (a) express  $\mathbf{V}$  and  $\mathbf{I}$  in polar form, and find (b)  $\mathbf{VI}$ , (c)  $\mathbf{VI}^*$ , (d)  $\mathbf{V}/\mathbf{I}$ , and (e)  $\sqrt{\mathbf{I}}$ .

**Solution:**

(a)

$$\begin{aligned}|\mathbf{V}| &= \sqrt{\mathbf{V}\mathbf{V}^*} \\ &= \sqrt{(3 - j4)(3 + j4)} = \sqrt{9 + 16} = 5, \\ \theta_V &= \tan^{-1}(-4/3) = -53.1^\circ,\end{aligned}$$

$$\mathbf{V} = |\mathbf{V}|e^{j\theta_V} = 5e^{-j53.1^\circ} = 5/\underline{-53.1^\circ},$$

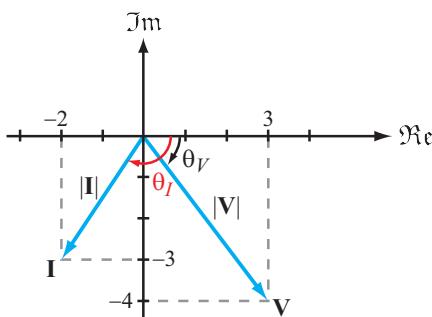
$$|\mathbf{I}| = \sqrt{2^2 + 3^2} = \sqrt{13} = 3.61.$$

Since  $\mathbf{I} = (-2 - j3)$  is in the third quadrant in the complex plane (**Fig. A-3**),

$$\begin{aligned}\theta_I &= -180^\circ + \tan^{-1}\left(\frac{3}{2}\right) = -123.7^\circ, \\ \mathbf{I} &= 3.61/\underline{-123.7^\circ}.\end{aligned}$$

Alternatively, whenever the real part of a complex number is negative, we can factor out a  $(-1)$  multiplier and then use Eq. (A.16a) to replace it with a phase angle of either  $+180^\circ$  or  $-180^\circ$ , as needed. In the case of  $\mathbf{I}$ , the process is as follows:

$$\begin{aligned}\mathbf{I} &= -2 - j3 = -(2 + j3) \\ &= e^{\pm j180^\circ} \cdot \sqrt{2^2 + 3^2} e^{j\tan^{-1}(3/2)} \\ &= 3.61e^{j57.3^\circ} e^{\pm j180^\circ}.\end{aligned}$$



**Figure A-3** Complex numbers  $\mathbf{V}$  and  $\mathbf{I}$  in the complex plane (Example A-1).

Since our preference is to end up with a phase angle within the range between  $-180^\circ$  and  $+180^\circ$ , we will choose  $-180^\circ$ . Hence,

$$\mathbf{I} = 3.61e^{-j123.7^\circ}.$$

(b)

$$\begin{aligned}\mathbf{VI} &= (5/-53.1^\circ)(3.61/-123.7^\circ) \\ &= (5 \times 3.61)/(-53.1^\circ - 123.7^\circ) = 18.05/-176.8^\circ.\end{aligned}$$

(c)

$$\mathbf{VI}^* = 5e^{-j53.1^\circ} \times 3.61e^{j123.7^\circ} = 18.05e^{j70.6^\circ}.$$

(d)

$$\frac{\mathbf{V}}{\mathbf{I}} = \frac{5e^{-j53.1^\circ}}{3.61e^{-j123.7^\circ}} = 1.39e^{j70.6^\circ}.$$

(e)

$$\begin{aligned}\sqrt{\mathbf{I}} &= \sqrt{3.61e^{-j123.7^\circ}} \\ &= \pm\sqrt{3.61} e^{-j123.7^\circ/2} = \pm 1.90e^{-j61.85^\circ}.\end{aligned}$$

**Exercise A-1:** Express the following complex functions in polar form:

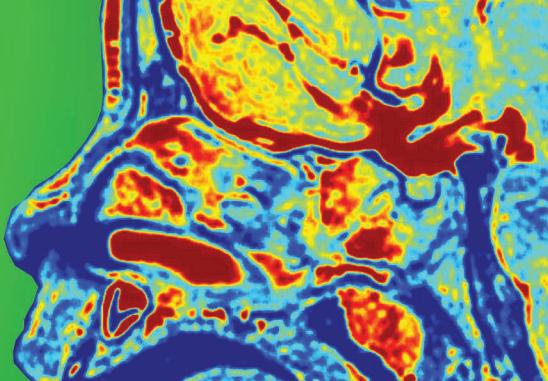
$$\mathbf{z}_1 = (4 - j3)^2,$$

$$\mathbf{z}_2 = (4 - j3)^{1/2}.$$

**Answer:**  $\mathbf{z}_1 = 25/-73.7^\circ$ ,  $\mathbf{z}_2 = \pm\sqrt{5}/-18.4^\circ$ . (See [IP](#))

**Exercise A-2:** Show that  $\sqrt{2j} = \pm(1 + j)$ . (See [IP](#))

# APPENDIX B



## B

## MATLAB® and MathScript

### A Short Introduction for Use in Image Processing

#### B-1 Background

“A computer will always do exactly what you tell it to do. But that may not be what you had in mind.”—a quote from the 1950’s.

This Appendix is a short introduction to MATLAB and MathScript *for this book*. It is not comprehensive; only commands directly applicable to signal and image processing are covered. No commands in any of MATLAB’s Toolboxes are included, since these commands are not included in basic MATLAB or MathScript. Programming concepts and techniques are not included, since they are not used anywhere in this book.

#### MATLAB

MATLAB is a computer program developed and sold by the Mathworks, Inc. It is the most commonly used program in signal processing, but it is used in all fields of engineering.

“MATLAB” (**matrix laboratory**) was originally based on a set of numerical linear algebra programs, written in FORTRAN, called LINPACK. So MATLAB tends to formulate problems in terms of vectors and arrays of numbers, and often solves problems by formulating them as linear algebra problems.

The student edition of MATLAB is much cheaper than the professional version of MATLAB. It is licensed for use by all undergraduate and graduate students. Every program on the website for this book will run on it.

#### MathScript

MathScript is a computer program developed and sold by National Instruments, as a module in LabVIEW. The basic commands used by MATLAB also work in MathScript, but higher-level MATLAB commands, and those in Toolboxes, usually do not work in MathScript. Unless otherwise noted, all

MATLAB commands used in this book and website also work in MathScript.

One important exception is `colormap(gray)`. To make this work in MathScript, `G=[0:64]/64;gray=G'*[1 1 1];` must be inserted at the beginning of the program.

Instructions on how to acquire a student version of MathScript are included on the website accompanying the book, as part of the student edition of LabVIEW. In the sequel, we use “M/M” to designate “MATLAB or MathScript.”

Freemat and GNU Octave are freeware programs that are mostly compatible with MATLAB.

#### Getting Started

To install the student version of MathScript included on the website, follow the instructions.

When you run M/M, a **prompt** `>>` will appear when it is ready. Then you can type commands. Your first command should be `>>cd mydirectory`, to change directory to your working directory, which we call “mydirectory” here.

We will use **this font** to represent typed commands and generated output. You can get help for any command, such as `plot`, by typing at the prompt `help plot`.

Some basic things to know about M/M:

- Inserting a **semicolon** “`;`” at the end of a command suppresses output; without it M/M will type the results of the computation. This is harmless, but it is irritating to have numbers flying by on your screen.
- Inserting **ellipses** “`...`” at the end of a command means it is continued on the next line. This is useful for long commands.
- Inserting “`%`” at the beginning of a line makes the line a **comment**; it will not be executed. Comments are used to explain what the program is doing at that point.
- `clear` eliminates all present variables. Programs should always start with a `clear`.
- `whos` shows all variables and their sizes.

- M/M variables are case-sensitive: `t` and `T` are different variables.
- `save myfile X, Y` saves the variables `X` and `Y` in the file `myfile.mat` for use in another session of M/M at another time.
- `load myfile` loads all variables saved in `myfile.mat`, so they can now be used in the present session of M/M.
- `quit` ends the present session of M/M.

## .m Files

An M/M program is a list of commands executed in succession. Programs are called “m-files” since their extension is “`.m`,” or “scripts.”

To write an `.m` file, at the upper left, click:

**File→New→m-file**

This opens a window with a text editor.

Type in your commands and then do this:

**File→Save as→myname.m**

Make sure you save it with an `.m` extension. Then you can run the file by typing its name at the prompt: `>>myname`. Make sure the file name is not the same as a Matlab command! Using your own name is a good idea.

You can access previously-typed commands using uparrow and downarrow on your keyboard.

To download a file from a website, right-click on it, select **save target as**, and use the menu to select the proper file type (specified by its file extension).

## B-2 Basic Computation

### B-2.1 Basic Arithmetic

- Addition: `3+2` gives `ans=5`
- Subtraction: `3-2` gives `ans=1`
- Multiplication: `2*3` gives `ans=6`
- Division: `6/2` gives `ans=3`
- Powers: `2^3` gives `ans=8`
- Others: `sin, cos, tan, exp, log, log10`
- Square root: `sqrt(49)` gives `ans=7`
- Conjugate: `conj(3+2j)` gives `ans=3-2i`

Both `i` and `j` represent  $\sqrt{-1}$ ; answers use `i, pi` represents  $\pi$ . `e` does not represent 2.71828.

### B-2.2 Entering Vectors and Arrays

To enter *row vector* [1 2 3] and store it in `A`, type at the prompt `A=[1 2 3];` or `A=[1,2,3];`

To enter the same numbers as a *column vector* and store it in `A`, type at the prompt *either* `A=[1;2;3];` or `A=[1 2 3];A=A'`; Note `A=A'` replaces `A` with its transpose. “Transpose” means “convert rows to columns, and vice-versa.”

To enter a vector of consecutive or equally-spaced numbers, follow these examples:

- `[2:6]` gives `ans=2 3 4 5 6`
- `[3:2:9]` gives `ans=3 5 7 9`
- `[4:-1:1]` gives `ans=4 3 2 1`

To enter an *array* or matrix of numbers, type, for example, `B=[3 1 4;1 5 9;2 6 5];` This gives the array `B` and its transpose `B'`

$$B = \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix} \quad B' = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 5 & 6 \\ 4 & 9 & 5 \end{bmatrix}$$

Other basics of arrays:

- `ones(M, N)` is an  $M \times N$  array of “1”
- `zeros(M, N)` is an  $M \times N$  array of “0”
- `length(X)` gives the length of vector `X`
- `size(X)` gives the size of array `X`  
For `B` above, `size(B)` gives `ans=3 3`
- `A(I, J)` gives the (I,J)th element of `A`.  
For `B` above, `B(2, 3)` gives `ans=9`

### B-2.3 Array Operations

Arrays add and subtract point-by-point:

`X=[3 1 4]; Y=[2 7 3]; X+Y` gives `ans=5 8 7`

But `X*Y` generates an error message.

To compute various types of vector products:

- To multiply element-by-element, use `X.*Y` This gives `ans=6 7 12`. To divide element-by-element, type `X./Y`

- To find the inner product of  $\mathbf{X}$  and  $\mathbf{Y}$ , which is  $(3)(2)+(1)(7)+(4)(3)=25$ , use  $\mathbf{X} * \mathbf{Y}'$ . This gives  $\text{ans}=25$
- To find the outer product of  $\mathbf{X}$  and  $\mathbf{Y}$ , which is

$$\begin{bmatrix} (3)(2) & (3)(7) & (3)(3) \\ (1)(2) & (1)(7) & (1)(3) \\ (4)(2) & (4)(7) & (4)(3) \end{bmatrix} \quad \text{use } \mathbf{X}' * \mathbf{Y}$$

This gives the above matrix.

A common problem is when you think you have a row vector when in fact you have a column vector. Check by using `size(X)`; in the present example, the command gives `ans=1, 3` which tells you that  $\mathbf{X}$  is a  $1 \times 3$  (row) vector.

- The following functions operate on each element of an array separately, giving another array:

`sin, cos, tan, exp, log, log10, sqrt  
cos([0:3]*pi)` gives `ans=1 -1 1 -1`

- To compute  $n^2$  for  $n = 0, 1 \dots 5$ , use `[0:5].^2` which gives `ans=0 1 4 9 16 25`
- To compute  $2^n$  for  $n = 0, 1 \dots 5$ , use `2.^[0:5]` which gives `ans=1 2 4 8 16 32`

Other array operations include:

- `A=[1 2 3; 4 5 6]; (A(:))'`  
Stacks  $\mathbf{A}$  by columns into a column vector and transposes the result to a row vector. In the present example, the command gives `ans=1 4 2 5 3 6`
- `reshape(A(:,), 2, 3)`  
Unstacks the column vector to a  $2 \times 3$  array which, in this case, is the original array  $\mathbf{A}$ .
- `X=[1 4 1 5 9 2 6 5]; C=X(2:8)-X(1:7)`  
Takes differences of successive values of  $\mathbf{X}$ . In the present example, the command gives `C=3 -3 4 4 -7 4 -1`
- `D=[1 2 3]; E=[4 5 6]; F=[D E]`  
This concatenates the vectors  $\mathbf{D}$  and  $\mathbf{E}$  (i.e., it appends  $\mathbf{E}$  after  $\mathbf{D}$  to get vector  $\mathbf{F}$ ). In the present example, the command gives  
`F=1 2 3 4 5 6`
- `I=find(A>2)` stores in  $\mathbf{I}$  locations (indices) of elements of vector  $\mathbf{A}$  exceeding 2.  
`find([3 1 4 1 5]<2)` gives `ans=2 4`

- `A(A>2)=0` sets to 0 all values of elements of vector  $\mathbf{A}$  exceeding 2. For example,  
`A=[3 1 4 1 5]; A(A<2)=0` gives `A=3 0 4 0 5`

M/M indexing of arrays starts with 1, while signal and image indexing starts with 0. For example, the DFT is defined using index  $n = 0, 1 \dots N-1$ , for  $k = 0, 1 \dots N-1$ . `fft(X)`, which computes the DFT of  $\mathbf{X}$ , performs

`fft(X)=X*exp(-j*2*pi*[0:N-1]'*[0:N-1]/N);`

## B-2.4 Solving Systems of Equations

To solve the linear system of equations

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix}$$

using

`A=[1 2; 3 4]; Y=[17; 39]; X=A\Y; X'`

gives `ans=5.000 6.000`, which is the solution  $[x \ y]'$ .

To solve the complex system of equations

$$\begin{bmatrix} 1+2j & 3+4j \\ 5+6j & 7+8j \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 16+32j \\ 48+64j \end{bmatrix}$$

`[1+2j 3+4j; 5+6j 7+8j]\*[16+32j; 48+64j]` gives

`ans= 2-2i  
6+2i ,`

which is the solution.

These systems can also be solved using `inv(A)*Y`, but this is a bad idea, since computing the matrix inverse of  $\mathbf{A}$  takes much more computation than just solving the system of equations. Computing a matrix inverse can lead to numerical difficulties for large matrices.

## B-3 Plotting

### B-3.1 Plotting Basics

To plot a function  $x(t)$  for  $a \leq t \leq b$ :

- Generate, say, 100 values of  $t$  in  $a \leq t \leq b$  using  
`T=linspace(a,b,100);`
- Generate and store 100 values of  $x(t)$  in  $\mathbf{X}$

- Plot each computed value of  $X$  against its corresponding value of  $T$  using `plot(T,X)`
- If you are making several different plots, put them all on one page using `subplot`.  
`subplot(324), plot(T,X)` divides a figure into a 3-by-2 array of plots, and puts the  $X$  vs.  $T$  plot into the 4th place in the array (the middle of the rightmost column).

One problem with MathScript that does not arise with MATLAB is that in MathScript `subplot(324)` opens 6 figures, even if only one or two of them will actually be used for plots. This is inelegant but harmless.

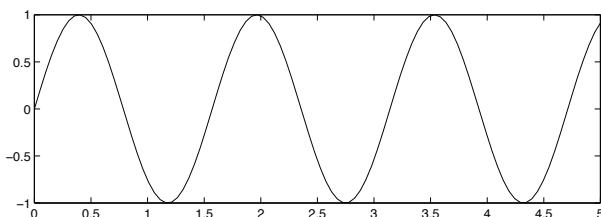
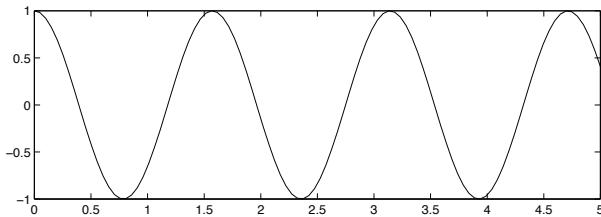
Print out the current figure (the one in the foreground; click on a figure to bring it to the foreground) by typing `print`

Print the current figure to a encapsulated postscript file `myname.eps` by typing `print -deps2 myname.eps`. Type `help print` for a list of printing options for your computer. For example, use `-depsc2` to save a figure in color.

To make separate plots of  $\cos(4t)$  and  $\sin(4t)$  for  $0 \leq t \leq 5$  in a single figure, use the following:

```
T=linspace(0,5,100);X=cos(4*T);Y=sin(4*T);
subplot(211),plot(T,X)
subplot(212),plot(T,Y)
```

These commands produce the following figure:



The default is that `plot(X,Y)` plots each of the 100 ordered pairs  $(X(I), Y(I))$  for  $I = 1, \dots, 100$ , and connects the points with straight lines. If there are only a few data points to be plotted, they should be plotted as individual ordered pairs, not connected by lines. This can be done using `plot(X,Y,'+')`.

## B-3.2 Plotting Problems

Common problems encountered using `plot`:

$T$  and  $X$  must have the same lengths; and neither  $T$  nor  $X$  should be complex; use `plot(T,real(X))` if necessary.

The above `linspace` command generates 100 equally spaced numbers between  $a$  and  $b$ , *including*  $a$  and  $b$ . This is *not* the same as sampling  $x(t)$  with a sampling interval of  $\frac{b-a}{100}$ . To see why:

- `linspace(0,1,10)` gives 10 numbers between 0 and 1 inclusive, spaced by 0.111;
- `[0:0.1:1]` gives 11 numbers spaced by 0.1.

Try the following yourself on M/M:

- `T=[0:10];X=3*cos(T);plot(T,X)`  
 This should be a very jagged-looking plot, since it is only sampled at 11 integers and the samples are connected by lines.
- `T=[0:0.1:10];X=3*cos(T);plot(T,X)`  
 This should be a much smoother plot since there are now 101 (not 11) samples.
- `T=[1:4000];X=cos(2*pi*440*T/8192);sound(X,8192)` This is musical note "A."  
`sound(X,Fs)` plays the vector  $X$  as sound, at a sampling rate of  $Fs$  samples/second.
- `plot(X)`. This should be a blue smear! It is about 200 cycles squished together.
- `plot(X(1:100))` This "zooms in" on the first 100 samples of  $X$  to see the sinusoid. It is also possible to zoom in by clicking on the figure.

## B-3.3 More Advanced Plotting

Plots should be labelled and annotated. Use:

- `title('Myplot')` adds the title "Myplot"
- `xlabel('t')` labels the x axis with "t"
- `ylabel('x')` labels the y axis with "x".
- `$\omega$` produces  $\omega$  in `title`, `xlabel` and `ylabel`. Similarly for other Greek letters. Note ', not ', should be used everywhere.

- `axis tight` contracts the plot borders to the limits of the plot itself.
- `axis([a b c d])` changes the horizontal axis limits to  $a \leq x \leq b$  and the vertical axis limits to  $c \leq y \leq d$ .
- `grid on` adds grid lines to the plot.
- `plot(T,X,'g',T,Y,'r')` plots on the same plot (`T, X, Y` must all have the same lengths) `X` vs. `T` in green and `Y` vs. `T` in red.

There is much, much more that can be done. Type `help plot` to see how to do it.

## B-4 Image Commands

### B-4.1 Reading Images into M/M

`X=imread('picture.jpg')` reads the  $(M \times N)$  JPEG image `picture.jpg` into the  $(M \times N)$  M/M array `X` if `picture.jpg` is a grayscale (black and white) image.

For color images, `X=imread('picture.jpg')` reads the  $(M \times N)$  JPEG image `picture.jpg` into the  $(M \times N \times 3)$  3-D M/M array `X` if `picture.jpg` is an RGB color image. Then `X(:,:,1)` is the red component of the image, `X(:,:,2)` is the green component, and `X(:,:,3)` is the blue component (see Chapter 10).

The following image formats can be handled using `imread`: JPEG, TIFF, BMP, PNG, HDF, PCX. The extension of the picture filename is used to determine the type of image.

### B-4.2 Displaying Images in M/M

An image will be displayed using the default axis setting. This usually alters the aspect ratio of the displayed image. This can be corrected by `axis image`, which will display the array as being  $(M \times N)$ .

`figure, imagesc(X), colormap(gray)` displays the  $(M \times N)$  array `X` as a grayscale (black-and-white) image. Adding `axis off` suppresses the numbers along the axes, which are in MATLAB format (see Chapter 3). `imagesc(X)` scales the array so that its values range from 0 to 1 for display purposes (see Chapter 5). `image(X)` omits this scaling, so in general it should not be used.

For color images, `figure, imagesc(X)` displays the  $(M \times N \times 3)$  3-D MATLAB array `X` as a color image, with colors determined as discussed above. Despite its name, `imagesc(X)` does **not** scale the array so that its values range

from 0 to 1 for color images; this must be done separately for each component:

```
X(:,:,1)=X(:,:,1)/max(max(X(:,:,1)));
X(:,:,2)=X(:,:,2)/max(max(X(:,:,2)));
X(:,:,3)=X(:,:,3)/max(max(X(:,:,3)));
```

prior to using `imagesc(X)`. This was done for all programs for Chapter 10.

## B-4.3 Image Computations in M/M

To process `X` using M/M, `X` must be converted from uint8 format to the double-precision, 64-bit, floating-point format used by M/M computation. This can be accomplished by inserting `X=double(X)` after reading in `X`.

`conv2(X,H)` computes the 2-D discrete-space convolution of `X` and `H`. If `X` is  $(M_1 \times M_2)$  and `H` is  $(L_1 \times L_2)$  then `Y` is  $(N_1 \times N_2)$ , where  $N_i = M_i + L_i - 1$  for  $i = 1, 2$  (see Section 3-6.2).

`conv2(X,H,'valid')` computes the 'valid' 2-D discrete-space convolution, defined in Section 7-12.3.

`fft2(X,M,N)` computes the  $(M \times N)$  2-D DFT of the image in array `X`. If the size of `X` is smaller than  $(M \times N)$ , it is zero-padded to size  $(M \times N)$ . If `M` and `N` are not specified (i.e., `fft2(X)`), then `M` and `N` are set to the size of the image. Section 3-9 discusses some issues in the computation and display of `fft2(X,M,N)`.

`real(ifft2(F,M,N))` computes the  $(M \times N)$  inverse 2-D DFT of `F`. Even if `F` is conjugate symmetric, roundoff error will incorrectly make `ifft2(F,M,N)` a complex array. So the real part of `ifft2(F,M,N)` should always be computed. Section 3-9 discusses some issues in the computation and display of `real(ifft2(X,M,N))`.

## B-5 Miscellaneous Commands

### B-5.1 Rectangular-to-Polar Complex Conversion

If an M/M result is a complex number, then it is presented in its rectangular form `a+bj`. M/M recognizes both `i` and `j` as  $\sqrt{-1}$ , so that complex numbers can be entered as `3+2j` or `3+2i`.

To convert a complex number `X` to polar form, use `abs(X), angle(X)` to get its magnitude and phase (in radians), respectively. To get its phase in degrees, use `angle(X)*180/pi`

Note `atan(imag(X)/real(X))` will **not** give the correct phase, since this formula is only valid if the real part is positive. `angle` corrects this.

The real and imaginary parts of  $\mathbf{X}$  are found using `real(X)` and `imag(X)`, respectively.

## B-5.2 Polynomial Zeros

To compute the zeros of a polynomial, enter its coefficients as a *row* vector  $\mathbf{P}$  and use `R=roots(P)`. For example, to find the zeros of  $3x^3 - 21x + 18$  (the roots of  $3x^3 - 21x + 18 = 0$ ) use `P=[3 0 -21 18]; R=roots(P); R'`, giving `ans=-3.0000 2.0000 1.0000`, which are the roots.

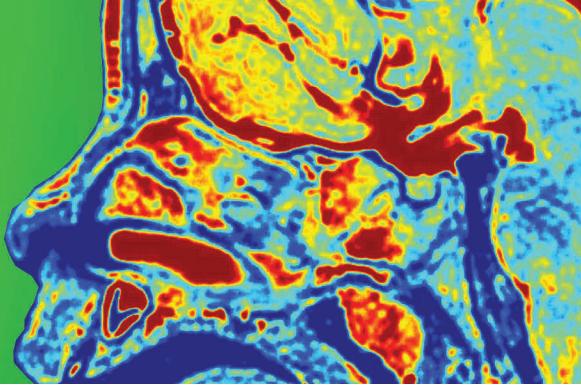
To find the monic (leading coefficient one) polynomial having a given set of numbers for zeros, enter the numbers as a *column* vector  $\mathbf{R}$  and use `P=poly(R)`. For example, to find the polynomial having  $\{1,3,5\}$  as its zeros, use `R=[1;3;5]; P=poly(R)`, giving `P=1 -9 23 -15`. The polynomial is therefore  $x^3 - 9x^2 + 23x - 15$ .

Note that polynomial are stored as row vectors, and roots are stored as column vectors.

## B-5.3 Discrete-Time Commands

- `stem(N, X)` produces a stem plot of  $\mathbf{X}$  vs.  $\mathbf{N}$
- `conv(X, Y)` convolves  $\mathbf{X}$  and  $\mathbf{Y}$ .
- `fft(X, N)` computes the  $N$ -point DFT of  $\mathbf{X}$ .
- `ifft(F)` computes the inverse DFT of  $\mathbf{F}$ . Due to roundoff error, use `real(ifft(F))`.
- `sinc(X)` compute  $\frac{\sin(\pi x)}{\pi x}$  for each element.

# INDEX



## Index

1-D Fourier transforms, 47–53  
1-D continuous-time signals, 41–43  
1-D continuous-time systems, 43–47  
1-D discrete-time signals and systems, 59–66  
1-D estimation examples, 300–303  
1-D fractals, 314–320  
2-D, 2  
2-D continuous-space Fourier transform, 94–107  
2-D continuous-space images, 91–93  
2-D convolution, 94  
2-D discrete Fourier transform, 119–121  
2-D discrete space, 113–118  
2-D discrete-space Fourier transform, 118–119  
2-D estimation, 309–313  
2-D fractals, 320–322  
2-D impulse, 91  
2-D sampling theorem, 107–113  
2-D slices, 19  
2-D spline interpolation, 149–150  
2-D wavelet transform, 228–232

*a posteriori*, 293  
*a posteriori* pdf, 293  
*a priori*, 293  
*a priori* information, 292  
*a priori* pdf, 293  
absorption coefficient, 18  
activation constant, 392  
activation function, 391  
active pixel sensors, 4  
additive, 336  
additive noise, 181, 184  
Airy disc, 7  
aliased, 54, 141  
aliasing, 73  
analysis filter bank, 216  
analysis filters, 214  
angular frequency, 21  
angular resolution, 9

antialias filtering, 141  
antialias lowpass filtering, 141–143  
application of MRF to image segmentation, 327–329  
APSSs, 4  
associative property, 45  
autocovariance functions, 286  
average, 213, 217, 221  
average image, 229  
axial resolution, 25

B-spline, 129  
B-splines interpolation, 143–149  
backpropagation, 397, 398  
bandlimited, 53, 107  
basis functions, 203, 206  
basis pursuit, 238  
basis pursuit denoising, 238  
batches, 402  
Bayes's rule, 293  
Bayesian estimators, 296  
bed of nails, 107  
Bessel function, 8, 103  
bilateral, 52  
bilateral power spectral density, 281  
bilinear, 149  
binary, 325  
binary image, 326  
binomial probability density function, 265  
black, 335, 337  
blackbody, 11  
blackbody radiation law, 11  
blurring, 181  
box image, 91  
brickwall lowpass filter, 184  
brightness, 5  
butterfly diagram, 77

Canny edge detector, 173  
CCDs, 4

- CDF, 168  
 center of mass, 295  
 central limit theorem, 144  
 centroids, 380  
 characteristic function, 275  
 charge-coupled device, 4, 335  
 circularly shift, 123  
 class index, 354  
 class-assignment algorithm, 323  
 classification, 322  
 classification by MAP, 358–360  
 classification by MLE, 357–358  
 classification of rotated images, 366–367  
 classification of spatially scaled images, 361–366  
 classification of spatially shifted images, 360–361  
 Classification Rule, 373  
 clique, 323  
 CMOS, 4  
 CMYK, 335  
 coin-flip experiment, 298–300  
 color, 335  
 color cube, 336  
 color image classification, 367–373  
 color science, 339  
 color systems, 335–340  
 color-image deblurring, 343–346  
 colorimetry, 339  
 commutative property, 45  
 comparison of 2-D interpolation methods, 150  
 complement, 255, 256  
 complex conjugate, 411  
 complex number, 411  
 compressed, 206  
 compressed sensing, 203, 236–238  
 compressed sensing examples, 242–249  
 compression, 203  
 computation of continuous-time Fourier transform using the DFT, 82–84  
 computation of the 2-D DFT using MATLAB, 121–124  
 computed tomography, 2, 18  
 computing solutions to underdetermined equations, 238–240  
 conditional expectation, 270  
 conditional pdf, 265  
 conditional pmf, 268  
 conditional probability, 259–261  
 confusion matrix, 358  
 conjugate symmetry, 50, 67, 95, 120  
 connecting the dots, 129  
 constant speed, 196  
 continuous random variables, 261  
 continuous space, 285, 361  
 continuous-space Fourier transform, 94  
 continuous-space systems, 93–94  
 continuous-time Fourier transform, 47  
 continuous-time random process, 278  
 continuous-time signals, 207  
 convolution integral, 45  
 convolved image, 166  
 cooled detectors, 12  
 coordinates, 376  
 correlation, 270, 354, 356  
 correlation coefficient, 270  
 cost function, 193  
 covariance, 270  
 covariance matrix, 273  
 cross-correlation, 360  
 cross-covariance matrix, 273  
 cross-validation, 404  
 CSFT, 94  
 CT, 2  
 CT scan, 18  
 CTFT, 47  
 cumulative distribution function, 168  
 cutoff index, 184  
 cyan, 335, 337  
 cyclic, 73  
 cyclic convolution, 209–213  
 Daubechies wavelet function, 203, 225  
 deblurring, 160, 181  
 decimating, 212  
 decimation, 141, 203  
 decomposed, 76  
 decomposition structure, 218  
 deconvolution, 160  
 deconvolution using the DFT, 80–82  
 deep learning, 393  
 degree of similarity, 354  
 denoising, 160, 181, 182  
 denoising by lowpass filtering, 183–188  
 denoising by thresholding and shrinking, 232–235  
 denoising color images, 346–347  
 density slicing, 9  
 derivation of backpropagation, 403–404  
 detail, 214, 217, 221  
 detail images, 229  
 detector resolution, 9

- deterministic, 181, 292  
deterministic deconvolution, 309  
deterministic estimate, 319  
deterministic versus stochastic Wiener filtering, 307–309  
DFT, 70, 119  
difference operator, 171  
diffraction pattern, 7  
dimensionality reduction, 377  
direct, 181  
direct and inverse problems, 181–183  
direct problem, 236  
discrete, 69  
discrete Fourier transform, 70–76, 119  
discrete random variables, 261  
discrete space, 113, 285  
discrete time, 59  
discrete-space Fourier transform, 118, 183  
discrete-space image, 113  
discrete-space spatial frequency, 184  
discrete-time eternal sinusoid, 61  
discrete-time Fourier transform, 66–70  
discrete-time frequency, 61  
discrete-time (Kronecker) impulse, 61  
discrete-time random process, 278, 279  
discrete-time rectangle function, 69  
discrete-time signal, 59  
discrete-time sinc function, 68  
discrete-time wavelet transforms, 218–223  
disjoint, 256  
disk-image, 92  
display dynamic range, 160  
displaying images, 90–91  
dissimilarity index, 325  
distribution function, 168  
distributive property, 45  
downsampled average signal, 214  
downsampled detail signal, 214  
downsampling, 140–141  
DSFT, 118, 183  
DSSF, 184  
DTFT, 66  
DTFT of the Smith-Barnwell condition, 221  
duration, 43, 61, 143  
dynamic range, 167
- edge, 171  
edge detection, 160, 171–176  
edge image, 341  
edge indicator, 173
- edge thinning, 173, 175  
edge-detection gradient, 173  
effects of shifts on pdfs and pmfs, 263–265  
electromagnetic, 3  
EM, 3  
emission, 11  
emissivity, 12  
energy, 43  
energy spectral density, 52, 67  
estimate, 234  
estimation, 255, 292  
estimation accuracy, 193  
estimation error, 303  
estimation methods, 292–298  
estimation problem, 292  
Euler's identity, 411  
even, 49, 77, 93  
even symmetry, 49  
even values of  $n$ , 77  
event, 255  
examples of image interpolation applications, 152–155  
excitation, 20  
expansion of signals in orthogonal basis functions, 206–209  
expectation, 269
- false-color, 335  
false-color image, 90  
fast Fourier transform, 70, 76–80, 119  
fast iterative shrinkage and thresholding algorithm, 242  
FFT, 70, 119  
field gradient, 20  
fill in the gaps, 129  
filtering, 213  
filtering of signals and images, 203  
finite-impulse response, 68  
FIR, 68  
first iteration, 381  
FISTA, 242  
focal underdetermined system solver, 240  
focusing, 23  
FOCUSS, 240  
four-color printing process, 338  
fractal, 290  
fractal-like, 314  
frequency exponent, 315  
frequency filtering, 50  
frequency response, 50, 51  
functions of random variables, 269–272  
fundamental frequency, 186

- fundamental period, 61
- gamma transformation, 162
- Gaussian, 266, 279
- Gaussian pdf, 293
- Gaussian random vector, 275
- Gaussian random vectors, 275–278
- geometric pmf, 259
- Gibbs distribution, 325
- global minimum, 397
- GPUs, 393
- gradient, 397
- gradient descent, 397, 402
- gradient fields, 21
- gradient threshold, 173
- graphical processing units, 393
- grayscale, 335, 339, 341
- grayscale image, 90
- gyromagnetic ratio, 21
- Haar, 203
- Haar transform, 226
- Haar wavelet transform, 213–218
- Hammersley-Clifford theorem, 325
- Hamming window, 68, 186, 188
- Hankel transform, 103
- hard thresholding, 242
- hexagonal sampling, 110
- hidden layers, 393
- histogram, 168
- histogram equalization, 160, 167–170
- histogram equalization and edge detection, 340–343
- histogram-equalized image, 167
- horizontal-direction vertical-edge detector, 172
- HSI, 339
- HSV, 339
- hue, 339
- hyperplane, 394
- ICM, 328
- ideal lowpass filter, 50
- iff, 412
- IID, 279, 300
- ill-posed problem, 237
- image, 107
- image array size, 131
- image classification by correlation, 354–356
- image deconvolution, 191–194
- image dynamic range, 160
- image enhancement, 160
- image physical size on computer screen, 131
- image pixel area, 131
- image plane, 5
- image recognition, 354
- image reconstruction fidelity, 107
- image recording, 196
- image restoration, 160, 181
- image sharpening filter, 166
- image shifting, 152
- image spatial resolution, 9
- image spectra, 92
- image texture, 323
- images, 92
- imaginary, 411
- implementation of upsampling using 2-D DFT in MATLAB, 137–140
- impulse, 41
- impulse response, 8, 44
- impulse train, 54
- in phase, 338
- incremental learning, 402
- independent and identically distributed, 279, 300
- independent events, 260
- independent random variable, 266, 271
- infrared, 9
- infrared catastrophe, 316
- input layer, 393
- interpolation, 129, 205
- interpolation using sinc functions, 129–130
- intersection, 255, 256
- interval, 61
- interval probability, 262, 263
- inverse, 181
- inverse problem, 236, 292
- IR, 9
- IRLS, 239
- Ising model, 325
- Ising-Gibbs distribution, 326
- ISTA, 241
- Iterated Conditional Modes, 328
- iterative reweighted least squares, 239
- iterative shrinkage and thresholding algorithm, 241
- jinc function, 106
- joint covariance matrix, 369
- joint pdf, 265, 357
- joint pdfs and pmfs, 265–269
- joint probability mass function, 266

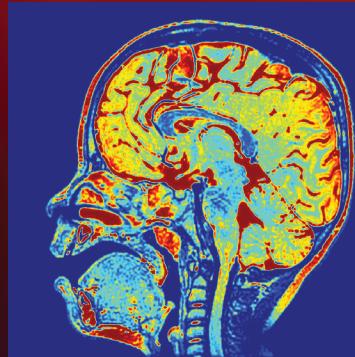
- jointly Gaussian random variables, 275  
K-Means clustering algorithm, 380–382  
K-stage decomposition, 221  
knots, 143  
  
L, 43, 44  
label, 399  
labeled, 390  
Lanczos interpolation formula, 130  
Landweber algorithm, 241–242  
Landweber iteration, 241  
Laplacian, 164  
Laplacian operator, 164  
Laplacian pdf, 293  
Laplacian’s spatial frequency response, 165  
Larmor frequency, 21  
LASSO cost functional, 234, 238  
lateral resolutions, 25  
LCD, 5, 91  
learning, 354, 390  
least absolute shrinkage and selection operator, 234, 238  
least-squares estimation, 303–307  
least-squares estimator, 293, 295  
lens law, 4  
letter recognition, 358  
likelihood, 255, 293  
likelihood function, 292, 370  
linear, 43, 44  
linear convolution, 73  
linear least-squares estimate, 304  
linear operator, 269  
linear programming, 238  
linear shift-invariant, 94, 117  
linear time-invariant, 41, 44  
linear transformation, 160  
liquid crystal display, 5, 91  
local gradient, 403  
local minimum, 397  
locality, 325  
localization, 21  
location-independent covariance matrix, 369  
log-likelihood function, 293, 302  
logarithmic, 162  
logarithmic warping, 362  
logarithmically transformed scale factors, 363  
long-wave IR, 11  
lowpass filter, 99  
LSE, 292, 293, 295  
  
LSI, 94, 117  
LTI, 41, 43, 44  
LTI filtering of random processes, 282–285  
LU decomposition, 239  
luminance, 338  
LWIR, 11  
  
MADs, 78  
magenta, 335, 337  
magnetic quantum number, 21  
magnetic resonance, 19  
magnetic resonance imaging, 2, 19–23, 236  
magnetized, 20  
MAP, 292, 293, 323  
marginal pdf, 357, 369  
marginal pdfs, 265, 357  
marginal pmfs, 268  
Markov random fields, 255, 292, 322–326  
“mask” image, 164  
maximum *a posteriori*, 293  
maximum likelihood estimate, 357  
maximum likelihood estimation, 293  
mean, 269  
mean square error, 295  
mean value, 269  
mean vector, 273  
mean-square error, 399  
measurement precision, 193  
median filtering, 194  
MEP, 359  
mesh plot, 90  
middle-wave IR, 11  
minimum error probability, 359  
minimum  $\ell_1$  norm, 237  
MLE, 292, 293, 323, 358  
MLE criterion, 370  
modified functions, 212  
modified MLE criterion, 371  
modulated signal, 78  
moment of inertia, 295  
morphing, 152  
motion blur, 195  
motion-blur deconvolution, 195–197  
MR, 19  
MRF, 292, 322  
MRI, 2, 19, 236  
MSE, 295  
multilayered perceptrons, 390  
multiplications and additions, 76, 78

- mutually exclusive, 260  
 MWIR, 11  
 N-D CSFT, 275  
 $N$ -dimensional continuous-space Fourier transform, 275  
 $N$ -dimensional spatial frequency vector, 275  
 $N$ -point, 71  
 near IR, 11  
 nearest-neighbor, 107, 145  
 neighborhood, 323  
 neural network training examples, 404–406  
 neural networks, 390–396  
 neurons, 390  
 NIR, 11  
 NMR, 19  
 NN, 107, 145  
 nodes, 393  
 noise, 181  
 noise variance, 182  
 noise-amplification problem, 193  
 noisy image, 184  
 non-WSS, 317  
 normal, 266  
 normal distribution, 301  
 normalized intensity, 161  
 notch filter, 188  
 notch filtering, 188–191  
 nuclear magnetic resonance, 19  
 Nyquist frequency, 54  
 Nyquist sampling rate, 54, 107  
 Nyquist-sampled version, 108  
 object plane, 5  
 observation, 292, 310  
 observed value, 292, 296, 301  
 octaves, 205, 218  
 odd, 49, 77, 93  
 odd symmetry, 49  
 odd values of  $n$ , 77  
 optical imagers, 3–13  
 original, 133  
 original sampled image, 133  
 orthogonal, 206, 373  
 orthogonal expansion, 206  
 orthogonality, 304  
 orthogonality property, 66, 71  
 orthonormal, 207  
 orthonormal expansion, 221  
 output layer, 393  
 overdetermined, 238  
 parallel-axis theorem, 295  
 Parseval’s theorem, 48, 67  
 partition function, 325  
 path attenuation, 19, 247  
 pdf, 182, 262  
 perceptrons, 390  
 perfect-reconstruction conditions, 219  
 perfectly red, 337  
 periodic, 69  
 periodogram spectral estimator, 314  
 perturbation, 208  
 perturbed, 208  
 phase shifting, 23  
 phase spectrum, 98  
 photometry, 339  
 piecewise- $M$ th-degree polynomial, 223  
 piecewise polynomial, 223  
 pixel, 113  
 pixel dimensions, 131  
 pixel value, 168  
 pixel-value transformation, 160–163, 168  
 pixels of the DFT image, 243  
 pmf, 262  
 point spread function, 8, 94, 173  
 point spread functions, 92  
 polar form, 411  
 potential energy, 325  
 power laws, 314  
 power spectral, 281  
 probabilistic, 181  
 probability, 255–259  
 probability density function, 182, 262  
 probability mass function, 262  
 probability tree, 257  
 projection-slice theorem, 249  
 pseudo inverses, 238  
 pseudo-inverse solution, 238  
 pseudocolor, 9  
 PSF, 8, 92, 94, 173, 181  
 pulse, 41  
 pulser, 23  
 QMF, 203, 219  
 QMF relation, 219  
 quadrature, 338  
 quadrature mirror filter, 203, 219

- radar imagers, 13–18  
radar shadow, 16  
radial brickwall lowpass filter, 106  
radial discrete-space frequency, 165  
radial frequency, 110  
radially bandlimited, 110  
radio frequency, 20  
Radon transform, 19, 247  
random, 255  
random fields, 255, 285–286  
random processes, 255, 278–282  
random variables, 255, 261–263  
random vectors, 272–275  
range, 13  
Rayleigh resolution criterion, 9  
Rayleigh’s theorem, 48  
real, 411  
realization, 278  
receive beamforming unit, 23  
reclining matrix, 373, 375  
reconstructed image, 107  
reconstruction structure, 218  
recorded, 181  
rectangle function, 41  
rectangular form, 411  
reflectance, 11  
regularity, 44  
regularization parameter, 193  
regularized, 193  
removing interference, 181  
resolution area, 131  
RF, 20  
RGB, 335  
ring impulse, 103  
rotated, 195, 366  
rotation cross-correlation, 366  
rotation matrix, 101  
rotationally invariant, 102  
RV, 273  
  
salt-and-pepper, 194  
sample function, 278  
sample mean, 300, 302  
sampled image, 107  
sampled signal, 54, 129  
sampled version, 107  
samples, 53  
sampling interval, 53, 129  
sampling length, 107  
  
sampling rate, 53, 107, 129  
sampling theorem, 53–59  
SAR, 14  
SAR PSF, 16  
saturation, 339  
scaling, 218, 223  
scaling constant, 314  
scene spatial resolution, 9, 14  
SD, 397, 403  
SDC, 306  
segment, 171, 322  
self-similarity, 314  
separable, 93, 228  
separating hyperplane, 394  
sharpened image, 164  
Shepp-Logan phantom, 230  
shift invariance, 94  
shorthand notation, 275  
shot noise, 194  
shrinkage, 203  
shrinking, 232, 233  
shrnking, 235  
SI, 94  
side information, 292, 293  
side-looking airborne radar, 14  
sifting, 41  
sifting property, 91  
signal flow graph, 77  
signal-to-noise, 184  
sinc, 50  
sinc function, 69  
sinc interpolation formula, 55, 107, 129  
singular value decomposition, 373  
singular values, 373  
sinusoidal interference, 188  
SLAR, 14  
slice, 18  
slices, 2  
Smith-Barnwell condition, 203  
Smith-Barnwell condition for perfect reconstruction, 220  
SNR, 184  
Sobel edge detector, 173, 341  
soft thresholding, 242  
SPARSA, 242  
sparse, 215  
sparse reconstruction by separable approximation, 242  
sparsification using wavelets of piecewise-polynomial signals, 223–228

- spatial frequency, 94  
 spatial frequency response, 95  
 spatial frequency response of the Laplacian operator, 164  
 spatial interval, 113  
 spatial resolution, 9  
 spatial sampling, 113  
 spatial-scaling factors, 362  
 spatially scaled, 361, 363  
 spatially shifted, 363  
 spatially warped images, 362  
 spectral estimation, 313–314  
 spectrum, 51  
 spin quantum number, 20  
 square wave, 186  
 squared  $\ell_2$  norm, 237  
 standard deviation, 270  
 static field, 20  
 statistical self-similarity, 314  
 steepest descent, 397, 403  
 steepest-descent, 397  
 steering, 23  
 steering the beam, 24  
 stochastic, 292  
 stochastic deconvolution, 306, 310  
 stochastic denoising, 305  
 stochastic denoising/deconvolution, 309  
 stochastic denoising filter, 305  
 stochastic gradient descent, 402  
 stochastic processes, 278  
 stochastic Wiener, 306  
 stochasticity, 292  
 sub-Nyquist-sampled version, 108  
 subband coding, 203  
 subband decomposition, 203  
 subtractive, 337  
 summary of image enhancement techniques, 176  
 superposition, 44  
 superposition integral, 45  
 supervised learning, 390  
 supervised training, 354  
 support, 43, 61, 143  
 SVD, 373  
 synthesis filters, 214  
 synthetic-aperture radar, 14  
 tall matrices, 373  
 terminals, 393  
 tetrahedral die, 258  
 thermal imagers, 11  
 thermal infrared imager, 9  
 three subtractive colors, 338  
 threshold, 376  
 threshold level, 232  
 thresholding, 203, 233, 235  
 thresholding and shrinking, 235  
 thumbnail image, 231  
 TI, 43, 44  
 Tikhonov regularization, 193  
 tiling, 110  
 time delaying, 23  
 time scaling, 41  
 time-invariant, 44  
 time-scaled, 43  
 Toeplitz blocks, 246  
 trade-off parameter, 234  
 training, 354, 390, 396  
 training images, 367, 373, 377  
 training matrix, 378  
 training neural networks, 396–403  
 transducers, 23  
 transform, 206  
 transmit beamforming unit, 23  
 transmit/receive switch, 23  
 transpose, 273  
 tree-structured filter banks, 203–206  
 true, 181  
 true resolution area, 131  
 true value of index  $k$ , 356  
 true-color, 335  
 truncated SVD, 377  
 TSFBs, 203  
 twiddle factors, 79  
 twiddle multiplications, 79  
 TWISTA, 242  
 two-dimensional, 2  
 two-step iterative shrinkage and thresholding algorithm, 242  
 ultrasound, 23  
 ultrasound imager, 23–27  
 ultraviolet catastrophe, 316  
 uncooled detectors, 12  
 uncorrelated, 270, 271, 273, 280  
 underdetermined, 236  
 underdetermined system, 237  
 unfiltered, 307  
 uniform, 272  
 uniform pdf, 299  
 union, 255

Universal Approximation Theorem, 393  
unknown, 292  
unsharp masking, 160, 163–167  
unsupervised learning, 373, 390  
unsupervised learning and classification, 373–377  
unsupervised learning examples, 377–380  
unsupervised training, 354  
upsampled version, 133  
upsampling and downsampling modalities, 130–132  
upsampling and interpolation, 133–137  
upsampling factor, 133  
  
valid convolution, 245  
van Cittert iteration, 241  
variance, 270  
VE, 171  
vertical edge, 171  
vertical-direction horizontal-edge detector, 172  
voltage outputs, 5  
Voronoi sets, 380  
voxels, 22  
  
warping, 152  
wavelength, 25  
wavelet, 218, 223  
wavelet transform, 203  
wavelet transform matrix, 237  
wavenumbers, 94  
weak-sense stationary, 280  
weighting coefficients, 76  
white, 282  
white random process, 282  
wide-sense stationary, 280  
Wiener filter, 193, 310  
Wiener process, 317  
windowed sinc functions, 130  
within-class, 367  
WSS, 280, 304  
  
X-ray computed tomography, 18–19  
  
yellow, 335, 337  
YIQ, 335  
  
zero-padded functions, 80  
zero-padding, 74  
zero-stuffing, 203, 205, 212

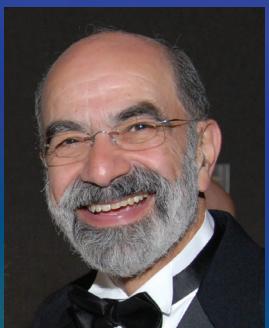


*Andrew E. Yagle   Fawwaz T. Ulaby*  
*University of Michigan*



Andrew E. Yagle is professor of Electrical Engineering and Computer Science at the University of Michigan. He is the recipient of several research and teaching awards including the NSF Presidential Young Investigator, ONR Young Investigator, College of Engineering Teaching Excellence Award, the Eta Kappa Nu Professor of the Year Award, and the Class of 1938E Distinguished Service Award.

He is a past member of the IEEE Signal Processing Society Board of Governors, the Image and Multidimensional Signal Processing Technical Committee, the Digital Signal Processing Technical Committee, and the Signal Processing Theory and Methods Technical Committee. He is a past associate editor of the *IEEE Transactions on Signal Processing*, *IEEE Signal Processing Letters*, *Multidimensional Systems and Signal Processing*, and the *IEEE Transactions on Image Processing*.



Fawwaz T. Ulaby is the Emmett Leith Distinguished Professor of Electrical Engineering and Computer Science and former Vice President for Research at the University of Michigan. He is a member of the National Academy of Engineering and recipient of the IEEE James H. Mulligan, Jr. Education Medal. His *Applied Electromagnetics* textbook is used at over 100 US universities.