

Project AC460

Image Processing
and
Pattern Recognition
with
Java

Jon Campbell and Fionn Murtagh

IVS, School of Computer Science
The Queen's University of Belfast

December 1998

Contents

1	Introduction	3
1.1	Motivation and Rationale	3
1.2	Applications	4
1.3	What is a Digital Image?	6
1.3.1	Other Examples of Digital Quantities	8
1.3.2	Raster Sampling or Scanning	8
1.3.3	Pixel	8
1.3.4	Spatial Resolution	8
1.3.5	Graylevel Resolution	8
1.4	Signal Processing	10
1.4.1	Colour and Spectral Bands	10
1.4.2	Sensing	10
1.5	General Concepts of Image Processing	11
1.6	Further Examples of Images	14
1.7	Exercises for Chapter 1	14
2	Digital Image Fundamentals	25
2.1	Visual Perception	25
2.2	An Image Model: a General Imaging System	26
2.2.1	Radiometric Measurement and Calibration	27
2.2.2	Motivation	27
2.2.3	Uneven Illumination	28
2.2.4	Uneven Sensor Response	28
2.3	Imaging Geometry	29
2.3.1	General	29
2.3.2	Geometric Distortion	29
2.3.3	Geometric Calibration	30
2.3.4	Object Frame versus Camera Frame	30
2.3.5	Lighting Angles	31
2.4	Sampling and Quantization	31
2.5	Colour	31
2.5.1	Electromagnetic Waves and the Electromagnetic Spectrum	31

2.5.2	The Visible Spectrum	32
2.5.3	Sensors	35
2.5.4	Spectral Selectivity and Colour	35
2.5.5	Spectral Responsivity	36
2.5.6	Colour Display	37
2.5.7	Additive Colour	37
2.5.8	Colour Reflectance	37
2.5.9	Exercises	38
2.6	Photographic Film	39
2.7	General Characteristics of Sensing Methods	39
2.7.1	Active versus Passive	39
2.7.2	Methods of Interaction	39
2.7.3	Contrast	39
2.7.4	Exercises	41
2.8	Worked Example on Calibration	42
2.9	CCD Calibration in Astronomy	46
2.9.1	CCD Detectors versus Photographs	46
2.9.2	CCD Detectors and Their Calibration	47
2.10	Questions on Chapters 1 and 2 – Fundamentals, Sensors, and Calibration	49
3	The Fourier Transform in Image and Signal Processing	53
3.1	Introduction	53
3.2	Digital Signal Processing	53
3.2.1	Introduction	53
3.2.2	Finite Sampled Signals.	54
3.2.3	Sampling Frequency	57
3.2.4	Amplitude Resolution	58
3.2.5	Frequencies	58
3.2.6	Phase	59
3.2.7	Periodic Signals	60
3.3	Fourier Series	60
3.3.1	General	60
3.3.2	Orthogonal Functions	62
3.3.3	Finite Fourier Series	63
3.3.4	Complex Fourier Series	63
3.4	The Fourier Transform	64
3.5	Discrete Fourier Transform	66
3.5.1	Definition	66
3.5.2	Discrete Fourier Spectrum	67
3.5.3	Interpretation	67
3.5.4	Frequency Discrimination by the DFT	67
3.5.5	Implementation of the DFT	75
3.6	Fast Fourier Transform	78

3.6.1	General	78
3.6.2	Software Implementation	79
3.7	Convolution	82
3.7.1	General	82
3.7.2	Impulse Response	86
3.7.3	Linear Systems	86
3.7.4	Some Interpretations of Convolution	87
3.7.5	Convolution of Continuous Signals	87
3.7.6	Two-Dimensional Convolution	87
3.7.7	Digital Filters	88
3.8	Fourier Transforms and Convolution	90
3.9	The Discrete Fourier Transform as a Matrix Transformation	93
3.10	Cross-Correlation	94
3.11	The Two-Dimensional Discrete Fourier Transform	95
3.12	The Two-Dimensional DFT as a Separable Transformation	96
3.13	Other Transforms	97
3.13.1	General	97
3.13.2	Discrete Cosine Transform	98
3.13.3	Walsh-Hadamard Transform	98
3.14	Applications of the Discrete Fourier Transform	99
3.14.1	Introduction	99
3.14.2	Frequency Analysis	100
3.14.3	Filtering	100
3.14.4	Fast Convolution	101
3.14.5	Fast Correlation	102
3.14.6	Data Compression	103
3.14.7	Deconvolution	103
3.15	Questions on Chapter 3 – the Fourier Transform	104
4	Image Enhancement	105
4.1	Introduction	105
4.2	Noise and Degradation	106
4.3	Point Operations	108
4.3.1	Grey Level Mapping by Lookup Table	108
4.3.2	Colour Lookup Tables	109
4.3.3	Greyscale Transformation	111
4.3.4	Thresholding and Slicing	112
4.3.5	Contrast Enhancement Based on Statistics	113
4.3.6	Histogram Modification	114
4.3.7	Local Enhancement	121
4.4	Noise Reduction by Averaging of Multiple Images	121
4.5	Spatial Operations	123
4.5.1	Neighbourhood Averaging	123
4.5.2	Lowpass Filtering	124

4.5.3	Median Filtering	127
4.5.4	Other Non-linear Smoothing	134
4.6	Image Sharpening – General	135
4.7	Gradient Based Edge Enhancement	135
4.7.1	Introduction	135
4.7.2	Gradient, Slope and Differentiation	135
4.7.3	Discrete Differentiation – Differences	136
4.7.4	Differentiation in 2-D – Partial Differentials	138
4.7.5	Windows for Differentiation	139
4.7.6	Other Gradient Windows	139
4.7.7	Gradient Magnitude and Direction	140
4.8	Laplacian	146
4.9	Edge Detection by Template Matching	147
4.10	Highpass Filtering	147
4.10.1	Marr-Hildreth Operators	147
4.11	Additional Exercises	148
4.12	Answers to Selected Questions	149
4.13	Examples of Image Enhancement Operations	156
4.14	Questions on Chapter 4 – Image Enhancement	164
5	Data and Image Compression	169
5.1	Introduction and Summary	169
5.2	Compression – Motivation	170
5.3	Context of Data Compression	172
5.4	Information Theory	173
5.4.1	Introduction to Information Theory	173
5.4.2	Entropy or Average Information per Symbol	175
5.4.3	Redundancy	176
5.4.4	Redundancy is Sometimes Useful!	177
5.5	Introduction to Image Compression	177
5.6	Run-Length Encoding	179
5.7	Quantization Coding	180
5.8	Source Coding	180
5.8.1	Variable Length Coding	180
5.8.2	Unique Decoding	182
5.8.3	Huffman Coding	183
5.8.4	Some Problems with Single Symbol Source Coding	187
5.8.5	Alternatives/Solutions	188
5.9	Transform Coding	191
5.9.1	General	191
5.9.2	Subimage Coding	192
5.9.3	Colour Image Coding	192
5.10	Image Model Coding	192
5.11	Differential and Predictive Coding	193

5.12 Dimensionality and Compression	194
5.13 Vector Quantization	195
5.14 The JPEG Still Picture Compression Standard	196
5.15 Error Criteria for Lossy Compression	196
5.16 Additional References on Image and Data Compression	197
5.17 Additional Exercises	197
5.18 Questions on Chapter 4 – Image Compression	199
6 From Image to Objects	205
6.1 Introduction	205
6.2 Introduction to Segmentation	205
6.2.1 Single Pixel Classification	207
6.2.2 Boundary-Based Methods	215
6.2.3 To Read Further on Image Segmentation	217
6.2.4 Exercises on Image Segmentation	217
6.3 Mathematical Morphology	218
6.3.1 Introduction to Mathematical Morphology	218
6.3.2 Scanned Operators	224
6.3.3 Grey-level Morphology	225
6.3.4 Composite Operations – Open and Close	226
6.3.5 Program Implementation	227
6.3.6 Examples of Morphological Operations	227
6.3.7 To Read Further on Mathematical Morphology	231
6.3.8 DataLab-J Demonstrations on Mathematical Morphology	232
6.3.9 Exercises on Mathematical Morphology	235
6.4 The Wavelet Transform	236
6.4.1 Introduction	236
6.4.2 The à trous Wavelet Transform	237
6.4.3 Examples of the À Trous Wavelet Transform	240
6.4.4 The Haar Wavelet Transform	244
6.4.5 Examples of the Haar Wavelet Transform	247
6.4.6 To Read Further on the Wavelet Transform	249
6.4.7 DataLab-J Demonstrations of the Wavelet Transform	249
6.4.8 Exercises on the Wavelet Transform	251
7 Pattern Recognition	253
7.1 Introduction	253
7.2 Features and Classifiers	254
7.2.1 Features and Feature Extraction	254
7.2.2 Classifier	255
7.2.3 Training and Supervised Classification	258
7.2.4 Statistical Classification	258
7.2.5 Feature Vector – Update	258

7.2.6	Distance	259
7.2.7	Other Classification Paradigms	259
7.2.8	Neural Networks	262
7.2.9	Summary on Features and Classifiers	262
7.3	A Simple but Practical Problem	263
7.3.1	Introduction	263
7.3.2	Naive Character Recognition	263
7.3.3	Invariance for Two-Dimensional Patterns	267
7.3.4	Feature Extraction – Another Update	267
7.4	Classification Rules	268
7.4.1	Similarity Measures Between Vectors	270
7.4.2	Nearest Mean Classifier	271
7.4.3	Nearest Neighbour Classifier	274
7.4.4	Condensed Nearest Neighbour Algorithm	274
7.4.5	k-Nearest Neighbour Classifier	275
7.4.6	Box Classifier	275
7.4.7	Hypersphere Classifier	276
7.4.8	Statistical Classifier	276
7.4.9	Bayes Classifier	279
7.5	Linear Transformations in Pattern Recognition and Estimation	279
7.5.1	Linear Partitions of Feature Space	280
7.5.2	Discriminants	282
7.5.3	Linear Discriminant as Projection	283
7.5.4	The Connection with Neural Networks	284
7.5.5	Fisher Linear Discriminant	285
7.5.6	Karhunen-Loëve Transform	286
7.5.7	Least-Square Error Linear Discriminant	288
7.5.8	Computational Considerations	290
7.5.9	Eigenimages	291
7.5.10	Other Connections and Discussion	293
7.6	Shape and Other Features	293
7.6.1	Two-dimensional Shape Recognition	293
7.6.2	Two-dimensional Invariant Moments for Planar Shape Recognition	294
7.6.3	Classification Based on Spectral Features	296
7.6.4	Some Common Problems in Pattern Recognition	297
7.6.5	Problems Solvable by Pattern Recognition Techniques	297
7.6.6	For Further Reading	298
7.7	Exercises	299
8	Neural Networks: From the Perceptron to the Multilayer Perceptron	309
8.1	Introduction	309
8.2	Historical Background	310

8.3	Neural Networks Basics	312
8.3.1	Introduction	312
8.3.2	Brain Cells	313
8.3.3	Artificial Neurons	314
8.3.4	Neural Networks and Knowledge Based Systems	316
8.3.5	Neurons for Recognising Patterns	318
8.3.6	Perceptrons	321
8.3.7	Neural Network Training	321
8.3.8	Limitations of Perceptrons	322
8.3.9	Neurons for Computing Functions	323
8.3.10	Complex Boundaries via Multiple Layer Nets	324
8.3.11	'Soft' Threshold Functions	325
8.3.12	Multilayer Feedforward Neural Network	326
8.3.13	Exercises	327
8.4	Implementation	334
8.4.1	Software	334
8.4.2	Hardware	334
8.4.3	Optical Implementations	335
8.5	Training Neural Networks	335
8.5.1	Introduction	335
8.5.2	Hebbian Learning Algorithm	335
8.5.3	The Perceptron Training Rule	335
8.5.4	Widrow-Hoff Rule	336
8.5.5	Statistical Training	337
8.5.6	Backpropogation	337
8.5.7	Simulated Annealing	338
8.5.8	Genetic Algorithms	338
8.6	Other Neural Networks	338
8.7	Conclusion	338
8.7.1	Exercises	339
8.8	Recommended Reading	343
8.9	References and Bibliography	344
8.10	Questions on Chapters 7, 8 and 9 – Segmentation, Pattern Recognition and Neural Networks	348
8.11	Recommended Texts and Indicative Reading	356
A	Appendix: Essential Mathematics	361
A.1	Introduction	361
A.2	Random Variables, Random Signals and Random Fields	361
A.2.1	Basic Probability and Random Variables	361
A.2.2	Random Processes	365
A.2.3	Further Background Reading	371
A.3	Linear Algebra	372
A.3.1	Basic Definitions	372

A.3.2	Linear Simultaneous Equations	374
A.3.3	Basic Matrix Operations	376
A.3.4	Particular Matrices	381
A.3.5	Complex Numbers	382
A.3.6	Further Matrix and Vector Operations	383
A.3.7	Vector Spaces	387

B Appendix: Image Analysis and Pattern Recognition in Java 391

Chapter 1

Introduction

1.1 Motivation and Rationale

Digital image processing and digital signal processing – this book covers both – are amongst the fastest growing computer technologies of the 1990s. With increasing computing power, it is increasingly possible to do numerically many tasks that were previously done using analogue techniques. More importantly, it is now feasible to perform processing on signals and images that were previously unthinkable. There are related advances in performance and cost of sensors. A room-full of computing equipment of yesteryear now fits in the palm of your hand. Likewise the capacity of communications channels and storage devices have grow dramatically. Digital television has arrived. We routinely transfer images over World-Wide Web.

It is now as valid to think of an image as data for processing just as a column of numbers in a bank balance, or strings of text in a database. Digital image processing is now state-of-the-art in many areas of high-technology: industrial inspection, monitoring of the earth and weather forecasting, document handling; and in areas of low(ish)-technology: personal computers and multimedia, digital cameras and electronic darkrooms.

Although some image processing requires special architectures (e.g. the real-time processing of video images – 25–30 per second), much practical work can be done on a general purpose computer, and even a modest PC. Thus, it is as valid to think of a picture as data for processing as a column of numbers in a bank balance, or strings of text in a database.

The principal objective of this unit is to lay a foundation for further study and research in this field.

On completion of this first chapter, you should be able to:

- understand the fundamentals of digital image representation and the elements of a digital image processing system.
- be familiar with a simple model of visual perception.

- describe and apply image enhancement techniques.
- understand the fundamental concepts of one-dimensional signal processing, and its link with image processing.
- understand the concepts and applications of image transforms.
- describe and apply image data compression techniques.
- appreciate the concept of data compression applied to text, signals, and images.
- describe and apply image segmentation techniques.
- understand the fundamentals of pattern recognition.
- be aware of current advances in, and applications of, image processing.

1.2 Applications

Throughout this book we will continuously refer to the applications of image processing technology. Some of the areas we shall touch are:

Machine Vision: How to get a machine to sense a scene and perform the perception, recognition, and knowledge acquisition tasks that are routine for human observers. Broadly speaking there are two important sub-areas of machine vision:

- 3-dimensional scene analysis, e.g. for automatic vehicle navigation. Difficult, except in very limited domains. Still a research area.
- automatic/automated inspection, e.g. quality control of computer printed circuit boards, or of pastry cases, metal parts, etc... The Signal and Image Processing Group at Magee is doing research into flaw detection in textiles. Here the world is essentially two dimensional. Now state of the art technology.

Character recognition: The grand-daddy of all image processing / pattern recognition tasks. How to convert ink marks on a page into text characters. Similar technology can be used for recognising any planar shape, e.g. for a robot to pick from a selection of parts on a conveyer belt. Printed (block) character recognition is more or less state-of-the-art; cursive, and handwritten script much more difficult - still a research area.

Medical: A small set of applications include:

- blood cell analysis; looking for abnormal shapes, or abnormal proportions of shapes.
- computer-aided tomography. Construction of 3D ‘image’ from a set of 2-d X-ray images of cross-sections.
- automatic screening of chest X-ray images.
- teleradiology and telemedicine, i.e. enabling a specialist to medical images and other measurements of a patient while they (doctor and patient) are separated by large distances.

Remote Sensing: Images of the earth, sensed from satellites and aircraft, can be processed to:

- assist in weather forecasting. Of course, fairly ‘raw’ unprocessed images are routinely used in weather processing, as can be noted in any TV weather forecast.
- automatically produce land-use maps,
- mineral exploration,
- evaluate the extent of global warming, – earth’s radiation budget studies.
- pollution monitoring.
- some of the earliest digital image processing work was done at the Jet Propulsion Laboratory, California, to ‘clean-up’ images sent from deep space probes (e.g. Venus).

Military: Some applications include:

- automatic guidance of heat seeking missiles; images are infrared,
- interpreting remotely sensed images from spy satellites and aircraft,
- determination if ‘friend’ aircraft, from ‘foe’ using, e.g., silhouette images,

Previously military applications were the primary instigators of most electronics related research and development, and this was especially so in image science. Not so important now that the cold-war is over.

Document Image Processing: Increasingly, business records (letters, balance sheets, etc...) are prepared on computer, and stored on them, i.e. as strings of digits, alphabetic characters, even computer-aided-design drawing codes. But, certain types of document originate outside the computer, e.g. cheques, delivery documents with receipt signatures, or are difficult to convert into symbols, so that the best that can be done is to make a digital picture record of them, and store that on a computer.

Entertainment and Consumer Items: We can mention:

- digital video,
- still images on computers,
- digital cameras.

Geographical Information Systems: The combination of many different sorts of spatial information in one data-base, e.g. Ordnance Survey map, satellite image (see above), census data, gas and electricity mains, geology maps etc...

As in general artificial intelligence, we shall find the irony that many of the image processing tasks that humans regard as routine ('child's-play') are difficult for machines. Fortunately, the reverse is true: some processing tasks that appear impossible to human eyes and brains, that require enormous attention to detail, that are boring and repetitive, or have to be done in hostile environments, can be automated quite simply.

1.3 What is a Digital Image?

Image: As used in most of these lectures, the term image or, strictly, monochrome image, refers to a two-dimensional brightness function $f(x, y)$, where x and y denote spatial coordinates, and the value of f at any point (x, y) gives the brightness (or, graylevel) at that point.

Monochrome versus colour: Mostly we shall deal with monochrome ('black-and-white') images – i.e. $f(., .)$ is a graylevel. In a colour image $f(., .)$ gives a colour. A colour image can be represented by three mono. images, each representing the intensity of a primary colour (eg. red, green, blue). Thus,

$$f_r(x, y), f_g(x, y), f_b(x, y).$$

More usefully, a colour image is represented by $f(b, x, y)$, where b denotes colour ($b = \text{band}$), where band = 0, 1, or 2, for red, green, blue.

Digital?

The monochrome image, $f(x, y)$, mentioned above is continuous (or analogue in some parlance), in *two* senses:

- $f(., .)$ is a real number, and,
- x and y are real numbers.

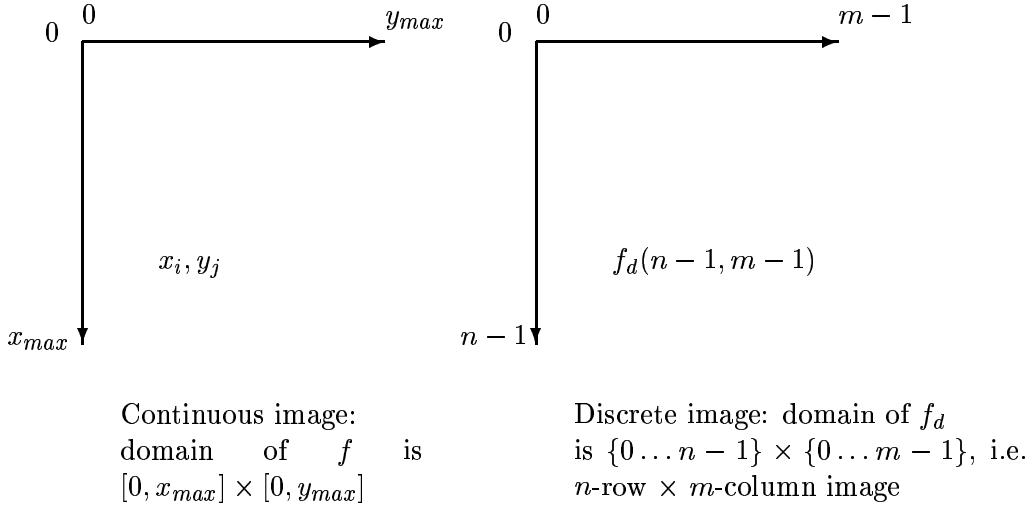


Figure 1.1: Correspondence of continuous and discrete axes. Normal usage of x - and y -axes. Conventionally $f(r, c)$, $r = \text{row}$, $c = \text{column}$.

Thus, you can achieve infinitesimally fine resolution in $f(., .)$, x , and y .

As always in computers we must use 'digital' or 'discrete' approximations. We approximate $f(., .)$ by restricting it to a discrete set of graylevels (often, in image processing systems, an 8-bit integer 0..255, and we sample $f(., .)$ at discrete points $x_i, i = 0 \dots n - 1$, and $y_j, j = 0 \dots m - 1$. See Figure 1.1.

Thus we arrive at a digital image: $f_d(r, c)$ where f_d can take on discrete values $\{0 \dots G - 1\}$ and r takes values $\{0 \dots n - 1\}$, and c takes values $\{0 \dots m - 1\}$.

The digital image can be viewed as a matrix (or two-dimensional array) of numbers:

$$f_d(i, j) = \begin{bmatrix} f_d(0, 0) & f_d(0, 1) & \dots & f_d(0, m - 1) \\ f_d(1, 0) & f_d(1, 1) & \dots & f_d(1, m - 1) \\ \vdots & \vdots & \ddots & \vdots \\ f_d(n - 1, 0) & f_d(n - 1, 1) & \dots & f_d(n - 1, m - 1) \end{bmatrix} \quad (1.1)$$

From now on we will drop the d , and use $f(r, c)$.

For ease of implementing certain algorithms, or due to the hardware configuration, digital images are often square ($m = n$). As well, (for storage or algorithmic convenience in digital computers) n is sometimes a power of 2: 16, 32, 64, 128...1024, etc.

Since the graylevel value will usually be stored in an integer computer word, G , also, will be a power of two; though, there is no reason not to use floating point or some other numerical representation.

1.3.1 Other Examples of Digital Quantities

Music on tape, or vinyl LP is continuous. Music on CD is digital. CD sampling rate is 44,100 samples per second, 12-bits per sample, 2 channels (stereo).

In modern telephone systems, speech is transferred digitally between major exchanges – here you can get away with 8,000 samples per second, and 8-bits per sample.

1.3.2 Raster Sampling or Scanning

The image model given above corresponds to the image model used in raster graphics, i.e. the image is formed by regular sampling of the x-, and y-axes.

1.3.3 Pixel

Each $f(r, c)$ in eqn. 1.1 is a picture element or pixel (the equivalent term ‘pel’ is used in some texts).

1.3.4 Spatial Resolution

Spatial resolution (or just resolution) is *high* if the samples x_i, y_j are closely spaced, and is low if they are widely spaced. Clearly, the closer the spacing, the more alike the digital image will be to the original, i.e. we are always demanding higher resolution. On the other hand, the higher the resolution, the larger are m, n – more data; data volume grows as the square of the resolution. Spatial resolution is illustrated in the image of Jon Campbell in Figure 1.2. The Java package, DataLab-J, accompanying this book has a script to zoom this image further, 16-fold, 32-fold, and so on. Try it!

1.3.5 Graylevel Resolution

With proper selection of the digitisation range, it is usually possible to represent, without any humanly perceivable degradation, monochrome images using just 8-bits; the psychologists tell us that humans can perceive no more than 160 levels at once. High-precision astronomers like to keep every bit

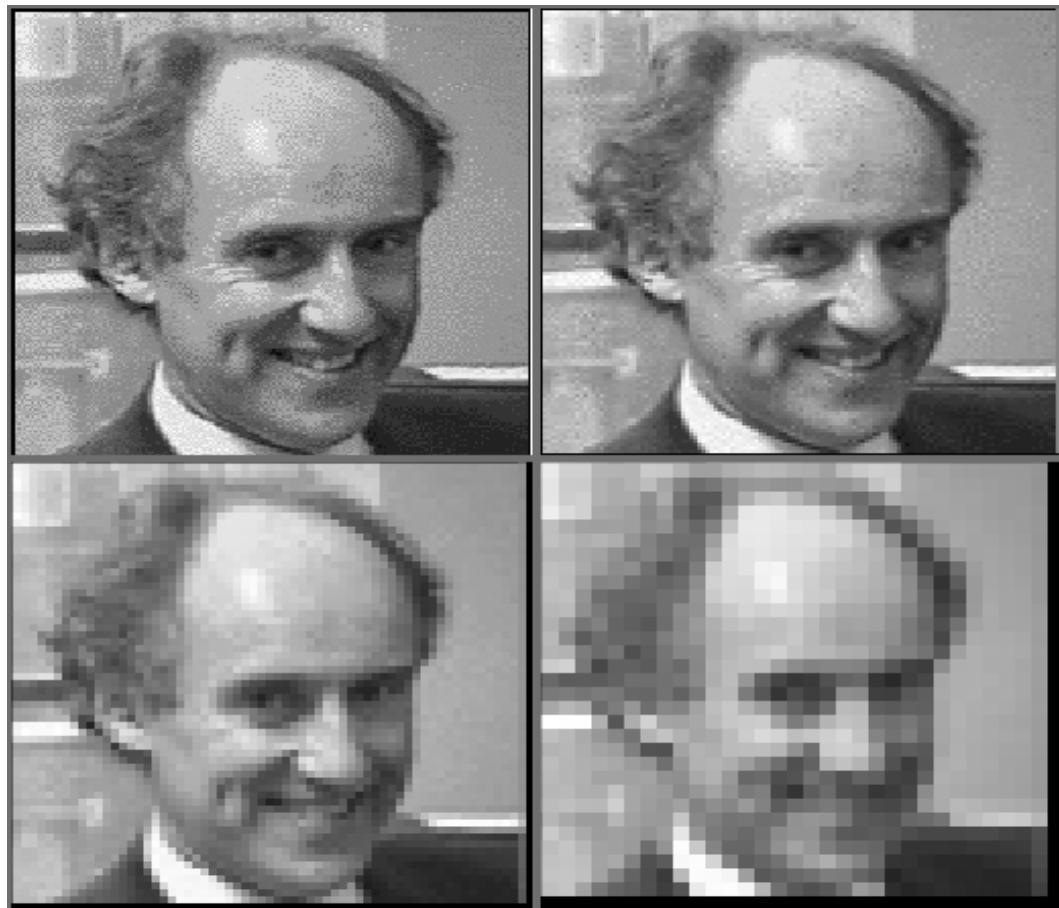


Figure 1.2: Upper left: original image. Upper right: zoomed by a factor 2.
Lower left and right: zoomed by 4 and by 8.

of information as they track wayward photons. The astronomical image storage format, FITS (Flexible Image Transport System) allows for 64-bit data. What is the largest pixel value which can be stored, then? What is the *dynamic range* of the data, i.e. the possible range of the data values? Ignore properties of the detector and take it that, in astronomy, the data must be non-negative.

1.4 Signal Processing

Signal processing – and digital signal processing - are closely related to image processing; whereas images are two-dimensional, signals are one-dimensional.

A music signal coming from a microphone, or going to a loudspeaker, is a continuous voltage signal – a continuous function of time: $f(t)$.

A **digital signal** is a sequence of numbers:

$$f_0, f_1, \dots, f_{n-1}$$

where f_0 may represent the (digitised) voltage at $t = t_0$, f_1 at t_1 , i.e. the function is sampled (and digitised) at t_0, t_1, t_2, \dots

We shall cover some aspects of signal processing because some processing techniques are more easily understood in one-dimension; and they are easily transferred to 2D.

1.4.1 Colour and Spectral Bands

As mentioned earlier, a colour image can be represented by three monochrome images or bands. But, image sensors are not just limited to visible light; some remotely sensed images are made up of 10 or more spectral bands.

1.4.2 Sensing

To produce $f(x, y)$ (forget about digitisation for a moment) from a scene, sensing must take place. That is, the brightness of each point (x, y) , in the field-of-view (FOV) of the observer, must be measured.

In a practical system this can be considered to be accomplished by passing a small aperture (opening) over the field of view, stopping when the aperture centre is over the discrete sample point (x_i, y_j) and taking the average brightness within the aperture. Usually, the width and height of the aperture are about the same as the horizontal and vertical sampling periods, respectively.

1.5 General Concepts of Image Processing

The general concept of a process involving image processing may be seen in Fig. 1.3. The flow of information is from right (the original scene) which reflects light into the sensor; the sensor converts light into a voltage; the voltage (a continuous quantity) is sampled and digitised to yield a number; some (numerical/digital) image processing is done; the numbers must be converted back into a voltage and transferred to a display which produces patterns of light that the user can see.

Quantities used	
Scene	Light
Sensor	Voltage
Digitize	Numbers
Image processing	Numbers/volts
Display (*)	Light
User's eyes	

* Alternatives: printer, file store, transmission.

Figure 1.3: Overview of imaging and image processing.

Typical image processing operations are:

- smooth out the graininess, speckle or noise in an image,
- remove the blur from an image,
- improve the contrast,
- segment the image into regions, e.g. on a printed circuit board, plastic region, copper region.
- remove warps or distortions,
- code the image in some efficient way for storage or transmission.

In later Chapters, we will look fairly often at actual pixel values. High values may appear as white, and very low values as black. A lookup table establishes this correspondence. We may similarly color grayscale images which is often a useful thing to do, in order to show up faint parts of the image.

Our images may well be noisy, reminiscent of a poor television or audio signal. One way to handle such noisy images is to smooth them. This we do by ‘passing a window’ over the image, changing the value of the pixel covered by this window. Each pixel in the output image, for example, becomes the average of the original pixel together with its eight neighbours. The smoothing may well be somewhat successful, but significant blurring of some edges and corners can also happen.

Contrast is very considerably enhanced by defining edges, or regions of sharp contrast in the image. Notionally we can turn an image into a line drawing version in this way. In practice, this is not so easy!

Appendix 1 describes the overall structure of DataLab-J which accompanies this book. Here we provide an example.

A short example of Java code for image negation follows. The program scripts `neg1.dlj` and `neg2.dlj` provide examples of such image negating, which is turning an image into its negative, or vice versa.

```
/**
 *
 *<p>
 *This method negates an image
 *
 *@param x input Im
 *
 */

public static Im negate(Im x){
    int nr= x.nrows();
    int nc= x.ncols();
    float mm = x.max() + x.min();
    Im z= new Im(nr,nc);
    for(int r= 0;r< nr;r++){
        for(int c= 0;c< nc;c++){
            z.put(mm - x.get(r,c),r,c);
        }
    }
    return z;
}
```

An input image, `rct1010.pgm` in the image collection supplied on disk, is as follows.

```
P2
10 10
3

0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 3 3 3 3 3 0 0 0
0 0 3 3 3 3 3 0 0 0
0 0 3 3 3 3 3 0 0 0
0 0 3 3 3 3 3 1 0 0
0 0 3 3 3 3 3 0 0 0
0 0 0 1 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0
```

This is in ASCII PGM (portable gray map) format. The output image produced is `r2neg.pgm`:

```
P2
10 10
3

3 3 3 3 3 3 3 3 3 3
3 3 2 2 3 2 3 3 3 3
3 3 3 3 3 3 3 3 3 3
3 3 0 0 0 0 0 3 3 3
3 3 0 0 0 0 0 3 3 3
3 3 0 0 0 0 0 3 3 3
3 3 0 0 0 0 0 2 3 3
3 3 0 0 0 0 0 3 3 3
3 3 3 2 3 3 2 3 3 3
3 3 3 3 3 3 3 3 3 3
```

The following list gives an indication of the range of functions available in DataLab-J.

- Arithmetic: typically, add two images to produce a third.
- Display: all graphic and text displays, including to printer and file.
- Miscellaneous Data Handling: includes creation and deletion of images, copying.

- Data Generation: generation of test data and patterns, random and deterministic.
- File Handling: file input-output, both formatted (ASCII) and unformatted.
- One-dimensional Digital Filters: simple recursive digital filters.
- Correlation and Convolution: 1- and 2-dimensional correlation using ‘fast’ FFT methods.
- Image Enhancement: smoothing, edge detection, median filtering etc.
- Fourier and other Transforms: one-dimensional DFT operations (using FFT), two-dimensional DFT, Wavelet transform, Walsh transform, Hough transform.
- Pattern Classification: numeric pattern classification, especially statistical pattern recognition; includes multilayer perceptron (backpropagation trained) neural network, and fuzzy rule-based classifier.
- Estimation: estimation, eg. multivariate linear regression; includes multilayer perceptron (backpropagation trained) neural network estimation, and fuzzy rule-based estimation.
- Feature Extraction and Discriminant Analysis: linear transformations for feature extraction and discrimination.
- Two-dimensional Texture Analysis: e.g. various measures obtained from co-occurrence matrix.
- Two-dimensional Shape Recognition: two-dimensional moments.
- Image Morphology: both binary and graylevel.
- Matrix and Vector Arithmetic: provides a basic calculator for matrix and vector arithmetic.

1.6 Further Examples of Images

Here are just a few interesting images, the analysis of which could require many dozen pages, and maybe even a book-length text.

1.7 Exercises for Chapter 1

1. For a monochrome image, $G = 255$, $n = 1024$, and $m = 1024$, how many bytes will the image occupy?

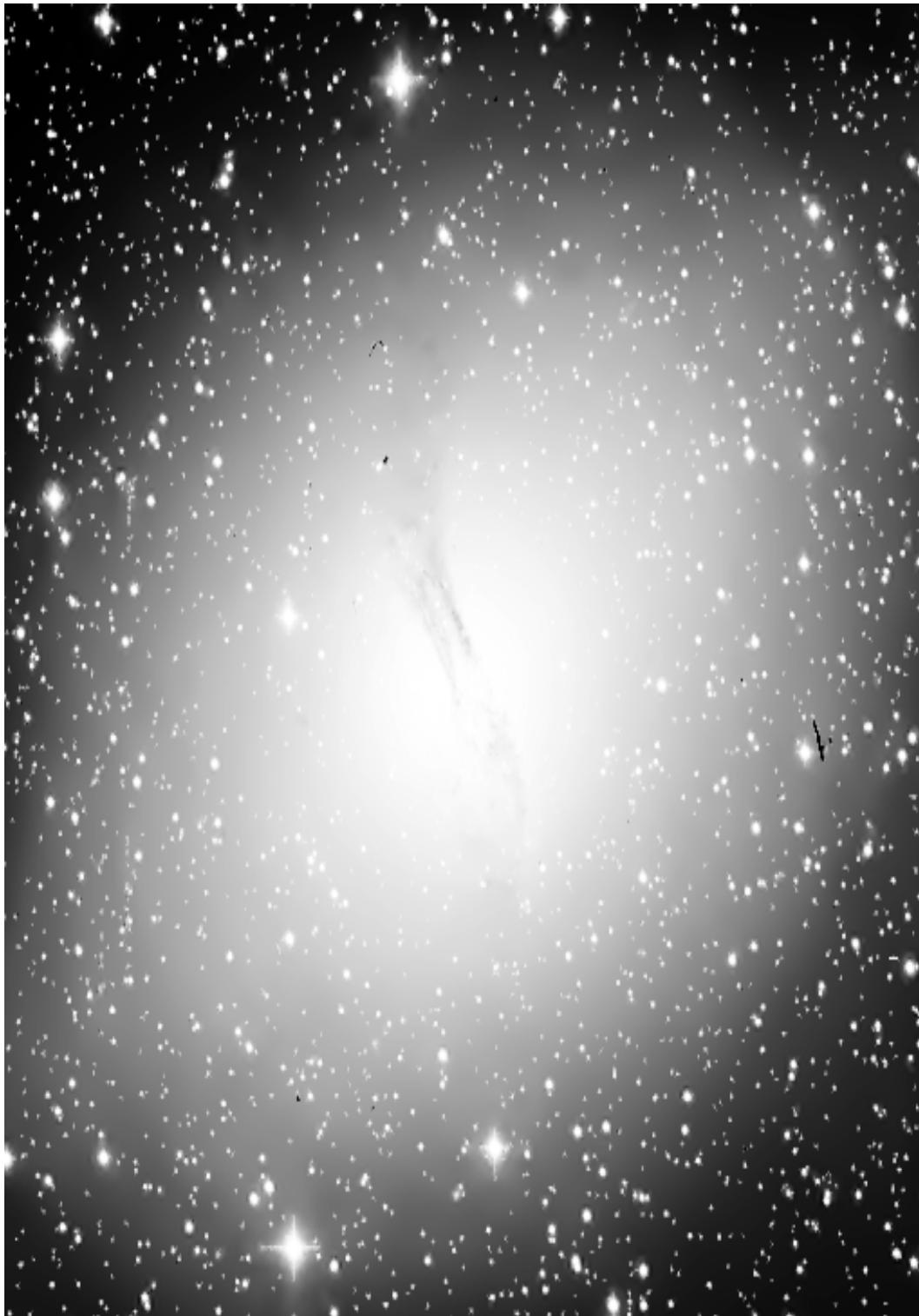


Figure 1.4: An image of a galaxy (NGC 5128, Cen A, from ESO Southern Sky Survey) following compression and uncompression.

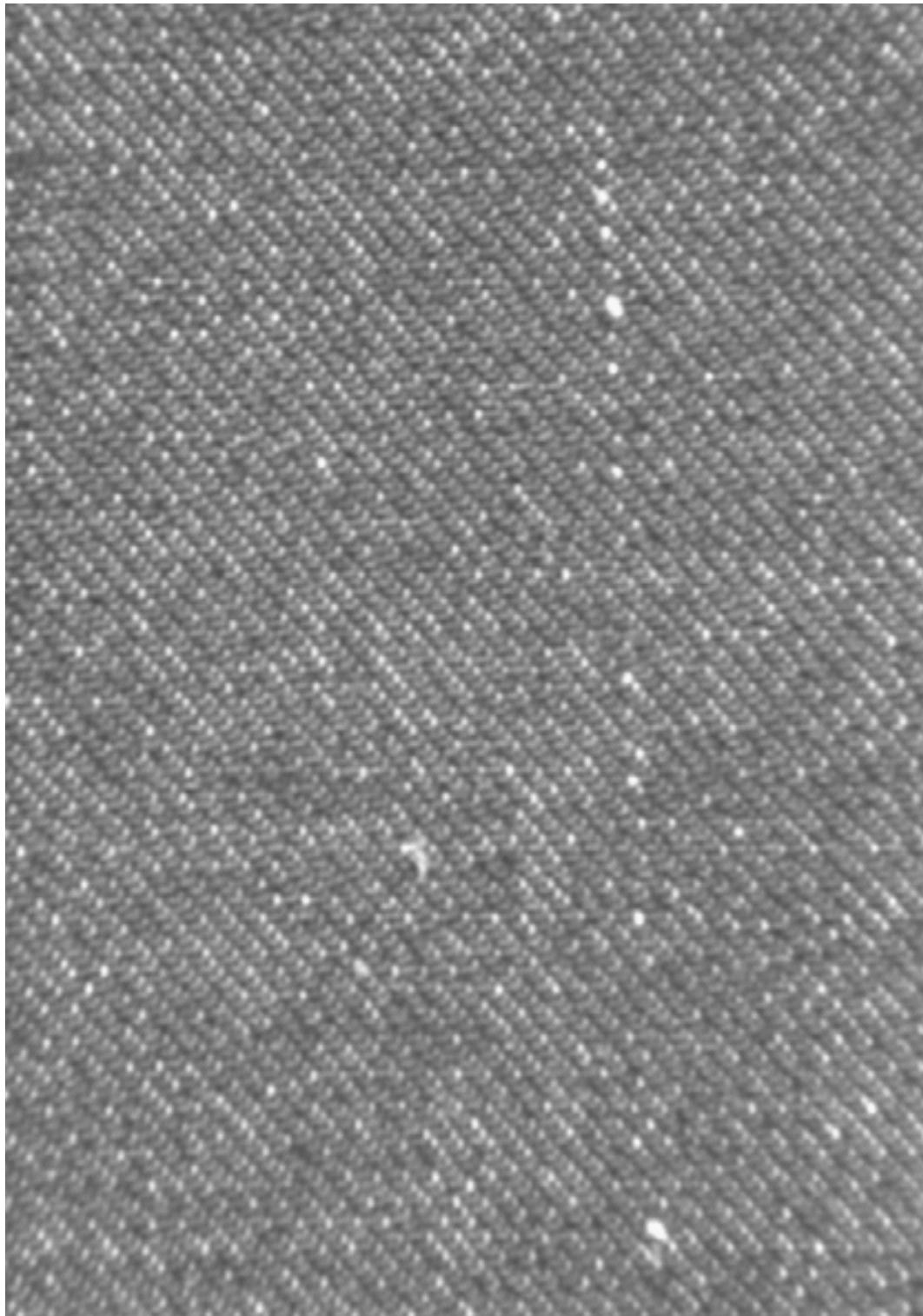


Figure 1.5: An image of a piece of textile – jeans – showing a production fault.

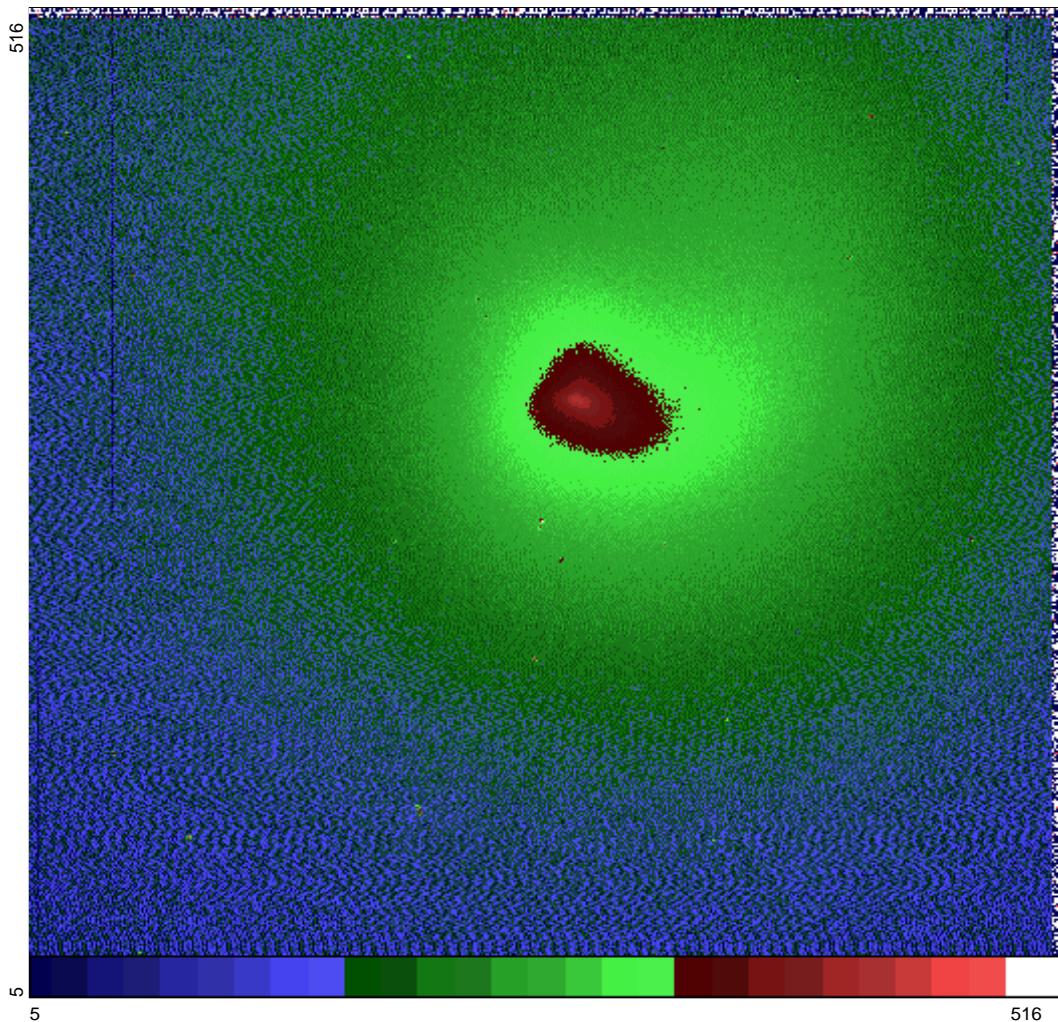


Figure 1.6: A comet nucleus.

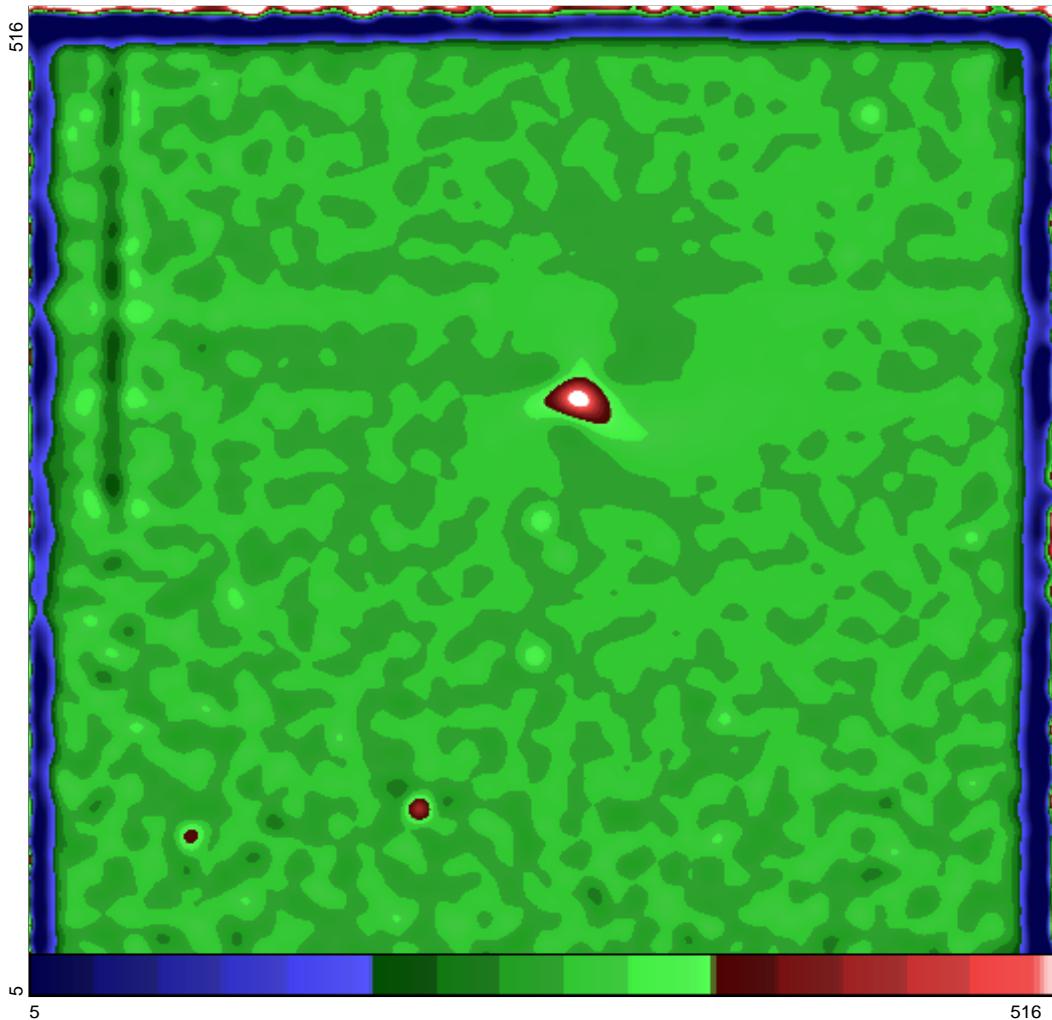


Figure 1.7: A plane from a wavelet transform of the previous image. Note the faint structures (outgassing) emanating from the nucleus.

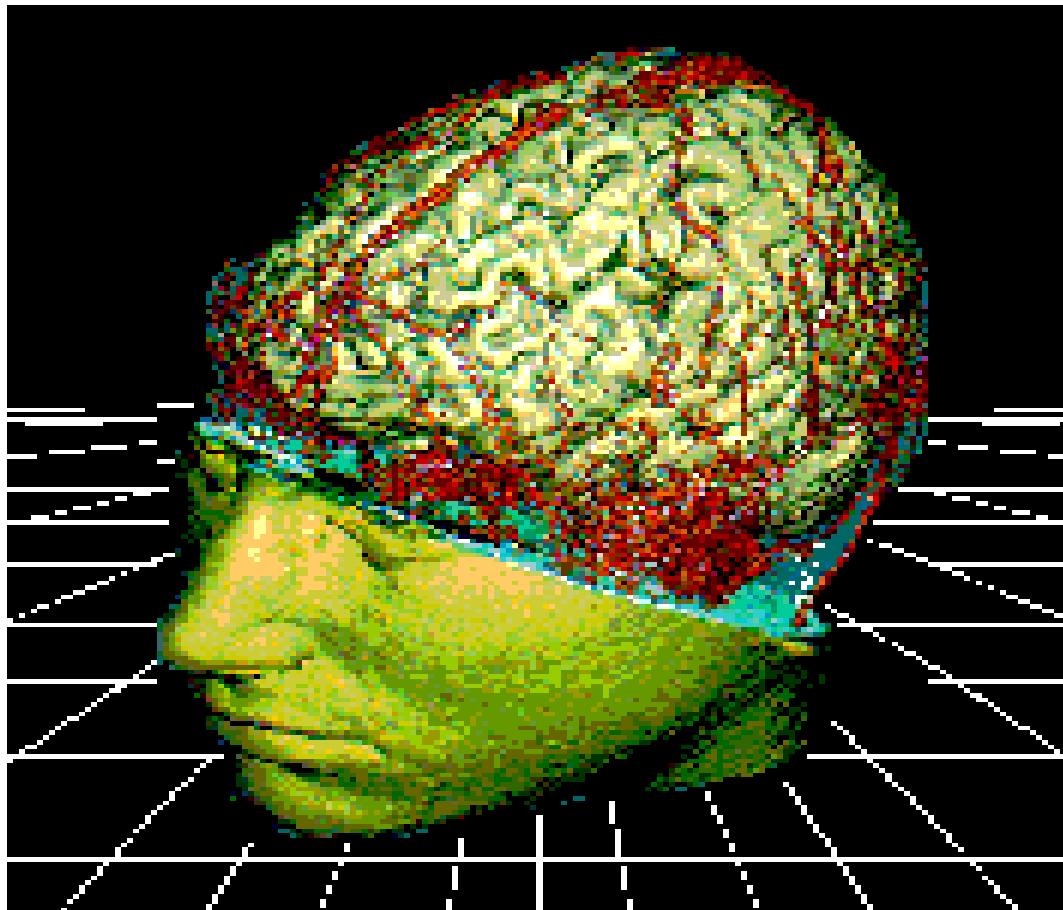


Figure 1.8: A reconstructed MRI (magnetic resonance) image of the brain.

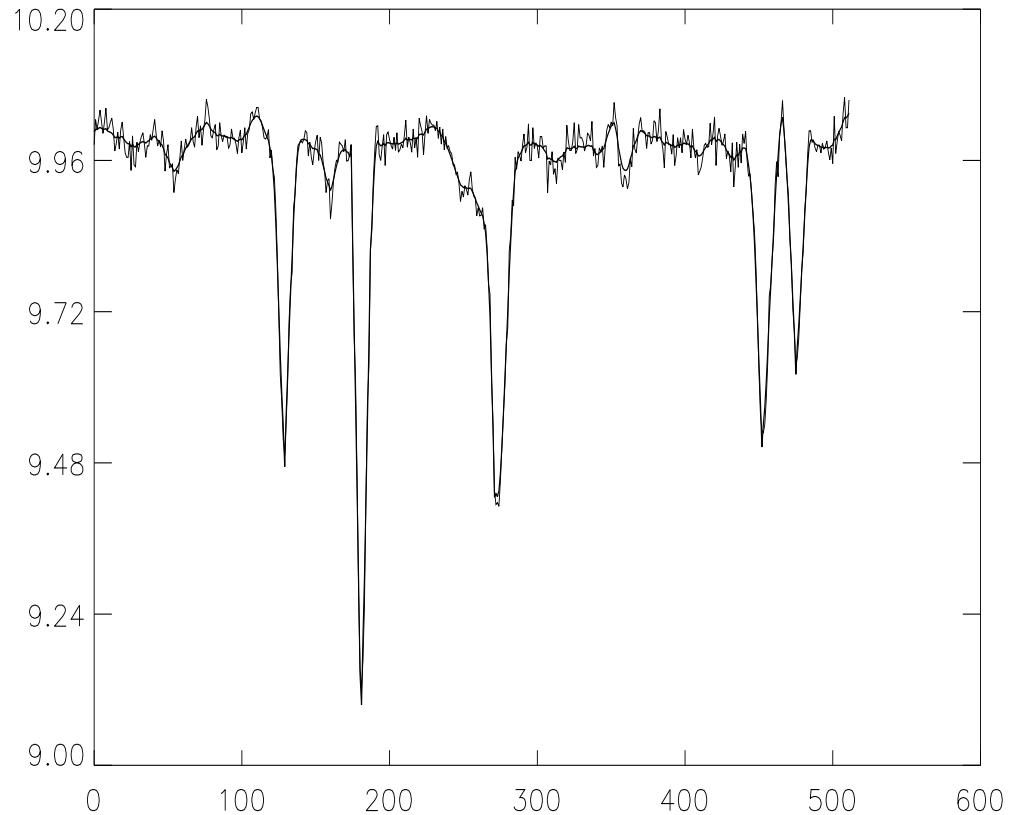


Figure 1.9: An astronomical spectrum.

2. Repeat Exercise 1 for a color image.
3. If you had to digitize a TV image, suggest suitable values of m , n .
4. Using the results of Exercise 3, and assuming 25 frames (images) per second, how much data for a one-hour film?
5. Laser printers commonly work at 600 dots per inch (dpi). How many pixels are there in an A4 page (assuming that an individual dot can be represented in a single pixel)? How many bits per pixel for a laser printer image? Suggest some more appropriate conversion factor between a laser printer and a raster-scanned image.
6. (a) From your conclusions from the previous question, how much data is there in a laser printer image of an A4 page?
(b) Compare with the data volume in an A4 page stored as ASCII text.
7. The ground resolution of weather satellites is about 1 to 5 kilometers. The ground resolution for some spy satellites is reported to be 15 centimeters.
 - (a) A satellite image has a ground resolution of 15 cm. What does a ground resolution of 15 cm mean?
 - (b) Using such an image, could a skilled interpreter
(i) read the number plate of a car?
(ii) determine the type of a car?
(iii) count the number of cars in a car park?
(iv) evaluate how many people at a football match?
(v) tell where the football was at the time the image was taken?
8. Assuming monochrome, ground resolution of 15 km and 8 bits per pixel, how much data in an image of your city or region? (You will have to estimate the dimensions you choose for your region.)
9. You are designing an image processing system to do quality control of lace fabric. The structure of the lace is such that thread separation is 0.5 mm. Suggest a suitable sampling resolution (in millimeters). The fabric is produced in 1.8 metres wide rolls, and it passes the inspection point at 1 metre per second). How many pixels per second? What does this suggest about the type of processor required?
10. Take some measurements and suggest a resolution for flaw detection in denim (jean) fabric.

11. Assuming 12-bits per sample, 44.1K samples per second (44.1 kilo-Hertz, KHz), 2 channels, how much data on a 1 hour CD? Compare this to the result obtained in Exercise 4 (one hour of video).
12. What are typical local area network speeds in the networked work-place? Would it be possible to send or receive digitised speech or CD music over such links?
13. What are typical modem speeds for dial-up access to your work-place, or to your local Internet provider, or to a local access point of an online system? Would it be possible to send speech or CD music over it? If you cannot send digital speech over that line, can you see any paradox?
14. If you had an image of a photographic negative, suggest a digital method of producing a ‘positive’ of this.
15. If you had an image of a face – in colour, i.e. three monochrome images f_r, f_g, f_b for red green and blue. You find that the flesh tones correspond to:

$$f_r = 100 \pm 10, f_g = 70 \pm 6, f_b = 50 \pm 10$$

Sketch an algorithm that will convert the flesh colour to pure bright white (255,255,255), while leaving other tones untouched.

16. You have a satellite image in blue, green, red, and infrared (4 mono. images this time, f_r, f_g, f_b , and f_{ir}). Let m_{12} be the mean colour for class 1 in band 1, etc. Let s_{14} be the standard deviation of the colours for class 1 in band 4, etc. You find that water (landuse type 1) is usually

$$(m_{11} \pm s_{11}, m_{12} \pm s_{12}, m_{13} \pm s_{13}, m_{14} \pm s_{14})$$

and farmland (type 2)

$$(m_{21} \pm s_{11}, m_{22} \pm s_{12}, m_{23} \pm s_{13}, m_{24} \pm s_{14})$$

Sketch an algorithm that will produce a label image (i.e. landuse map), f_l , containing 1 where there is water, 2 where there is farmland, and 0 otherwise.

17. Suggest a window (or algorithm) for detecting bright spots one pixel wide.

18. Take the example DataLab-J code used in this Chapter for image negation and do the following.
- (a) Add 2 to each pixel of image.
 - (a) Add two images `f1` and `f2` to produce a third `f3`. What precautions should you take?
 - (c) Apply a 3×3 smoothing window.
 - (d) Implement the algorithm that is the answer to the previous questions, assuming three input images `fr1`, `fg1`, `fb1` and three output images `fr2`, `fg2`, `fb2`.
 - (e) Re-implement your algorithm assuming your parameters, `w` and `s`, are stored in two-dimensional arrays, and similarly your multispectral image, `f`, is stored in a three-dimensional array.

Samples of most of this question are available in the script directory (`dljbook`) on the disk.

Chapter 2

Digital Image Fundamentals

2.1 Visual Perception

Here, briefly, are some points about human visual perception:

- the perceived image may differ from the actual light image (i.e. the perceived *brightness* image is a considerably modified ‘copy’ of the physical light intensity emanating from the scene),
- there are two types of light sensors on the retina – rods and cones,
- rods are more sensitive than cones; rods are used for night (scotopic) vision; rods are largely colour insensitive (e.g. no colour evident in moonlight),
- cones are used for brighter light, cones can sense colour,
- perceived (subjective) brightness (B_s) is roughly a logarithmic function of light intensity (L): thus, if you increase L by 10, B_s increases by only 1 unit, increase L by 100 B_s increases by 2 units, 1000 increases by 3 etc.
- the visual system can handle a range of about 10^{10} (10 thousand million) in light intensity (from the threshold of scotopic vision to the glare limit). (Question: how many bits is that?)
- to handle this range, the pupil must adapt by opening and closing the pupil; opening the pupil – in darkness – lets more light in; closing it – in bright light – lets less light in,
- the eye can handle only a range of about 160 at any one instant, i.e. where there is no opening and closing of the pupil; of course, this explains why 8-bits (256 levels) usually suffice in a display memory,

2.2 An Image Model: a General Imaging System

Note: in this chapter we treat physical units somewhat informally – we do not deal rigorously with the physics of light radiation.

A general camera-based sensing arrangement is shown in Figure 2.1: the object, some distance from the camera lens, is projected onto the image plane. At the image plane there is a mosaic of light sensitive sensors; these have the effect of transforming the two-dimensional continuous image lightness function, $f_i(y, x)$, into a discrete function, $f'[r, c]$, where r (ow) and c (olumn) are the discrete spatial coordinates; as in Chapter 1, we use square brackets, [,], to indicate a discrete domain; eventually, $f'[\cdot]$ gets digitised to yield a digital image, $f_d[r, c]$, (where digital often connotes discrete space, in addition to integer valued; since all the images under discussion will be digital, we drop the ‘ d ’ subscript in normal usage).

Figure 2.1 Image Capture Schematic

In most image sensors, the mosaic of light cells completely cover the image plane, and the light cell corresponding to $f[r, c]$ has a finite area, say $A = (yr - sy)/2 \leq y \leq (yr + sy)/2, (xr - sx)/2 \leq x \leq (xr + sx)/2$, and so the sensing process involves integration (averaging) as well as spatial sampling:

$$f_d[r, c] = \int_A f_i(y, x) dy dx \quad (2.1)$$

Thus, we arrive at a digital image: $f_d[r, c]$ where f_d can take on discrete values $[0, 1, \dots, G - 1]$ and $r \in [0, 1..n - 1]$, $c \in [0, 1..m - 1]$.

From now on we drop the ‘ d ’ i.e. $f_d[,]$ is written $f[,]$.

Thus

$$f : [0, N - 1] \times [0, m - 1] \rightarrow [0, G - 1]$$

The *domain* is the set of pixels, the *range* is the set of pixel values, and f maps the domain onto the range. Question: if Z^+ are the positive integers and R are the reals, to what extent is it true that $f : Z^+ \times Z^+ \rightarrow R$?

This can be viewed as a matrix (two-dimensional array) of numbers:

$$f[r, c] = \begin{bmatrix} f[0, 0] & f[0, 1] & \dots & f[0, m-1] \\ f[1, 0] & f[1, 1] & \dots & f[1, m-1] \\ \vdots & \vdots & \ddots & \vdots \\ f[n-1, 0] & f[n-1, 1] & \dots & f[n-1, m-1] \end{bmatrix} \quad (2.2)$$

In many image processing applications, $f(., .)$ is represented by an 8-bit byte ($f \rightarrow [0 \dots 255]$); the range $[0 \dots 255]$ derives not only from storage convenience, but from the facts that:

- human eyes can, simultaneously, perceive only about 160 light levels (see section 2.1), and,
- most optical sensors are troubled to exceed a signal-to-noise ratio of 48 decibels [$48 \text{ dB} = 20 \log(1/256)$].

Mostly we will be dealing with monochrome images – i.e. $f[r, c]$ represents a grey level. In a colour image $f(., .)$ must give a colour. From the point of view of image processing, a colour image can be represented by three monochrome images, each representing the intensity of a primary colour (eg. red, green, blue). Thus, $f_r[r, c]$, $f_g[r, c]$, and $f_b[r, c]$, for red, green and blue. Of course, we can generalise to any number of ‘wavebands’/‘colours’, in or out of the visible spectrum. A generalised ‘colour’ image is represented by $f[b, x, y]$, where b denotes colour ($b = \text{band}$), where, normally, band = 0, 1, and 2, for red, green, and blue.

2.2.1 Radiometric Measurement and Calibration

2.2.2 Motivation

In Chapter 1 we defined an image thus: “... monochrome image, refers to a two-dimensional brightness function $f(x, y)$, where x and y denote spatial coordinates, and the value of f at any point (x, y) gives the brightness (or, grey level) at that point”.

For this section it would be better to talk of light intensity or lightness (instead of brightness). Correct terms: *lightness* describes the real physical light intensity, brightness is only in the mind.

Think now of the scene as a flat two-dimensional plane – a sheet of coloured paper. Its lightness, $f(x, y)$, is the product of two factors:

- $i(x, y)$ – the illumination of the scene, i.e. the amount of light falling on the scene, at (x, y) ,
- $r(x, y)$ – the reflectance of the scene, i.e. the ratio of reflected light intensity to incident light.

$$f(x, y) = i(x, y)r(x, y) \quad (2.3)$$

Naturally occurring ranges of values of i and r :

Illumination (i)	units
Sunny day at surface of earth	9000
Cloudy day	1000
Full Moon	0.01
Office lighting	100
Reflectance (r)	units
Snow	0.93
White paint	0.80
Stainless steel	0.65
Black velvet	0.01

Note: pure white ($r=1$) and pure black ($r=0.0$) are hard to achieve.

2.2.3 Uneven Illumination

More often than not, when we sense a scene, we want to measure $r(x, y)$, so we assume that $i(x, y)$ is constant I_0 , so that $f(x, y) = r(x, y)I_0$. Thus except for the multiplicative constant, we have $r(x, y)$.

If illumination is *not* constant across the scene, then we have problems disentangling what variations are due to r , and what are due to i .

2.2.4 Uneven Sensor Response

Most modern electronic cameras are charge-coupled device (CCD) based. In a CCD you have a rectangular array of light sensitive devices $i = 0, 1, \dots, n-1$, $j = 0, 1, \dots, m-1$ at the image plane. The voltage given out by these is proportional to the amount of light falling on it.

Often it is assumed that an image $f(x, y)$ arriving at the cameras image plane, is converted into values (analogue or digital), $f_c(x, y)$, which are proportional to $f(x, y)$, i.e.

$$f_c(x, y) = K f(x, y) \quad (2.4)$$

If $K = K(x, y)$, i.e. it varies across the image plane, then we have non-even illumination. However, in this case, if $K(x, y)$ can be relied on to stay constant with time, we can estimate it, e.g. by imaging a sheet of constant reflectance, and constant illumination. This is *radiometric calibration*. An example is given at the end of the chapter.

2.3 Imaging Geometry

2.3.1 General

Figure 2.2 shows the imaging geometry (Rosenfeld and Kak, 1982b). The reference frame (x, y, z) is based on the image plane, with z being the optical axis. P , at coordinates (x_0, y_0, z_0) is a general point in the scene, and P_c , $(x_1, y_1, 0)$, its projection onto the image plane. The lens centre is $(0, 0, f)$. By similar triangles, the following relationships hold:

$$\frac{x_1}{f} = -\frac{x_0}{z_0 - f} \quad \frac{y_1}{f} = -\frac{y_0}{z_0 - f}$$

Reference frame (x, y, z) is based on the centre of the image plane; 0 is the origin.

Figure 2.2 Imaging Geometry.

2.3.2 Geometric Distortion

The equation for x_1 and y_1 above yields two important pieces of information: first, the image is inverted (x_1, y_1 are negative), and, second, there is a scale change, the larger z_0 , the smaller the image. Normally, camera users are unaware of the inversion, the recording process takes care of it. Clearly, however, scaling is a problem, since the size of the image changes with distance from the camera; it is not easy to ensure that the object remains at a fixed distance.

The problem is exacerbated if the object is tilted with respect to the image plane, there are different scalings for x_0 , and y_0 , and we have perspective distortion, see Figure 2.3 (a). In addition, due to imperfections,

lens systems may be subject to other forms of geometric distortion, involving non-linear terms in x_0 , y_0 and cross terms; typical are barrel distortion, and pincushion distortion, see Figures 2.3 (b), and 2.3 (c).

These show the distorted images of an object consisting of orthogonal, parallel, and equally spaced lines ruled on a plane.

(a) Perspective distortion	(b) Barrel distortion	(c) Pin-cushion distortion
-------------------------------	--------------------------	-------------------------------

Figure 2.3 Geometric Distortion

The existence of a range of distortion types, as well as parameters has serious implications for machine vision and pattern recognition: essentially they increase the ‘search-space’ for any matching procedure; an alternative, but equivalent, interpretation is that they introduce extra ‘invariance’ requirements on a recognition algorithm (see Chapter 8).

2.3.3 Geometric Calibration

In cases where we must make accurate spatial measurements from an image, it may be necessary to geometrically calibrate it. Essentially, this entails performing, numerically, the inverse of the image creation distortion.

2.3.4 Object Frame versus Camera Frame

Figure 2.2 shows two reference frames, (x, y, z) based on the camera, and (x', y', z') based on the object. The position of the origin of the object frame (its range) with respect to the camera frame *and* the relative orientation of the object frame is called its pose; in the aerospace industry this is called attitude).

Thus, pose = range vector ($r = (rx, ry, rz)$) and attitude which consists of: pitch = rotation about the x-axis, yaw = rotation about the y-axis, and roll = rotation about the z-axis.

2.3.5 Lighting Angles

As mentioned in section 2.2, the spatial and colour distribution of the light source are important factors. In addition directionality may be important: e.g. (a) directional light from a single oblique source causes shadows; (b) a light source close to the camera axis may cause specular reflection from shiny surfaces.

2.4 Sampling and Quantization

See Chapter 1. Be aware of:

- the squared increase in data volume with increase in spatial resolution; i.e. go from 2 mm \times 2 mm pixels to 1mm \times 1mm and the number of pixels increases by *four* (not two),
- ditto as the image size increases.

Look at pages Gonzalez and Woods, pp. 35–37 to see the effects of reducing resolution (sampling grid), and of reducing grey levels; notice how contouring becomes evident in Figure 2.10 (e) (16 levels, 4 bits), and (f) (8 levels, 4 bits).

2.5 Colour

2.5.1 Electromagnetic Waves and the Electromagnetic Spectrum

Light is a form of energy conveyed by waves of electromagnetic radiation. The radiation is characterised by the length of its wavelength; the range of wavelengths is called the *electromagnetic (EM) spectrum*. Visible light occupies a very small part of the spectrum.

Table 2.1 shows the EM spectrum: the left hand column gives the wavelength in meters, the middle gives the name of the band, and the right gives the frequency of the radiation in Hertz (cycles per second).

Thus, crudely, if you were to ‘speed-up’ the frequency of vibration of a TV signal, you would get microwaves, speed-up microwaves \rightarrow heat radiation, \rightarrow light \rightarrow UV \rightarrow X-rays, etc. (Incidentally, microwave cookers work at approximately 900 MHz, which happens to be the frequency at which the water molecule, H₂O, resonates).

Wavelength (m)		Name	Frequency (Hz)
10^{-15}	1 femtometer (fm)	gamma rays	3×10^{23} Hz
10^{-12}	1 picameter	X-rays	3×10^{20} Hz
10^{-9}	1 nanometer X-rays		3×10^{17} Hz
10^{-8}	10 nm Ultraviolet		3×10^{16} Hz
10^{-7}	100 nm U-V		
4×10^{-7}	400 nm	Visible light (violet)	
7×10^{-7}	700 nm	Visible (red)	
10^{-6}	1 micrometer	Infrared (near)	3×10^{14} Hz
10^{-5}	10 micrometers	Infrared	3×10^{13} Hz
		Infrared (heat)	
10^{-3}	1 millimeter	Infrared (heat) +	3×10^{11} Hz
10^{-1}	0.1 meters	microwaves	(300 GigaHz)
1 meter		microwaves	3×10^9 (3 GigaHz)
	FM radio is ~ 100 Mhz (VHF)	TV etc. (UHF)	3×10^8 (300 MegaHz)
10 meters		radio (shortwave)	30 Mhz
100 meters		radio (shortwave)	3 MHz
200 – 600 m		radio (medium wave)	1.5 MHz to 500 KHz
1500 m (1 Km)		radio (long wave)	200 KHz

Table 2.1: The electromagnetic spectrum.

It is possible to use various parts of the EM spectrum for imaging: e.g. X-rays, microwaves, infrared (near), and thermal infrared. Our major interest will be in visible light.

2.5.2 The Visible Spectrum

The visible spectrum stretches from about 400 nm to 700 nm. The reason why this part of the spectrum is visible is that the rods and cones in our retinas are sensitive to these wavelengths, and insensitive to the remainder; e.g. if you look at a clothes iron in the dark, you may ‘feel’ the heat radiated from it, but your eyes will not convert that energy into a light sensation; similarly, microwaves and X-rays, they may cause damage, but you will not ‘see’ them.

The relative spectral sensitivity of human eyes within the visible spectrum is shown in Figure 2.4, with approximate indication of corresponding colours. Figure 2.5 shows the sensitivity on the earth of light from space, resulting from the blocking effects of the earth’s atmosphere. This explains why X-ray and radio astronomy were fairly late developments, compared to optical (i.e., visual light) astronomy.

The term ‘spectral’ is often used – it refers to the electromagnetic radiation frequency *spectrum* – the range of frequencies which make up the light;

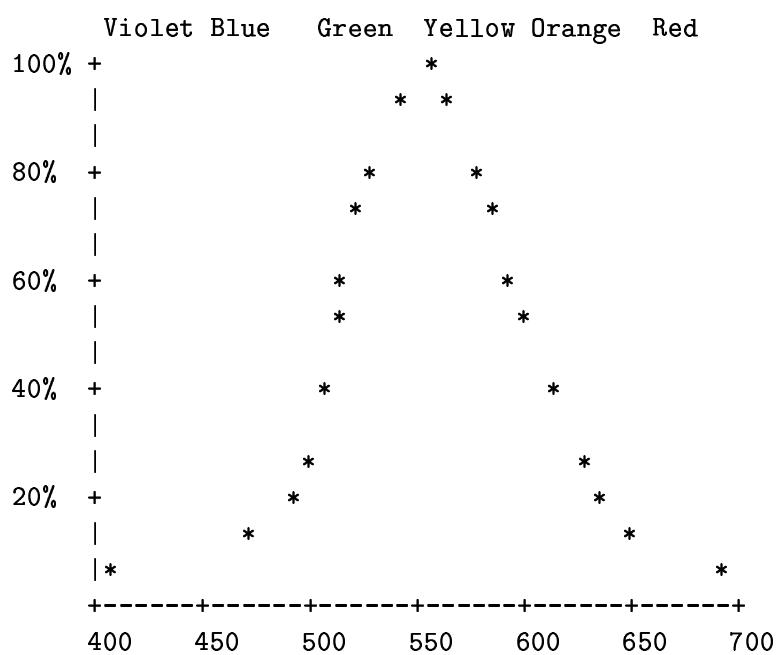


Figure 2.4 Relative spectral sensitivity of the eye.

Figure 2.5 The earth's atmosphere blocks out different parts of the EM spectrum -- fortunately for humankind.

we will have cause to cover other forms of spectra (see Chapter 3).

From Figure 2.4 we can see that the eye is very sensitive to radiation in the green-yellow range (peak at 550 nm), and relatively insensitive to blue, violet, and deep red; a blue light around 475 nm (relative sensitivity approx. 10%) would have to put out 10 times more power than the equivalent green-yellow light. Why did the human evolve this way? Well, the energy emitted by the sun (at least that part that reaches the earth) has an energy spectrum graph similar to Figure 2.4.

2.5.3 Sensors

A light sensor is likely to have a similar spectral response curve to Figure 2.4, though usually flatter and wider – i.e. more equally sensitive to wavelengths, and sensitive to UV and to near infrared.

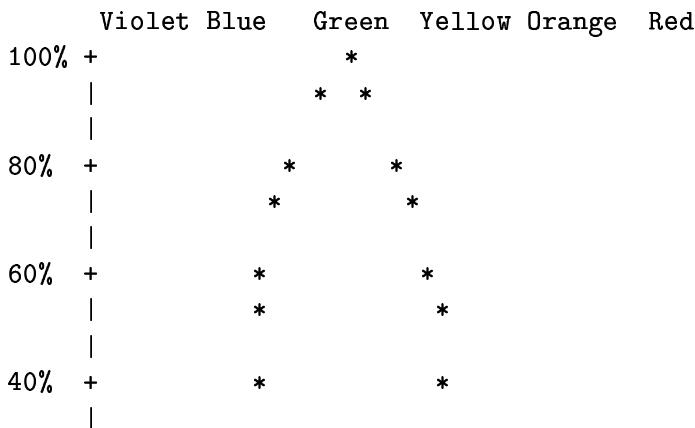
If Figure 2.4 was the spectral response of a sensor, then a blue light (see above), compared to a green-yellow light of the same power, would produce a sensor output of 10% of the voltage of the green-yellow.

2.5.4 Spectral Selectivity and Colour

We have already mentioned that a colour sensor (e.g. in a colour TV camera) is merely three monochrome sensors: one which senses blue, one green, and one red.

What is meant by sensing blue, green, or red? What we do is arrange for the sensor to have an effective response curve that is high in green (for example) and low elsewhere. But, we have already said that sensors have a fairly flat curve (maybe 200–1000 nm), so we must arrange somehow to block out the non-green light.

Wavelength sensitive blocking is done by a colour *filter*. A green filter allows through green light but absorbs the other; similarly blue and red.



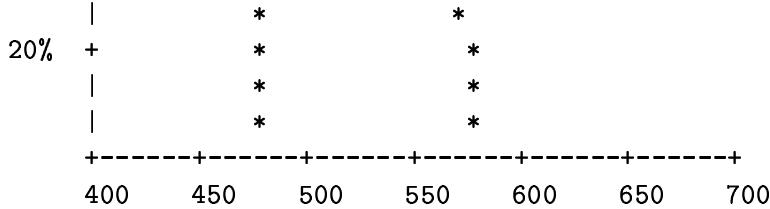


Figure 2.6 Relative sensitivity of a green filter.

So, we use three separate sensors, each with its own filter (blue, green, and red) located somewhere between the lens and the sensor. I.e. we have $f[d, r, c]$, $d = 0$ (blue), 1 (green), and 2 (red).

2.5.5 Spectral Responsivity

The relative response of a sensor can be described as a function of wavelength (forget about (x, y) or (r, c) for the present): $d(\lambda)$, where λ denotes wavelength.

The light arriving through the lens can also be described as a function of λ : $g(\lambda)$, and the overall output is found by integration:

$$\text{voltage} = \int_0^\infty d(\lambda)g(\lambda)d\lambda \quad (2.5)$$

Obviously, the integral can be limited to (say) 100 nm to 1000 nm.

If we have a filter in front of the sensor, relative transmittance (the amount of energy it lets through), $t(\lambda)$, then the light arriving at the sensor, $g'(\lambda)$, is the product of $g()$ and $t()$:

$$g'(\lambda) = g(\lambda)t(\lambda) \quad (2.6)$$

and equation 2.5 changes to:

$$\text{voltage} = \int_0^\infty d(\lambda)g(\lambda)t(\lambda)d\lambda \quad (2.7)$$

or,

$$\text{voltage} = \int_0^\infty d(\lambda)g'(\lambda)d\lambda \quad (2.8)$$

2.5.6 Colour Display

So now we have three images stored in memory; how to display them to produce a proper sensation of colour?

Similarly to our model of a colour camera as three monochrome cameras, a colour monitor can be thought of as three monochrome monitors: one which gives out blue light, one green and one red.

A monochrome cathode ray tube display works by using an electron gun to squirt electrons at a fluorescent screen; the more electrons the brighter the image; what controls the amount of electrons is a voltage that represents brightness, say $f_v(r, c)$.

A monochrome screen is coated uniformly with phosphor that gives out white light – i.e. its energy spectrum is similar to Figure 2.4.

A colour screen is coated with minute spots of colour phosphor: a blue phosphor spot, a green, a red, a blue, a green, ... following the raster pattern mentioned in Chapter 1. The green phosphor has a relative energy output like the curve in Figure 2.6; the blue has a curve that peaks in the blue, etc. There are three electron guns – one controlled by the blue image voltage (say, $f_b(0, r, c)$), one by the green ($f_g(r, c)$) and one by the red ($f_r(r, c)$). Between the guns and the screen, there is an intricate arrangement called a ‘shadow-mask’ that ensures that electrons from the blue gun reach only the blue phosphor spots, green → green spots, etc.

2.5.7 Additive Colour

If you add approximately equal measures (we are being very casual here, and not mentioning units of measure) of blue light, green light and red light, you get white light. That’s what happens on a colour screen when you see bright white: each of the blue, green, and red spots are being excited a lot, and equally. Bring down the level of excitation, but keep them equal, and you get varying shades of grey.

Your intuition may lead you to think of subtractive colour; filters are subtractive: the more filters, the darker; combine blue, green and red filters and you get black. However, with additive colour, the more light added in, the brighter; the more mixture, the closer to grey – and eventually white.

2.5.8 Colour Reflectance

This subsection may be skimmed at the first reading.

All this brings a new dimension to the discussion of illumination and reflectance in section 2.2. Now we can think of illumination (i) and reflectance(r) as functions of λ as well as (x, y) .

Thus, the lightness function is now spectral (and therefore a function of λ), i.e.

$f(\lambda, x, y)$ is the product of two factors:

- $i(\lambda, x, y)$ – the spectral illumination of the scene, i.e. the amount of light falling on the scene, at (x, y) , at wavelength λ ,
- $r(\lambda, x, y)$ – the reflectance of the scene, i.e. the ratio of reflected light intensity to incident light

$$f(\lambda, x, y) = i(\lambda, x, y)r(\lambda, x, y) \quad (2.9)$$

Why does an object look green (assuming it is being illuminated with white light)? Simply because its $r(\lambda, ..)$ function is high for λ in the green region (500-550 nm), and low elsewhere (again, see Figure 2.6). Of course, illumination comes into the equation: a white card illuminated with green light (in this case $i(\lambda, ..)$) looks like Figure 2.4) will look green, etc.

2.5.9 Exercises

Ex. 2.5-1 Write down cases where you might want to use very narrow band filters, i.e. you want to be very selective about the colour of light you let into the sensor.

Ex. 2.5-2 A coloured card whose reflectivity is $r(\lambda, x, y)$ is illuminated with coloured light with a spectrum $i(\lambda)$ (constant over spatial coordinates (x, y)); this is sensed with a camera whose CCD sensor has a responsivity $d(\lambda)$ (again constant over x, y); a filter with transmittance $t(\lambda)$ is used. Show that the overall voltage output is

$$v(x, y) = \int r(\lambda, x, y)i(\lambda)t(\lambda)d(\lambda)d\lambda$$

Ex. 2.5-3 A blue card is illuminated with white light; explain the relative levels of output from a colour camera for blue, green, red.

Ex. 2.5-4 A blue card is illuminated with red light; explain the relative levels of output from a colour camera for blue, green, red.

Ex. 2.5-5 A blue card is illuminated with blue light; explain the relative levels of output from a colour camera for blue, green, red. What, if any, will be the change from Ex. 2.5-4 ?

Ex. 2.5-6 A white card is illuminated with yellow light; explain the relative levels of output from a colour camera for blue, green, red.

Ex. 2.5-7 A white card is illuminated with both blue and red lights; explain the relative levels of output from a colour camera for blue, green, red.

Ex. 2.5-8 A blue card is illuminated with both blue and red lights; explain the relative levels of output from a colour camera for blue, green, red; what, if any, will be the change from Ex. 2.5-6.

2.6 Photographic Film

Many images start off as photographs, so film cannot be ignored. Realise that:

- just like the eye, film is limited in the range of illumination that it can handle,
- a camera adapts by opening / closing the lens diaphragm, – or, by increasing decreasing exposure time.

2.7 General Characteristics of Sensing Methods

2.7.1 Active versus Passive

Active methods require, in addition to a sensor, a source of energy which illuminates or otherwise probes or excites the object. See Figures 2.7 (a), (b), and (c).

Passive methods operate by sensing some emission that emanates naturally (e.g. reflected sunlight) from the object, see Figure 2.8.

2.7.2 Methods of Interaction

1. Absorption.

Here we assume that the object is relatively transparent, see Figure 2.7 (c). This is how X-rays work.

2. Reflection.

See section 2.5.8, and Figures 2.7 (a), (b) and Figure 2.8 (a).

3. Emission.

See Figure 2.8 (b); here the sensed object creates the sensed energy (e.g. a piece of hot metal, the sun).

2.7.3 Contrast

For sensing to be effective the sensed signal must change for different parts of the object (otherwise we have the equivalent of a blank screen); *contrast* defines the magnitude of sensed signal change that differentiates (generally speaking) between object present and not present. E.g. X-rays, let G_0 be

Figure 2.8 Active sensing configurations.

Figure 2.9 Passive sensing configurations.

the image grey level corresponding to just soft tissue, let G_b be the grey level for bone, then the contrast for bone, C_b , is

$$C_b = (G_b - G_0)/G_0 \quad (2.10)$$

2.7.4 Exercises

Ex. 2.7-1 (a) What is meant by *active* sensing.

(b) Explain how, and why, active infra red sensing cameras could be used by wildlife film-makers.

Ex. 2.7-2 (a) What is meant by *passive* sensing.

(b) Explain how, and why, passive infrared sensing cameras could be used by wildlife film-makers.

(c) In a military application, why would passive sensing be preferred to active.

Ex. 2.7-3 Identify and explain one application of aerial thermal infrared sensing.

Ex. 2.7-4 (a) Explain how a medical X-ray system works.

(b) Identify and explain uses of X-ray images, other than medical.

Ex. 2.7-5 Referring to Figures 2.7 and 2.8 identify a suitable sensing arrangement for detecting flaws (small holes) in paper manufacture.

Ex. 2.7-6 In problem 2.7-5, assume that you have a single line of sensors (512 of them across the moving roll of paper). The sensor is sampled rapidly, giving out 512 samples for every millimetre of paper longitudinal movement; the sensor width also corresponds to a transverse extent of 1 mm.

(a) Assuming that you have a function, say `sread(f)`, that reads the samples into an array `f` (`unsigned char f[512]`), and that your computer can keep up with the processing, suggest processing to detect small holes. [Assume background readout (normal) of 10, and much higher when there is a hole].

Hint:

```
#define NPIXELS 512
unsigned char f[NPIXELS];

while(1){ /*do forever*/
```

```

waitForSignal(); /*waits for sampling signal*/
sread(f);

for(i=0;i<NPIXELS;i++){
    a = s[i];
    b = s[i+1];
    if( a ???? b ???) flaw[i]=TRUE;
    else                  flaw[i]=FALSE;
}
}

```

- (b) Revise your program to distinguish between holes that are (i) about 1 to 2 mm wide, and (ii) more than 3 mm wide.

Ex. 2.7-7 Repeat 2.7-6 (b) now bringing the longitudinal dimension into consideration, i.e. we want to distinguish holes whose area is 4 sq mm (4 pixels) or less and those above that.

Ex. 2.7-8 How would you make your answer to Ex. 2.7-6 generalise to the case of holes *and* flaws caused by dark marks in the paper.

2.8 Worked Example on Calibration

A monochrome CCD camera (followed by a digitiser etc.) is monitoring parts passing along a conveyer belt; the scene is illuminated from above. There are four major difficulties:

1. uneven illumination,
2. bias (an uneven bias) in the CCD cells,
3. uneven gain in the CCDs,
4. in addition, there is noise (assume Gaussian and zero mean – like the noise generated by DataLab function ‘ggn’).

Develop a technique (a program of measurements, followed by calibration computation) by which the effects of the uneven illumination and the bias may be removed (calibrated). This calibration may take place, for example, once a day, or however often the effects change.

There are two equations (or models) that are applicable:

- Equation (i): $g(x, y) = r(x, y).i(x, y)$, [$g()$ =light entering camera].
i.e. light entering camera is a product of $r()$, reflectance of the scene, and $i()$ illumination.

- Equation (ii): $f(x, y, t) = b(x, y) + h(x, y).g(x, y) + n(x, y, t)$

This says that the output from the cell corresponding to position (x, y) is a function of:

- the bias of the cell (x, y) , $b(x, y)$, i.e. b is called *bias*, because it is added to all output; you have b even for zero bias,
- the gain of the system, $h(x, y)$, for cell (x, y) ; one cell may be more responsive than another, i.e. more output for the same input – its amplifier is turned up more!
- the noise.

I expect that your answer will contain correction tables for each (x, y) .

Assume that you have constant reflectance white and grey cards (where $r(x, y) = \text{constant}$, for all (x, y)). Assume also that you can shut out all light from the camera (e.g. using a lens cap).

Describe how you would compensate for:

1. Uneven illumination on its own; i.e. assume that $b(x, y) = 0.0$, $h(x, y) = 1.0$ for all cells, and that $n(x, y, t) = 0.0$ for all (x, y, t) .
2. Variable bias on its own; i.e. all the other effects are missing.
3. Variable gain on its own.
4. Noise on its own. Hint: the conveyer belt is moving very slowly, and you have enough time to capture a large number of images of the object.
5. The whole lot, together.
6. How should the calibration results be used (in operational mode)?

Answer:

Extract from chapter 4.

Assume you have the possibility of obtaining many still, ‘identical’, images of a scene; but, the images are picking up noise in transmission, or from the sensor system. Then averaging together these images pixel by pixel:

$$f_a(r, c) = (1/N_a) \sum_{i=1..N_a} f_i(r, c)$$

for $r = r_l..r_h, c = c_l..c_h$, where $N_a = \text{number of images averaged}$, and $f_i(., .)$ is the i th image. Note: this is done for each pixel independently; we are not smoothing across pixels. Thus, we can talk about the mean, $m(r, c)$, and variance $v(r, c)$ of each pixel.

If our model is that the only distortion is noise, then $f_i(r, c)$ can be written:

$$f_i(r, c) = f(r, c) + n_i(r, c)$$

i.e. the i th image is the true, noiseless image, plus the i th noise image. Most naturally occurring noise has the characteristic (or is assumed to have) that it is random and uncorrelated. Often too, it is zero mean.

Roughly speaking, these last two statements indicate that for every positive noise value, you will eventually get a negative one, and if you take enough values in the average you end up with zero.

It can be shown that if the noise level (e.g., as indicated by the standard deviation of the pixel values (at (r, c)) is s_1 for 1 image (no averaging), then it is

$$s_n = s_1 / \sqrt{N_a}$$

for N_a images averaged.

You can easily experience two examples of this:

1. On a noisy stereo radio reception, switch the tuner to mono; this causes the system to add the left and right signals to produce one signal; the noise standard deviation reduces by $1/\sqrt{2} = 1/1.414$, i.e. reduces to 0.707 of what it was,
2. freeze frame on a videoplayer, see how noisy the image is compared to moving: the eye tends to average over a number of the 25 images painted on the screen per second.

The process involved here is *calibration*, in particular photometric or radiometric calibration (to do with grey level values). The other sort is *geometric calibration*, i.e. correcting the geometric shape of the image.

It would be relatively safe to assume that, except for the noise, all the other factors are invariant with time (at least, after the system has warmed up). If this is not the case, at least the factors will vary slowly, so that you can stop and recalibrate often enough to catch the changes.

Even though the question mentions only bias ($b()$), we will also consider gain ($h()$) in this answer. We have:

$$f(x, y, t) = b(x, y) + h(x, y).g(x, y) + n(x, y, t),$$

Using eqn. 2.9:

$$f(x, y, t) = b(x, y) + h(x, y).r(x, y).i(x, y) + n(x, y, t),$$

Now, $h()$ and $i()$ can be combined

$$d(x, y) = h(x, y).i(x, y)$$

so we only need determine $d(x, y)$ for all cells.

Noise: we must assume that the noise is random and uncorrelated (from $t = t_1$, to another $t = t_2$), and zero mean. I.e. if we start off with, on one image, with fluctuations of standard deviation s_1 , if we average over P images the standard deviation reduces to $s_P = s_1/\sqrt{P}$. We will always have some fluctuation; this corresponds to an error. The average size of the error can be reduced from (say) 10 units to 1 unit if (say) we average over $P=100$ images (see notes in section 4.10).

You would probably assume that the noise was the same level for all cells.

How to find out s_1 , the standard deviation for 1 image?

Collect many images of (say) a white card. Calculate the average pixel value for each (x, y) . Calculate the variance $v(x, y)$ for each (x, y) , then SD $s_1(x, y) = \sqrt{v(x, y)}$. Let this be s_1 .

Then if you want your error level to be s' , calculate P from

$$s' = s_1/\sqrt{P}, \text{ i.e.}$$

$$(c) \quad P = s_1^2/s'^2$$

so if s_1 is 10, and you want to get to $s' = 2$, P must be 25.

Calibration steps

1. Measure noise SD s_1 .
2. Decide on acceptable error for calibration data: s' .
3. Estimate P from s_1, s' (foregoing equation above).
4. Calibrate bias:
 - 4.1. Put the lens cap on the camera. Measure P images.
 - 4.2. Estimate $b(x, y) \pm$ error by averaging pixel at (x, y) over all P images.
Result: $b_e(x, y)$, for all cells x, y
5. Calibrate gain *and* illumination:
 - 5.1. Put a white card in the scene ($r(x, y) = \text{constant}$, for all x, y)
 - 5.2. Estimate $d(x, y)$ by averaging pixel at (x, y) over P images, and subtracting $b_e(x, y)$. (if $r()$ does not equal 1, but some other constant, K, then this will be included in our estimate – but no matter – the image is not absolute units anyway.)
Result: $d_e(x, y)$, for all cells x, y .

Use of calibration in operation

Note: if the noise level is unacceptable, then you will have to average multiple images in the manner described for ‘calibration mode’.

1. Measure $fa_1(x, y) = \text{average of } P f(x, y, t) \text{ images.}$
2. Remove bias: $fa_2(x, y) = fa_1(x, y) - b_e(x, y)$
3. Remove (‘cancel-out’) effects of uneven gain and illumination:

$$fa_3(x, y) = fa_2(x, y)/d_e(x, y)$$

$fa_3()$ is the calibrated image (noise reduced as well)

There will be errors/fluctuations, but the level of these can be controlled by the appropriate choice of P_s .

2.9 CCD Calibration in Astronomy

2.9.1 CCD Detectors versus Photographs

In astronomy, CCDs (charge-coupled device detectors) are widely-used. Photographic plates are also used for particular purposes and we will say a few words about images which are produced from photographs.

Photographic plates are subsequently digitized, using microdensitometers – the name is indicative of how the relative density of the photographic emulsion, following chemical perturbation effects caused by the detected light, is sampled at regular, very fine intervals at the micrometer level. These microdensitometer machines are only available in certain institutes world-wide. One such machine (at Space Telescope Science Institute, Baltimore), it was once pointed out to me, was positioned on huge concrete blocks in the bottom floor of the building, to avoid even the most minute vibration from the ambient environment. Needless to say, Baltimore does not suffer from any earthquakes. The level of precision obtainable with such machines is far beyond that offered by regular scanners.

The photographic plates themselves are of very high quality. A chemical emulsion known as IIIa-J is often used. This has very high-quality behaviour. About 3 years, the one company in the world which supplies such photographic plates, Kodak, was going to cease to do so, and this caused a lot of worry among professional astronomers. Photographic plates are used for wide fields of view, – hence survey work (i.e. surveys of lots of astronomical objects – stars, galaxies, etc.). CCDs, especially big ones (lots of pixels) are costly.

The dynamic range of photographs is very good. However, they suffer from nonlinear response effects, e.g. as one looks towards the central area of a star on a photographic, there is a levelling off of the responsiveness of the photographic plate. For so-called wide-field (large field of view) survey work, this is not too much of a problem. CCDs have the property of being much more linear in their response, i.e. output representation proportional to input.

Digitized photographic images may have to be corrected for geometric distortion. In addition there may be impurities detected on the photograph (e.g. dust), unevenness in the chemical constituents of the photographic plate, etc.

2.9.2 CCD Detectors and Their Calibration

Arriving photons of light dislodge electrons, which are read off a regular sampling grid. The electrons are converted to so-called data numbers, and these give us pixel values.

In the course notes, we have mentioned two equations of relevance:

$$g(x, y) = r(x, y)i(x, y)$$

where g is the light entering the camera; r is the reflectance; and i is the illumination.

$$f(x, y, t) = b(x, y) + h(x, y)g(x, y) + n(x, y, t)$$

where b is the bias, a constant added to the output even for zero input light; h is the gain of the system, the responsiveness of an individual pixel; and n is the noise. Note the time-dependence of some of these terms which we will ignore. It is also possible that there would be dependence on the wavelength of the incoming light.

We can merge these equations, getting

$$f(x, y) = b(x, y) + h(x, y)r(x, y)i(x, y) + n(x, y)$$

or simplifying by representing hi by d , the product of detector gain and illumination:

$$f(x, y) = b(x, y) + d(x, y)r(x, y) + n(x, y)$$

There may be added complications but we ignore them. These include the time dependence mentioned above; additional small non-linear terms; and charge transfer inefficiencies. So we have the nominal output of the detector, f . We want to correct this, to get estimates of bias, b , and of the gain as expressed in the term, d .

Now, we use three different items of information:

1. We use a so-called *dark frame*, an image with output current measured in the absence of any input. E.g. we put a lens cap on the detector.
2. We also use a so-called *flat-field*, an image (or a set of images) with the detector turned towards a uniformly-emitting source. This could be the inside of the observatory dome, for a ground-based observatory, or images of the ocean for Hubble Space Telescope. For a digital camera, we could use a white card. The flat-field describes the sensitivity over the CCD which is not uniform.
3. We take many images of the same scene, for the same duration, and average them. We do likewise for the darks and flat-fields. Why?

Because in averaging them, we dramatically decrease the noise. The relationship is as follows. Let n_1, n_2, \dots be the noise associated with images 1, 2, etc. Each of these will be a standard deviation (which can be understood as the spread of possible values) for each pixel. For pixels i and j we are just representing the noise images $n_i(x, y)$ for convenience by n_i . Over a given image, the simplest case to consider is when the noise is constant over all pixels.

Let the average of these noise values be $\hat{n} = 1/N \sum_{i=1}^N n_i$. This is the usual definition of the (statistical) mean.

Now consider averaging N noisy images $f_1 + n_1, f_2 + n_2, \dots, f_N + n_N$. We get the following interesting and not immediately intuitive result: $\hat{f} + \hat{n}/\sqrt{N}$. We have considerably decreased the noise, all the more so if N is large. What is the reason for this? To fully explain this we would have to step back, and start by clearly stating that the noise is Gaussian (normally-distributed). Consider the following: a noisy image means an uncertainty in measurement. Having lots of images provides us with a good deal of information, and hence helps to reduce the uncertainty. That is what the above expression for the averaged noisy image tells us.

Let us now approximate getting rid of the noise by taking N large enough, i.e. by having many identical images (note: the images will never be identically-valued in regard to noise! This is axiomatic.)

We now tackle the problem in the following stages: (i) use the darks to estimate the bias; (ii) use the flats to estimate the gain-times-illumination detector response; and (iii) use these in practice subsequently on 'real' images.

- Consider a stack (or set of N) dark images. Since there is no incoming light, we have $f_1 = b + n_1$. Averaging N such images, with N large enough so that the noise gets quite small, yields the average dark frame: $\hat{f} = b$. This is a way to estimate the bias, in practice. In this paragraph, each f was a dark. In the next paragraph, each f will be different, – a flat-field.
- Consider now a stack (or set) of N flat-fields. By definition, the reflectivity, r , is constant, say r_0 . The first such flat-field is $f_1 = b + d_1 r_0 + n_1$. Averaging N such flat-fields, and again ignoring noise which is very much lessened by the averaging, gives $\hat{f} = b + r_0 \hat{d}$. This gives enough information to estimate $\hat{d} = (\hat{f} - b)/r_0$. If we don't have a value for r_0 , this gives us relative calibration. If we have some absolute reference values for the pixels in our image, then we can estimate the constant reflectance r_0 and have an absolute calibration.

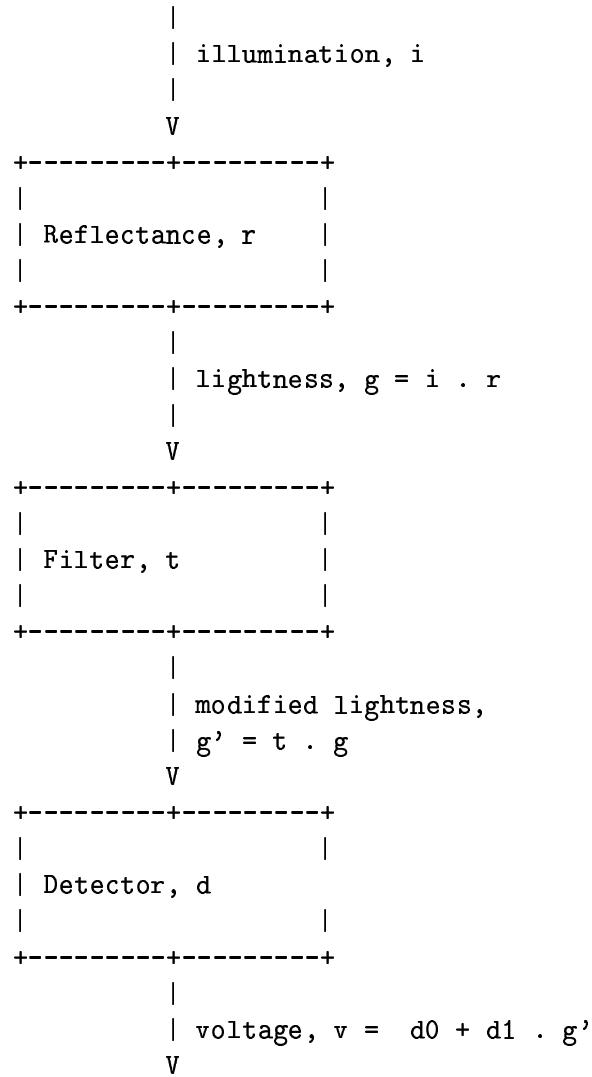
- Finally, having estimates of bias b , and of uneven detector response (gain times illumination) d , we put these results to use in the original imaging equation, $f = b + dr + n$, to determine r in practice. Needless to say, we again average frames to reduce the noise term.

Final remarks on this analysis: CCD detectors produce images with accompanying noise which to a good approximation is Gaussian. CCD detectors suffer from defective columns or rows (remember, the ejected electrons are made to run out along these), and these need to be allowed for, in the darks and flats as well as the ‘substantive’ images

2.10 Questions on Chapters 1 and 2 – Fundamentals, Sensors, and Calibration

1. (a) Explain in detail the steps (and data flow) in the progression from a 3-dimensional scene to a two-dimensional digital image.
 (b) Explain, using appropriate examples, the necessity to carefully tradeoff spatial resolution versus data volume.
 (c) An industrial inspection system is monitoring lace manufacture; the fabric is lace, and you want to detect very small stich flaws in it. The structure of the lace is such that thread separation is 0.5 mm.
 - Supporting your work with appropriate illustrations, derive a suitable sampling resolution (in millimeters).
 - Discuss data-rates and data volumes for this problem.
2. (a) Explain in detail the steps (and data flow) in the progression from a 3-dimensional scene to a two-dimensional digital image.
 (b) Explain, using appropriate examples, the necessity to carefully tradeoff spatial resolution versus data volume.
 (c) A satellite earth observation system is required to be able to monitor landuse in fields of two hectares and above.
 - Supporting your work with appropriate illustrations, derive a suitable spatial ground sampling resolution (in meters). [1 Hectare = 100m × 100m].
 - Discuss data-rates and data volumes for this problem.
3. (a) Explain the components of the (monochrome) image sensing model given in the figure below. [8 marks]
 (b) Essentially, a CCD camera is comprised of many such sensors. Explain how the problem of ‘calibration’ arises. [4 marks]

- (c) Distinguish between photometric (or radiometrical) calibration, and geometric rectification. [3 marks]
- (d) Briefly, explain how you would calibrate / measure detector 'bias' [Hint: d_0], and how you would use this result in practice to compensate for uneven bias. [5 marks]



4. (a) Explain the components of the (monochrome) image sensing model given in the Figure. [8 marks]
- (b) Essentially, a CCD camera is comprised of many such sensors. Explain how the problem of 'calibration' arises. [4 marks]
- (c) Distinguish between photometric (or radiometrical) calibration, and geometric rectification. [3 marks]

- (d) Briefly, explain how you would calibrate / measure detector ‘gain’
[Hint: d1], and how you would use this result in practice to compensate for uneven gain. [5 marks]
5. Q3. with (d) focussing on noise reduction.
6. Q3. with (d) focussing on uneven illumination.
7. (a) Explain the components of the (monochrome) image sensing model given in the Figure. [8 marks]
- (b) Extend the equations to include wavelength sensitivity. [NB. the detector will now have an integration]. [6 marks]
- (c) Hence, use appropriate diagrams and/or equations to explain the operation of a ‘colour’ sensing (ie. one which senses ‘red’, ‘green’ and ‘blue’). [6 marks]

Chapter 3

The Fourier Transform in Image and Signal Processing

3.1 Introduction

In Section 3.1 we will briefly introduce digital signal processing concepts (the processing one-dimensional sequences of data). Section 3.2 introduces the concept of the Fourier series expansion of functions, and section 3.3 the one-dimensional integral Fourier transform. Section 3.4 defines and describes the discrete Fourier transform (DFT). Section 3.5 introduces a fast algorithm for computing the DFT, the Fast Fourier transform – FFT.

Section 3.6 discusses convolution, impulse responses, linear systems, and convolution in continuous systems, and two-dimensional convolution. Section 3.7 describes how the FFT can be used as a tool for performing ‘fast’ convolutions.

Section 3.8 looks at the ‘matrix transformation’ nature of the DFT.

Section 3.9 shows how the DFT can be used to compute correlations.

3.2 Digital Signal Processing

3.2.1 Introduction

As introduced in Chapter 1, a continuous image is a two-dimensional lightness function $f(x, y)$, where x and y denote spatial coordinates, and the value of f at any point (x, y) gives the lightness (or, grey level) at that point.

Likewise a (continuous) signal is a one-dimensional function (usually of time) $f(t)$, where the value of $f(\cdot)$ at t gives the intensity at (time) t .

The function, $f(\cdot)$, could represent anything: the loudness of sound on a record, on a telephone line; it could represent temperature, pressure, the

retail-price-index, the price of sliced pan-loaves..., although these last two are liable to be sampled (discrete) rather than continuous.

The sorts of objectives we have in signal processing match those of image processing:

- clean up signals so that humans find them more useful, e.g. remove noise from a telephone signal, to make it more audible.
- automate human tasks, e.g. recognition of spoken words.

Whilst the main impetus for digital image processing came from the need for cleaning up pictures from space probes, and tasks like optical character recognition, much of early digital signal processing research was based on seismic data processing for oil prospecting; such was the complexity of the required processing, that available analogue (continuous) methods were inadequate and the processing had to be done (digitally) on computers.

We have already indicated the forces that have driven systems from analogue/continuous to discrete/digital; signal processing has also been subject to these forces and more and more signal processing is becoming digital.

3.2.2 Finite Sampled Signals.

The domain of the function $f(\cdot)$ is, strictly, $-\infty$ to $+\infty$, but, for practical purposes, attention is often limited to some piece of the infinite domain:

$$t_0 \leq t \leq t_1, \text{ where } t_0, t_1 \text{ are finite.}$$

This is the same as in imaging, where we limit our attention to some finite rectangular part of an infinite image.

Notation: the literature on signal processing commonly uses $x(t)$ as the ‘general’ signal. We use $x_c(t)$ to explicitly state that $x_c()$ is continuous.

If $x_c(t)$ is sampled (cf. discussion of images in Chapter 1) we obtain an array of numbers (a discrete or digital data sequence),

$$x[n], \quad n = 0, 1, 2, \dots, N - 1$$

where N is the number of samples.

Normally we use the notation $[\cdot]$ to denote a discrete function.

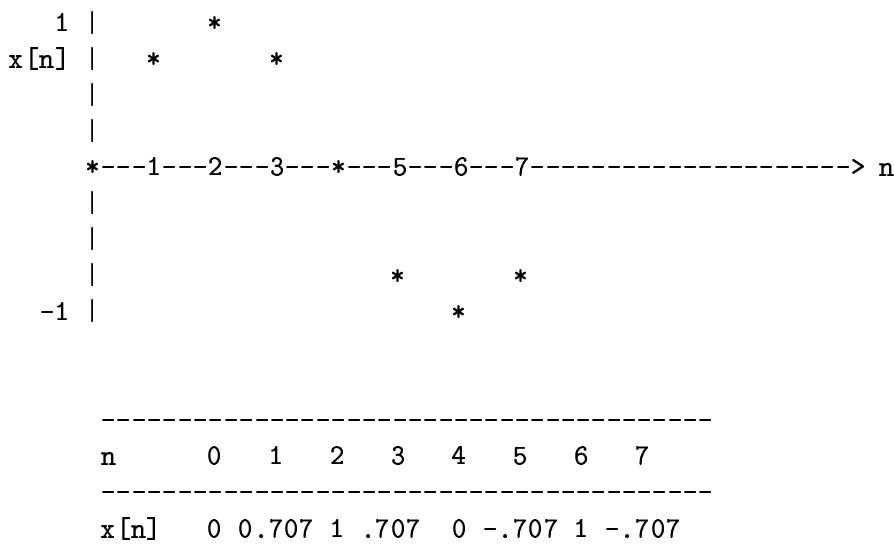
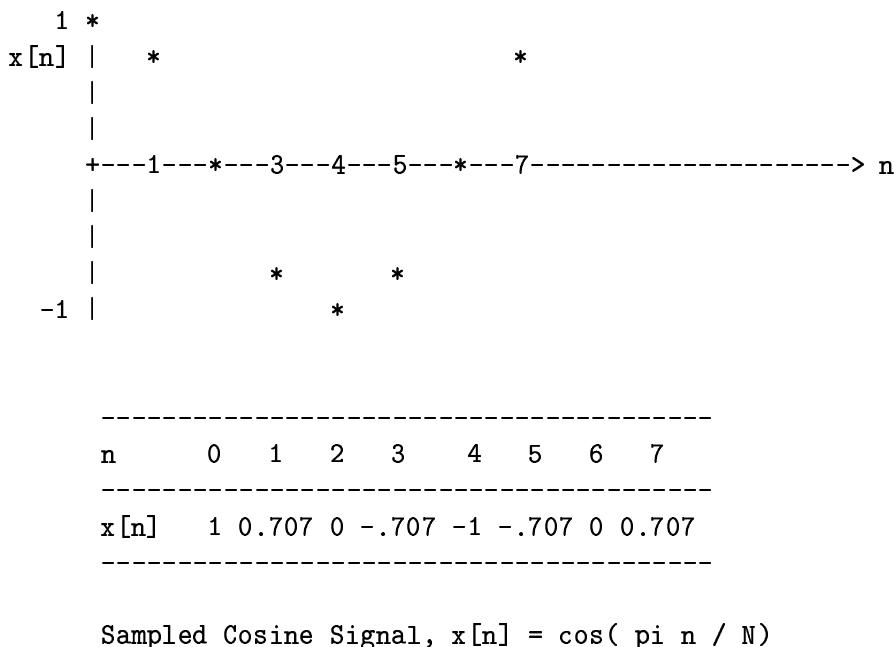
Typically, $x[n]$ is obtained by sampling $x(t)$ *periodically* at intervals T , i.e.

$$x[n] = x_c(nT), \quad n = 0, 1, 2, \dots, N - 1$$

Of course, $x[n]$ is some digital representation of a real number, i.e. in addition to time sampling it is digitized.

The following figures show examples of some sampled signals. If these were continuous, we would have two changes:

1. instead of existing just at discrete sample points, $n = 0, 1, 2$ etc. they would exist continuously – there would be a continuous line joining the points, and
 2. the discrete/integer abscissa n would be replaced by a real valued one, e.g. t (for time).

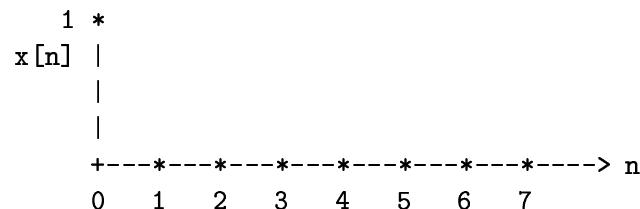


Sampled Sine Signal, $x[n] = \sin(\pi n / N)$

- Ex. 3.3.2-1** (a) Use a calculator to verify the values in the figures showing the sampled sine and cosine signals.

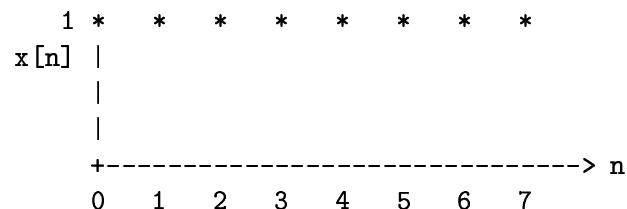
(b) Verify that $0.707 = \sqrt{2}/2$.

(c) Verify that both cosine and sine are periodic, i.e. the repeat themselves for $n = 8, 9, \dots$



n	0	1	2	3	4	5	6	7
x[n]	1	0	0	0	0	0	0	0

Impulse Signal



DC (Direct Current), i.e. constant value, Signal

3.2.3 Sampling Frequency

The symbol T in the sampling equation, $x[n] = x_c(nT)$, is the sampling period. The sampling frequency (f_s) is

$$f_s = 1/T$$

Nyquist Sampling Theorem: If a continuous signal contains only frequencies in the range 0 to f_{\max} (the *bandwidth*), then by sampling it at twice f_{\max} we obtain a discrete sequence from which it is possible to *exactly* reconstruct the original continuous signal; that is, there is no loss of information.

Therefore

$$f_s = 2f_{\max}$$

for no loss of information.

Ex. 3.3.2-2 For most voice recognition tasks, the effective bandwidth is 5 kHz (5,000 cycles per second). Thus, $f_s = 10$ kHz. Sampling period, T :

$$T = 1/f_s = 1/10,000 \text{ seconds} = 100 \text{ microseconds}$$

Ex. 3.3.2-2 If hi-fi music has a bandwidth of 15 kHz, calculate the minimum allowable sampling frequency f_s ; what is the corresponding sampling period T ?

Ex. 3.3.2-3 Compact discs (CDs) actually use $f_s=44.1$ kHz.

(a) What is sampling period, T ?

(b) For one second of music, what is N in the sampling equation, $x[n] = x_c(nT)$, $n = 0, 1, \dots, N - 1$?

Answer: 44,100. But note – this requires two channels – 88,200 – for stereo.

(c) one minute?

(d) one hour?

(e) Assuming 16 bits per sample, how many Megabytes are there on a one-hour CD.

Answer: 635 Megabytes.

3.2.4 Amplitude Resolution

If the dynamic range is properly chosen, 12-bits ($[0 \dots 4095]$), is adequate for hi-fi reproduction of audible signals; CDs use 16-bits.

3.2.5 Frequencies

When we say f_{\max} is 15 kHz, we mean that the most rapidly varying component ($x_m(t)$) of the signal corresponds to a ‘pure’ tone, or sinusoid (sine function), whose frequency is 15 kHz. I.e.

$$x_m(t) = a_m \sin(2\pi f_{\max} t)$$

where $\pi = 3.1415926\dots$ and a_m is the ‘amount’ of that frequency present.

Ex. 3.3.4-1 If f_{\max} is 1 Hz, work out the values of $x_m(t)$ for time samples

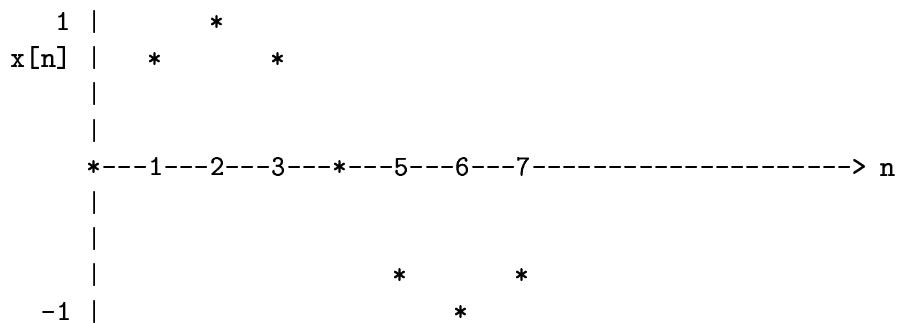
$$t = 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0$$

Assume $a_m = 1$.

t	$2\pi t$	$x_m(t) = \sin()$
0.0	0.0	0.0
0.125	0.785	0.707
0.25	1.57	1.0
0.375	2.36	0.707
0.5	3.14	0.0
0.625	3.93	-0.707
0.75	4.71	-1.0
0.875	5.50	-0.707
1.0	6.28	0.0

Ex. 3.3.4-2 Plot $x_m(t)$ from the foregoing Ex.

Answer:



n	0	1	2	3	4	5	6	7
x[n]	0	0.707	1	.707	0	-.707	1	-.707

Sampled Sine Signal, $x[n] = \sin(\pi n / N)$

Ex. 3.3.4-3 If f_{\max} is 2 Hz, work out the values of $x_m(t)$ for $t = 0, \frac{1}{16}, \frac{1}{8}, \frac{3}{16}, \frac{1}{4}, \dots, \frac{15}{16}, 1$. Assume $a_m = 1$.

Plot x_m .

Ex. 3.3.4-4 If f_{\max} is 0.5 Hz, work out the values of $x_m(t)$ for $t = 0, \frac{1}{8}, \frac{1}{4}, \dots, \frac{7}{8}, 1$. Assume $a_m = 1$.

Plot x_m .

3.2.6 Phase

Ex. 3.3.5-1 If f_n is 1 Hz, work out the values of $x_{cn}(t) = \cos(2\pi f_n t)$ for $t = 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0$.

Answer:

t	$2\pi t$	$x_{cn}(t) = \cos()$
0.0	0.0	1.0
0.125	0.785	0.707
0.25	1.57	0.0
0.375	2.36	-0.707
0.5	3.14	-1.0
0.625	3.93	-0.707
0.75	4.71	0.0
0.875	5.50	0.707
1.0	6.28	1.0

If you plot this you will find it is the same shape as $\sin()$, but is shifted along the t -axis by 0.25 (i.e. shifted by $\pi/2$).

Thus, $\cos()$ and $\sin()$ are ‘out of phase’ by $\pi/2$. Phase is a term for ‘where-you-are-in-the-cycle’. It is usually measured in radians:

$$\text{phase} = \phi \in [0, 2\pi]$$

or sometimes degrees: $\phi \in [0, 360]$.

Phase is usually written with a Greek letter lower-case phi, ϕ .

Human auditory perception is not usually sensitive to phase.

3.2.7 Periodic Signals

A signal $x(t)$ is periodic with period T_d if:

$$x(t) = x(t + T_d)$$

Thus, the 1 Hz sines and cosines, above, are periodic with period 1 sec. The 2 Hz signals have a period 0.5 seconds. A 4 Hz signal would have a period 0.25 sec., etc.

In analysis of signals it is often convenient to assume that signals are periodic. This can be done by taking a sufficiently large chunk of signal and assuming that it is repeated.

3.3 Fourier Series

3.3.1 General

Roughly speaking, *any* signal can be represented as a weighted sum of pure sinusoids:

$$x_n(t) = \sin(2\pi f_n t), \quad t = t_0 \dots t_1$$

$$x(t) = \sum_{n=0}^{n_{\max}} a_n x_n(t)$$

where a_n is the ‘weight’ of the contribution from frequency f_n .

Ex. 3.4-1 What does component 0 ($n = 0$) look like?

Thus if the signal is $\sin(2\pi 1 ft)$, i.e. a pure 1 Hz sinusoid, we have:

$$n_{\max} = 1,$$

$$a_0 = 0,$$

$$a_1 = 1.0,$$

$$f_{\max} = 1.0$$

Notation: it is useful to introduce a symbol for $2\pi f$ (the *radian frequency*). Greek lower-case ‘omega’, ω , is normally used. Thus

$$\sin(\omega t) = \sin(2\pi f t)$$

Ex. 3.4-2 Add the following components:

$$1 \sin(\omega t) + (1/3) \sin(3\omega t) + (1/5) \sin(5\omega t) \dots$$

where $\omega = 2\pi 1$.

What does the result look like?

As you add more terms you will get something which proceeds from the ‘sine-wave’ shape, to a ‘square-wave’ shape.

Strictly, this decomposition in terms of sinusoidal functions will not completely work. We must:

1. include cosines as well,
2. insist that the signal $x(t)$ is periodic.

$$x(t) = a_0 + \sum_{n=1}^{\text{nmax}} (a_n \cos(n\omega_d t) + b_n \sin(n\omega_d t)) \quad (3.1)$$

where $\omega_d = 2\pi/T_d$, T_d is the (repetition) period, i.e. ω_d corresponds to the frequency (f_d) of repetition.

Unless you know otherwise, nmax is infinity.

The infinite series,

$$x(t) = a_0 + a_1 \cos(1\omega_d t) + b_1 \sin(1\omega_d t) + a_2 \cos(2\omega_d t) \dots$$

is called the *Fourier series* corresponding to $x(t)$. Note that a_0 corresponds to the $n = 0$ term in the sinusoidal decomposition seen earlier in this section.

The coefficients a_n, b_n above are evaluated by the expressions,

$$\begin{aligned} a_0 &= 1/T_d \int_{-T_d/2}^{+T_d/2} x(t) dt \\ a_n &= 2/T_d \int_{-T_d/2}^{+T_d/2} x(t) \cos(n\omega_d t) dt \\ b_n &= 2/T_d \int_{-T_d/2}^{+T_d/2} x(t) \sin(n\omega_d t) dt \end{aligned}$$

These are the Fourier coefficients of $x(t)$.

Ex. 3.4-2 If $x(t) = \sin(1.t)$, $\omega_d = 1$, and $T_d = 2\pi/\omega_d = 2\pi$ there is only one component, b_1 :

$$b_1 = 2/T_d \int_{-T_d/2}^{+T_d/2} x(t) \sin(1.1.t) dt$$

This is just the same as integrating from 0 to T_d ,

$$= 2/T_d \int_0^{+T_d} \sin(1.t) \sin(1.1.t) dt$$

(Integral of $\sin^2(t) = t/2 - \sin(2t)/4$.)

$$= (2/T_d)[t/2 - \sin(2t)/4] \text{ evaluated at } t = T_d$$

$$-[t/2 - \sin(2t)/4] \text{ evaluated at } t = 0\}$$

$$= (2/T_d)[T_d/2 - \sin(2T_d) - 0 - \sin(0)]$$

$= (2/T_d)[T_d/2 - 0]$, since $2T_d = 4\pi$ and $\sin(4\pi) = 0$. Hence $b_1 = 1$, as expected!

All the above says is that: *any periodic waveform can be represented by weighted sum of a number of harmonics of the fundamental repetition frequency.*

A *harmonic* is any integer multiple of a *fundamental frequency*.

3.3.2 Orthogonal Functions

[Recall orthogonal vectors – section 3.2.13]

In Ex. 3.4-2 all the other coefficients are zero because the functions, $\cos(n\omega_d t), \sin(n\omega_d t)$ are *orthogonal* in the range $-T_d/2$ to $+T_d/2$, i.e.

$$2/T_d \int_{-T_d/2}^{+T_d/2} \sin(k\omega_d t) \sin(n\omega_d t) dt = 1$$

for $k = n$, and equal to zero otherwise.

Similarly,

$$2/T_d \int_{-T_d/2}^{+T_d/2} \cos(k\omega_d t) \cos(n\omega_d t) dt = 1$$

for $k = n$, and equal to zero otherwise.

Finally,

$$\frac{2}{T_d} \int_{-T_d/2}^{+T_d/2} \sin(k\omega_d t) \cos(n\omega_d t) dt = 0$$

for all k, n .

3.3.3 Finite Fourier Series

The Fourier series is given by eqn. 3.1,

$$x(t) = a_0 + \sum_{n=1}^{\text{nmax}} [a_n \cos(n\omega_d t) + b_n \sin(n\omega_d t)]$$

where $\omega_d = 2\pi/T_d$, T_d is the (repetition) period, i.e. ω_d corresponds to the frequency (f_d) of repetition.

If the largest frequency present is just less than $N\omega_d$, i.e. N times the repetition frequency, then we can set $\text{nmax} = N - 1$ and we can represent $x(t)$ with: a_0 , $(N - 1)$ a -coefficients and $(N - 1)$ b -coefficients.

This gives a total of $2N - 1$ coefficients.

3.3.4 Complex Fourier Series

$$x(t) = a_0 + \sum_{n=1}^{\text{nmax}} [a_n \cos(n\omega_d t) + b_n \sin(n\omega_d t)]$$

can be written even more compactly as:

$$x(t) = \sum_{-n\text{max}}^{+n\text{max}} [c_n \exp(jn\omega_d t)]$$

Now the coefficients are complex numbers $c_n = a_n + jb_n$, ($j = \sqrt{-1}$)

Note that:

$$\exp(jB) = \cos(B) + j \sin(B)$$

$$\cos(B) = (\exp(jB) + \exp(-jB))/2$$

$$\sin(B) = (\exp(jB) - \exp(-jB))/2 \quad j$$

Now the coefficients are given by:

$$c_n = (1/T_d) \int_{+T_d/2}^{-T_d/2} x(t) \exp(-jn\omega_d t) dt$$

These equations introduce nothing new - they are just shorthand for the earlier definition of the Fourier series.

Ex. 3.4-3 Verify the last statement by substituting the equation for $\exp(jB)$ above in

$$c_n = (a_n - jb_n)/2$$

$$c_{-n} = (a_n + jb_n)/2$$

and

$$c_0 = a_0$$

3.4 The Fourier Transform

If we now relax the ‘periodic’ restriction, i.e. let $T_d \rightarrow \infty$, we have

$$x(t) = \sum_{-\infty}^{+\infty} c_n \exp(jn\omega_d t)$$

and

$$c_n = (1/T_d) \int_{+T_d/2}^{-T_d/2} x(t) \exp(-jn\omega_d t) dt$$

$$T_d \rightarrow \infty \implies \omega_d = 2\pi/T_d \rightarrow dw$$

(i.e. *very* small),

$$n\omega_d \rightarrow n dw$$

(call this ω).

The above equation for c_n now becomes:

$$c_n = (dw/2\pi) \int_{+\infty}^{-\infty} x(t) \exp(-j\omega t) dt$$

and the equation for $x(t)$ is now a sum of infinitely many terms, i.e. an integral:

$$x(t) = \int_{+\infty}^{-\infty} c_n \exp(j\omega t) dt$$

Substituting for c_n gives:

$$x(t) = \int_{-\infty}^{+\infty} d\omega \left[\int_{-\infty}^{+\infty} x(t) \exp(-j\omega t) dt \right] \exp(j\omega t) / (2\pi)$$

The expression in the square brackets [.] is a function of angular frequency (ω) and is called the *Fourier transform* of $x(t)$.

Fourier transform:

$$X(\omega) = (1/2\pi) \int_{-\infty}^{+\infty} x(t) \exp(-j\omega t) dt$$

If we substitute $X(\omega) = \dots$ in the equation for $x(t)$ above, we get:

Inverse Fourier transform:

$$x(t) = (1/2\pi) \int_{-\infty}^{+\infty} X(\omega) \exp(+j\omega t) d\omega$$

The simplest way of thinking about the Fourier transform is that it gives an estimate, at each value of ω , of the ‘amount’ of

- $\cos(\omega t)$ in the signal
- $\sin(\omega t)$ in the signal

Here you can get an estimate for continuous ω . On the visual display of a ‘graphic-equaliser’ you see much larger ‘ ω -blocks’. Further, no distinction is made between $\sin()$, and $\cos()$ – human ears being insensitive to phase cannot tell the difference between $\cos()$ and $\sin()$!

$X(\omega)$ is a complex number:

$$X(\omega) = X_r(\omega) + jX_i(\omega)$$

where X_r , and X_i , are the real part, and imaginary part, respectively.

The X_r s correspond to the cosines, the X_i s correspond to sines.

Negative Frequencies?? Just a mathematical convenience. Physically, a negative frequency is interpreted as the same as a positive frequency.

The Fourier Transform, $X(\omega)$, is often called the *spectrum* of $x(t)$.

If you take the modulus of the complex numbers:

$$\begin{aligned} X_a(\omega) &= |X(\omega)| = |X_r(\omega) + jX_i(\omega)| \\ &= \sqrt{(X_r * X_r + X_i * X_i)} \end{aligned}$$

$X_a(\omega)$ is called the *amplitude spectrum*;

$X_a * X_a$ is called the *power spectrum*.

Likewise, by calculating the *argument* (angle) of the complex number

$$X_r + jX_i$$

$$X_p(\omega) = \arg(X(\omega)) = \arctan(X_i(\omega)/X_r(i))$$

we get the *phase* spectrum.

3.5 Discrete Fourier Transform

3.5.1 Definition

The Discrete Fourier Transform (DFT) of the digital signal/sequence $x[n], i = 0, 1 \dots N - 1$, is

$$X[u] = (1/N) \sum_{n=0}^{N-1} x[n] \exp(-j \pi u n/N)$$

for $u = 0, 1 \dots N - 1$.

The Inverse DFT (IDFT) is

$$x[n] = \sum_{u=0}^{N-1} X[u] \exp(+j \pi u n/N)$$

Interpretation of DFT:

If $x[n]$ has been produced by sampling at $f_s = 1/T$, the largest frequency present in $x[n]$ is $f_s/2$ (section 3.3.3).

Examine the DFT equation, and look at just the cosine part,

$$X[u] = (1/N) \sum_{n=0}^{N-1} x[n] \exp(-j \pi u n/N)$$

Cosine part:

$$X_c[u] = (1/N) \sum_{n=0}^{N-1} x[n] \cos(\pi u n/N)$$

At $u = 0 : \cos(0) = \cos(\pi (0) n/N) = \cos(0) = 1$

i.e. we are matching $x[n]$ with a constant (sometimes called the DC component – DC = direct current) signal.

At $u = 1 : \cos(1) = \cos(\pi 1 n/N)$.

This term varies slowly from $\cos(0)$ at $n = 0$ to $\cos(\pi (N-1)/N) = \cos(\pi)$ i.e. just one cycle in the sequence.

At $u = 2$: we get 2 cycles in the sequence.

At $N/2$: we get $N/2$ cycles in the sequence, i.e. 2 samples per cycle, i.e. very rapidly varying, this corresponds to $f_s/2$.

3.5.2 Discrete Fourier Spectrum

As with the continuous Fourier Transform, $X(\omega), X[u]$ is often also called the *spectrum* of $x[n]$ – or the *discrete spectrum*.

If you take the modulus of the complex numbers, $X[u] = X_c[u] + jX_s[u]$:

$$X_a(w) = |X[u]| = |X_c[u] + jX_s[u]| = \sqrt{(X_c * X_c + X_s * X_s)}$$

$X_a[u]$ is called the *amplitude* spectrum;

$X_a * X_a$ is called the *power* spectrum.

Likewise, by calculating the *argument* (angle) of the complex number

$$X_p[u] = \arg(X[u]) = \arctan(X_s[u]/X_c(u))$$

we get the *phase* spectrum.

3.5.3 Interpretation

The components near the beginning of the DFT correspond to the ‘slowly varying’ parts of the signal; the components near the middle, the ‘fast moving’ parts.

Domains: Often when working with sequences $x[n]$, or signals $x(t)$ we say we are in the *time domain*; in image processing, this becomes the *spatial domain*. If we are working with $X[u]$, or $X(\omega)$, then we are in the *frequency domain*.

3.5.4 Frequency Discrimination by the DFT

In general, if we have N samples, the spacing of each frequency sample is f_s/N .

$$df = f_s/N$$

The larger the N , the finer is our discrimination of frequency. If $N \rightarrow \infty$, then we have a continuous spectrum, i.e. the continuous Fourier transform.

Above $N/2$ we have no additional information; these terms correspond to the ‘negative’ frequencies of the continuous transform.

Ex. 3.6-1 Consider a CD signal sequence; it (one channel) is sampled at $f_s = 44.1$ KHz. Take a one second sequence, i.e. 44,100 samples.

Take a DFT of $x[n], n = 0, 1, \dots, 44099$.

We have $d_f = 44100/44100 = 1$ Hz, i.e. a sampling of 1 Hz of the spectrum.

The highest frequency is $N/2$; this is

$$f_{\max} = 44100/2 = 22050$$

i.e. the highest frequency represented ($f_s/2$).

Ex. 3.6-2 Show how the DFT obtained in Ex. 3.6-1 could be used to create ‘graphic-spectrum-analyser’ output in 10 ‘bands’ representing the frequency range 0 to 15 KHz?

0 to 15 KHz, so we are interested in the first 15000 frequency samples. There are 10 bands so each band will have 1500 Hz.

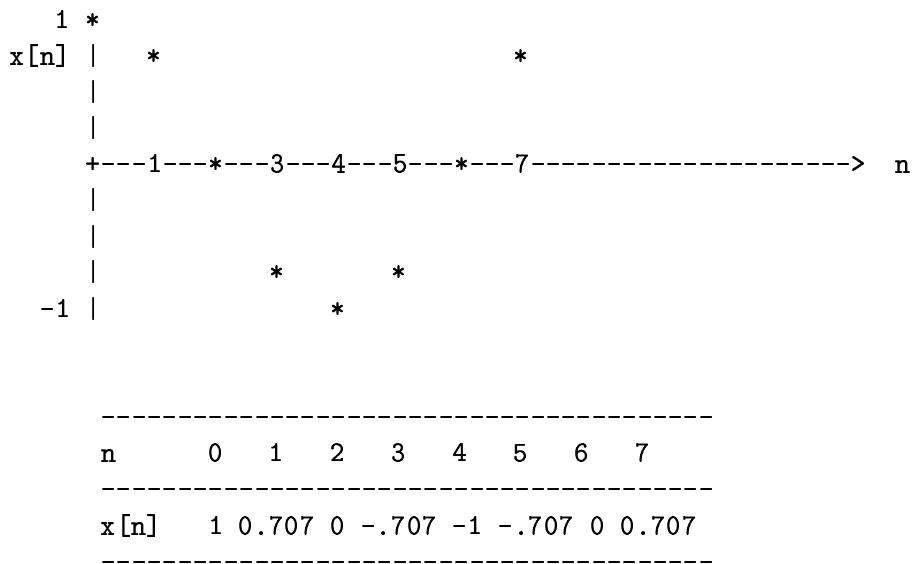
First band: 0 – 1500 Hz

2nd band: 1501 – 3000 Hz etc.

Therefore, for band 1, sum the *amplitude* spectrum from $\omega = 0$ to $\omega = 1500$.

Band 2: sum for $\omega = 1501$ to 3000 etc.

Ex. 3.6-3 (a) What is the DFT of a sequence containing the following single cycle of a cosine wave. Think, before you calculate.



Answer: Note all real, imaginary parts = 0, for all u.

u	0	1	2	3	4	5	6	7
Xr [u]	0	1/2	0	0	0	0	0	1/2
Xi [u]	0	0	0	0	0	0	0	0

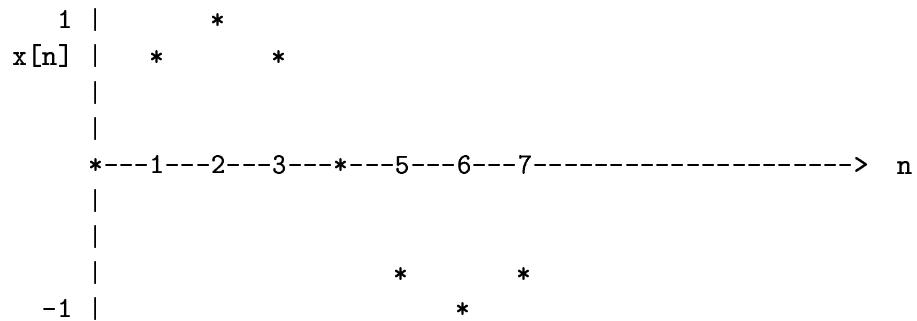
(b) What is the amplitude spectrum?

Answer:

u	0	1	2	3	4	5	6	7
Xa [u]	0	1/2	0	0	0	0	0	1/2

(c) Phase spectrum?

Ex. 3.6-4 (a) What is the DFT of a sequence containing the following single cycle of a sine wave. Again, think, before you calculate.



n	0	1	2	3	4	5	6	7
$x[n]$	0	0.707	1	.707	0	-.707	1	-.707

Answer: Note all imaginary, real parts = 0, for all u , i.e. just the opposite of the cosine.

u	0	1	2	3	4	5	6	7
Xr [u]	0	0	0	0	0	0	0	0
Xi [u]	0	1/2	0	0	0	0	0	1/2

(b) What is the amplitude spectrum?

Answer:

u	0	1	2	3	4	5	6	7
Xa [u]	0	1/2	0	0	0	0	0	1/2

(c) Phase spectrum?

Ex. 3.6-5 (a) What is the DFT of a sequence containing an ‘impulse’ $x[n] = \{1, 0, 0, 0, 0, \dots\}$; work out the first few terms.

Answer: Recall the definition of the DFT,

$$X[u] = (1/N) \sum_{n=0}^{N-1} x[n] \exp(-j \pi u n/N)$$

for $u = 0, 1, \dots, N-1$

We can separate this into a cosine part, $X_c[.]$, and and a sine part, $X_s[.]$:

$$X_c[u] = (1/N) \sum_{n=0}^{N-1} x[n] \cos(\pi u n/N)$$

$$X_s[u] = (1/N) \sum_{n=0}^{N-1} x[n] \sin(\pi u n/N)$$

And, since $\exp(-jB) = \cos(B) - j \sin(B)$,

$$X[u] = X_c[u] - j X_s[u]$$

Thus, work out $X_c[.]$:

$$\begin{aligned} X_c[u] &= (1/N) \sum_{n=0}^{N-1} x[n] \cos(\pi u n/N) \\ X_c[0] &= (1/N)[x[0] \cos(\pi 0 0/N) + x[1] \cos(\pi 0 1/N) + \dots] \\ &= (1/N)[1.1 + 0.1 + \text{etc. all zeros}] \end{aligned}$$

since $x[0] = 1$, and $x[i] = 0$, $i = 1 \dots N - 1$ and, $\cos(0) = 1$.

$$= 1/N$$

$$\begin{aligned} X_c[1] &= (1/N)[x[0] \cos(\pi 1 0/N) + x[1] \cos(\pi 1 1/N) + \dots] \\ &= (1/N)[1 1 + 0.(\text{don't care}) + \text{etc. all zeros}] \end{aligned}$$

since $x[0] = 1$, and $x[i] = 0$, $i = 1..N - 1$

$$= 1/N$$

Similarly $X_c[2] \dots X_c[7]$, all equal $1/N$.

Now $X_s[.]$:

$$X_s[u] = (1/N) \sum_{n=0}^{N-1} x[n] \sin(\pi u n/N)$$

$$X_s[0] = (1/N)[x[0] \sin(\pi 0.0/N) + \dots]$$

$$= (1/N)[1.0 + 0 + 0 \dots]$$

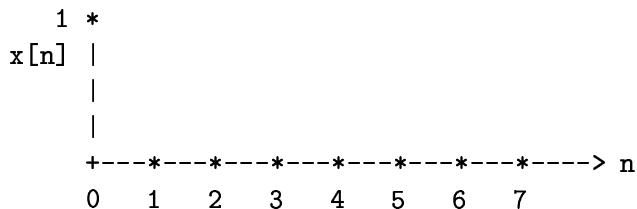
since $\sin(0) = 0$

$$X_s[0] = 0$$

Similarly $X_s[1] \dots X_s[7]$ all equal 0.

Thus, $X[0] = 1/N + j.0 = 1/N = 1/8 = 0.125$ (the imaginary part here being zero).

Thus:



n	0	1	2	3	4	5	6	7
$x[n]$	1	0	0	0	0	0	0	0

DFT – real, imaginary part = 0.

u	0	1	2	3	4	5	6	7
$X[u]$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$

(b) What is the amplitude spectrum? Work out the first few terms, you will be able to guess the remainder.

Recall the definition of the discrete amplitude spectrum:

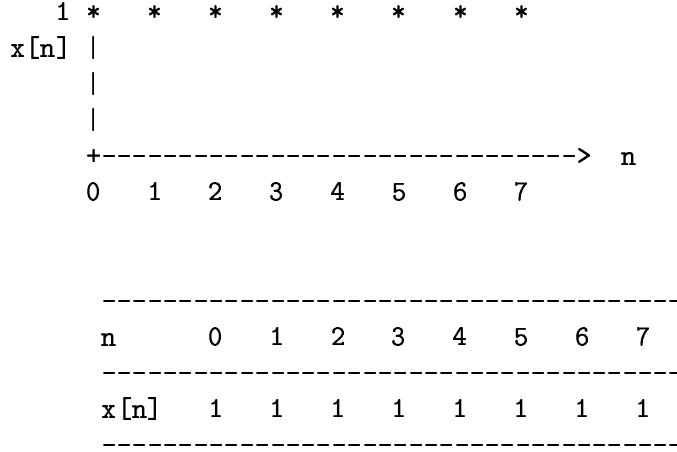
$$X_a[u] = |X[u]| = |X_c[u] + jX_s[u]| = \sqrt{(X_c * X_c + X_s * X_s)}$$

Thus, the amplitude spectrum of the impulse is simply:

u	0	1	2	3	4	5	6	7
$X[u]$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$

I.e. an ‘impulse’ has equal amounts of all frequencies. So, if you fed impulses into your amplifier, the ‘graphic-spectrum-analyser’ would show an equal reading on all bands.

Ex. 3.6-6 Work out the DFT, amplitude spectrum of a DC signal, i.e. $\{1, 1, 1, 1, \dots\}$. A pattern should emerge for $u = 1, 2, 3, \dots$.



Hint: this sequence correlates exactly with the constant sequence $[\cos(0), \cos(0), \cos(0) \dots] = [1, 1, 1, \dots]$

and does not correlate with any other of the transform sequences, e.g. $\cos(\pi \cdot 1 \cdot i/N), i = 0, 1, \dots 7$ the most slowly varying cos():

$$\cos(\pi \cdot 2 \cdot i/N), i = 0, 1, \dots 7 \text{ the next cos()}]$$

Thus, work out $X_c[.]$:

$$X_c[u] = (1/N) \sum_{n=0}^{N-1} x[n] \cos(\pi u n/N)$$

$$X_c[0] = (1/N)[x[0] \cos(\pi \cdot 0 \cdot 0/N) + x[1] \cos(\pi \cdot 0 \cdot 1/N) + \text{etc...}]$$

$$= (1/N)[1.1 + 1.1 + \text{etc. all 1s}]$$

since $x[0] = 1$, and $x[1] = 1$, etc. 2, 3..7 and, $\cos(0) = 1$.

$$= (1/8)[1 + 1 \dots 8 \text{ of them}] = (1/8)[8] = 1$$

$$X_c[1] = (1/N)[x[0] \cos(\pi \cdot 1 \cdot 0/N) + x[1] \cos(\pi \cdot 1 \cdot 1/N) + \text{etc.}]$$

$= (1/N)$ (for every positive $\cos()$ we have a negative), i.e. they all cancel, $= (1/8)[0] = 0$.

Similarly $X_c[2] \dots X_c[7]$, all equal 0.

Similarly, the $X_s[.]$ – they are all 0 – following the argument for $X_c[1]$.

Thus:

u	0	1	2	3	4	5	6	7
X[u]	1	0	0	0	0	0	0	0

Thus, the DFT of the constant sequence [1,1,1,1...] is an ‘impulse’.

Note: we have already seen that the DFT of an ‘impulse’ is a constant sequence; i.e. the impulse and the constant sequence are *duals* of one another.

(b) What is the amplitude spectrum?

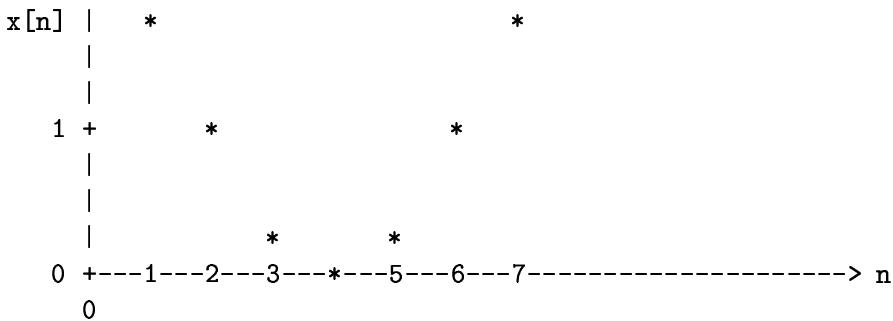
Answer:

u	0	1	2	3	4	5	6	7
X[u]	1	0	0	0	0	0	0	0

(c) What is the phase spectrum?

Ex. 3.6-7 What is the DFT of the following sequence. Look at Exs. 3.6-3, and 3.6-6, and think, before you calculate; the DFT is a linear transformation, i.e.

$$DFT(x[.] + y[.]) = DFT(x[.]) + DFT(y[.])$$



n	0	1	2	3	4	5	6	7
$x[n]$	2	1.707	1	-0.293	0	-0.293	1	1.707

Answer: Note all real, imaginary part = 0, for all u .

u	0	1	2	3	4	5	6	7
$X_r [u]$?	?	?	?	?	?	?	?
$X_i [u]$	0	0	0	0	0	0	0	0

Ex. 3.6-8 From what has been said above about ‘square waves’, give a qualitative estimate of the amplitude spectrum of a single cycle square-wave

$$x[n] = \{1, 1, 1, \dots \text{ up to } N/2 - 1; 0 \text{ at } N/2; -1, -1, -1 \text{ up to } N - 1\}$$

3.5.5 Implementation of the DFT

Let us write a program to perform the DFT. Recall the definition of the DFT:

Forward:

$$X[u] = (1/N) \sum_{n=0}^{N-1} x[n] \exp(-j\pi u n/N)$$

for $u = 0, 1 \dots N - 1$.

Inverse DFT:

$$x[n] = \sum_{n=0}^{N-1} X[u] \exp(+j\pi u n/N)$$

First, we have to solve the problem of how to represent complex numbers; this can be done simply by splitting $X[u]$ into the real, $X_r[u]$, and imaginary, $X_i[u]$, parts:

$$X[u] = X_r[u] + jX_i[u]$$

(for all the cases we deal with, $X_r = X_c$ (cosine component) $X_i = X_s$ (sine))

Actually, see below, there will be a negative sign on the sin term.

Now, using,

$$\exp(jB) = \cos(B) + j \sin(B) \quad \exp(-jB) = \cos(B) - j \sin(B)$$

and, in general, the sequence to be transformed may be complex (actually for a signal, $xi[n]$ will be zero, but we must be general especially for the inverse transform).

$$x[n] = xr[n] \pm jxi[n]$$

Therefore,

$$x[n] \exp(jB) = (xr[n] + jxi[n])(\cos(B) + j \sin(B))$$

$$= xr[n] \cos(B) - xi[n] \sin(B) + jxi[n] \cos(B) \pm xr[n] \sin(B)$$

the terms being respectively the real and imaginary parts. Hence,

Real part:

$$X_r[u] = (1/N) \sum_{n=0}^{N-1} \{xr[n].\cos(\pi.u.n/N) - xi[n].\sin(\pi.u.n/N)\}$$

for $u = 0, 1 \dots N - 1$.

If $x[.]$ is real, i.e. $xi[n] = 0$ for all n , this simplifies back to the same as the cosine part, $X_c[.]$

$$X_r[u] = (1/N) \sum_{n=0}^{N-1} \{xr[n] \cos(\pi.u.n/N)\}$$

for $u = 0, 1 \dots N - 1$.

Imaginary part:

$$X_i[u] = (1/N) \sum_{n=0}^{N-1} x_i[n] \cos(\pi.u.n/N) + x_r[n] \sin(\pi.u.n/N)$$

for $u = 0, 1 \dots N - 1$.

Again if $x[.]$ is real, this simplifies to:

$$X_i[u] = -(1/N) \sum_{n=0}^{N-1} x_r[n] \sin(\pi.u.n/N)$$

for $u = 0, 1 \dots N - 1$.

Both forward and inverse DFTs can be done by the same code; for inverse ($X[u] \rightarrow x[n]$), we simply replace the negative sign on the sin part, by positive, and we don't divide across by N .

But to simplify things, let us assume that we are transforming forward, and that the signal is real. Hence, the software, function **rfdft** (Real, Forward DFT):

```
void rfdft(
    float x[.], /*- real input -/
    float xro[.], float xio[.], /*- real and imag. outputs -/
    int N) /*- length -/
{
    float pi = 3.1415926, tpi;
    int n, u;

    tpi=2.0*pi;

    for(u=0;u<N;u++){
        xro[u]=0.0;
        xio[u]=0.0;
        for(n=0;n<N;n++){

            /*-- X_r [u] = (1/N) # {xr[n].cos( PI .u.n/N)} --*/
            xro[u] = xro[u] + x[n]*cos(tpi*u*n/N);

            /*-- X_i [u] = - (1/N) # xr[n].sin( PI .u.n/N) --*/
            xio[u] = xio[u] - x[n]*sin(tpi*u*n/N);
        }
        xro[u]=xro[u]/N;
        xio[u]=xio[u]/N;
    }
}
```

}

Ex. 3.6-8 Implement `rfdft`, and compare its results with the DFT in DataLab.

Ex. 3.6-9 Compare the running time for `rdftr` with its FFT counterpart in DataLab, for $N = 512, 1024, 2048, 4096, 8192$. (See section 3.7).

3.6 Fast Fourier Transform

3.6.1 General

For N point sequences the DFT,

$$X[u] = (1/N) \sum_{n=0}^{N-1} x[n] \exp(-j\pi u n / N)$$

will require

$$N \times (\text{multiply} + \text{add})$$

for each u , see section 3.6.5. That is, for $u = 1, 2..N - 1$, a total of N^2 operations.

A very clever algorithm called the Fast Fourier transform (FFT) allows $X(u), u = 0, 1..N - 1$ to be computed in $N \log_2(N)$ operations, but *only* for N = powers of 2, i.e. $N = 2, 4, 8, 16 \dots 1024, 2048 \dots$

Note: Otherwise the FFT is identical to the DFT.

Ex. 3.7-1 Consider the number of operations for an $N = 1024$ DFT.

$$N^2 = 1024^2 = 1 \text{ Million,}$$

Say a (complex add + multiply) takes 100 microsecs on a PC, then to a first approximation, the DFT will take $1 \text{ M} \times 100 \text{ microsec} = 100$ seconds.

If we use the FFT, the number of operations is $N \log_2(N) = 1024 \cdot 10 = 10,000$ approx.

So, to a first approximation, the FFT will take

$$10,000 \times 100 \text{ microsec} = 1 \text{ sec.}$$

If $N = 16384$ the saving is $15,000 \times 100$ microsec (1.5 secs) versus $270,000,000 \times 100$ microsec (27,000 secs), i.e. 1.5 secs versus 7.5 hours.

Before the FFT computation of the DFT was practically impossible. The FFT was one of the inventions that really opened up the territory of digital signal processing.

3.6.2 Software Implementation

[This may be skipped by those who have no need to apply an FFT; the reason this section is included is that, in the past, some students have undertaken final year projects that needed the FFT.]

The following is the ‘prototype’ of a C function that computes the DFT – using the FFT algorithm; if you are interested, see Press et al (1992; *Numerical Recipes in C*), or any book on image processing or signal processing giving some description of FFT algorithms.

```
void four1(float data[.],int N,int isign)
```

The arguments are:

`data[.]`: is an array containing the input data; because we must accomodate complex data, the FFT expects both real and imaginary parts; these are interleaved (and, in addition, the arrays are ‘1’ offset, i.e. start at index 1):

```
xre0 -> data[1], xim0 -> data[2], ... i.e.
```

```
xre0, xim0, xre1, xim1, ..., xreN-1, ximN-1;
```

Likewise, the output (transformed) data are interleaved:

```
Xre0, Xim0, Xre1, Xim1, ..., XreN-1, XimN-1; or using our earlier notation:
```

```
Xc0, Xs0, Xc1, Xs1, ..., XcN-1, XsN-1;
```

`N`: is the length of the array – which must be a power of 2 (2, 4, 8, 16, 32, 1024, ... etc.)

`isign`: defines whether the transform is Inverse (+1) or Forward (-1); all this does is set the sign in the `exp()` to

select the appropriate equation:

$$X[u] = (1/N) \sum_{n=0}^{N-1} x[n] \exp(-j\pi u n / N)$$

$$x[n] = \sum_{u=0}^{N-1} X[u] \exp(+j\pi u n / N)$$

A typical calling program:

```

/* get data from image store */
d=0; /*dimension 0 - assume there is only one */
r=0; /*row 0 - assume there is only one row */

for(j=1,c=c1;c<=ch;c++,j+=2){
    IMget(&val,d,r,c,ims);
    data[j]=val;
    data[j+1]=0.0; /*zero complex part*/
}
/* now do fft */

four1(data,len,-1);/* we use -1 for forward*/

/*
Now extract the DFT re. and imag. parts. and, if
required, compute the amplitude and phase spectrum
- recall these equations:

```

$$X_a(w) = |X[u]| = |X_c[u] + jX_s[u]| = \sqrt{(X_c * X_c + X_s * X_s)}$$

$X_a[u]$ is called the *amplitude* spectrum;
 $X_a * X_a$ is called the *power* spectrum.
 $X_p[u] = \arg(X[u]) = \arctan(X_s[u]/X_c(i))$ = *phase* spectrum.

```

*/
/* now extract re and cmplx parts*/

for(j=1,c=c1;c<=ch;c++,j+=2){
    re=data[j]; /* Real and Imaginary are interleaved*/

```

```

im=data[j+1];
if(ftrri){
    IMput(&re,d,r,c,imd0);
    IMput(&im,d,r,c,imd1);
}
else if(ftrap){ /*amplitude and phase reqd. */
    amp=sqrt(re*re+im*im);
    pha=0.0;
    if(fabs(re)>100.0*FLT_EPSILON)pha=atan2(im,re);
    IMput(&amp,d,r,c,imd0);
    IMput(&pha,d,r,c,imd1);
}
else if(ftra){ /*amplitude only reqd.*/
    amp=sqrt(re*re+im*im);
    IMput(&amp,d,r,c,imd0);
}
}

```

Frequencies associated with output data.

Positive frequencies:

```

data[1] = Xc [0], cosine(0) - DC term
data[2] = Xs[0], sine(1) term; always 0 for real input.
data[3] = Xc [1], cosine(1) term; freq 1 => 1. df
data[4] = Xs[1], sine(1) term
data[5] = Xc [2], cosine(2) term freq 2 => 2. df
data[6] = Xs[2], sine(2) term
...
data[N+1] = Xc [N/2], cos(N/2) term      => (N/2). df
data[N+2] = Xs[N/2], sin(N/2) term

```

Negative Frequencies.

```

data[N+3] = Xc [N/2 +1], cos(-[N/2+1]) term
data[N+4] = Xs[N/2 +1], sin(-[N/2+1]) term
...
data[2N] = Xc [N-1]      cos(-1) term
data[2N+1]= Xs[N-1]      sin(-1) term.

```

If you look at any amplitude or power spectrum, you will see that the

negative frequencies contain exactly the same information as the positive, i.e. the spectrum is symmetrical about $(N/2).d_f$ term.

Recall section 3.6.4, where we defined $d_f = f_s/N$ thus, the highest frequency in the DFT/FFT:

$$(N/2).f_s/N = f_s/2$$

which is reassuring, since the whole of sampled data theory is based on the assumption that we cannot represent frequencies higher than sampling frequency/2.

The larger the N, the finer is our discrimination of frequency. If $N \rightarrow \infty$, then we have a continuous spectrum, i.e. the continuous Fourier transform.

Above $N/2$ we have no additional information; these terms correspond to the ‘negative’ frequencies of the continuous transform.

Ex. 3.7-1 If, in the program above, the sampling frequency is 10,000 Hz, work out d_f and, hence, write the few lines of program that compute the amplitude of frequencies (get as close as possible): 0 Hz, 50 Hz, 100 Hz, 3000 Hz, 5000 Hz, 6000 Hz (be careful!).

Ex. 3.7-2 If we have data from a CD, and an FFT of length 32768, write the few lines of program that compute the average amplitude in five bands of frequencies between 0 and 15000 Hz. What is the highest frequency available?

3.7 Convolution

3.7.1 General

A great many signal processing operations can be defined in terms of convolution. Here we will deal with discrete convolution, using sums. There is an equivalent continuous convolution which is defined in section 3.8.5 below.

The discrete convolution of $x[.]$, and $h[.]$ is given by,

$$y[n] = x[n] \circ h[n] = \sum_{-\infty}^{+\infty} x[n-m] h[m]$$

If we agree that $h[m]$ is zero outside the range $[0..N-1]$, we have,

$$y[n] = x[n] \circ h[n] = \sum_{m=0}^{N-1} x[n-m] h[m]$$

Convolution is commutative, so that,

$$x[.] \circ h[.] = h[.] \circ x[.]$$

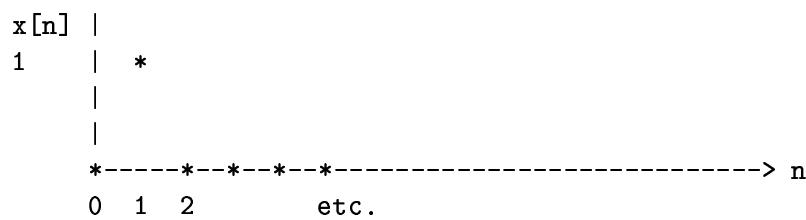
i.e.

$$y[n] = x[n] \circ h[n] = \sum_{-\infty}^{+\infty} x[m] h[n-m]$$

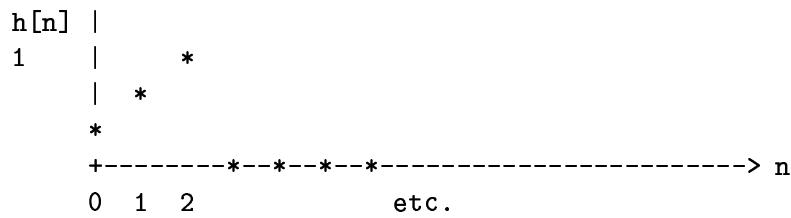
You will find these two alternative equations for $h[n]$ used interchangably in the literature.

Ex. 3.8-1 Convolve the sequence $a_x[.] = \{0, 1, 0, 0, \dots\}$ with the sequence $h[.] = \{\frac{1}{3}, \frac{2}{3}, 1, 0, 0, \dots\}$

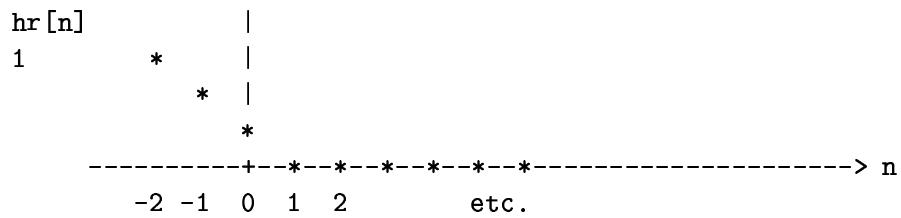
These are shown in the following figure:



(a) $x[.]$ an 'impulse'



(b) $h[.]$ a 'half-triangle'



(c) $h[.]$ reversed - flipped about the $n = 0$ axis

Sequences for convolution – apply the following equation:

$$y[n] = x[n] \circ h[n] = \sum_{m=0}^{N-1} x[n-m].h[m]$$

$$\begin{array}{llll} y[n=0]: & m=0 & x[n-m].h[m] = x[0-0].h[0] = 0.1/3 & =0 \\ & m=1 & x[0-1].h[1] = 0.2/3 & =0 \\ & m=2 & x[0-2].h[2] = 0.1 & =0 \end{array}$$

$$y[0] = 0$$

=====

$$\begin{array}{llll} y[n=1]: & m=0 & x[n-m].h[m] = x[1-0].h[0] = 1.1/3 & =1/3 \\ & m=1 & x[1-1].h[1] = 0.2/3 & =0 \\ & m=2 & x[1-2].h[2] = 0.1 & =0 \end{array}$$

$$y[1] = 1/3$$

=====

$$\begin{array}{llll} y[n=2]: & m=0 & x[n-m].h[m] = x[2-0].h[0] = 0.1/3 & =0 \\ & m=1 & x[2-1].h[1] = 1.2/3 & =2/3 \\ & m=2 & x[2-2].h[2] = 0.1 & =0 \end{array}$$

$$y[2] = 2/3$$

=====

$$\begin{array}{llll} y[n=3]: & m=0 & x[n-m].h[m] = x[3-0].h[0] = 0.1/3 & =0 \\ & m=1 & x[3-1].h[1] = 0.2/3 & =0 \\ & m=2 & x[3-2].h[2] = 1.1 & =1 \end{array}$$

$$y[2] = 1$$

=====

If you continue, you will find that the remainder of y s are zero.

Thus convolution can be done mechanically as follows:

- reverse ('flip') the convolution sequence $h[n]$ about $n = 0$, i.e. $\{\frac{1}{3}, \frac{2}{3}, 1\}$ becomes $\{1, \frac{2}{3}, \frac{1}{3}\}$, call this $hr[.]$, see above figure (c).
- For each point, n , in the input sequence ($x[n]$):

- overlay $hr[.]$ on $x[.]$, with $hr[.]$'s rightmost point at n (i.e. $hr[.]$ slides along by one for each iteration),
- multiply the corresponding values, $hr[.]x[.]$,
- sum the products
- the sum is the convolved result at $[n]$

Summary: ‘flip’,

sum of products, slide right,

sum of products, slide right,

....

Ex. 3.8-2 Show that is the convolution of the ‘rectangular’ sequence 1,1,1,0,0,0,0,
 ... with itself is a ‘triangular’ sequence; here we do it out mechanically:

$$\begin{array}{c}
 \{1,1,1,0,0,0,0\ldots\} \\
 \{1,1,1,0,0,0,0\ldots\} \quad (\text{flip}) \\
 \quad \quad \quad 1
 \end{array}$$

$$\begin{array}{c}
 \{1,1,1,0,0,0,0\ldots\} \\
 \{1,1,1,0,0,0,0\ldots\} \quad (\text{slide, multiply, add}) \\
 \quad \quad \quad 2
 \end{array}$$

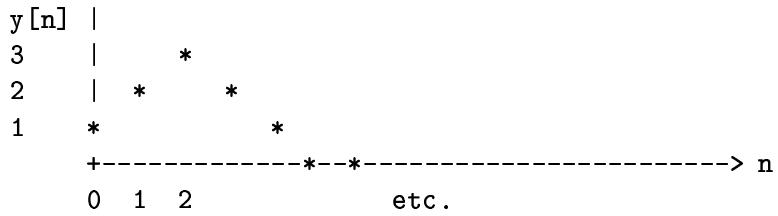
$$\begin{array}{c}
 \{1,1,1,0,0,0,0\ldots\} \\
 \{1,1,1,0,0,0,0\ldots\} \quad (\text{slide, multiply, add}) \\
 \quad \quad \quad 3
 \end{array}$$

$$\begin{array}{c}
 \{1,1,1,0,0,0,0\ldots\} \\
 \{1,1,1,0,0,0,0\ldots\} \quad (\text{slide, multiply, add}) \\
 \quad \quad \quad 2
 \end{array}$$

$$\begin{array}{c}
 \{1,1,1,0,0,0,0\ldots\} \\
 \{1,1,1,0,0,0,0\ldots\} \\
 \quad \quad \quad 1
 \end{array}$$

$$\begin{array}{c}
 \{1,1,1,0,0,0,0\ldots\} \\
 \{1,1,1,0,0,0,0\ldots\} \\
 \quad \quad \quad 0
 \end{array}$$

Thus, output is $y[.] = \{1,2,3,2,1,0,0,0\ldots\}$



3.7.2 Impulse Response

In digital signal processing, a sequence consisting of a single ‘1’ is called an ‘impulse’ sequence, see Ex. 3.8-1.

In the convolution operation, $h[.]$ is called the *impulse response*. That is, if we convolve an impulse sequence with $h[.]$ we get an output identical to $h[.]$.

Likewise, see Ex. 3.9-3, if we convolve any sequence, $x[.]$, with an $h[.]$ that is an impulse, we get an output identical to $x[.]$. This is easy to verify – replace $h[.]$ in Ex. 3.8-2 with [1,0,0, …] and work it out.

3.7.3 Linear Systems

Let $y_1[n]$ be the result of passing $x_1[n]$ through a ‘system’, and $y_2[.]$ the result of $x_2[.]$ passing through the same system. Then if the system is *linear*, the result of passing

$$x[.] = x_1[.] + x_2[.]$$

through the system is

$$y[.] = y_1[.] + y_2[.]$$

That is, we can add before or after the system.

The term system is very general. We mean anything which applies processing to a sequence.

In this chapter we deal almost exclusively with linear systems (system = filter or other similar processing); in Chapter 4 we will deal with some operations that are not linear (e.g. median filtering).

Any filter or operation that can be applied by a straight convolution (one-dimensional, or two-dimensional) is sure to be linear; but if you do anything like squaring, or taking absolute values, it becomes non-linear.

3.7.4 Some Interpretations of Convolution

1. Weighted sum: the output is a weighted sum of past inputs; i.e. weighted according to the impulse response.
2. Delayed impulses: any sequence is just a sum of delayed and weighted (i.e. have values other than 1) impulse sequences. Thus the convolution is just a weighted sum of delayed impulse responses. This is a consequence of the linearity of convolution. Linear systems can *always* be completely defined by their impulse response.
3. Tapped Delay Line: a method of applying a weighted sum; of interest only to engineers.

Ex. 3.8-3 What is the result of convolving *any* sequence, $x[.]$, with $h[n] = \{1, 0, 0, 0, \dots\}$, i.e. $h[.]$ is an impulse.

It may help to recall that: $x[.] \circ h[.] = h[.] \circ x[.]$.

3.7.5 Convolution of Continuous Signals

For continuous signals convolution is given by:

$$y(t) = x(t) \circ h(t) = \int_{-\infty}^{+\infty} x(t_1)h(t - t_1)dt_1$$

3.7.6 Two-Dimensional Convolution

Convolution can be extended to two-dimensions in a straightforward manner, if $f[r, c]$ is the input image, $h[r, c]$ the convolution kernel (or convolution mask), and $g[r, c]$ the output image,

Recall that:

$$\begin{aligned} y[n] &= x[n] \circ h[n] = \sum_{m=0}^{N-1} x[n-m]h[m] \\ g[.] &= f[.] \circ h[.] \end{aligned}$$

$$g[r, c] = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} f[r-k, c-l]h[k, l]$$

As mentioned in the one-dimensional case, convolution is commutative, so that,

$$g[.] = h[.] \circ f[.]$$

i.e. $g[r, c]$ can equally be written

$$g[r, c] = \sum_{k=r-N+1}^r \sum_{l=c-M+1}^c f[k, l]h[r - k, c - l]$$

In section 4.11, we will encounter convolution again; there we will have $h[.]$ extending from

$k = -w$ to $+w$ instead of 0 to $N - 1$

and $l = -v$ to $+v$ instead of 0 to $M - 1$

It is only a matter of convention (cf. storage of $h[.][.]$ in an array that does not allow negative subscripts). Thus $g[.]$ can be further rewritten as:

$$g[r, c] = \sum_{k=r-w}^{r+w} \sum_{l=c-v}^{c+v} f[k, l]h[r - k, c - l]$$

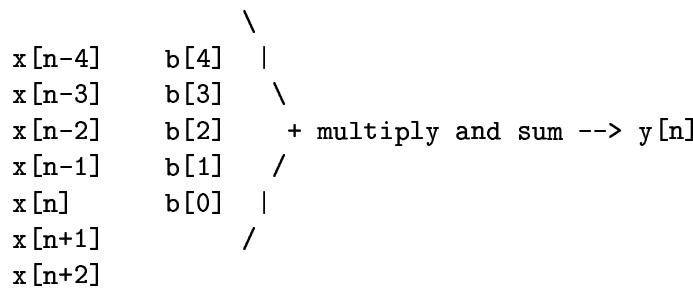
3.7.7 Digital Filters

Finite Impulse Response

Recall the discrete convolution equation, rewritten with minor changes of notation (simply: $N \rightarrow M$ and $h[.] \rightarrow b[.]$ – this is just to agree with common usage):

$$y[n] = x[n] \circ b[n] = \sum_{m=0}^{M-1} x[n - m]b[m]$$

This is, in fact, a digital filter. If we allow the index n to represent discrete time ($t = n.dt$), then this equation says that the output of the filter at n (e.g. now) is a weighted sum of the current input ($x[n]$) and the last $N - 1$ inputs ($x[n-1], x[n-2], \dots, x[n-N+1]$), i.e. see the following figure:



Another Way of Looking at Convolution

The above digital filter is called a *finite impulse response (FIR)* filter, because the impulse response is $b[0] \dots b[M - 1]$, which is clearly finite in duration; or non-recursive – for reasons which will become evident soon. If we were doing running averages over the last five samples, this would be

$$y[n] = \frac{1}{5}(x[n] + x[n - 1] + x[n - 2] + x[n - 3] + x[n - 4])$$

i.e. $b[i] = \frac{1}{5}$, $i = 0, \dots, 4$, and 0 otherwise.

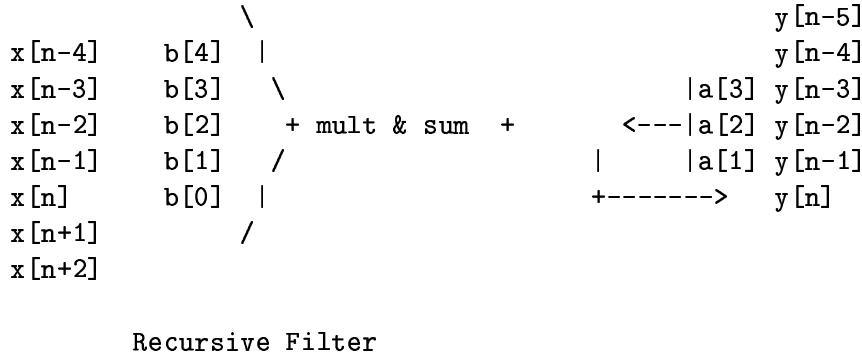
Recursive or Infinite Impulse Response (IIR)

The general form for a recursive filter is:

$$y[n] = \sum_{m=0}^{M-1} x[n-m]b[m] - \sum_{k=1}^K y[n-k]a[k]$$

That is, to get the current output, we do a weighted sum of the last M inputs (as before), plus, we also use the last K outputs. Thus, the filter is called *recursive*; also *infinite impulse response* – because of the recursion, the impulse response is, theoretically, infinite in duration.

For $M = 5$ and $K = 3$, we have the situation depicted in the following figure:



Ex. 3.8-4 Express the moving average

$$y[n] = \frac{1}{5}(x[n] + x[n - 1] + x[n - 2] + x[n - 3] + x[n - 4])$$

as a recursive filter.

This can be reworked recursively as:

$$y[n] = \frac{1}{5}(5y[n - 1] - x[n - 5] + x[n])$$

$$= y[n - 1] - \frac{1}{5}x[n - 5] + \frac{1}{5}x[n]$$

i.e. $K = 1, a[1] = 1$

$$N = 5, b[0] = \frac{1}{5}, b[1] = 0, b[2] = 0, b[3] = 0, b[4] = 0, b[5] = \frac{1}{5}.$$

3.8 Fourier Transforms and Convolution

Although the DFT is useful in its own right, one of its greatest uses is as a tool for efficient computation of convolutions.

Let

$$y[n] = x[n] \circ h[n] = \sum_{m=0}^{N-1} x[n-m]h[m]$$

Then,

$$Y[u] = NX[u]H[u]$$

where:

$X[.]$ is the DFT of $x[.]$

$H[.]$ is the DFT of $h[.]$

$Y[.]$ is the DFT of $y[.]$

N is the length of the DFT.

That is, convolution in the time or spatial domain is replaced by multiplication in the frequency domain (except for the multiplicative factor $1/N$).

Thus, we can do convolution as follows:

1. Take DFT of $x[.]$: $X[u] = \text{DFT}(x[.])$
2. Take DFT of $h[.]$: $H[u] = \text{DFT}(h[.])$
3. Multiply DFTs : $Y[u] = X[u].H[u], u = 1, \dots, N - 1$
4. Take InverseDFT of Y , multiplied by N : $y[n] = \text{IDFT}(NY[.])$

Why go to all this bother, when we have an easily applied formula for the convolution? Unless we implement the DFT using the FFT there is no reason. But with the FFT there is great saving to be made.

Consider $N = 1024, \log_2(1024) = 10(2^{10} = 1024)$. Let us convolve $h[.]$ and $x[.]$, each 1024-long sequences.

- (a) Calculations for straight convolution: $N * N$ (multiplies + add) = 1 Million
- (b) Using FFT:

Step 1. FFT of x : $N \log_2(N)$ complex operations

Step 2. FFT of h : ditto

Step 3. Multiply $X.H$: N complex multiplies.

Step 4. IFFT of Y : $N \log_2(N)$ complex ops.

Thus, a total of $(3N \log_2(N) + N)$ complex operations, assuming multiply takes nearly the same time as mult. + add., yielding = $3000.10 + 1000$ complex ops. = 31000.

Now if complex operations take about *twice* the time of ordinary, we have: 62000 time units for the FFT method, compared to 1000000 for straight convolution.

Thus, the FFT is 16 times faster.

With larger signals, and with images (as we shall see), the savings are even greater.

Ex. 3.9-1 The convolution of a rectangular sequence (see Ex. 3.8-2) $\{1, 1, 1, 1, 0, 0, 0, 0, \dots\}$ with itself is a triangular sequence (i.e. shaped like a triangle).

If the DFT of the rectangular sequence is a $\sin(x)/x$ function – see Gonzalez & Woods p.83, what is the DFT of a ‘triangle’.

Ex. 3.9-2 Verify that for $h[.] = [1, 0, 0, \dots]$ (an impulse), and for $x[.] =$ anything, the following equation makes good sense:

$$Y[u] = NX[u]H[u]$$

We will work with a sequence of length 8.

Recall Ex. 3.6-3, where we worked out the DFT of an impulse $h[.] = [1, 0, 0, 0, \dots]$:

u	0	1	2	3	4	5	6	7
H[u]	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

Therefore the right-hand side of the foregoing equation becomes:

$$= 8.X[u].\frac{1}{8} = X[u]$$

Therefore $Y[u] = X[u]$ $u = 0, 1, \dots, 7$ and if we take the Inverse DFT (IDFT):

$$\text{IDFT}(Y[.]) = \text{IDFT}(X[.]) = x[.]$$

i.e. we have shown, via this exercise that the convolution of $x[.]$ with an impulse is itself (unchanged). And, see section 3.8.2, we know this to be true. So the equation to be demonstrated has been shown to be valid in this case.

Here is an example, from DataLab, of convolution via DFT, see section 3.7.2 for a better description of the FFT function, ‘four1’.

```

for(i=1,c=c1;c<=ch;c++,i+=2){
    IMget(&val,d,r,c,ims1);
    data1[i]=val;
    data1[i+1]=0.0; /*zero complex part*/
    IMget(&val,d,r,c,ims2);
    data2[i]=val;
    data2[i+1]=0.0; /*zero complex part*/
}
/* now do fft */

four1(data1,len,-1);/* see Num. Rec. p 411
                      NB. use of -1 for forward*/
four1(data2,len,-1);

/* now extract re and cmplx parts and multiply*/

for(i=1,c=c1;c<=ch;c++,i+=2){
    re1=data1[i];
    im1=data1[i+1];
    re2=data2[i];
    im2=data2[i+1];
    /* --- multiply DFTs --- */
    re=re1*re2-im1*im2;
    im=im1*re2+im2*re1;
}
four1(dt1,len,1); /* -- inverse DFT --- */

for(i=1,c=c1;c<=ch;c++,i+=2){
    re=dt1[i];
    IMput(&re,d,r,c,imd);
}
}

```

3.9 The Discrete Fourier Transform as a Matrix Transformation

Why is the DFT called a ‘transformation’?

In section 3.2.1 we said that multiplying a vector by a matrix is a transformation, i.e.

$$y = Ax$$

with respective dimensions of these terms being $(m \times 1)$, $(m \times n)$, and $(n \times 1)$.

If y is $(N \times 1)$ and x is $(N \times 1)$, A must be $(N \times N)$,

$$y = Ax$$

with respective dimensions of these terms being $(N \times 1)$, $(N \times N)$, and $(N \times 1)$.

This can be expressed using summation as,

$$y[u] = \sum_{n=0}^{N-1} a_{un}x[n]$$

$u = 0, 1, \dots, N - 1$ where a_{un} is the element of A in row u , column n

$y[u]$ is the u th element of vector y , and

$x[n]$ is the n th element of vector x .

This equation is precisely the same form as one met earlier:

$$X[u] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \exp(-j\pi un/N)$$

$u = 0, 1, \dots, N - 1$ where a_{un} is now complex,

$$a_{un} = \frac{1}{N} \exp(-j\pi un/N)$$

But, as pointed out in sections 3.2.8 and 3.2.9, this makes no difference to the general principle.

We repeat an earlier interpretation of the DFT, as follows: row u of the transformation matrix is made up of

$$\cos(\pi un/N) - j \sin(\pi un/N), \quad n = 0, 1, \dots, N - 1$$

i.e. the $\cos()$ part is the cosine-wave with frequency u , and ditto the sine part, again with the same frequency. The values $y[u]$ depend on how well $x[.]$, the input signal, matches the sine and cosine waves.

Of course $y[.]$ is now complex, with,

- the ‘real’ part corresponding to $\cos(u\dots)$
- the imaginary part corresponding to $\sin(u\dots)$.

If you prefer, you can think of the transformation as being non-complex and having $2N$ rows, N rows corresponding to cosine terms, and N corresponding to sine terms. However, you can lose a lot of mathematical flexibility in this move.

As can be imagined, the Inverse DFT just applies the inverse matrix, $B = A^t$,

$$b_{nu} = \exp(j\pi nu/N)$$

Ex. 3.10-1 Check that the DFT and IDFT are indeed inverses of each other, i.e. find a few elements of

$C = AB$ where A and B are as defined earlier in this section.

Check that C is, indeed, the identity matrix (see section 3.2.6), i.e. it has 1s along the diagonal, zero elsewhere.

Ex. 3.10-2 Verify that the DFT matrix is orthogonal, i.e. its transpose is its inverse (see section 3.2.7).

3.10 Cross-Correlation

The cross-correlation of two sequences is given defined as:

$$c[k] = x[n](c)y[n] = \sum_{-\infty}^{+\infty} x[m]y[m+k]$$

What we obtain from the cross-correlation is how well $y[.]$, shifted right by k points, matches $x[.]$.

For example, if $x[.]$ is exactly the same shape as $y[.]$, only delayed by 10 points, then $c[.]$ above will exhibit a strong peak at $c[k = 10]$.

Cross-correlation is the same as convolution only we do not ‘flip’ the correlation signal, $y[.]$, prior to application; see section 3.8.1 for a discussion of convolution, (and the ‘flip’).

Thus cross-correlation can be done efficiently using the FFT – see section 3.9; we just reverse one of the signals, before applying the DFT.

As we shall see later, there are a great many applications of cross-correlation in image processing:

1. Template matching.

We want to find if (or where) an object of a certain shape is in an image. First, we create an image containing the object – a ‘template’ ($w[.]$, say). Then we cross-correlate the template with the image ($f[.]$), to produce the cross-correlation image $c[.]$. Those pixels in $c[.]$ that are greater than a threshold, correspond to centres of ‘matching’ objects in the image, i.e. objects that match the template.

Obviously, ‘template-matching’ has many applications in 1-dimensional signal processing, e.g. radar signal analysis, electrocardiogram analysis.

2. Image registration.

Two images, $f_1[\cdot]$, and $f_2[\cdot]$, represent the same scene, perhaps satellite images taken at different dates. But, they are not registered, i.e. pixel $[r, c]$ of $f_1[\cdot]$ does not correspond to pixel $[r, c]$ of $f_2[\cdot]$. Assume they are shifted in rows and columns, with respect to one another. Solution: cross-correlate the two images, find the point of maximum correlation, the indices of that point give the shifts. Clearly, also, there are many similar applications in 1-d signals.

Ex. 3.11-1 The *autocorrelation* is the cross-correlation of a signal with itself,

$$a_{xx}[k] = x[n](c)x[n] = \sum_{-\infty}^{+\infty} x[m]x[m + k]$$

Explain how the DFT can be used to compute this.

3.11 The Two-Dimensional Discrete Fourier Transform

The two-dimensional DFT can be written down simply by extending the one-dimensional equation.

However, if we left it at that, we would miss a lot, in fact the two-dimensional DFT can be applied by successive applications of the one-dimensional DFT – this is why we spent so much time on the one-dimensional version.

For an N row $\times M$ column input image $f[r, c], c = 0 \dots M - 1, r = 0 \dots N - 1$.

Two-Dimensional DFT:

$$F[u, v] = (1/MN) \sum_{r=0}^{N-1} \sum_{c=0}^{M-1} f[r, c] \exp[-j\pi(ur/(N+v)c/M)]$$

for $u = 0, 1 \dots N - 1$, and $v = 0, 1 \dots M - 1$.

Two-Dimensional Inverse DFT:

$$f[r, c] = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F[u, v] \exp[j\pi(ur/(N+v)c/M)]$$

for $r = 0, 1 \dots N - 1$, and $c = 0, 1 \dots M - 1$.

Interpretation of Two-dimensional DFT:

Remember that the u th component of the 1-dimensional version corresponds to how well the input sequence matches the sine-wave, $\sin(\pi un/N)$ – imaginary part, and $\cos(\pi un/N)$ – real part.

The u, v th component of the 2-dimensional DFT corresponds to how well the input image (2-dimensions) matches an image made up by multiplying a sine-wave, $\sin(\pi ur/N)$, along the columns, by a sine-wave, $\sin(\pi vc/M)$ along the columns; ditto the cosine part. Thus, recalling our interpretation of the one-dimensional DFT in section 3.6:

“The components near the beginning of the DFT correspond to the ‘slowly varying’ parts of the signal; the components near the middle, the ‘fast moving’ parts”.

The same is true for the two-dimensional, only now we have to think of relative ‘speed of variation’ in two-dimensions.

3.12 The Two-Dimensional DFT as a Separable Transformation

In section 3.10 we saw that the one-dimensional DFT could be written as a matrix transformation,

$$y = Ax$$

of respective dimensions $(N \times 1), (N \times N), (N \times 1)$.

Things are more complicated for the two-dimensional version, but only just. To write the two-dimensional DFT

$$F[u, v] = (1/MN) \sum_{r=0}^{N-1} \sum_{c=0}^{M-1} f[r, c] \exp[-j\pi(ur/(N+v)c/M)]$$

in matrix form, we have to observe that the transformation *kernel*,

$$\exp[-j\pi(ur/(N+v)c/M)]$$

can be separated into a ‘row part’, $\exp[-j\pi(ur/N)]$, and a ‘column part’, $\exp[-j\pi(vc/M)]$, by noting that,

$$\exp[-j\pi(ur/(N+v)c/M)] = \exp[-j\pi ur/N] \exp[-j\pi vc/M]$$

The consequence is that $F[u, v]$ can be written in matrix form as

$$F[u, v] = PfQ$$

of respective dimensions $(N \times M)$, $(N \times N)$, $(N \times M)$, and $(M \times M)$.

Matrix Q is just the $(M \times M)$ one-dimensional DFT matrix (kernel) that corresponds to DFTing each row. Matrix P is the $(N \times N)$ one-dimensional DFT matrix that corresponds to DFTing each column. For this reason the two-dimensional DFT can be said to be *separable*.

We will encounter other separable operators, i.e. those that can be separated into two sets of one-dimensional operations (one for rows, one for columns). Separable operators are particularly important because they generally require $O(N^3)$ operations, while non-separable operations require $O(N^4)$.

Note that, if $N = 1024$, $N^3 = 1000$ million, $N^4 = 1,000,000$ million (i.e. 1000 times more). If an $O(N^3)$ operation takes one minute, an $O(N^4)$ operation takes nearly 17 hours.

Of course, for the DFT, we can use the FFT and achieve even greater savings, i.e. $O(N^3) \rightarrow O(N^2 \log N)$. If $N = 1024$, $N^2 \log N = 10,000,000 = 10$ million.

Thus, 1 million million, reduces to 10 million!

The matrix form of the DFT looks a bit horrendous, but becomes more placid if we approach it one step at a time:

(a) DFT each column, individually, forgetting that it is part of an image; i.e. apply Q to f .

Result: $(N \times M)$ image f' with the columns now DFTs.

(b) Now DFT each row of image f' , again individually.

Result: Image $F[u, v]$ that is the two-dimensional DFT of $f[r, c]$.

The same considerations apply to the Inverse transform.

3.13 Other Transforms

3.13.1 General

A good many other transforms are used in image and signal processing. They all obey the same principles seen for the DFT in the one-dimensional case or in the two dimensional case; only the matrix kernels, A , in one-dimensional, or P and Q in 2-dimensional, change.

$$y = Ax$$

of respective dimensions $(N \times 1)$, $(N \times N)$ and $(N \times 1)$.

$$F[u, v] = PfQ$$

of respective dimensions $(N \times M)$, $(N \times M)$, $(N \times M)$ and $(M \times M)$.

As mentioned above, only transformations that are *separable* are of much interest; indeed, again for performance reasons, we are often interested only in transforms that have ‘fast’ implementations, like the FFT; i.e. $O(N \log N)$ operations instead of $O(N^2)$.

3.13.2 Discrete Cosine Transform

The one dimensional Discrete Cosine transform (DCT) is given by,

$$X_c(u) = (2a_u/N) \sum_{n=0}^{N-1} x[n] \cos \pi u(2n+1)/(2N)]$$

where $a_u = 1/\sqrt{2}$, for $u = 0$; $= 1$, for $u = 1, 2, \dots, N-1$.

That is we are using cosine functions, only, as a basis for the transform.

The Inverse DCT is given by

$$x[n] = \sum_{u=0}^{N-1} a_u X_c[u] \cos \pi u(2n+1)/(2N)]$$

The major interest in the DCT is for image compression, where, for a large class of images, it can be shown to offer better compression than the DFT.

The DCT forms the basis of a major image compression standard being promoted by a world authority, the Joint Photographic Experts Group (JPEG).

The DCT can be implemented using a double sized FFT, see Rosenfeld and Kak Vol 1. p. 155; the only significance of this statement is that we have a ‘fast’ algorithm for the DCT.

3.13.3 Walsh-Hadamard Transform

We return to the matrix version of transforms, as given by

$$y = Ax$$

of respective dimensions $(N \times 1)$, $(N \times N)$ and $(N \times 1)$.

We can write the matrix for the Walsh-Hadamard transform entirely in terms of +1s and -1s, e.g.

2x2 Hadamard:

$$\mathbf{A}(2 \times 2) = \begin{matrix} 1 & 1 \\ 1 & -1 \end{matrix}$$

(8 × 8) Walsh-Hadamard transform: a_{un} is given by the following table:

n	0	1	2	3	4	5	6	7
u								
0	+	+	+	+	+	+	+	+
1	+	-	+	-	+	-	+	-
2	+	+	-	-	+	+	-	-
3	+	-	-	+	+	-	-	+
4	+	+	+	+	-	-	-	-
5	+	-	+	-	-	+	-	+
6	+	+	-	-	-	-	+	+
7	+	-	-	+	-	+	+	-

Walsh-Hadamard Transform Kernel for N=8

Examination of this table indicates the nature of the Walsh-Hadamard transform – the basis waveforms are ‘square-wave’ in form, compared to the sine/cosine waves of the DFT. In Walsh-Hadamard transforms we speak of *sequency*, as a generalised frequency.

The Walsh-Hadamard transform has a ‘fast’ implementation; in addition, since the elements of the matrices are either +1, or -1, no multiplication is required.

As with the DCT, the major interest in the Walsh-Hadamard transform is image compression.

3.14 Applications of the Discrete Fourier Transform

3.14.1 Introduction

This section discusses the applications of the DFT. Since we have shown the strong relationship between the one-dimensional DFT and the two-dimensional version, and, having noted their analogous properties, we will sometimes discuss just the DFT and make no distinction between one-, and two-dimensions. Often we work in one-dimension for simplicity of notation; extension to two-dimensions is generally straightforward.

The major applications are:

1. Analysis of images and signals. That is analyse the frequency content, for example, compute the amplitude spectrum, or power spectrum.
2. Filtering.
3. As a tool for ‘fast’ computation of convolutions, and correlations.
4. In data compression.
5. In template matching and image registration (i.e. using correlation).
6. Various applications related to description of one-, and two-dimensional shapes; of particular importance, here, is the *shift invariant* property of the DFT power-spectrum.
7. Image restoration and deblurring.

Since Chapter 3 is already very long, we will try to be brief! You are strongly encouraged to glance at the recommended texts – these give many good examples, especially pictures, which we cannot reproduce here.

In what follows,

- $F[u]$ is the DFT of sequence $f[n]$,
- $F[u, v]$ is the 2-dimensional DFT of image $f[u, v]$,
- $F_c[u]$ refers to cosine ‘parts’,
- $F_s[u]$ refers to sine.

3.14.2 Frequency Analysis

The power spectrum given by

$$P[u] = F_c[u]^2 + F_s[u]^2$$

This gives an estimate of the relative ‘strength’, in the sequence, of frequency u .

In two-dimensions, $P[u, v]$ gives the relative strength of cross terms, e.g. a low (slowly varying) frequency along the rows, a high (rapidly varying) frequency along the columns.

3.14.3 Filtering

Filtering refers to frequency filtering. A low-pass filter is an operation which, when applied to a sequence, rejects all high-frequencies, and ‘passes’ all low, i.e. it ‘smoothes’ the input. A low-pass filter is defined by two parameters:

- the cut-off frequency,
- the rate of cutoff.

A high-pass filter does the opposite, it rejects low-frequencies, and passes high, i.e. ‘sharpens’ the input.

A band-pass filter passes intermediate frequencies, e.g. in a telephone system very low frequencies, below about 300 Hz, and high frequencies, above about 3000 Hz, are attenuated. That is, a band-pass filter with cut-off frequencies at 300 Hz and 3000 Hz is used.

Filters can be applied in the spatial domain (or time domain) or in the frequency domain.

Spatial domain filters are applied using convolution, see Chapter 4.

Frequency domain filters are applied by:

1. taking the DFT, $x[n] \rightarrow X[u]$,
2. multiplying the DFT array by an array of weights,

$$X'[u] = F[u]X[u], u = 0, 1, \dots, N - 1$$

3. applying the IDFT, $X'[u] \rightarrow y[n]$, the filtered output sequence.

For a low-pass filter: $F[u] = 1.0$, for $u <$ cutoff, and $F[u] = 0.0$, above cutoff.

For a high-pass filter: $F[u] = 0.0$, for $u <$ cutoff, and $F[u] = 1.0$, above. In practical cases more tapered weighting may be applied.

In the two-dimensional case, similarly, we define pass and reject regions in the (u, v) plane.

3.14.4 Fast Convolution

We have already indicated in section 3.9 that the FFT can be used to perform fast convolutions:

If

$$y[n] = x[n] \circ h[n] = \sum_{-\infty}^{+\infty} x[n-m]h[m]$$

then,

$$Y[u] = X[u]H[u]$$

That is, convolution in the time or spatial domain is replaced by multiplication in the frequency domain.

Thus, we can do convolution as follows:

1. Take DFT of $x[.]$: $X[u] = \text{DFT}(x[.])$
2. Take DFT of $h[.]$: $H[u] = \text{DFT}(h[.])$
3. Multiply DFTs : $Y[u] = X[u]H[u]$, $u = 1, \dots, N - 1$
4. Take InverseDFT of Y : $y[n] = \text{IDFT}(Y[.])$

For $N \times M$ images, each of the arrays above, $X[.], H[.], Y[.]$ are two-dimensional. The multiplication in step (3) becomes:

$$Y[u, v] = X[u, v]H[u, v]$$

for $u = 0, 1, 2 \dots N - 1, v = 0, 1, 2 \dots M - 1$.

3.14.5 Fast Correlation

If we observe that cross-correlation (sometimes called just correlation), defined as,

$$c[k] = x[n](c)y[n] = \sum_{-\infty}^{+\infty} x[m]y[m + k]$$

is ‘convolution’ with one of the sequences reversed, then the method given in section 3.15.4 can be used to provide ‘fast’ correlation, i.e. we only require the preliminary reversal,

$$x[i] = x[N - 1 - i], \quad i = 0, 1 \dots N - 1$$

Again, the extension to two-dimensions is straightforward.

Note: the reversal is not really necessary. Examine the DFT as given by

$$X[u] = (1/N) \sum_{n=0}^{N-1} x[n] \exp(-j\pi un/N)$$

Arguing qualitatively, reversing $x[n]$ is equivalent to setting $n = -n$, in $\exp()$. Now,

$$\exp(jB) = \cos B + j \sin B$$

$$\exp(-jB) = \cos B - j \sin B$$

i.e. $\exp(jB)$ is the complex conjugate of $\exp(-jB)$, so that the DFT of the reversed sequence is the complex conjugate of $X[u]$, written $X^*[u]$.

Thus, for correlation, we replace,

3. Multiply DFTs : $Y[u] = X[u]H[u]$, $u = 1, \dots N - 1$

with,

3. Multiply complex conjugate of DFT of $x[.]$, with DFT of $H[.]$: $Y[u] = X[u]H^*[u]$, $u = 1, \dots N - 1$

3.14.6 Data Compression

Generally speaking, observe that the ‘large’ components of an image are more important to human viewers, in the majority of cases, for the majority of uses. Observe, also, the ready acceptance of fairly blurred pictures in newspapers, or as family snaps – the usefulness of the picture is only slightly degraded by lack of sharpness.

The essence of data compression (image data compression, signal compression, or any other) is to reduce the amount of data to be transmitted, stored, etc.

Now recall that the DFT of an $N \times N$ image has, itself, $N \times N$ data elements, and that the original image can be reconstructed from the DFT by applying the IDFT. Consider, then, dispensing with some of the higher frequencies, say K of them. Now we have only $N - K$ DFT data elements to transmit. We can reconstruct using the IDFT, the only loss will be in a slight smoothing of the image (equivalent to low-pass filtering).

This is a very qualitative argument – but you can see what is happening. If we removed K rows, or K columns, or K arbitrary pixels from an image, the effect would be enormous, by removing K elements from the DFT, the effect is much less, or, at least, much more subtle.

The Discrete Cosine and Walsh-Hadamard transforms can be used in an analogous manner.

3.14.7 Deconvolution

Deconvolution is the process of restoring an image (or signal) that has been subjected to (unwanted) degradation by convolution. Usually this takes the form of blurring.

The convolution,

$$y[n] = x[n] \circ h[n] = \sum_{-\infty}^{+\infty} x[n-m]h[m]$$

is not, in general, reversible.

If we have $y[n]$, the result of convolving a wanted signal $x[n]$ with some ‘unwanted’ operator (e.g. blurring), and we don’t know the operator’s impulse response, $h[.]$, we have very little chance – but see below. It is as if we have the sum of ten numbers, and are asked to estimate what are the individuals that were used to make up the sum – impossible!

If, however, we know $h[.]$, or can estimate it, we can use the DFT to *deconvolve* $h[.]$ (invert the convolution) as follows:

1. If we observe that $Y[u] = X[u].H[u]$ (convolution \rightarrow multiplication in frequency domain),

2. therefore, using $h[n]$, or an estimate of it, compute using the DFT, $H[u]$.
3. Compute $X'[u] = Y[u]/H[u]$, $u = 0, 1 \dots N - 1$ (i.e. you can ‘cancel-out’ the effects of the convolution, in the frequency domain),
4. obtain the ‘restored’ $x[n]$, using the IDFT.

The extension to two-dimensions is again straightforward.

Ex. 3.15-1 An amateur photographer chances upon a bank robbery. As the robbers’ van speeds past him he takes a photograph of the side of the van. When printed the photograph turns out to be blurred – the photographer forgot to ‘pan’ with the moving van; thus the photograph has been ‘convolved’ with a smoothing function. The sign on the van cannot be read.

Assuming the the smoothing was along the horizontal (i.e. along the rows) suggest a digital image processing method for restoring the image. (The image is scanned at 100 pixels per inch, the smoothing appears to be about 1/10 inch wide, i.e. 10 pixels).

Suggest a restoration technique.

Ex. 3.15-2 If, in the previous question, the smoothing was *not* along the horizontal, how could we remedy the situation. (The processing is simpler if the smoothing is along one dimension, only).

Ex. 3.15-3 Suggest a way in which we might estimate the blurring function. Hint: if the van was black, and there was a shiny bright spot somewhere in the field of view of the photograph,...?

The Hubble Space Telescope was launched by the shuttle in 1989. Unfortunately, the optics had not been tested properly, and the resulting images were blurred. From a knowledge of the source defect, and, by examining actual blurred results, it was possible to estimate the blurring function, and, by deconvolution, to partially restore the images.

Deconvolution (1-D sound signals) is also very important in oil prospecting.

3.15 Questions on Chapter 3 – the Fourier Transform

TBD

Chapter 4

Image Enhancement

4.1 Introduction

Whenever an image passes through a ‘system’, i.e. gets sensed, and/or digitised, and/or compressed and decompressed, and/or transmitted, and/or displayed, the ‘quality’ of the image may be degraded in some way – for example, the addition of noise. Image enhancement is about the improvement of the ‘quality’ of the image – usually an attempt at returning it to some ‘original’ state.

Of course, quality is subjective – it depends on what the (human or machine) user/observer intends to use the image for. Indeed, the original image may be of low quality from a few points of view, but we are content with this, and all we want to do is highlight (enhance) some features of the image – i.e. improve the quality with respect to one particular criterion.

In general, the improvement of quality – no matter what the criterion for improvement – will usually be called *image enhancement*; and so to some extent this chapter contains a fairly mixed bag of techniques and processes.

In this chapter we will cover two broad categories of image enhancement: point (or pixel, or grey level) operations, and spatial operations (or neighbourhood operations).

1. Point operations

We apply some function to each pixel value, individually, to yield a new value; there is a net enhancement effect on the image viewed as a whole, even though the global relationship between pixel values is ignored. In this category, the enhancement is usually for the benefit of a human observer, and the enhancement is often connected with contrast enhancement (e.g. turning up the contrast on a TV monitor – see section 2.7 for a definition of contrast).

We differ from Gonzalez and Woods in calling these *point* operations – they (incorrectly, and in disagreement with most authors) call them

spatial operations.

2. Spatial operations

Here we apply a function to a pixel and its neighbours, to yield a new value for the pixel; the function takes into account the relationship between the value of the pixel and the values of its neighbours – thus *spatial* or *neighbourhood* operations. Broadly speaking, spatial operations come in two categories: smoothing, and sharpening.

Sections 4.3–4.10 will cover point operations, and from section 4.11 onwards we will be dealing with spatial operations.

In general this chapter will concentrate on the ‘enhancement’ operations, e.g. smoothing out noise, enhancement of edges. We will leave it to later chapters and lectures to apply further processing to the enhanced images, e.g. application of a threshold to an edge-enhanced image, to produce edge points, tracing a path of edge points, segmentation etc.

But first we must look a little more closely at noise.

4.2 Noise and Degradation

Figure 4.1 shows an image passing through a system. At any stage in the system ‘noise’ may be added. Here we are using ‘noise’ in a very general sense – i.e. any degradation. It could be the type of noise mentioned in Chapter 1 – i.e. random numbers added to the actual image numbers; it could be thumbprint on a photograph, or grain, or streaks on a negative, or any form of sensor noise, or as on a TV electrical noise, caused by, e.g., electromagnetic noise from a starter motor that gets onto the TV transmission signal.

In general, *noise* is data that we don’t want!

Additive noise: More often than not, noise is additive, i.e. the ‘noisy’ image is the *true*, noiseless, image with noise added; thus

$$f(r, c) = f_{\text{true}}(r, c) + n(r, c) \quad (4.1)$$

where $f_{\text{true}}(.,.)$ is the true, noiseless image $n(.,.)$ is a noise image, e.g. as generated by DataLab ‘ggn’.

Incidentally, not only is additive noise very common, but it is also the most easy analyzed and dealt with.

Reduction of additive noise is covered in Section 4.10.

Non-additive noise: Some of image *restoration* deals with removal of non-additive noise; examples are: noise ‘convolved’ into an image – blur is a good example; gain calibration errors on a CCD camera is an example of multiplicative noise. (Note – bias is additive).

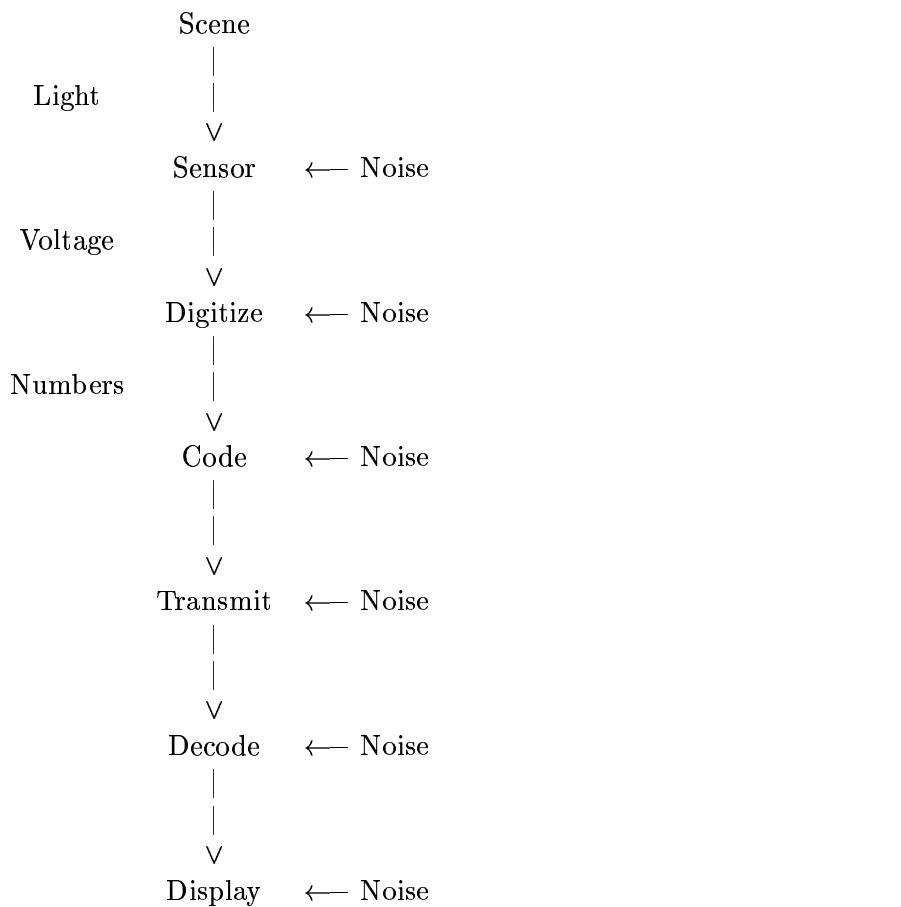


Figure 4.1: From top to bottom: image passing through a system.

If you think of an audio channel as conveying a sequence of data (albeit analogue), then hiss, crackle, or clicks are all undesirable data *added* to the ‘wanted’ data – the voice or music – i.e. noise. The noise obscures the wanted data. The noise that causes hiss on an audio channel shares many characteristics with the noise that causes ‘snow’ on a TV screen; and both are very similar to the image noise shown in Chapter 1.

It is not surprising, therefore, that many noise reduction processes in image processing have been borrowed from analogue (and, more recently, digital) signal processing.

4.3 Point Operations

4.3.1 Grey Level Mapping by Lookup Table

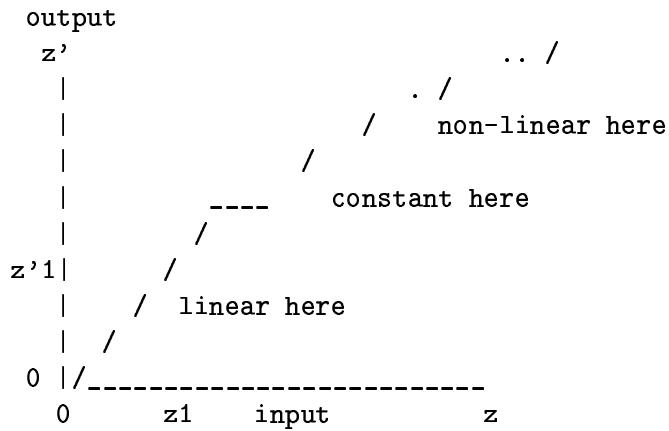
The simplest form of point operation. Suppose you have an image whose data is all squashed in the range 0 to 10; it won’t look like much when displayed on a 0–255 display; it will have low contrast and look grey and muddy.

A simple solution is to form and apply a lookup table as follows:

```
#define MAX 11
int lut[MAX]={0,25,50,...250};

#define NROW 64
#define NCOL 64
unsigned char f1[NROW][NCOL],f2[NROW][NCOL];
int rl,rh,cl,ch,r,l;
float val;int ival,ival2;

/* read in image...*/
.....
rl=0;rh=NROW-1;
cl=0;ch=NCOL-1;
for(r=rl;r<=rh;r++){
    for(c=cl;c<=ch;c++){
        ival=f1[r][c];
        if(ival<=0)ival2=lut[0];
        else if(ival>MAX)ival2=lut[MAX];
        else ival2=lut[ival];
        f2[r][c]=ival2;
    }
}
```



For any input pixel value z_1 , find the point on the horizontal (z_1) axis, then move vertically to the curve, then horizontally to get $z'_1 = t(z_1)$.

Figure 4.2: Non-linear contrast adjustment.

The example above is rather trivial, since multiplying by 25 would do the same. But, lookup tables are useful in many circumstances:

1. it may be desirable to leave the original image unaffected and send the data to display via a LUT,
2. in most hardware implemented image processing machines, there is a LUT between the memory and display; it may start off as $\{0, 1, 2, 3, \dots, 255\}$ i.e. ‘straight-through’, but it can be modified by user or program,
3. especially in colour, it may be useful to have a selection (palette) of LUTs, cf. changing the palette on an IBM PC,
4. any ‘point’ operation may be implemented by LUT,
5. they are easy to specify interactively – and you see the results immediately,
6. they can be used for application of non-linear contrast adjustments, see Figure 4.2.

4.3.2 Colour Lookup Tables

Suppose you wanted to do ‘pseudo-colour’ coding – just as in the radar rainfall maps in the TV weather forecast. E.g. code 0 as black, ... 5 as deep

blue ... 100 as red ... 255 as bright white, etc. Assume 16 levels for red, green, and blue (typical for a cheaper system – i.e. in total a 4096 colour palette; top of range would have 256, 256, 256).

The following gives the gist (the Red, Green, and Blue values can each be e.g. 0..63, the same as IBM PC with SuperVGA):

```

Red Green Blue
int lut[3][256]={ 0,    0,    0, /*0,0,0 = black*/
                  0,    0,   10, /*add some blue*/
                  0,    0,   20, /*and some more*/
                  ...
                  0,   10,   0, /*now green*/
                  0,   20,   0, /*and some more*/
                  ...
                  0,   63,   0, /*very bright green*/

                 15,   0,   0, /* red*/
                  ...
                 63,   0,   0, /*bright red */

                 10,   10,  10, /*dark grey*/
                 30,   30,  30, /*lighter grey*/
                 63,   63,  63 /* bright white*/
               }
```

Now, apply LUT:

```

for(r=r1;r<=rh;r++){
  for(c=c1;c<=ch;c++){
    ival=f1[r][c];
    for{b=0;b<=2;b++}{

      f2[b][r][c]=lut[b][ival]; /*f2 is the output
                                     colour image*/
    }
  }
}
```

Again, as noted in section 4.3, LUTs are:

- (1) easy to do in hardware,
- (2) easy to specify interactively.

Exercises: [see Chapters 1 and (especially) 2 for some discussion of colour].

Ex. 4.4-1 Describe, in English, what colour would appear for:

- (a) red=63, green=63, blue=0.
- (b) red=5, green=5, blue=5.
- (c) red=63, green=63, blue=63.
- (d) if red=30, green=30, blue=50 is light blue, what is red=50, green=30, blue=30 Hint: think additive (see Chapter 2).

Ex. 4.4-2 Suggest, in English, a colour coding scheme for coding magnitude (examples: temperature on a map, altitude on a map, rainfall, etc.).

4.3.3 Greyscale Transformation

This section is a slight generalization of section 4.3.

A lookup table is nothing more than a discrete function; again, the terms function and transformation are used interchangeably. Expressed as a function, the transformation from input image to output image is $z' = t(z)$ where z' is output pixel value, z the input value, i.e. $z = \{0 \dots z_G\}$, $z' = \{0 \dots z'_G\}$, replacing G of Chapter 1 with z_G , and $t(\cdot)$ is the transformation function.

Suppose, as in section 4.3, the input image does not occupy the full dynamic range, e.g., an “underexposed” photograph.

Suppose, $a \leq f_1(x, y) \leq b$, i.e. the range of $f_1()$ is $[a, b]$, and you want the range stretched to $z_0 \leq z' \leq z_G$, then the transformation is

$$z' = (z_G - z_0).(\frac{z - a}{b - a}) + z_0 = (z_G - z_0).z/(b - a) + (z_0.b - z_G.a)/(b - a) \quad (4.2)$$

The right hand side terms are respectively *scale* and *shift* terms.

Equation 4.2 is a simple linear transformation which stretches and shifts the original greyscale $[a, b]$ to $[z_0, z_G]$. If you are familiar with graphics transformations you will recognise a one-dimensional shift and scale.

Now suppose that *most* of the input pixels are in the range $[a, b]$, and those outside are only due to noise, or some uninteresting artifact. Then we can use $z' = (z_G - z_0).(\frac{z - a}{b - a}) + z_0$, for $a \leq z \leq b$, $z' = z_0$ for $z < a$, and finally $z' = z_G$ for $z > b$.

This stretches $[a, b]$ as before, but *compresses* intervals $[z_0, a]$ and $[b, z_G]$ into single points, z_0 , and z_G , respectively.

In addition, there is nothing to stop you applying different linear transformations to many regions of the greyscale: see exercises.

Ex. 4.5-1 Let $a = 0$, $b = 10$, $z_0 = 0$, and $z_G = 255$ – we want to stretch the input range $[0, 10]$ to an output range of $[0, 255]$. Thus equation 4.2 becomes:

$$z' = (255 - 0).(z - 0)/(10 - 0) + 0 = 25.5 * z \quad (4.3)$$

I.e. we need no shift, and a scaling by a factor of 25.5.

Ex. 4.5-2 Let $a = 15$, $b = 35$, $z_0 = 0$, and $z_G = 255$ – we want to stretch the input range $[15, 35]$ to an output range of $[0, 255]$.

(a) Work out the transformation,

(b) apply it to the input values 15, 25, 35; does it check out? – it should be very clear to you what each of these values should map to.

Ex. 4.5-3 Let $a = 0$, $b = 1023$, $z_0 = 0$, and $z_G = 255$.

(a) Work out the transformation,

(b) apply it to the input values 0, 255, 511, 1023.

If an input image has been subjected to a known greyscale transformation, $t()$, then applying the inverse, $t^{-1}()$, can remove the effect.

E.g. in an X-ray image the image values are known to follow the rule:

$$f(x, y) = I_0 \cdot \exp(-c(x, y))$$

where I_0 = intensity of X-ray beam $c(x, y)$ is a function of the density and thickness of the X-rayed material.

The inverse of $\exp()$ is $\ln()$ (natural log, log base e), so that if you apply a $\ln()$ transformation to $f(x, y)$ you get an image which is proportional to $c(x, y)$ – which may be more useful in some cases.

The logarithm is also generally useful where you want to stretch out the lower pixel values, while compressing the upper.

Ex. 4.5-4 Plot a graphical representation of a \log_2 (log base 2) transformation for $[0..255]$. Verify that the previous statement is true – stretch ... compress.

4.3.4 Thresholding and Slicing

Suppose you have a monochrome satellite image of the earth. Water is fairly dark – all water pixels less than bw (say), and the land brighter ($> bw$). Then the thresholding function $z' = 0$ for $z < b_w$ and $z' = 1$ for $z \geq b_w$ will clearly segregate land and water.

The requirement to reduce a quantity to binary (true – false, yes – no, X – notX) is common in image processing, as in many computer applications; we shall come across this again – especially in image segmentation.

Thresholding generalises to density slicing. Suppose in the same satellite image water values are $[a_w, b_w]$, urban landuse $[a_u, b_u]$, grassland $[a_g, b_g]$, forest $[a_f, b_f]$, and none of the ranges overlap. The rule given in the following equations forms the basis of a land-use mapping algorithm:

$$z' = 1 \text{ for } a_w \leq z \leq b_w \text{ (class 1 = water)}$$

$$z' = 2 \text{ for } a_u \leq z \leq b_u \text{ (class 2 = urban)}$$

...

$$z' = 4 \text{ for } a_f \leq z \leq b_f \text{ (class 4 = forest)}$$

$$z' = 0 \text{ otherwise.}$$

Further, if the pixels are 20 meters \times 20 meters, then counting all the 4s in the output image (=nf, say) and multiplying by 0.04, will give the number of hectares of forestry in the image. [Why 0.04 ?]

Also, you can apply a colour lookup table (see section 4.4) to produce a colour map of the result.

4.3.5 Contrast Enhancement Based on Statistics

Sometimes we want a non-subjective method of equalising the average grey level and contrast of a number of images; the method given in this section and the next (histogram modification) meet this aim.

Suppose we have an input image whose mean grey level is m and standard deviation is s . We desire m' and s' . Often the notation μ (mu) and σ (sigma) are used for m and s . This is simply shifting and scaling again (see section 4.5) and the transformation given in equation 4.4 produces an output image with the desired mean and standard deviation:

$$z' = (z - m).s'/s + m' \quad (4.4)$$

The term $(z - m)$ on the right hand side is a shift to mean = 0. The next term, s'/s , is a rescaling. The final term, m' is a shift to mean = m' .

This can be compared with

$$z' = (z_G - z_0).(z - a)/(b - a) + z_0$$

Some definitions:

Mean, $\mu = \sum_{r=0\dots N-1, c=0\dots M-1} f(r, c)/MN$ i.e. the average grey level for whole image

Variance, $v = \sigma^2 = \sum_{r=0\dots N-1, c=0\dots M-1} (f(r, c) - \mu)^2/MN$ i.e. the average squared deviation from the mean – a measure of the contrast present in the image; small v – low contrast.

Standard deviation, $\sigma = \sqrt{v}$

Ex. 4.7-1 An input image has mean $m = 5$, standard deviation $s = 3$. Derive the contrast stretching transformation to produce output image with $m' = 128$, $s' = 80$.

Ex. 4.7-2 Input image:

2	2	2	2	2
4	4	4	4	4
6	6	6	6	6
8	8	8	8	8
10	10	10	10	10

- (a) What is mean, m ? (b) What is standard deviation, s ? (c) Derive the transformation to yield mean, $m' = 128$, standard deviation $s' = 80$. (d) Then compute the output image. (e) Will it fit into $[0, 255]$? If not, ‘clip’ values < 0 to 0, and values > 255 to 255.

4.3.6 Histogram Modification

First some definitions.

A *histogram* is a table, with an entry for each possible pixel value, giving the number of pixels which have that value; usual notation:

$$h(z), z = [z_0 \dots z_G]$$

$$\sum_{z=z_0 \dots z_G} h(z) = M.N$$

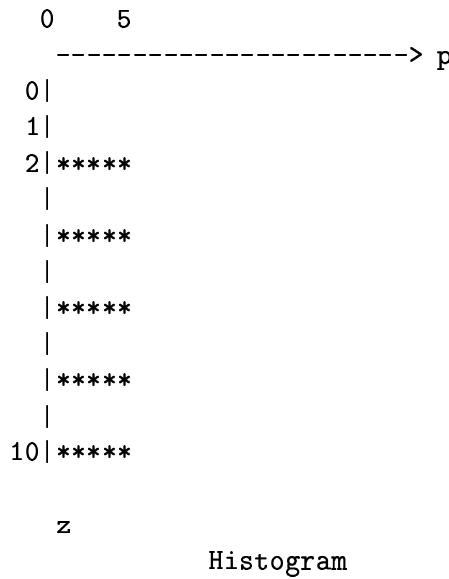
for an $M \times N$ image.

Ex. 4.8-1 Input image

2	2	2	2	2
4	4	4	4	4
6	6	6	6	6
8	8	8	8	8
10	10	10	10	10

Thus: $h(0) = 0$, $h(1) = 0$, $h(2) = 5$, $h(4) = 5, \dots$ $h(10) = 5$, all others = 0.

A histogram is usually shown as a graph as follows (graph on its side):



A *Probability Density Function (pdf)* is the limit case of a histogram when we assume a lot of data, and a totalled area equal to 1 (unity).

Note: Whereas we deal mostly with discrete numbers, e.g. the integers between 0 and 255, probability density function normally refers to the probability of *real* numbers; strictly we should use the term probability function for discrete values. However, it is convenient to use pdf for either.

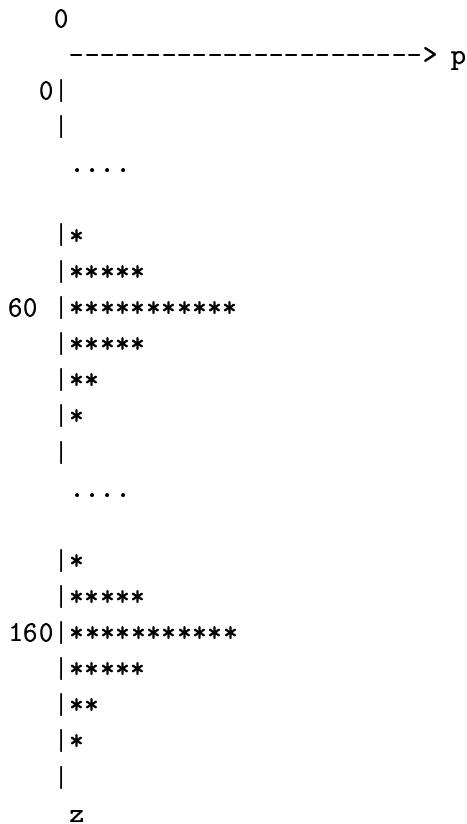
If you divide $h(z)$ by $M.N$ for all $z = [z_0, z_G]$, you get something which gives an estimate of the probability distribution (or probability density) of the grey levels in the image: $p(z) = h(z)/(M.N)$, and $\sum_{z=z_0..z_G} p(z) = 1$ (from definition of probability)

The probability density $p(z_i)$ lets you know, for a stream of pixels coming from an image, the probability that the pixel value will be z_i . In the example given above,

$$p(0) = 0, \quad p(1) = 0, \quad p(2) = 1/5, \quad \dots p(10) = 1/5, \quad p(11) = 0\dots$$

One of the motivations behind the invention of probability theory was that of gambling. If you knew that all images followed the previous density, then it would be foolish to bet on the value 12, or 0 (say), coming up.

Ex. 4.8-2 You will find that the noisy image shown in Chapter 1 has a histogram something like:



Histogram of noisy image.

Ex. 4.8-3 What is the histogram of the clean cross image in Chapter 1?

Ex. 4.8-4 A fair dice is thrown 2400 times. What, approximately, will be the histogram?

Answer: $h(1) = 2400/6 = 400$, $h(2) = 400$, ..., $h(6) = 400$.

Ex. 4.8-5 What is the probability density for any single number (not a 6 number ‘play’) on a lottery with numbers labelled 1 to 39? Assume fair!

Ex. 4.8-6 (a) What is the probability of the numbers (1,5,11,15, 22, 33) coming up in a six-number lottery – numbers labelled 1 to 39?

(b) Hence, at what stage does the lottery become a good bet? i.e. better than evens?

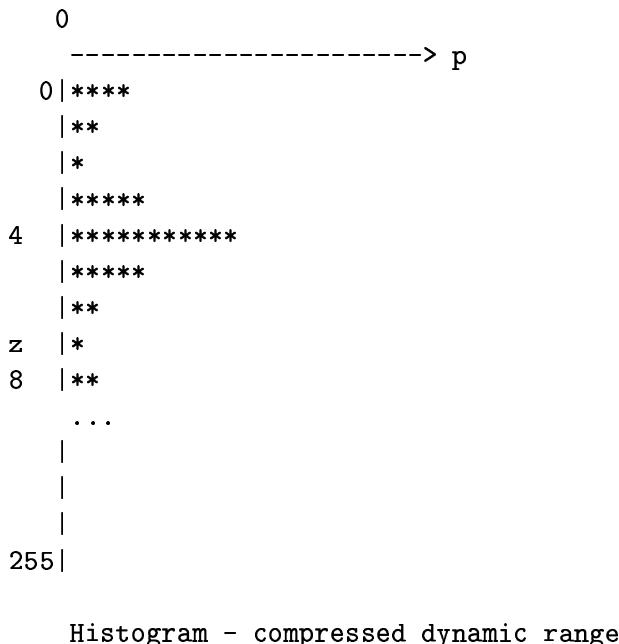
Answer: when the pool becomes better than about 1,000,000 pounds. Why? (Note: this was originally written before the so-called bonus number appeared!)

The *Cumulative Distribution Function* (cdf), $F(z)$, gives the accumulated probability from minus infinity (or z_0 in our case) up to z , i.e. $F(z') = \sum_{z=z_0 \dots z'} p(z)$. Thus, $F(z_G) = 1$.

Ex. 4.8-7 (a) Prove $F(z_G) = 1$. (b) Prove $F(z_0) = p(z_0)$.

Histogram Transformation: suppose you had an image with a histogram as shown in the figure below, i.e. with a severely compressed dynamic range. Shown on a 255 grey level display this would look ‘flat’ – all dull grey, ‘muddy’ – with no detail evident. You could use one of the contrast enhancement methods mentioned above or you could set up a lookup table based on the requirement to modify the histogram to a more suitable shape, $q(z)$, instead of $p(z)$.

In what follows, we will consider arbitrary $q(z)$, but often we will want *histogram equalization*, i.e. a flat histogram, or *uniform* probability distribution function. Histogram equalization usually gives a satisfactory contrast enhancement: $q(z) = 1/G$ for $z = z_0 \dots z_G$, where G = number of possible grey levels.



Although there are no values above 8 in the above figure, a similar state of affairs can exist if you have a few values all over the range 0 to 255, but with most of them clustered at one small sub-range. A linear contrast stretch

will not work in this latter case (cf. equation 4.2). Histogram equalization is what we need.

The treatment here differs from that in G&W (which is more mathematical); it follows Lim (Chap 8.1) closely. See also Rosenfeld and Kak (Chap. 6.2.4). You will also find contrary treatments in Boyle and Thomas, and Low; these work *only* for histogram equalization, whereas the method given below will work for *any* desired histogram, including *uniform* (equalization).

Notation: Input grey levels z ,
 Output grey levels w ,
 Transformation (lut) $w = T(z)$,
 Input probability distribution $p(z)$, cdf $P(z)$
 Desired (output) distribution $q(w)$, cdf $Q(w)$
 N_p = number of pixels in image.

Algorithm:

- (1) Estimate frequency of grey levels (histogram), $ph(z)$.
- (2) From $ph(z)$ compute $p(z) = ph(z)/N_p$.
- (3) Form cdfs $P(z)$, and $Q(w)$, from $p()$ and $q()$, respectively,

$$\text{i.e. } P(z') = \sum_{z=z_0}^{z'} p(z)$$

- (4) for each z in the input do:
 choose w such that $Q(w)$ is closest to $P(z)$,
 set $T(z) = w$;
- (5) Apply $T(z)$ to input image $f1[r,c]$: $f2[r,c] = T(f1[r,c])$

Exercise 4.8-8 Suppose an 8-level image, ($G = 8$); $M = 8$, $N = 16$ ($M.N = \text{no. of pixels} = 128$). Input histogram $ph()$, output $q()$ – see table:

Input and Output Histograms:

z	0	1	2	3	4	5	6	7
<hr/>								
$ph(z)$	1	7	21	35	35	21	7	1
$q(w)$	16	16	16	16	16	16	16	16

- (1) Form $P(z)$, and $Q(w)$, from $p()$ and $q()$:

Input and Output cdfs:

z	0	1	2	3	4	5	6	7
$P(z)$	1	8	29	64	99	120	127	128
$Q(w)$	16	32	48	64	80	96	112	128

(2) for each z in the input do:

(2.1) choose w such that $Q(w)$ is closest to $P(z)$, $T(z) = w$; (i.e. set up lookup table/function)

```

z=0 : must choose T(0) = 0  (closest)
z=1 : T(1) = 0
z=2 : T(2) = 1
z=3 : T(3) = 3
z=4 : T(4) = 5
z=5 : T(5) = 6
z=6 : T(6) = 7
z=7 : T(7) = 7

```

Thus, resulting histogram:

Input and Output Histograms:

z	0	1	2	3	4	5	6	7
$p(z)$	1	7	21	35	35	21	7	1
$q'(w)$	8	21	0	35	0	35	21	8

Notice that $q'(w)$ is only an approximation to $q(w)$. In general, for discrete levels, this will be the case.

Remark: We could make the equalization exact by modifying the algorithm to allocate the pixels at input grey level z_i , not just to one output level w_j , but put some in w_{j-1} , and some in w_{j+1} – to produce an exact $q(w)$ histogram. Rosenfeld and Kak, Chap 6.2, do this. To me it doesn't make much sense to map one pixel value 3 (say) to 1, another to 2, and maybe another to 4. Furthermore, the lookup table algorithm would have to be made considerably more complex – for very little gain.

Ex. 4.8-9 Suppose that a 64×64 , 8-level image has the greylevel distribution values of $p(z)$, the density calculated from the histogram, shown below. Calculate the transformation function, $T(z)$, to give histogram

equalization, using the algorithm given above. Then calculate $q'(w)$, the actual output histogram, and plot it.

Recommendation: Proceed as follows. Work out $q(w)$. The work out the cumulatives $P()$, and $Q()$; then apply the algorithm. Note: this example is taken from G&W 2nd ed., *Don't* copy their answer, it's wrong!

Histogram for Ex. 4.8-9:

z	$ph(z)$	$p(z) = ph(z)/M.N$	$q(w)$	$P(z)$	$Q(z)$	$T(z)$
0	790	0.19				
1	1023	0.25				
2	850	0.21				
3	656	0.16				
4	329	0.08				
5	245	0.06				
6	122	0.03				
7	81	0.02				

Ex. 4.8-10 Using the same input histogram as in Ex. 4.8-9, produce a transformation to yield the density shown, $q(w)$ in the following table. Again, be wary of the G&W answer for this.

Densities for ex. 4.8-10:

z	$ph(z)$	$p(z) = ph(z)/M.N$	$q(w)$	$P(z)$	$Q(z)$	$T(z)$
0	790	0.19	0.00			
1	1023	0.25	0.00			
2	850	0.21	0.00			
3	656	0.16	0.15			
4	329	0.08	0.20			
5	245	0.06	0.30			
6	122	0.03	0.20			
7	81	0.02	0.15			

Low (Chap. 5), G&W (Chap. 4.2) give good examples of effectiveness of histogram modification.

Ex. 4.8-11 Someone has asserted: “histogram modification (or any form of contrast stretching) is unlikely to be of assistance in a purely *automatic* machine vision application (i.e. one in which there is no human intervention), since these techniques introduce no new information – in fact, they often destroy information”.

On the basis of sections 4.4 to 4.8, comment on this assertion. Hint: Look carefully at the histogram equalization transformation function in Ex. 4.8-8.

Compare the input and output histograms. How many grey levels in the input? How many in the output?

4.3.7 Local Enhancement

The point (grey level) methods in sections 4.5 to 4.8 are applied *globally*, i.e. to the whole image. But suppose that the contrast differs across the image, e.g. due to shadow, or parts of a TV camera sensing surface having lost sensitivity. Or, as in the example in G&W (p. 159), you have a global dynamic range that is greater than the display can handle, i.e. you would like to, adaptively, focus on a different part of the greyscale, at different parts of the image.

Thus, you need to allow the transformation to vary across the image.

This can be done by defining an $m \times n$ neighbourhood. The neighbourhood is ‘scanned’ across the image pixel by pixel. It stops at each pixel, computes the transformation for the current $m \times n$ pixels, and applies that transformation to the centre pixel (only).

Obviously, this is a lot of computing, which can be reduced by not overlapping the neighbourhoods; however, a checkerboard effect may result.

See G&W for some results of local enhancement.

4.4 Noise Reduction by Averaging of Multiple Images

Assume you have the possibility of obtaining many still, ‘identical’, images of a scene; but, the images are picking up noise in transmission, or from the sensor system. Then averaging together these images pixel by pixel:

$$f_a(r, c) = (1/N_a) \sum_{i=1..N_a} f_i(r, c)$$

for $r = r_l..r_h, c = c_l..c_h$, where N_a = number of images averaged, and $f_i(.,.)$ is the i th image.

Note: this is done for each pixel independently; we are *not* smoothing across pixels. Thus, we can talk about the mean, $m(r, c)$, and variance $v(r, c)$ of each pixel.

If our model is that the only distortion is noise, then $f_i(r, c)$ can be written:

$$f_i(r, c) = f(r, c) + n_i(r, c)$$

i.e. the i th image is the *true*, noiseless image, plus the i th noise image. Most naturally occurring noise has the characteristic (or is assumed to have) that it is random and uncorrelated. Often too, it is zero mean.

Roughly speaking, these last two statements indicate that for every positive noise value, you will eventually get a negative one, and if you take enough values in the average you end up with zero.

Ex. 4.10-1 Throw a dice many times; for each throw you get one of 1,2,..6; from each throw subtract 3.5 and add the number to a running sum (which was initialised with zero); as the number of throws gets larger, the sum approaches 0.0 more closely.

It can be shown that if the noise level (e.g., as indicated by the standard deviation of the pixel values (at (r, c)) is s_1 for 1 image (no averaging), then it is

$$s_n = s_1 / \sqrt{N_a}$$

for N_a images averaged.

You can easily experience two examples of this:

1. On a noisy stereo radio reception, switch the tuner to mono; this causes the system to add the left and right signals to produce one signal; the noise standard deviation reduces by $1/\sqrt{2} = 1/1.414$, i.e. reduces to 0.707 of what it was,
2. freeze frame on a videoplayer, see how noisy the image is compared to moving: the eye tends to average over a number of the 25 images painted on the screen per second.

Ex. 4.10-2 In DataLab, generate a rectangle, (grect) add Gaussian noise to it (ggna) – observe the noiselike mottled effect; now add more noise (ggna) say 10 times; notice how the mottled effect decreases; the noise is cancelling itself out.

Ex. 4.10-3 You are working with a noisy TV camera and a digitiser. You observe that the noise standard deviation, after the digitiser, is about 16 grey levels. You have detail in the image that requires better than 5 grey levels precision, to be sure of resolving it; thus, in single images, the detail is ‘buried’ in the noise. If possible, how many images should you average?

Incidentally, image averaging does not directly smooth the image, nor sharpen, or otherwise spatially affect it.

4.5 Spatial Operations

4.5.1 Neighbourhood Averaging

Suppose that small neighbourhoods (e.g. 3×3) of an image are known to exhibit very smooth variations in pixel values; so that any abrupt variations have to be due to noise; of course, this is not always true. For this sort of image, if you average over the neighbourhood (window), you will average the noise, which (in the average) tends to zero.

The window (or mask) can be any shape, but is usually square, and often 3×3 . The following formally describes the process.

$$g(r, c) = (1/N_w) \sum_{\text{window } (r,c)} f(i, j)$$

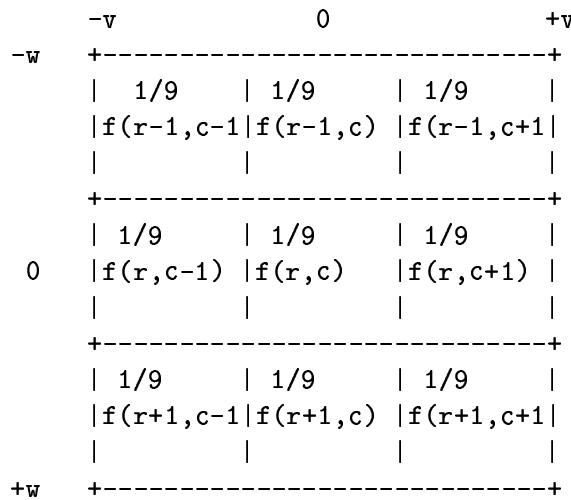
where the window is centred on (r, c) , g is the output image, and f is the input image.

For a window which stretches $-w$ to $+w$ on either side of the central pixel, in the vertical dimension (rows) and $-v$ to $+v$ in the horizontal (columns) this becomes:

$$g(r, c) = \sum_{k=r-w}^{r+w} \sum_{l=c-v}^{c+v} f(k, l) \cdot (1/N_w)$$

where $N_w = (2.w + 1) \cdot (2.v + 1)$

Window ($w=1$, $v=1$, 3×3) centred on pixel (r, c) :



The above equation for $g(r, c)$ allows for any window whose dimensions are odd numbers $(2.w+1, 2.v+1)$ – so that the window extends symmetrically on each side of the central pixel. The factor $1/N_w (= 1/9$ for 3×3), is the so-called *weight* of the *window* or *mask*. For averaging, all weights are equal.

The formal style of the equation for $g(r, c)$ will allow us extend the notion of windows to perform any function, (smooth, sharpen, filter); generally this is called *convolution*. Equivalent terms are: window, mask, template, convolution kernel, kernel, weighting function; also, see later, impulse response.

The major problem with averaging is that real spikes and discontinuities are smoothed as well, not just the noise induced ones. Thus edges and points are blurred. If you start off with a resolution of $10m \times 10m$ (say), and use a 3×3 window, you effectively reduce the resolution to $30m \times 30m$.

4.5.2 Lowpass Filtering

If you use a window like the one seen in the previous section the process is given the general term *lowpass filtering*; it is called this because you are filtering out high frequency components from the image (for more about frequency, filtering etc. see Chapter 3).

Sometimes these windows produce better results than straight averaging.

$$\begin{array}{ccc} 1/10 & 1/10 & 1/10 \\ 1/10 & 2/10 & 1/10 \\ 1/10 & 1/10 & 1/10 \end{array} \quad \begin{array}{ccc} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{array} \quad \begin{array}{ccc} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{array}$$

(a)

(b)

(c)

$$\begin{array}{ccc} 1/5 & & 1/6 \\ 1/5 & 1/5 & 1/5 \\ 1/5 & & 1/6 \end{array}$$

(d)

(e)

Lowpass Filtering Windows

In these filters, (c) is averaging, (d) and (e) are so-called ‘plus shaped’ windows, i.e. zero at corners.

Convolution: in the case of filtering, the more general form of passing a window function over an image is called *convolution*:

$$g(r, c) = \sum_{k=r-w}^{r+w} \sum_{l=c-v}^{c+v} f(k, l) h(r - k, c - l)$$

where $h(\cdot)$ are the weights.

In shorthand this can be written: $g = f \circ h$ where \circ denotes convolution.

The arrangement for convolution is shown as follows:

Convolution Window Centred on Pixel (r,c):

	-v	0	+v
-w	-----+		
	h(1,1) h(1,0) h(1,-1)		
	f(r-1,c-1) f(r-1,c) f(r-1,c+1)		
0	-----+		
	h(0,1) h(0,0) h(0,-1)		
	f(r,c-1) f(r,c) f(r,c+1)		
+w	-----+		
	h(-1,1) h(-1,0) h(-1,-1)		
	f(r+1,c-1) f(r+1,c) f(r+1,c+1)		

Convolution Window - itself:

	-v	0	+v
-w	-----+		
	h(-1,-1) h(-1,0) h(-1,+1)		
0	-----+		
	h(0,-1) h(0,0) h(0,+1)		
+w	-----+		
	h(+1,1) h(+1,0) h(+1,+1)		

The observant reader will notice that in the convolution equation, and in the two foregoing window representations, the window is placed as a mirror image of itself – rotated by 180 degrees; of course, usually it doesn't

really matter, if the window is symmetric. Again, we adopt this formal style because it helps unify all mask/window based operations into one single concept: *convolution*. Much more about convolution later.

The function $h(i, j)$ in the convolution equation is called the *impulse response* of the filter. That is, if you have an image with a single pixel, of value 1 (an impulse), at (r_0, c_0) , and zeros elsewhere, then applying the convolution equation will produce an output image with the values of $h(., .)$ centred around (r_0, c_0) – and zero everywhere else.

Ex. 4.12-1 Prove the previous assertion by applying each of the lowpass filters shown above in section 4.7.1, using convolution, to the following ‘impulse’ image. You should set the values around the edge of the output image to 0.

Image (5x5) with impulse at (r=2, c=2):

```
0 0 0 0 0
0 0 0 0 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
```

Ex. 4.12-2 Given an input image $f[r_1..rh][c_1..ch]$, an output image $g[r_1..rh][c_1..ch]$, a filter window (impulse response) $h[-w..+w][-v..+v]$, write a computer program fragment to perform convolution.

Hint 1: There will be *four* nested loops.

Hint 2: Start off with:

```
for r:= r1+w to rh-w do
    for c:=...
        sum:=0.0
```

Test the fragment by dry running it to do Ex. 4.12-1.

Ex. 4.12-3 You want to save memory. You decide to use only one image, $f()$, and store the results of filtering back in $f()$, i.e. $f()$ is input (source) and output (destination).

- (a) Explain why this will not work.
- (b) For what sort of operations will this work okay?
- (c) try out on DataLab (e.g. functions esobx, esm).

Ex. 4.12-4 Adapt the algorithm of Ex. 4.12-2 to work in a programming language that insists that all array indices start at 0 (or 1).

4.5.3 Median Filtering

As mentioned in section 4.11, a problem with averaging is its tendency to blur edges. *Median filtering* is a method which largely avoids this problem. In addition median filtering is better suited to removing ‘impulse’ noise, i.e. noise characterised by large irregular spikes, or so-called ‘salt-and-pepper’ noise. In an audio signal this is the type of noise that comes from a scratch on a record, rather than the more normal ‘hiss’.

Median filtering works by taking all the values in a neighbourhood, sorting them according to magnitude, and taking the middle ranked value (median) as the output value.

Consider this one-dimensional example which will use a 3 wide kernel or window. Input values:

```
0 0 6 0 0 0 12 0 0 0 15 15 15 15 33 15 15 15 0 0 0 0 0
```

If we apply a 3 wide averaging window to this we get

```
0 2 2 2 0 4 4 4 0 5 10 15 15 21 21 21 15 10 5 0 0 0 0
```

(1) The noise ‘spikes’ (6,12,33) are smoothed – rather smeared – but not removed.

(2) Step edges (0 0 0 15 15 15) are blurred (to 0 5 10 15 15).

Application of a 3 sample wide median filter produces:

(original first)

```
0 0 6 0 0 0 12 0 0 0 15 15 15 15 33 15 15 15 0 0 0 0 0
```

(median filtered)

```
0 0 0 0 0 0 0 0 0 15 15 15 15 15 15 15 15 15 0 0 0 0 0
```

Thus the median filter completely removes the spikes, but the edges are preserved.

See the texts, especially Low (p. 76), G&W for good examples.

As with the weighted filtering windows, the median filter window can be any shape or size; but is often square and 3×3 .

Median filtering is a non-linear filter – we have covered the distinction linear versus non-linear in Chapter 3.

Ex. 4.13-1 (a) Given an input image $f[r_1..r_h][c_1..c_h]$, write a computer program fragment to perform a 3×3 median filter – output image $g[r_1..r_h][c_1..c_h]$. (See Ex. 4.12-2: you should probably adapt the results of that).

As before there will be at least *four* nested loops, starting off with:

```
for r:= rl+w to rh-w do
    for c:=...
```

If you wish, you may assume you have a function

```
sort(x[] : integer, var, n : integer)
```

which sorts an array of integers: $x[0]$ = largest, $x[n-1]$ smallest. n = size of array.

- (c) Test the fragment by dry running it on Ex. 4.13-2.
- (d) How could you speed up the algorithm? Hint: not all of the array (to be sorted) changes as the window is swept over the image.

Ex. 4.13-2 Perform 3×3 median filtering on the image shown below. Draw a picture of the result. What happens at corners?

Letter 'T' with impulse noise:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 9 0 0 6 0 0 0 0 0 9 0 0
0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 9 1 1 1 9 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 9 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 9 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 9 9 0 1 1 1 1 1 0 0 0 9 0 0
0 0 0 0 0 0 1 1 9 1 0 0 0 0 0 0 0
0 0 0 9 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Ex. 4.13-3 Perform 3×3 averaging on the above image of the letter 'T' with impulse noise. Compare the result with that obtained by median filtering. Which would be best if you wanted to recognise the character?

Ex. 4.13-4 A so-called 'separable median filter' is performed by sweeping a 1×3 median filter along each of the rows, followed by a 3×1 median filter down each of the columns.

- (a) Apply this to the noisy letter 'T'.
- (b) Compare the result with that for Ex. 4.13-2.

Ex. 4.13-5 The separable median filter should be much faster; make a brief comparison of the likely speed performance of separable, and square median filters.

Another example: the following figures show the following sequence of images:

- (a) 16×16 image with rectangle
- (b) Rectangle with random spots added.
- (c) Result of 3×3 smoothing
- (d) Result of 3×3 median filter.

In each case the corresponding data are printed.

16 x 16 image with rectangle:

	0123456789012345	
0		
1		
2		
3		
4	MMMMMM	
5	MMMMMM	
6	MMMMMM	
7	MMMMMM	
8	MMMMMM	
9	MMMMMM	
0	MMMMMM	
1	MMMMMM	
2	MMMMMM	
3		
4		
5		
+-----+		

Data for previous figure:

Rectangle image with random spots added:

0123456789012345

0| /
1| /
2| // / /
3| / / / /
4| ///////////////
5| //M/////////
6| ///////////////
7| ///////////////
8| ///////////////
9| ////////////// /
0| ///////////////
1| ///M//////// /
2| ///////////////
3|
4| /
5|
+-----
+0123456789012345

Data for previous figure:

Previous image smoothed with 3x3 window:

```

0123456789012345
-----
0|       ...   |
1|       ...   |
2| ... . . ,.. |
3| .,-/-/-=//. |
4| ,/XBX+=+++, |
5| ,=MMMBBBB+- |
6| ,=MMMMMMB=, |
7| ,=BBBMMMB=, |
8| ,=BBBMMMB=- |
9| ,=BBBBBBB=- |
0| ,=BMMMBBB=/..|
1| ,=BMMMBBB=- |
2| .-=+++=----, |
3| .,-,,,,,.    |
4|                   |
5|                   |
-----+
+0123456789012345

```

Data for previous image:

10	10	10	0	0	0	0	0	10	10	21	21	21	10	0	0
10	10	10	0	0	0	0	0	10	10	21	21	21	10	0	0
21	21	21	10	10	10	21	10	21	10	32	21	21	0	0	0
10	10	10	21	32	43	54	43	54	43	65	54	54	21	10	10
10	10	10	32	54	87	98	87	76	65	76	76	65	32	10	10
0	0	0	32	65	109	109	109	98	98	98	98	76	43	10	10
0	0	0	32	65	109	109	109	109	109	109	98	65	32	0	0
0	0	0	32	65	98	98	98	109	109	109	98	65	32	0	0
0	0	0	32	65	98	98	98	109	109	109	98	65	43	10	10
0	0	0	32	65	98	98	98	98	98	98	98	65	43	10	10
0	0	0	32	65	98	109	109	109	98	98	98	65	54	21	21
10	10	0	32	65	98	109	109	109	98	98	98	65	43	10	10
10	10	0	21	43	65	76	76	76	65	65	65	43	32	10	10
10	10	0	21	32	43	32	32	32	32	32	32	21	10	0	0
0	0	0	10	10	10	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0	0	0	0	0	0	0

Note smudging of edges.

Noisy rectangle median filtered:

0123456789012345

Data for previous image:

4.5.4 Other Non-linear Smoothing

Mode

The output value is the most common in the neighbourhood. Gives similar results to median in many cases. Very useful if the image is made up of labels (e.g. of landuse), i.e. after classification or segmentation. In that case you are taking a vote as to the most common label in the neighbourhood; certainly in that case averaging or median would make no sense. Why?

k-Nearest Neighbour (kNN)

Set $g(r, c)$ to the average of the k pixels, in $n(r, c)$ (the neighbourhood, centred on (r, c)) whose values are closest to $f(r, c)$. Typically use $k = 6$ for a 3×3 neighbourhood. Preserves edges.

Sigma Filter

Set $g(r, c)$ to the average of all input pixels, in $n(r, c)$, whose values are within T of $f(r, c)$. T is a parameter. Called sigma (greek letter, the symbol for standard deviation) because T may be based on the standard deviation. Similar performance to kNN.

Out-of-range filtering

Compute average in the neighbourhood ($n(r, c)$) of (r, c) ; average = $f_a(r, c)$. If the difference between $f(r, c)$ and $f_a(r, c)$ is greater than some threshold, set the output $g(r, c)$ to the average, otherwise set it to $f(r, c)$ (i.e. leave it the same). I.e.

```
If |f(r,c) - fa(r,c)| > Thresh
    then g(r,c) = fa(r,c)
    else g(r,c) = f(r,c)
```

Ex. 4.14.1 Repeat Ex. 4.13-1 for the out-of-range filter.

Ex. 4.14.2 Repeat Ex. 4.13-2 for the out-of-range filter.

Closest-of-minimum-and-maximum

Compute the min and max values in $n(r, c)$. Set $g(r, c)$ to the one that is closest to $f(r, c)$. Useful for sharpening boundaries. Usually iterated. Will leave isolated spikes – these can be removed by median filter.

Min-Max

Compute minimum (or maximum) value in $n(r, c)$ and set $g(r, c)$ to that. Useful for shrinking or expanding 1s in a binary picture. More on this when we talk about segmentation.

4.6 Image Sharpening – General

Often called ‘edge detection’ – but, strictly, we need thresholding to follow the sharpening to qualify for this name. The following shows a full edge detection system, based on gradient.

$$f(x, y) \rightarrow \text{Gradient} \rightarrow | \cdot | \rightarrow \text{Threshold} \rightarrow \text{Edge thinning} \rightarrow \text{Edge}$$

Here $| \cdot |$ denotes absolute value.

Most edge sharpening processes will also sharpen spikes.

In contrast with smoothing operations, which are usually lowpass filters, sharpening operations are associated with highpass filtering.

Noise is usually high frequency, so sharpening will often increase noise effects.

The generalized windowing / masking introduced in section 4.12 is easily applied to sharpening – you just change the weights.

4.7 Gradient Based Edge Enhancement

4.7.1 Introduction

The first part of this section introduces differentiation (of continuous functions), indicates how gradient and ‘edginess’ are related, then shows how to construct windows that do the discrete (digital) version of differentiation.

The use of gradient is based on the reasoning that edges are characterised by large slopes in the image function $f(x, y)$ (view $f(x, y)$ as forming the height coordinate on the $x - y$ plane).

4.7.2 Gradient, Slope and Differentiation

The following figure illustrates gradient of a one-dimensional function $f(\cdot)$ of variable x , $f(x)$.



$$\begin{array}{c}
 | & & / \\
 | & & \text{---} \\
 f(x_2) & 12 & | & / \\
 | & & / | \\
 f(x_1) & 6 & | & \text{Df} \\
 | & & / & \text{Dx} \\
 0 & | & / & \text{---} \\
 0 & & x_1 & x_2 \\
 & & =2 & =4
 \end{array}
 \quad (2)$$

Illustration of 1-D gradient (D stands for delta)

The gradient (or slope) at a point (x) is determined by dividing the increase in f , $\Delta f = f(x_2) - f(x_1)$ (in the example) by the increase in the argument, x , $\Delta x = x_2 - x_1$ (in the example):

gradient $f(x) = \Delta f / \Delta x$

at halfway between x_1, x_2 .

In the limit, as $x_2 - x_1$ approaches 0, ($x_2 \rightarrow x_1$), you end up with:

$$df/dx = \lim_{\Delta x \rightarrow 0} (\Delta f / \Delta x)$$

df/dx is called the derivative of the function f with respect to the variable x . In the figure above, df/dx is about 1.0 at (1), a bit less at (3) and 0.0 at (2); thus df/dx measures slope or gradient.

4.7.3 Discrete Differentiation – Differences

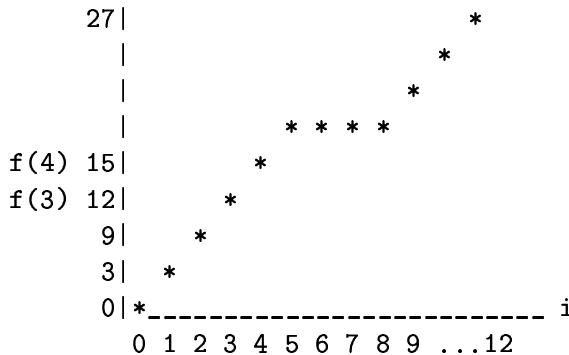
If we now go discrete, as in the following figure, we change from continuous variable x to discrete variable i ; $f(x)$ becomes a sequence of numbers, $f(i)$:

i	0	1	2	3	4	5	6	7...	12
f(i)	0	3	6	9	12	15	18	18	27

Discrete Function

$f(i)$ is shown graphically below.

$f(i)$



Graph of Discrete Function

The discrete version of differentiation is *differencing*. If we have a sampled version of an analogue function, then the difference gives the best available approximation of differential. At any point i , the difference of $f(i)$ is $\Delta f(i) = f(i) - f(i - 1)$.

If the sampling period (Δx) is 1 then this gives a direct replacement for the differential. $\Delta f(i)$ gives an output value centred on $(i + 1/2)$; to make it symmetric we can use: $\Delta f(i) = f(i + 1) - f(i - 1)$.

Ex. 4.16-1 In the figure illustrating the 1-D gradient above, assume that, in region (1), $f(x) = 3x$.

(a) What is df/dx in this region?

In the figure showing a sampled (discrete) version of $f(x)$, the sampling period is $\Delta x = 1$.

(b) What is $\Delta f(i)$ at $i = 3$?

(c) Compare the results of (a), and (b).

(d) What are *two* circumstances in which the results will differ?

Second Differences

If you differentiate df/dx , you get d^2f/dx^2 . In terms of differences, assume you have the difference sequence:

$$\Delta f(0), \Delta f(1), \dots, \Delta f(i - 1), Df(i), \dots, \Delta f(n - 1)$$

Taking differences on this (equivalent to d^2f/dx^2 in continuous functions) yields (just apply the definition of Δ again): $\Delta^2 f(i) = \Delta f(i) - \Delta f(i - 1)$.

But we have seen that $\Delta f(i) = f(i) - f(i - 1)$, and similarly, $\Delta f(i - 1) = f(i - 1) - f(i - 2)$.

Substituting gives:

$$\Delta^2 f(i) = f(i-2) - 2f(i-1) + f(i)$$

This is offset by 1 sample, so a better (symmetric) approximation of $d^2 f / dx^2$ is:

$$\Delta^2 f(i) = f(i-1) - 2f(i) + f(i+1)$$

Usually, the signs are negated.

4.7.4 Differentiation in 2-D – Partial Differentials

Moving to two dimensions introduces a very minor addition. Here we now have *two* gradients:

$\partial f(x, y) / \partial x$ in the x direction, and
 $\partial f(x, y) / \partial y$ in the y direction,

the so-called partial differentials. $\partial f(x, y) / \partial x$ is $f(x, y)$ differentiated with respect to x , with y held constant, i.e.

$$f_x(x, y) = \partial_{\text{at } x, y=y_1} f(x, y) / \partial x = df(x, y = y_1) / dx$$

The notation $f_x(x, y)$ is just a shorthand way of expressing this.

Similarly $\partial f(x, y) / \partial y$ ($= f_y(x, y)$).

A similar $\partial^2 f(x, y) / \partial x^2$ exists, $f_{xx}(x, y)$.

If we want the magnitude of the complete gradient, we use Pythagoras' theorem to add them vectorially:

$$G(f(x, y)) = \sqrt{f_x()^2 + f_y()^2}$$

Sometimes the quicker addition:

$$G(f(x, y)) = |f_x()| + |f_y()|$$

is used.

If we convert the foregoing equation to discrete form, we get:

$$G(f(r, c)) = |f_r(r, c)| + |f_c(r, c)|$$

where here $f_r()$ are now differences (cf. discrete differentiation above):

- (a) $f_r(r, c) = f(r, c) - f(r-1, c)$, and
- (b) $f_c(r, c) = f(r, c) - f(r, c-1)$

These can be computed by simple differencing along the rows, then the columns, and adding the absolute values of the results.

4.7.5 Windows for Differentiation

In terms of ‘windows’ (see section 4.12, ‘Low-Pass Filtering’ above), the foregoing equations suggest

- (a) $f_r(r, c) = f \circ h_r$
- (b) $f_c(r, c) = f \circ h_c$

where \circ denotes convolution, where h_r is a 2×1 window,

$$h_r = \begin{matrix} -1 \\ +1 \end{matrix}$$

and h_c is a 1×2 window:

$$h_c = -1 \quad +1$$

If we base the edge enhancement of the symmetric difference of the expressions for h_r and h_c , we get

$$h_r = \begin{matrix} -1 \\ 0 \\ +1 \end{matrix}$$

and

$$h_c = -1 \quad 0 \quad +1$$

4.7.6 Other Gradient Windows

Prewitt Operators:

$hr = -1 \ 0 \ +1$ $-1 \ 0 \ +1$ $-1 \ 0 \ +1$ (vertical edges)	$hc = -1 \ -1 \ -1$ $0 \ 0 \ 0$ $+1 \ +1 \ +1$ (horiz. edges)
--	--

Sobel Operators: More contribution from central pixel, compared to Prewitt.

$hr = -1 \ 0 \ +1$ $-2 \ 0 \ +2$ $-1 \ 0 \ +1$ (vertical edges)	$hc = -1 \ -2 \ -1$ $0 \ 0 \ 0$ $+1 \ +2 \ +1$ (horiz. edges)
--	--

Roberts Operators: Not vertically, horizontally aligned.

$h1 = 1 \ 0$ $0 \ -1$	$h2 = 0 \ 1$ $-1 \ 0$
--------------------------	--------------------------

4.7.7 Gradient Magnitude and Direction

Often, we are not interested in separating vertical and horizontal edges, we just want the total ‘edginess’. For magnitude we apply the following equation:

$$\begin{aligned} G_m(f(r, c)) &= |G_x| + |G_y| \\ &= |f(r, c) \circ h_r| + |f(r, c)| h_c \end{aligned}$$

or,

$$G_m(f(r, c)) = \sqrt{G_r^2 + G_c^2}$$

Likewise, you can add the Roberts gradients.

If we want direction: $G_a(f(r, c)) = \arctan(G_r/G_c)$. (As usual, r corresponds to y , c corresponds to x , in the notation of some texts.)

Direction is sometimes useful, if you are trying to trace the path of a continuous edge, based on an edge enhanced image.

Ex. 4.16.8-1 Apply a Sobel vertical gradient operator to the letter ‘T’ image given below. Draw a picture of the result. What happens at corners?

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Letter ‘T’ image.

Ex. 4.16.8-2 Apply a Sobel horizontal gradient operator to the letter ‘T’ image. Draw a picture of the result.

Ex. 4.16.8-3 Add the results (absolute values) of Ex. 4.16.8-1, and -2. Draw a picture of the result.

Ex. 4.16.8-4 Given an input image $f[r_1..r_h][c_1..c_h]$, an output image $g[r_1..r_h][c_1..c_h]$, write a computer program fragment to apply the Sobel vertical operator.

Hint: Look at your answer for Ex. 4.12-2.

Another example:

The following Figures show the following sequence of images:

- (a) 16×16 image with rectangle,
- (b) Sobel vertical gradient of (a),
- (c) Sobel horizontal gradient of (a),
- (d) Overall gradient (b) + (c).

In each case the corresponding data are printed.

0123456789012345

0| |
1| |
2| |
3| |
4| MBBBBBBBBB |
5| MBBBBBBBBB |
6| MBBBBBBBBB |
7| MBBBBBBBBB |
8| MBBBBBBBBB |
9| MBBBBBBBBB |
0| MBBBBBBBBB |
1| MBBBBBBBBB |
2| MBBBBBBBBB |
3| |
4| |
5| |
+-----+
+0123456789012345

(a) 16 x 16 image with rectangle

0123456789012345

0| |
1| |
2| |
3| +,BBBBBB,+ |
4| +,BBBBBB,+ |
5| |
6| |
7| |
8| |
9| |
0| |
1| |
2| +,BBBBBB,+ |
3| +,BBBBBB,+ |
4| |
5| |
+-----+
+0123456789012345

(b) Sobel vertical gradient

(b.2) Data for (b)

0		
1		
2		
3	++	++
4	, ,	, ,
5	MM	MM
6	MM	MM
7	MM	MM
8	MM	MM
9	MM	MM
0	MM	MM
1	MM	MM
2	, ,	, ,
3	++	++
4		
5		

+-----+

+0123456789012345

(c) Sobel horizontal gradient

0123456789012345

```
-----
|          |
|          |
|          |
| M++++++M |
| +-+-----+ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| ++      ++ |
| +-+-----+ |
| M++++++M |
|          |
|          |
+-----+
+0123456789012345
```

(d) Sobel gradient - overall

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	198	396	396	396	396	396	396	396	396	396	396	396	198	0	0
0	0	0	396	594	396	396	396	396	396	396	396	396	396	594	396	0	0
0	0	0	396	396	0	0	0	0	0	0	0	0	0	396	396	0	0
0	0	0	396	396	0	0	0	0	0	0	0	0	0	396	396	0	0
0	0	0	396	396	0	0	0	0	0	0	0	0	0	396	396	0	0
0	0	0	396	396	0	0	0	0	0	0	0	0	0	396	396	0	0
0	0	0	396	396	0	0	0	0	0	0	0	0	0	396	396	0	0
0	0	0	396	396	0	0	0	0	0	0	0	0	0	396	396	0	0
0	0	0	396	396	0	0	0	0	0	0	0	0	0	396	396	0	0
0	0	0	396	594	396	396	396	396	396	396	396	396	396	594	396	0	0
0	0	0	198	396	396	396	396	396	396	396	396	396	396	396	198	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(d.2) Data for (d)

4.8 Laplacian

As mentioned above, edge enhancement is based on the highlighting of regions where the values are changing rapidly. Usually, we say we have an edge when the gradient is greater than a threshold. An alternative is to look for points where the gradient reaches a local extremum, i.e. where the second differential has a zero crossing. That is, compute the second differential and mark as edge those points where it changes sign.

The Laplacian gives a method of computing the discrete equivalent of the second differential. Recall that in section 4.16.4, we found the approximation to this, d^2f/dx^2 , to be $\Delta^2 f = f(i-1) - 2f(i) + f(i+1)$. If we extend second differences to two dimensions we get the following plus-shaped window:

$$\text{hlap4} = \begin{matrix} & & -1 \\ & -1 & 4 & -1 \\ & & -1 \end{matrix}$$

This is called the Laplacian operator, after Laplace's differential equation:

$$\partial^2 f / \partial x^2 + \partial^2 f / \partial y^2 = 0$$

even though it is really the negative of the Laplacian.

The following square windows also approximate the Laplacian:

$$\begin{aligned} \text{hlap8} &= \begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix} \\ \text{hlap8} &= \begin{matrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{matrix} \end{aligned}$$

If you need direction (or separate x -, and y -components), you can use:

$$\begin{aligned} \text{hlapr} &= \begin{matrix} -1 \\ 2 \\ -1 \end{matrix} \\ \text{hlapc} &= \begin{matrix} -1 & 2 & -1 \end{matrix} \end{aligned}$$

The Laplacian is sometimes called ∇^2 , *grad squared*.

The Laplacian tends to go a bit crazy on noisy images (i.e. noise is high frequency – the Laplacian enhances high frequencies), so some sort of noise reduction – prior to applying the Laplacian – is advisable.

Lim (p. 488) gives a method of eliminating some of the noise susceptibility from the edge detection process:

1. Compute the Laplacian,

2. Compute the local variance (within a neighbourhood),
3. If Laplacian crosses zero, then: potential edge point,
4. If potential edge point *and* local variance above a threshold, *then:* definite edge point.

Ex. 4.17-1 Apply a Laplacian (hlap4) to the noisy ‘T’ image used in the Figure in exercise 4.13-2.

Ex. 4.17-2 Smooth this same image, then apply the Laplacian, and finally detect zero crossings.

4.9 Edge Detection by Template Matching

Edge enhancement may also be approached by inserting in the h operators templates of edges, e.g. the first of the following operators enhances grey level variations along the ‘north-west, south-east’ diagonal, and the second of the following operators enhances vertical edges (this is one of the Prewitt operators).

$$\begin{aligned} A &= \begin{matrix} 0 & +2 & +1 \\ +2 & 0 & -2 \\ -1 & -2 & 0 \end{matrix} \\ A &= \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} \end{aligned}$$

4.10 Highpass Filtering

Smoothing operations are associated with lowpass filters, sharpening operations are associated with highpass filtering. Thus, more general highpass filtering operations (later lectures) may give alternative edge enhancement methods.

4.10.1 Marr-Hildreth Operators

See Lim (p. 488), Niblack (p. 86).

These are nicknamed ‘Mexican hat’ operators – because of their shape; also called ‘difference-of-Gaussian’ (DOG). They are based on an initial smoothing with a Gaussian lowpass filter window, to remove noise and spurious edges:

$$h(r, c) = \exp(-(r^2 + c^2)/2\pi s^2)$$

where s is a parameter, which gives the width of the smoothing window.

Then apply the Laplacian, followed by detection of zero crossings.

The smoothing, Laplacian processes can be combined by taking the (continuous) Laplacian of the Gaussian function:

$$\nabla^2 h(r, c) = \exp(-(r^2 + c^2)/2\pi s^2)(r^2 + y^2 - 2\pi s^2)/(\pi s^2)^2$$

Ex. 4.20-1 For $s = 1.0$ calculate $h(r, c)$ for:

- (a) $c = 0$, $r = 0$ to 10 in steps of 1.
- (b) $r = 0$, $c = 0$ to 10 in steps of 1.
- (c) Plot results on a graph.

Ex. 4.20-2 Repeat this for the Marr-Hildreth operator, i.e. $\nabla^2 h(r, c)$.

4.11 Additional Exercises

Ex. 4.21-1 A monochrome TV camera (followed by a digitiser etc.) is monitoring parts passing along a conveyer belt. This time, the illumination is even, but unfortunately, the image is ‘noisy’. If you continuously image a grey card (of constant grey level across the field of view, you get an estimate of 7 for the standard deviation of a pixel value (i.e. by estimating the standard deviation for a sequence of values for the same pixel).

Suggest a method by which you can reduce the standard deviation to a level of 1. Assume that you can halt the conveyer belt, under control of the image acquisition computer.

Ex. 4.21-2 A monochrome CCD TV camera (followed by a digitiser etc.) is monitoring parts passing along a conveyer belt. The CCD has 512×512 cells. Unfortunately, each cell has a bias (different for each); e.g. for a white card scene, cell (100,101) might give a value of 200, cell (50,29) might give 210, etc.

Suggest how you would estimate the bias ‘image’.

How would you apply a correction?

Ex. 4.21-3 A monochrome TV camera is monitoring tomatoes passing along a conveyer belt. Green tomatoes are frowned upon by the supermarket buyers. Suggest a technique by which green tomatoes may be highlighted. (Assume you can have the luxury of a separate camera for green tomato detection).

Ex. 4.21-4 A 7×7 image is shown below.

- (a) Show that the Sobel gradient operator, applied to this Figure will yield the image data following, $| G(r, c) |$; the image boundary pixels are not shown – because you cannot apply the Sobel operator at these points.
- (b) If you choose a threshold of 100, what are the candidate edge points?
- (c) It is necessary to perform edge thinning on the result of (b), i.e. to wide strips of edges. Suppose we decide that any point among the candidate edge points is a true edge point if it is a local maximum of $| G |$ in either horizontal, or vertical directions. On this basis, determine edge points.

60	60	62	65	68	70	70
60	60	62	65	68	70	70
70	70	72	75	78	80	80
100	100	102	105	108	110	110
130	130	132	135	138	140	140
140	140	142	145	148	150	150
140	140	142	145	148	150	150

40.8	44.7	46.6	44	7	40.8
160.2	161.2	161.8	161.2	160.2	
240.1	240.8	241.2	240.8	240.1	
160.2	161.2	161.8	161.2	160.2	
40.8	44.7	46.6	44	7	40.8

Ex. 4.21-5 The input greyscale is [0,30];

- (a) give the algorithm (three transformations) to compress the greyscale by a factor of 2 in the ranges [0,10] and [20,30], while stretching it by a factor of 2 in [10,20],
- (b) draw a graph of this transformation; output on vertical axis, input on horizontal (cf. figure in section 4.3).

4.12 Answers to Selected Questions

Ex. 4.21-5 The input greyscale is [0,30];

- (a) give the algorithm (three transformations) to compress the greyscale by a factor of 2 in the ranges [0,10] and [20,30], while stretching it by a factor of 2 in [10,20],

- (b) draw a graph of this transformation; output on vertical axis, input on horizontal (cf. figure in section 4.3).

Answer:

This is slightly tricky, in that the question is not fully specific; it should have been added that the transformation should be continuous – although that would normally be assumed anyway.

Continuous answer:

```
(i) [0..10] -> [0..5]
(ii) [10..20] -> [5..25]
(iii) [20..30] -> [25..30]

if(betw(z,0,10))z' :=(z-0)/2 + 0
if(betw(z,10,20))z' :=(z-10)*2 + 5
if(betw(z,20,30))z' :=(z-20)/2 + 25
where betw(z,z0,z1) = if(z>=z1 and z<=z2)
```

Here I have made explicit the shift to zero *before* the scale, followed by a shift back (to the appropriate start point – the end point of the previous transformation).

A non-continuous answer would be:

```
(i) [0..10] -> [0..5]
(ii) [10..20] -> [20..40]
(iii) [20..30] -> [10..20]
```

though this does not make too much sense.

Ex. 4.7-2 Input image:

```
2 2 2 2 2
4 4 4 4 4
6 6 6 6 6
8 8 8 8 8
10 10 10 10 10
```

- (a) What is the mean, m ? What is the standard deviation, s ?
- (b) Transform to mean, $m' = 128$, standard deviation $s' = 80$.
- (c) Will it fit into $[0,255]$?
- (d) If not, ‘clip’ to 0 below 0 and to 255 above 255.

Answer:

$$\begin{aligned} m &= (1/25)(\text{sum}) = (1/25)(2+2+2\dots+10+10) \\ &= (1/25)(10+20+30+40+50) = 6 \end{aligned}$$

$$\begin{aligned} v &= (1/25)((2 - 6)^2 \dots) = (1/25)(5.16 + 5.4 + 5.4 + 5.16) = 200/25 = 8 \\ s &= \sqrt{v} = 2.8 \end{aligned}$$

$$\text{Formula: } z' = (z - m) \times s'/s + m' = (z - 6) \times 80/2.8 + 128$$

$$\text{at } 2 \ z' = -4 \times 80/2.8 \dots + 128 = 13.7 \text{ etc.}$$

...

$$\text{at } 10 \ z' = 4 \times 80/2.8 + 128 = 242.$$

i.e. the range is reasonably filled.

You can see that the purpose of such a transformation is to shift the data into the middle of the range, and expand the range.

Ex. 4.8-9 Suppose that a 64×64 , 8-level image has the grey level distribution shown below. We have also included in the table $p(z)$, the density calculated from the histogram. Calculate the transformation function, $T(z)$, to give histogram equalization, using the algorithm given above. Then calculate $q'(w)$, the actual output histogram, and plot it.

Recommendation: Proceed as follows. Work out $q(w)$. Then work out the cumulatives $P()$, and $Q()$; then apply the algorithm.

z	$\text{ph}(z)$	$p(z) = \text{ph}(z)/M.N$	$q(w)$	$P(z)$	$Q(z)$	$T(z)$
0	790	0.19	.125	.19	.125	1
1	1023	0.25	.125	.44	.25	3
2	850	0.21	.125	.65	.375	4
3	656	0.16	.125	.81	.5	5
4	329	0.08	.125	.89	.625	6
5	245	0.06	.125	.95	.75	7
6	122	0.03	.125	.98	.875	7

7	81	0.02	.125	1.0	1.0	7
---	----	------	------	-----	-----	---

Histogram for exercise 4.8-9

Answer:

See above. Note, you can work with histograms or with ‘probabilities’, so long as $p()$ and $q()$ are both *histograms* or *probabilities*.

$q() = .125$ is merely dividing the total probability (1.0) into 8 for 8 slots.

Ex. 4.8-10 Someone has asserted: “histogram modification (or any form of contrast stretching) is unlikely to be of assistance in a purely *automatic* machine vision application (i.e. one in which there is no human intervention), since these techniques introduce no new information – in fact, they often destroy information”

On the basis of sections 4.4 to 4.8, comment on this assertion.

Hint: Look carefully at the histogram equalization transformation function in Exercise 4.8-8. Compare the input and output histograms. How many grey levels in the input? How many in the output?

Answer:

(i) If you look at the transformation for Exercise 4.8-9 above you will see that there are *eight* grey levels in the input image, and only *six* in the output (0, 2 are missing). Thus information was destroyed in the transformation. Thus features that were previously evident, may now be obscured.

(ii) The purpose of histogram modification is to better match the picture to the human visual system – i.e. stretch the contrast. In a machine the most common activity will be comparing the values of two pixels; transforming the input (in the manner described above) cannot improve the outcome – but it can disimprove the outcome – i.e. by making pixels equal that were not equal in the input.

There may be some cases where the reason for the transformation has to do with the physics of the problem.

Ex. 4.12-2 Given an input image $f[r1..rh][c1..ch]$, an output image $g[r1..rh][c1..ch]$, a filter mask (impulse response): $h[-w..+w][-v..+v]$, write a computer program fragment to perform convolution (section 4.12).

Hint 1: there will be *four* nested loops.

Answer:

Start off with:

$$g(r, c) = \sum_{k=r-w}^{r+w} \sum_{l=c-v}^{c+v} f(k, l) h(r - k, c - l)$$

where $h()$ are the weights.

Note: this assumes that $h[]$ is stored in an a rectangular array, $h[-w..+w] [-v..+v]$. In most computer languages an array must start at 0 or 1; in C this is 0. Thus, in C the formula would have to read

$\dots f[k][l] * h[r-k+w][c-l+v]$

so that the indices go from $0..w*2 + 1$, $0..v*2 + 1$

Code: (in C)

```

for(r= r1+w;r<= rh-w; r++)
    for(c=c1+v;c<=ch-v;c++){
        sum=0.0;
        for(k=r-w;k<=r+w;k++)
            for(l=c-v;l<=c+v;l++)
                sum+=f[k][l]*h[r-k+w][c-l+v];
        g[r][c] = sum;
    }
}

```

Test the fragment by dry running on an ‘impulse’ image.

If you operate on an impulse with $h[]$ you will get a copy of $h[]$ where the impulse was. I.e. $h[]$ is the impulse response of the operator.

- Ex. 4.13-2** Perform 3×3 median filtering on the image given in Ex. 4.13-2 (letter ‘T’ with impulse noise). Draw a picture of the result. What happens at corners?

Answer:

Median filtering removes all the ‘spikes’, and leaves the ‘T’ intact, *except* it removes the corners.

An averaging filter is not well suited to this sort of noise, *or* this sort of image – where there are sharp edges, which presumably should be preserved. The noise is just smeared, not removed; the edges are blurred.

Ex. 4.16.8-1 Apply a Sobel vertical gradient operator to the image of the letter ‘T’ in Exercise 4.13-2. Draw a picture of the result. What happens at corners?

Ex. 4.16.8-2 Apply a Sobel horizontal gradient operator to the ‘T’ image given in Exercise 4.13-2. Draw a picture of the result.

Answer:

The Sobel operator gives 1,3, or 4 for positive edges, and $-1, -3, -4$ for negative edges; the magnitude depends on whether we are at the end or middle of an edge line.

Usually sign does not matter; although, if you were following edges, using a similarity criterion, it might.

When adding vertical and horizontal edges, the signs *must* be removed – otherwise you may lose edge points at corners etc.

Ex. 4.16.8-4 Given an input image $f[r_1..r_h][c_1..c_h]$, an output image $g[r_1..r_h][c_1..c_h]$, write a computer program fragment to apply the Sobel vertical operator.

Hint: Look at your answer for Ex. 4.12-2.

Answer:

$$h = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

Declare $h[]$ as follows:

```
int h[3][3] = {1,0,-1,2,0,-2,1,0,-1};
```

and use answer to Ex. 4.12-2.

Note the way C stores such arrays.

Of course, you can save time by avoiding multiplying by 1, and avoiding the 0s altogether – but that is a minor implementation concern, secondary to getting the program to work, and to making it readable.

Given that C stores the array like it does, why is it better to have ‘c’, the second index, in the inner loop?

Answer:

If you are in a virtual memory system (e.g. paged), such an arrangement will generate far less virtual memory handling. Consider $NR = NC = 512$ = page size. Done the right way you may get away with 1 page fault. Done the wrong way, you may get 512 page faults.

But, PCs don't have paging, I hear you say. Maybe, but, increasingly, they have caches – which operate in the same way to paging – only the pages are smaller.

4.13 Examples of Image Enhancement Operations



Figure 4.3: Lena image, widely used test image.



Figure 4.4: Noisy version of image – Gaussian noise added. All remaining operations are on this noisy image.



Figure 4.5: Histogram-equalized version of image.



Figure 4.6: Blurred version of image – low-pass filter used.



Figure 4.7: Sharpened version of image – high-pass filter used.

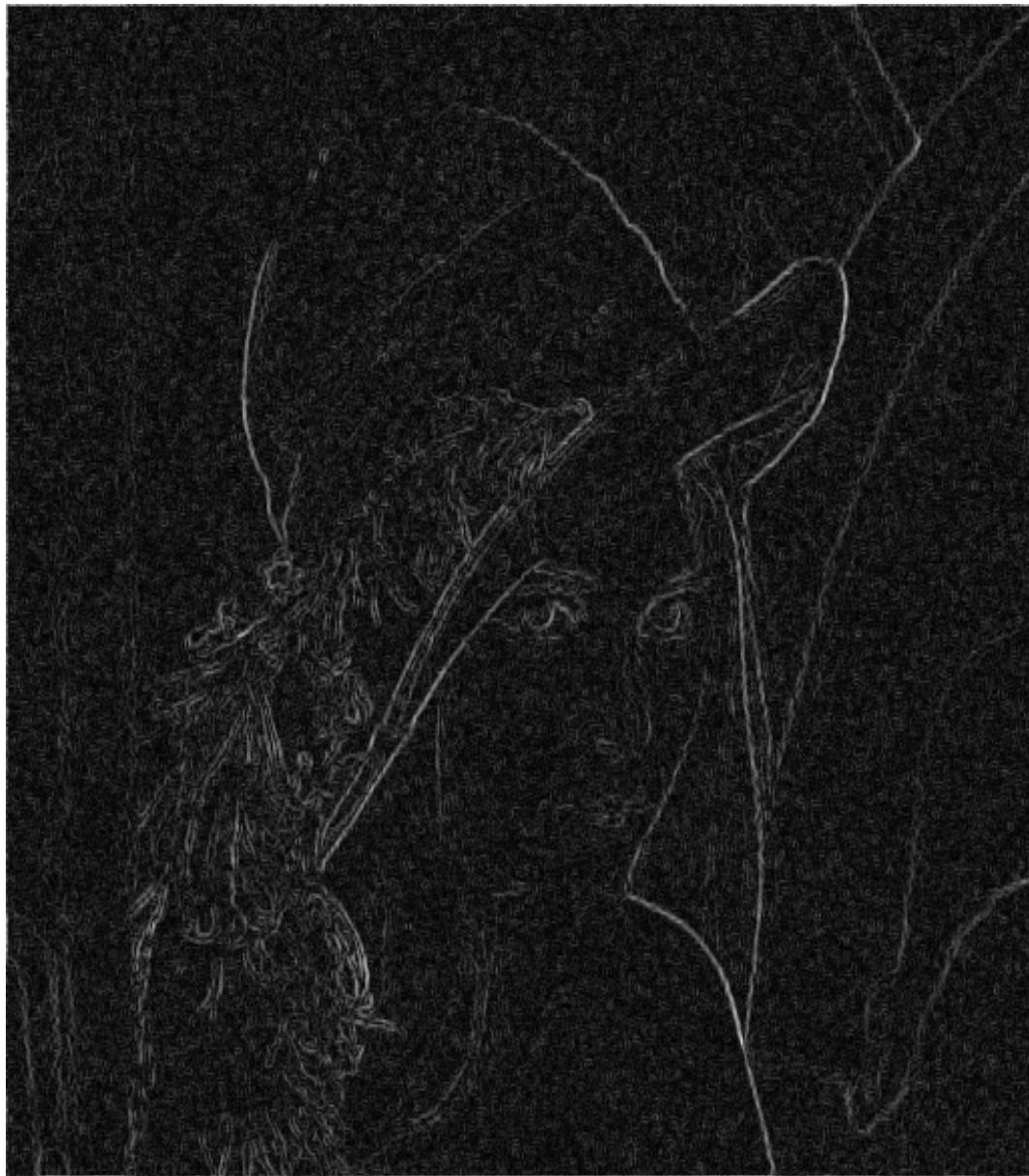


Figure 4.8: Edge detector used on image.



Figure 4.9: Edge detector used on image, followed by histogram equalization.

4.14 Questions on Chapter 4 – Image Enhancement

1. (a) Explain two contrast enhancement techniques. In your answer, be careful to mention the type of image, and applications, to which these techniques are particularly suitable.
- (b) The histogram of an 8×16 3-bit (8-level) image is shown in Figure Q1. Compute the grey level transformation, $T(z)$, required for histogram equalisation. Apply $T(z)$ to obtain the enhanced output image. *Comment on all of your results.* [Remark – this is looking for the extra sparkle that gets the last 3 or 4 marks.]

z	0	1	2	3	4	5	6	7
<hr/>								
$p(z)$	1	7	21	35	35	21	7	1

Figure Q1

(More likely, you would be given an image and so would have to work out the histogram.)

2. (a) Explain and contrast ‘point’ and ‘spatial’ operations for image enhancement. In your answer, be careful to mention, for each type, one form of image / noise to which it is appropriate.
- (b) Explain, using appropriate numerical or pictorial examples, the linear contrast transformation given as follows:

$$z' = (z_G - z_0)(z - a)/(b - a) + z_0$$

- (c) Explain how the linear contrast equation may be used to contrast stretch *or* contrast compress.
3. (a) Explain how the contrast transformation $z' = (z - m).s'/s + m'$ may be used to transform the grey level distribution of an image from having mean m , standard deviation s , to a distribution having mean m' , standard deviation s' . Give a qualitative explanation of an application of such a transformation.
- (b) Input image:

2	2	2	2	2
4	4	4	4	4
6	6	6	6	6
8	8	8	8	8
10	10	10	10	10

What is mean, m ? What is standard deviation, s ? Transform to mean, $m' = 128$, std. dev $s' = 80$. Will it fit into [0,255]? If not, ‘clip’ to 0 below 0 and to 255 above 255.

4. (a) Explain how certain types of noise may be reduced by averaging multiple images.
(b) Local or ‘point’ enhancement operations have the disadvantage that they operate uniformly over an image, however, the image may not be uniform. Discuss, and explain methods of overcoming this problem.
 5. (a) Using appropriate numerical and/or pictorial illustrations, compare and contrast the algorithms for spatial smoothing and edge enhancement. (You should employ appropriate equations and masks provided in the appendix to the exam paper.)
(b) Explain applications of each.
 6. (a) Using appropriate numerical and/or pictorial illustrations, compare and contrast the algorithms for spatial smoothing and median filter. (You should employ appropriate equations and masks given in the appendix to the exam paper.)
(b) Explain applications of each.
 7. (a) Using appropriate numerical and/or pictorial illustrations, compare and contrast the algorithms for spatial smoothing and median filter. (You should employ appropriate equations and masks given in the appendix to the exam paper.)
(b) Explain why any convolution operation is called *linear*, yet median filtering is called *non-linear*.
(c) Explain any other non-linear filter (other than median).
 8. (a) Explain Sobel edge enhancement.
(b) Explain how the results of edge enhancement may be used to complete edge *detection*.
 9. (a) Explain the roles of ‘gradient’ and ‘differentiation’ in edge enhancement. (Use one- or two-dimensional examples to illustrate your answer.)
(b) Apply Sobel edge enhancement to the image given in Figure Q2. Draw a picture of the result.

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Figure Q2

10. (a) Explain the connection between the Prewitt operators and differentiation.
- (b) Given an input image $f[r1..rh][c1..ch]$, an output image $g[r1..rh][c1..ch]$, write a computer program fragment to apply the Prewitt vertical operator.
11. (a) Explain the connection between the Sobel operators and differentiation.
- (b) Given an input image $f[r1..rh][c1..ch]$, an output image $g[r1..rh][c1..ch]$, write a computer program fragment to apply the Sobel vertical operator.
12. (a) Explain the connection between cross-correlation and convolution.
- (b) Given an input image $f[r1..rh][c1..ch]$, a convolution mask $h[0..w-1][0..v-1]$ and an output image $g[r1..rh][c1..ch]$, write a computer program fragment to apply two dimensional convolution.
- (c) (Assignment question only: give at least *two* sets of sample inputs and outputs).
13. A 7×7 image is shown in Figure Q3.
 - (a) Show that the Sobel gradient operator, applied to Figure Q3 will yield Figure Q4, $|G(r, c)|$; the image boundary pixels are not shown in Figure Q4 - because you cannot apply the Sobel operator at these points.
 - (b) If you choose a threshold of 100, what are the candidate edge points?
 - (c) It is necessary to perform edge thinning on the result of (b), i.e. to wide strips of edges. Suppose we decide that any point among the candidate edge points is a true edge point if it is a

local maximum of $|G|$ in either horizontal, or vertical directions.
On this basis, determine edge points.

60	60	62	65	68	70	70
60	60	62	65	68	70	70
70	70	72	75	78	80	80
100	100	102	105	108	110	110
130	130	132	135	138	140	140
140	140	142	145	148	150	150
140	140	142	145	148	150	150

Figure Q3

40.8	44.7	46.6	44	7	40.8
160.2	161.2	161.8	161.2	160.2	
240.1	240.8	241.2	240.8	240.1	
160.2	161.2	161.8	161.2	160.2	
40.8	44.7	46.6	44	7	40.8

Figure Q4

Chapter 5

Data and Image Compression

5.1 Introduction and Summary

The following are the important concepts introduced in this chapter:

- Data compression should be considered whenever large amounts of data are stored on a limited storage resource (e.g. a magnetic disk), or transmitted over a communications channel that has limited capacity (e.g. a communications line with restricted data rate).
- Data compression exploits the fact that you need only store or transmit ‘real information’.
- ‘(Real) information’ is strongly related to uncertainty; if no uncertainty exists, there cannot be any flow of ‘real’ information; you give no ‘information’ if you tell someone something that he/she knows already.
- Entropy is a suitable mathematical measure of uncertainty.
- Formally, the ‘size’ of a piece of ‘information’ (its entropy) can be measured in terms of the number of binary-digits (bits) required to represent it unambiguously.
- The number of bits is exactly the number of ‘twenty-questions’ type questions that need to be asked to unambiguously determine the information.
- The measure is zero if there is no uncertainty, and there is no point in sending or storing anything.
- The measure is small if there is small uncertainty (small variation), e.g. a hexadecimal digit can be expressed in 4 bits.

- The measure is large if there is big uncertainty, a big number requires a large number of bits, e.g. a list of the populations of Irish towns and cities would range (say) 100 to 1,000,000 (a large variation) and would require about 20 bits for each number.
- In sequences of pieces of information, sometimes one piece of information ‘gives the game away’ as to another piece of information; e.g. given the occurrence of the letter ‘q’, surely ‘u’ follows; this is variously called ‘redundancy’ (the ‘u’ is redundant), ‘correlation’, and ‘dependence’.
- Simple entropy-based compression schemes look at the probability (and hence the uncertainty/entropy) of single symbols (a symbol can be e.g. a pixel value, or a text character).
- However, the true entropy of the source must be measured using all possible strings of symbols; hence, single symbol entropy-based methods (e.g. Huffman coding) do not achieve optimum efficiency.
- In signals and images, there usually is dependence (correlation) between neighboring points.
- The bigger the dimensionality (signals: dimensionality = 1, images = 2, moving sequences of images = 3) the more neighbors, so the more correlation, and so the greater the scope for correlation.
- Various compression methods exploit correlation in markedly different ways:
 - run-length encoding,
 - differential (predictive) encoding,
 - transform coding.

5.2 Compression – Motivation

We mentioned in the introductory chapter that technological advances (sensors, display devices, speed of processors, size of memories) are allowing image processing into the arena of data processing (i.e. processing of text, numbers).

Nevertheless, the vast amount of data required to represent a digital image is still a major obstacle:

- Storage: disk and tape devices are getting large and fast (and smaller in size), but not infinitely so. The amount of data produced is increasing ever more rapidly.

- Communications channel capacity: the capacity of a communication channel is finite; fiber optics with their vastly increased capacity are on the way, but copper lines will prevail for most of the next 10 years.

So storage and communication of images can still be prohibitively costly, even though the remaining components of the system are readily and cheaply available. This is especially true of image sequences (video, movies).

For example, a single digitized TV image requires (approximately) $600 \times 600 \times 3$ colors $\times 256$ levels $= 1.08$ Megabyte. A 35mm image (e.g. as used in conventional photography, and in the cinema) requires more than 10 times that.

A good 35mm film – like those used in most movie cameras – will give a resolution of around 50 lines per mm (here, lines = resolvable points); thus, if we allow 2 pixels per resolved point, we have $35 \times 50 \times 2 \Rightarrow 3500 \times 3500 \times 3$ pixels $= 36.75$ Megabytes.

Hence, given our preference for powers of two, most digital animation studios that work for the cinema use 4096×4096 pixel images.

Image data compression (often called just image coding) is about the reduction in the number of bytes (say) required to represent an image.

Ex. 5.1 A data mountain! Consider a satellite which images the earth's surface at 10 meters resolution, and in 10 (color) bands, 1 byte per band. Each point on the earth is covered every 20 days. How many bytes per year? How many magnetic tapes, assuming 5 GB per tape. [Average radius of the earth $R_e = 6378$ km; area of a sphere $= 4\pi r^2$].

Answer:

$$\text{Area} = 511,185,932 \text{ km}^2 = 0.5 \times 10^9 \text{ km}^2 = 0.5 \times 10^{15} \text{ m}^2$$

$$(1 \text{ km}^2 = 10^3 \times 10^3 = 10^6 \text{ m}^2.)$$

Convert to pixels:

$$= 0.5 \times 10^{13} \text{ pixels (i.e., 1 pixel every } 100 \text{ m}^2),$$

$$= 0.5 \times 10^{14} \text{ bytes (1 pixel = 10 colors = 10 bytes).}$$

Coverages per year $= 365/20 =$ approximately 18.

Bytes per year

$$= 0.5 \times 18 \times 10^{14} \text{ bytes}$$

$$= 9 \times 10^{14} \text{ bytes}$$

One 8mm Exabyte tape $= 5G$ bytes, i.e., 5×10^9 bytes,

Therefore, bytes per year $= 9 \times 10^{14}/5 \times 10^9 \approx 2 \times 10^5$

which is a lot of tapes. A large digital tape would contain about 100G bytes. Scientific remote sensing or astronomical missions – provide data measured in terabytes (1×10^{12} bytes). For such data stores, one would need optical disk storage devices, with a juke box (or a human operator) for access.

5.3 Context of Data Compression

The following gives the context of the problem. An image exists at the *source* in direct storage / raw / original form.

SOURCE -----> ENCODER -----> CHANNEL -----> DECODER ---> RECEIVER



Information Transmission Model

First, the data are *encoded*, presumably to occupy less storage than the original. Conceptually, encoding can be split into three steps, as shown in the Figure below. Note, however, that any of the three steps may be missing. If there is no compression/encoding, all three are missing!

TRANSFORMATION -----> QUANTIZATION -----> CODING

Encoder Model

The first step is to transform the image into some form where parts of the transform space are more important (carry more information) than others – however, the transformation is not essential to the concept, so for the purposes of this introduction, assume that the transformation passes the raw data straight on to the quantizer.

The next step is to quantize the data into a finite number of bits. Finally, the data can be coded (e.g. using a ‘codebook’). This is sometimes called ‘source coding’.

The coded data are then sent via a *channel*. Channel is a general term – it could be a transmission line, or it could be a storage device.

While in the channel, the data may be subject to noise corruption, e.g. additive random noise like we have seen. Reduction of the effects of noise is the objective of *error correction codes*; we shall not be concerned with noise – leaving that problem to another part of the system. The channel has limited *capacity*: for a transmission line – bits per second, for a data file – perhaps some agreed limitation in size.

Channel bandwidth: The bandwidth of a channel is measured in Hertz, Hz. The capacity of the channel – its ability to pass information – is proportional to bandwidth. Hertz is measured in cycles/second.

The capacity of a channel depends on (a) its bandwidth, and (b) its signal-to-noise ratio. Noise reduces capacity.

On leaving the channel, the data is not in a directly usable form: it must be *decoded*. Obviously the decoding process is strongly dependent on the encoding process – clearly, there must be cooperation between sender and receiver.

Finally, the image arrives at the *receiver*. Ideally, the receiver should receive the data as they left the source, or as close as possible.

Source Coding versus Channel Coding

Most books introduce two extra stages: *channel encoding* and *channel decoding* – just before, and just after, the channel. These are the equivalent of modulation and demodulation; they are associated with getting the signal into a form in which it can be carried by the so-called ‘physical-layer’. For the sort of approach used here, it is much better to include these in the channel – and to associate any noise due to them with the channel. If necessary we will make the distinction by explicitly naming the encoder and decoder as *source* encoder, and *source* decoder.

Next, we introduce the theoretical notion of information.

5.4 Information Theory

5.4.1 Introduction to Information Theory

We need a precise definition of *information*, particularly a quantitative unit and a method for measuring it.

Information theory defines information as the reduction of uncertainty.

Thus, the sentence “IBM make computers” conveys little or no information – because there is no uncertainty in your minds about the matter.

Informal discussion of Information:

Roughly speaking, the more variance, the more information. More precisely, the information (in bits) in a message is the number of ‘twenty questions’ type questions you would have to use to get the information.

Thus, if I tell you I have written down a number between 0 and 15 (say 5), you can ask me: is it less-than-or-equal-to 7? (yes), is it less-than-or-equal-to 3? (no), less-than-or-equal-to 5? (yes), therefore it is 5 or 4, and the next question clinches it; thus, 4 questions or 4 bits; i.e. a hexadecimal digit contains 4 bits of information.

In this last example, each value was equally likely, and the information for a value can be calculated as:

$$I = -\log_2(p) = \log_2(1/p)$$

where $p = \text{probability of the value} = 1/16$. $\log_2(1/16) = -4$.

Information and entropy (to follow) is associated with the pioneering work of Shannon in the late 40s.

Review of logarithms: $\log_2 x = y$ implies that $2^y = x$. Note that $y = 0 \iff x = 1$. Prove that $\log_2 x = k_1 \log_1 0x = k_2 \log_e x$ for constants k_1 and k_2 .

What is the difference between information and data? This is a bit subtle, and can lead to problems and apparent paradoxes. For example, if you wanted to transmit Hamlet’s ‘To be or not to be’ speech to a friend and you knew they had a collected works on their bookshelf, all you need to transmit is ‘Hamlet Act 3 Scene 2, speech 4’ i.e. about 30 alphabetic characters, or 240 bits – instead of the 2000 or so characters in the speech.

Magic? paradox? No. The friend had received the major part of the information previously – the book.

Most of data compression depends on this principle: don’t waste resources on sending ‘information’ that is already known.

Ex. Fair coin: $I = -\log_2(0.5) = 1$ bit.

Ex. Dice: $I = \log_2(6)$, approx = 2.5 bits. NB $\log_2(1/6) = -\log_2(6)$

Ex. Hexadecimal digit, see section above,

$$I = -\log_2(1/16) = -\log_2(1/2^4) = -\log_2(2^{-4}) = 4.$$

Ex. One letter from an equally probable 26-letter alphabet:

$$I = -\log_2(1/26) = 4.7 \text{ bits i.e. } 5 \text{ bits would do.}$$

Ex. ASCII code (128 allowable symbols) – information of specified symbol:

$$I = -\log_2(1/128) = 7 \text{ bits}$$

which is precisely the number of bits used.

5.4.2 Entropy or Average Information per Symbol

Information theory uses a measure of information called *entropy*; entropy is associated with ‘mixed-up-ness’ – uncertainty.

Entropy (H) is measured in units of *bits* – BIrary digiTs. A bit is the fundamental quantum of information required to distinguish between two equally likely alternatives, e.g. the result of the toss of a fair coin.

Mathematically, for N equally probable outcomes,

$$H = -\log_2(1/N) \quad (= \log_2 N)$$

H is information in bits.

Consider, now, n symbols (say 26) each with a different probability, p_i = probability of i th symbol. The average information conveyed by a symbol is:

$$\begin{aligned} H &= -p_1 \log_2(p_1) - p_2 \log_2(p_2) - \cdots - p_n \log_2(p_n) \\ &= -\sum_{i=1}^n p_i \log_2(p_i) \end{aligned}$$

Note: the average of any function of i , $f(i)$, can be determined from:

$$f_a = -\sum_{i=1}^n p_i \cdot f(i)$$

If the probabilities were equal for each letter, then the average information per letter for English would be 4.7 bits. However, the relative frequencies of occurrence – the frequencies of occurrence (approx. = probability) – of English letters are unequal (e: 0.131, t: 0.105, ...down to z: 0.00077 i.e. 770 times every million), which leads to a reduction in average information to about 4.15 bits per letter.

Clearly, we would like to use 4.15 bits instead of 4.7. In fact, there are coding methods (so-called entropy encoding) that can exploit unequal entropies in symbols; we will see an example of this later.

In images, the pixel values correspond to symbols. And we have already seen, from histograms, that there are unequal probabilities, and, therefore, unequal entropies that can be exploited, usually by appropriate coding – recall the Figure above showing the sequence: Transformation → Quantization → Coding.

Some probabilities of letters are given in the following Table (from Edwards, 1969).

Letter	p
a	.082
b	.014
c	.028
d	.038
e	.131
...	
h	.053
i	.063
...	
q	< .001
...	
u	.025
...	
z	< .001

Probabilities of letters in English text.

If all symbols (or system components) have the same probability, then the entropy is maximum and equal to $\log_2 p$.

5.4.3 Redundancy

We mentioned “independent” symbols above. In a message composed of English text, the symbols are not independent: there are varying degrees of dependence between successive symbols (and indeed between groups of them). If the letter “q” has just been transmitted, then the probability of “u” surely increases and the information contained in the next letter decreases accordingly; therefore, very little information is contained in the “u” because the receiver had a fair idea what it was before it arrived.

Redundancy is the information theory term for ‘dependence’ (or ‘lack of independence’) between symbols; literally, redundancy means ‘not needed’. Of course, redundancy is not always as clear-cut as in the case of the ‘q’ and ‘u’ mentioned above – given ‘q’, the probability of ‘u’ might jump from 0.025 to 0.95 (95%) (not 100%), and given ‘t’ the probability of ‘h’ jumps from 0.053 to maybe 0.2. A measure of redundancy can be defined, in terms of the ratio of ‘lost’ entropy to entropy with no redundancy, as follows:

$$\text{redundancy} = (H_{\text{ind}} - H_{\text{red}})/H_{\text{ind}} = (1 - H_{\text{red}}/H_{\text{ind}})$$

where H_{red} = average entropy for dependent symbols, taking into account redundancy, H_{ind} = average entropy for independent symbols.

Overall, English is more than 50 percent redundant.

Here, we have just been discussing dependency/correlation between pairs of symbols (so called ‘digrams’); there is additional redundancy if we look at triplets (so called ‘trigrams’); e.g. given ‘in’, ‘g’ becomes much more likely.

In images redundancy crops up as correlation between neighboring pixels.

Deep down, most data compression algorithms are based on removal of redundancy – don't send parts of the message that are already known – or on non-uniform entropy (and these are related). Transformation coding and predictive coding are examples of redundancy removal – but the redundancy removal is implicit and not too obvious.

5.4.4 Redundancy is Sometimes Useful!

In most forms of communication redundancy can sometimes be useful; humans use it all the time; e.g. you can often reply before the previous speaker has finished, misspellings in text are an irritation, but the message still gets across.

Of course, parity and checksums are classical forms of redundancy – used for error checking and correction.

Note: You will find that some authors, e.g. Edwards (1969), include, in the definition of redundancy, the loss of entropy due to unequal probabilities; we will reserve redundancy for loss of entropy due to dependence – as above in the equation defining redundancy.

Ex. 5.4-1 Derive an estimate of the redundancy contained in the original of the following sentence (from Cover and Thomas, 1991, p. 136).

TH_R_ _S _NLY _N_ W_Y T_ F_LL _N TH_ V_W_LS _N TH_S S_NT_NC_ .

5.5 Introduction to Image Compression

Image data compression techniques are very much problem-oriented. Nevertheless, there are general methods, and general categorizations that can be made.

In some cases there will be a difference (error) between what leaves the source, and what arrives at the receiver. If perfect reconstruction (after the decoder) is demanded, i.e. zero error, the encoder-decoder (compression) system is called *information preserving*, or *lossless*; typically, for text communication, we require lossless compression; often, for image communication, we may tolerate some loss, after all, we are used to fuzzy pictures in newspapers, badly set-up video recorders, etc.

Therefore, if the nature of the application allows some error, we can use a compression technique that merely maximises some *fidelity criterion*; these techniques are called *lossy*.

Note: Chapter 8 (Pattern Recognition) mentions feature extraction. Feature extraction is strongly related to compression, because:

- *we want to minimize the number of features,*

- we want them to ‘classify’ as well as possible (the fidelity criterion).

It is important to realise that, for error-free compression, there is ‘no free lunch’. As mentioned in the previous section, there are two major sources of savings: non-uniform entropy, and redundancy. If each pixel truly contains 8-bits of information and is truly independent, no compression is possible. But, if the grey levels are not equally likely (say, the average entropy is 6 bits) then compression is possible; however, it will not be possible to reduce below the theoretical limit of 6-bits. In this case the savings can be achieved through proper coding.

If there is correlation, it will be possible to *transform* the data such that there is less correlation (or none) – recall the Figure showing the encoder model: transformation → quantization → coding. E.g. transformation to the frequency domain, via the Fourier transform.

Summary:

There are two major categories of data compression:

- Lossless; the data are reconstructed as if no compression had taken place.
- Lossy; here distortion/errors are tolerated to a limited extent; minimize the errors by maximising some fidelity criterion.

Major compression principles:

- transform compression,
 - linear transforms, e.g. DFT, Discrete Cosine Transform.
 - general transforms – usually decorrelating.
- predictive compression (sometimes contained in the transform category),
- source coding, e.g. entropy encoding,
- quantization coding,
- ad-hoc structural methods, e.g. run-length encoding.
- image model coding, again, may be rather ad-hoc.

Many compression schemes combine more than one of these principles within an encoding scheme – e.g. JPEG, MPEG.

5.6 Run-Length Encoding

If we stretched the definition, run-length encoding could be considered a transformation; however, it is best to consider it as ‘ad-hoc’.

Consider the image in the figure below. Run length encoding exploits the highly repetitive nature of the image. It detects ‘runs’ of the same value, and codes the image as a sequence of: length-of-run, value;.... See below.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16,0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16,0;
0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0	3,0;10,1;3,0;
0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0	3,0;10,1;3,0;
0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0	3,0;10,1;3,0;
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0	6,0;4,1;6,0;
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0	6,0;4,1;6,0;
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0	6,0;4,1;6,0;
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0	6,0;4,1;6,0;
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0	6,0;4,1;6,0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16,0;

(a) Direct storage

(b) Run length

Let us assume that the pixels can be any of 255 levels. The image is 11 × 16, therefore 176 bytes.

Assuming that value and length-of-run can be quantized into 8 bits, run-length encoding reduces 176 bytes to 54 bytes.

A form of run-length encoding is used in fax machines. First, the image is quantized to 1 bit (black-white), and then runs of black (or white) are determined.

If we recognise the two-dimensionality of the image, we can run-length encode both horizontally and vertically.

In its plain form, run-length encoding is error free – lossless.

Ex. 5.6-1 Obviously, run length encoding will not provide any compression for ‘busy’ images (i.e. where the pixel values are rapidly varying, and there is very little correlation between adjacent pixels). Considering the model of a checker-board (alternate black (0), white (1), squares, and a 16 x 16 image, and starting with (1) a pure white image, next, (2) an image with 4 squares

white, black

`black, white`

and so on, at what stage does run-length encoding start to *increase* the data (i.e. more than 256 bytes)?

Ex. 5.6-2 Assume images are first quantized to 1 bit, and that maximum run length is 128; this allows the length-of-run and value to be both coded in *one* byte. Calculate the number of bytes for the ‘T’ Figure above.

Ex. 5.6-3 Using the scheme of Ex. 5.6-2, recalculate the cut-off point for Ex. 5.6-1 (checker-board).

5.7 Quantization Coding

For example, in the case of faxed documents, we agree that there are only two grey levels in the image (`black=0, white=1`). Thus we can quantize the source image to 1 bit. Thus, a saving of 8 times (over 256 levels).

If we have a pictorial image with 255 levels, but know that it will only ever be used to print on a newspaper with 16 grey levels (say), then we can quantize it to 4 bits.

Choice of quantization should depend on knowledge of the *source*, and of the *receiver* – the expected use of the image. E.g. if the image is known to be from (source) a black/white page, then one bit is enough; likewise, if (any) image is only ever going to be printed on such a page, again one bit will suffice.

From our knowledge of vision, we know that humans can, simultaneously, only cope with 160 levels at once; therefore we can restrict most images to 256 levels, or 8 bits. Nevertheless, in an X-ray image, a clinician may view parts of the image *non-simultaneously*; X-ray images tend to be stored using 12 bits.

In general, quantization coding is *lossy*.

5.8 Source Coding

We mention only variable length codes, particularly the Huffman code.

5.8.1 Variable Length Coding

The basic principle of variable length coding is: use short codewords for frequent symbols, use longer for less frequent symbols; in the average your message will be shorter than for equal length codes. Note: this principle is not new – consider the Morse code, which used ‘.’ for the most frequent symbol, the letter ‘E’. However, ASCII uses an equal length code.

Variable length codes can be usefully characterized by the average length of their codewords:

$$L_{av} = \sum p_i \cdot L_i$$

where L_i is the length, in bits, of symbol i and, p_i = probability of symbol i .

The theoretically optimum variable length code would give $L_{av} = H$, the average entropy per symbol. However, in general, quantization effects may reduce the efficiency of the variable length code (for the simple reason that we can use only integer numbers of bits; the optimum code may call for (say) 3.2 bits, we have to use 4, see Exercise 5.8-1). It is easy to show that an optimum code should use, for symbol z – with a probability $p(z)$ – a codeword of length as close as possible to $\log_2(1/p(z))$. Clearly, we will only have $L_{av} = H$ when all the $1/p(z_i)$ are integer powers of 2 (see Exercise 5.8-1).

In general, we can set bounds on the code length, $L(z)$, for a symbol z :

$$H(z) \leq L(z) < H(z) + 1$$

where $H(z)$ is the entropy of the symbol.

Ex. 5.8-1 Let the distribution of values in an image be:

z	0	1	2	3
$p(z)$	0.5	0.25	0.125	0.125

The average information per symbol, or entropy, is given by:

Note: $\log_2(1/2) = -1$, $\log_2(1/4) = -2$, $\log_2(1/8) = -3$

$$\begin{aligned} H &= -[0.5 \log_2(0.5) + 0.25 \log_2(0.25) + 2 \times 0.125 \log_2(0.125)] \\ &= -[-0.5 - 0.5 - 0.75] = 1.75 \end{aligned}$$

This compares with the 2 bits per pixel that would be required without source coding.

If we allocate codewords as follows:

z	0	1	2	3
$p(z)$	0.5	0.25	0.125	0.125
code	0	10	110	111
Length	1	2	3	3

we get an average codeword length of:

$$L_{av} = 0.5 \times 1 + 0.25 \times 2 + 0.125 \times 3 + 0.125 \times 3 = 1.75$$

i.e. the code is optimal.

Note: it is just by chance that 0 is mapped onto 0.

5.8.2 Unique Decoding

Obviously, we need codes to be uniquely decodable; and, we don't want the enormous overhead of 'start' and 'stop' codes !

At a first glance, the codes in the previous example, Example 5.8-1, might look as if a sequence of them would be difficult to disentangle. Consider

1101110101110

well, in fact, this uniquely separates into

110 111 0 10 111 0

and, hence, can be uniquely decoded as: 2 3 0 1 3 0

Ex. 5.8-2 Using the results of Ex. 5.8-1, derive a variable length code for the following symbols – given their probabilities; hence compute the compression achieved.

z	1	2	3	4
$p(z)$	0.5	0.25	0.125	0.125

Ex. 5.8-3 Using the results of Example 5.8-1, derive a variable length code for the following symbols – given their probabilities; hence compute the compression achieved.

z	1	120	122	250
$p(z)$	0.5	0.25	0.125	0.125

Ex. 5.8-4 Using the results of Example 5.8-1, derive a variable length code for the following symbols – given their probabilities; hence compute the compression achieved.

z	122	120	250	1
$p(z)$	0.5	0.25	0.125	0.125

5.8.3 Huffman Coding

The Huffman coding algorithm is an efficient algorithm for optimal source encoding.

The Huffman procedure is given as follows:

1. List the symbols, along with their probability. Elements of the list are nodes – we are building a *binary* tree.
2. Pick the two least probable nodes.
3. Make a new node out of these two – adding the probabilities.
4. Repeat steps 2, 3, until only one node remains – the root (effectively, we are building a binary rooted tree).
5. Bit allocation. Starting at the root, allocate 1 to one branch (left, say), and 0 to the other; until all branches have been allocated.
6. Read out the codes. Starting at the root, travel to the leaf that represents each level, reading off the bits that make up the code for that level.

Ex. 5.8-4 In an image, the 8 symbols z_0, z_1, \dots, z_7 occur with respective probabilities: 0.4, 0.08, 0.08, 0.2, 0.12, 0.08, 0.03, 0.01;

1. derive the Huffman code,
2. derive the average entropy per symbol,
3. using the results of (a) derive the average length of the Huffman generated code symbols,
4. hence, compare the efficiency of the Huffman code with the optimum.

Solution:

Figure 5.1 shows steps (1) to (4), and Figure 5.2 step (5).

Level	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
Starting								
Probs.	0.4	0.08	0.08	0.2	0.12	0.08	0.03	0.01

1st Iter	0.4	0.08	0.08	0.2	0.12	0.08	0.04	

2nd Iter	0.4	0.08	0.08	0.2	0.12	0.12		

3rd Iter	0.4		0.16	0.2	0.12	0.12		

4th Iter	0.4		0.16	0.2		0.24		

5th Iter	0.4		0.36		0.24			

6th Iter	0.4			0.6				

7th Iter				1.0				

Figure 5.1 Steps (1) to (4) of Huffman Procedure.

Level	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
Code	1	0111	0110	010	001	0001	00001	00000
Probs.	0.4	0.08	0.08	0.2	0.12	0.08	0.03	0.01
							+++	
							1	0
						+++++		
						1		0
	+++	---						
		1		0				

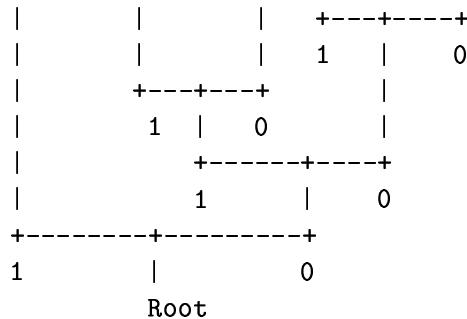


Figure 5.2 Step (5) of Huffman Procedure.

Note: (1) the work shown above – Figures 5.1 and 5.2 – would have been a lot easier, and clearer, if we had ordered the values/symbols according to probability, i.e. 0.4, 0.2, 0.12, 0.08, 0.08, 0.03, 0.01; the result would have been the same.

(2) It should be clear that the abstract symbols $z_0 \dots z_7$ could represent *any* actual values, e.g. 0, 1, 2, ... 6, 7; or, equally validly: 24, 61, 93, 119, 121, 150, 200, 250.

Rosenfeld and Kak (p. 187) give an efficient computational version of this procedure; or Sedgwick (p. 328).

Despite first appearances, the Huffman code is *uniquely decodable*. Suppose a decoder receives, from the channel, the bit stream:

0110100001

Examine this from the left. Neither 0, 01, nor 011 correspond to any code in Figure 5.2. But, 0110 corresponds to z_2 . The next bit is 1, which corresponds unambiguously to z_0 , etc.

0110	1	00001
z_2	z_0	z_6

Notice that the length of the code corresponds, as closely as discrete (integer) lengths can do, to the entropy of the symbol/level.

The entropy of a message composed of the symbols above is:

$$H = -\sum p_i \log_2(p_i)$$

Note: to compute log-to-base-2, $\log_2()$, use the following:

$$\log_2(x) = \log_2(10) \log_{10}(x)$$

$$\log_2(10) = 3.322$$

Thus, $\log_2(x) = 3.322 \times \log_{10}(x)$

For this exercise, we have:

$$\begin{aligned} H &= -[0.4 \log_2(0.4) + 3(0.08 \log_2(0.08)) + 0.2 \log_2(0.2) + 0.12 \log_2(0.12) + 0.03 \log_2(0.03) + 0.1 \log_2(0.01)] \\ &= -[0.529 + 0.874 + 0.464 + 0.37 + 0.152 + 0.066] = 2.45 \text{ bits} \end{aligned}$$

This is the theoretical limit. No source code can achieve better. An uncompressed code would use 3 bits (8 levels). For so few levels the savings are not great, but as the number of levels gets greater, so do potential savings. But anyway, which would you prefer, to pay 300 pounds for something, or 245?

The average length of the Huffman code words is:

$$1.0.4 + 3.4.0.08 + 3.0.2 + 3.0.12 + 5.0.03 + 5.0.01 = 2.52 \text{ bits}$$

which is not far off the optimum.

We conclude that: *Source encoding is error-free.*

Ex. 5.8-6 (a) Compute the histogram of the 10×10 image given below; (b) hence, compute the probability density; (c) hence, derive the average entropy per symbol/pixel; (d) derive a Huffman code for this image; (e) compute the average length of the Huffman code (averaged over all pixels in the image); (f) compare (e) with the theoretical optimum; (g) assuming the image is originally coded with 3 bit pixels, how many bits will the full image occupy? (h) neglecting the space occupied by the code table, how many bits will the Huffman code image occupy? (i) what saving will the Huffman code give?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3
1	2	2	1	2	2	1	1	2	1
2	2	2	1	3	3	1	1	3	3
3	0	0	4	0	0	4	4	0	4
4	4	4	5	4	5	4	4	5	5
5	5	4	4	6	6	7	6	5	5
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Hint: look carefully at Exercise 5.8-5.

Ex. 5.8-7 (a) Compute the histogram of the 10×10 image given below; (b) hence, compute the probability density; (c) hence, derive the average entropy per symbol/pixel; (d) derive a Huffman code for this image; (e) compute the average length of the Huffman code (averaged over all pixels in the image); (f) compare (e) with the theoretical optimum; (g) assuming the image is originally coded with 8 bit pixels, how many bits will the full image occupy? (h) neglecting the space occupied by the code table, how many bits will the Huffman code image occupy? (i) what saving will the Huffman code give?

119	119	119	119	119	119	119	119	119	119
119	119	119	119	119	121	121	121	121	121
121	121	121	121	121	121	121	121	121	121
24	250	250	24	250	250	24	24	250	24
250	250	250	24	121	121	24	24	121	121
121	119	119	93	119	119	93	93	119	93
93	93	93	150	93	150	93	93	150	150
150	150	93	93	61	61	200	61	150	150
119	119	119	119	119	119	119	119	119	119
119	119	119	119	119	119	119	119	119	119

Hint: look carefully at Ex. 5.8-6 and Ex. 5.8-5.

5.8.4 Some Problems with Single Symbol Source Coding

1. Symbols are *not* independent, redundancy exists in strings of symbols. Single symbol coding cannot exploit this potentially rich source of compression. Ideally, we would like to compute the probabilities for all pairs of characters $\{x_1, x_2\}$, hence requiring 65536 (why this number?) probabilities to be estimated; in this case, we would expect $\{'t', 'h'\}$ to have a high probability value, and, as mentioned earlier, $\{'q', 'u'\}$ to have nearly as high a probability as a single 'q'. But, we wouldn't stop at pairs, we would go on to triples $\{x, x_2, x_3\}$ – in which case $\{'i', 'n', 'g'\}$, $\{'t', 'h', 'e'\}$ would be high in the league table; of course, the table of probabilities has now $256 \times 256 \times 256$ entries = 16.9×10^6 which is getting a bit large!

Example. If, say, 'th' has a probability of 0.03125 ($= 1/32 = 2^{-5}$), we can code it with a 5-bit code. On the other hand, if we take single letters, 't' has probability 0.104, and 'h' 0.053; if we round these up to powers of two, we get $p('t') = 0.125 = 1/8 = 2^{-3}$, and $p('h') = 0.0625 = 2^{-4}$, i.e. for an optimum code, we get a total of 7

bits for the two of them. On the other hand, using $p('th') = 0.03125$, we arrive at an optimum code of 5 bits for the pair.

2. If the probabilities used to make the code are wrong, then compression performance can drop badly. And, since it is difficult to make such coding schemes adaptive, i.e. make them adapt to changing probabilities, then the coding can stray from optimum for large sections of the message. For example, in this part of the notes, there is a lot of English, so the probabilities for normal English text may work acceptably. However, if we switch to a C program, the probabilities will be wrong.
3. The average symbol length (L_{av}) of single symbol coding will achieve average entropy only if the probabilities are integer powers of 2; i.e. if we have a symbol i whose probability, p_i , is $1/4$, we can code it with $-\log_2(1/4) = 2$ bits; however, if we have probability, $p_i = 1/5$, $-\log_2(1/5) = 2.3$, so we need to round up and use 3 bits. We waste 0.7 bits every time that symbol occurs.

5.8.5 Alternatives/Solutions

1. Dictionary Codes.

Here we have a dictionary agreed between encoder and decoder. Commonly used strings are held in the dictionary. When a dictionary string appears at the encoder, the encoder emits (1) an ‘escape’ code that says: dictionary code follows, (2) the dictionary code.

Example. Assume a dictionary optimized for text about data compression, with entries as follows:

Code	String
0	the
1	encoder
2	decoder
3	code
4	dictionary
5	compression
	etc.

Every time we encounter ‘decoder’ we send just two data – the ‘escape’ code, and the code ‘1’; hence, assuming we are limited to bytes, we can send ‘decoder’ for the cost of two bytes – a saving of $(7 - 2) = 5$ bytes.

2. Sliding Window Coding.

The dictionary method has the problem of lack of adaptivity; again, a dictionary set up for a textbook on data compression may not work too well for a novel, or for a C program.

In sliding window compression, both encoder *and* decoder keep a buffer containing the last N characters sent/received (typically N = 2048).

The scheme works very similarly to the dictionary method, except now the decoder searches the buffer for the longest match between the incoming string and a string already in the buffer; if no match of length greater than (say) 2 is found, the characters are sent as single characters; however, if a match is found, an ‘escape’ code is sent, together with the position and length of the matched string. Hence, the buffer works as a sort-of adaptive dictionary. One of the Lempel-Ziv (LZ) compression schemes (LZ77) uses a sliding window method.

Lempel and Ziv are responsible for another adaptive method – LZ78 – in which both the encoder and decoder builds a dictionary in the form of a tree. This is used in the CCITT standard V42 bis for data compression in modems.

Most archiving/compression software for text uses some form of Lempel-Ziv code. E.g. DoubleSpace for PCs uses a sliding window compression. Originally, Huffman coding was the default choice – for English text it gave a compaction of about 43%. However, Lempel-Ziv give 55% compaction (hence the name ‘DoubleSpace’).

However, dictionary and sliding window methods normally work much better for text than for numerical data (e.g. image data) – where single symbol methods like Huffman, or arithmetic, see below, perform relatively well.

The principle of Lempel-Ziv coding scheme can be seen from the following example. Essentially, in LZ, encoding is performed by parsing the source data stream into the shortest substrings that have not already encountered.

The encoder and decoder keep in step and maintain codebooks that are identical.

Example. (S. Haykin, Communication Systems, Wiley, 1994, pp. 629-631).

Input data stream: 000101110010100101...

Assume that the symbols 0 and 1 are already in the codebook.

Hence, we can proceed as follows:

Sequences stored: 0, 1

```

Stream to be parsed:          000101110010100101...
Shortest substring not yet encountered: --

Sequences stored:           0, 1, 00
Stream to be parsed:         0101110010100101...
Shortest substring not yet encountered: --

Sequences stored:           0, 1, 00, 01
Stream to be parsed:         01110010100101...
Shortest substring not yet encountered: --

Sequences stored:           0, 1, 00, 01, 011
Stream to be parsed:         10010100101...
Shortest substring not yet encountered: --
etc.

```

At the end of the data stream shown, we will end up with a situation as follows, where ‘data sent’ is simply the shortest-substring-not-yet-encountered coded as:

`previous-substring, followed by the ‘new’ symbol`

Numerical position /									
codebook index:	1	2	3	4	5	6	7	8	9
Substring/codebook entry:	0	1	00	01	011	10	010	100	101
Data Sent:	1	1	1	2	4	2	2	1	4
(i.e. codebook indexes)	1	6	1	6	2				

It should be easy for you to persuade yourself that frequently occurring very long strings can be ‘learned’ by the system, with consequent large savings.

You should also convince yourself that Lempel-Ziv can be done ‘on-the-fly’, and, hence, is suitable for use in modems; again, the encoder and decoder must keep in step.

3. Arithmetic Coding.

Arithmetic coding is an alternative to Huffman coding that *can* cater for probabilities that are not integer powers of two (problem 3 of previous section 5.5.4, “Some Problems with Single Symbol Source Coding”).

5.9 Transform Coding

5.9.1 General

In some of the literature, transform covers a multitude of processes; in what follows we will restrict ourselves to linear transformations – see Chapter 3.

Generally speaking, linear transforms exploit the correlations between pixels; they remove the redundancy, by transforming into a domain where the values are not correlated.

There is a statistical procedure called Karhunen-Loëve Transform (also called Hotelling Transform, Principal Components Analysis, Eigenvector Analysis, or Factor Analysis) which is optimal in producing decorrelated values. However, we have not the prerequisite statistics and mathematical background required to cover it. Also, no fast algorithms ($O(n \log n)$) exist for K-L.

Currently (see the section below on “The JPEG Still Picture Compression Standard”), the Discrete Cosine transform (DCT) is the most commonly used for compression; the DCT is a very close relative of the DFT (see Chapter 3). It can be shown that the DCT closely approaches the K-L. In the past the Hadamard transform has been used.

It is possible to justify transform compression mathematically, but we will avoid that. There are three intuitive ways of explaining transform compression.

1. Frequency selection. If the adjacent pixels of an image are correlated, this means that the pixel values change by only a small amount between neighboring pixels; which, in turn, means that the lower frequencies in the image frequency spectrum are dominant. Therefore, we code the data in terms of frequencies, and we can throw away some of the high frequency elements; or, you can code the higher frequencies using less bits.

Thus, the encoder takes the DFT of the input image, picks the dominant (say) 10% of frequencies and sends these. The decoder reconstructs the DFT ‘image’ (setting the missing 90% to zero), and takes the Inverse DFT. (There will be some overhead, since the encoder must tell the decoder which frequencies were left out).

2. Mathematical transformation – simply by saying that we are transforming to a domain in which the values are less correlated. Since the information content must remain the same, we can represent the image, in the decorrelated domain, using fewer values; for more details, see any description of the Karhunen-Loëve transform.
3. Description in terms of basis images. The image is expressed in terms of its correlation with (similarity to) basis images; e.g. the image can

be described as 0.5 parts of basis-image 1, 0.2 of basis-image 2, etc. You send the 0.5, 0.2, ... ; the receiver knows the basis-images and can reconstruct. If the transform used is Fourier or Cosine, these basis-images correspond to sine and cosine functions at different frequencies – see Chapter 3 (DFT).

Generally, transform image compression is *lossy*.

Although we will not mention them further here, *wavelet* transforms are now attracting great interest for image compression. Chapter 13 of *Numerical Recipes* by Press et al. have a readable introduction using Daubechies wavelets, and also provide accompanying code.

5.9.2 Subimage Coding

Usually transform coding is applied, not to the full image, but to subimages, e.g. 8×8 , 16×16 . See section below on “The JPEG Still Picture Compression Standard”.

5.9.3 Colour Image Coding

For naturally occurring scenes, there is usually very strong correlation between colors; do a scatter plot (function d2s of DataLab) of the green and red bands (bands 1 and 2 – band 0 is infrared) of the SPOT image in DataLab – you will see that the data lie very close to a straight line along the diagonal, thereby indicating very high correlation. Decorrelating transforms can help here, too. But, note that we are decorrelating the vector formed by the bands, *not* the vector/image formed by the spatial array of pixels.

5.10 Image Model Coding

Again, image model coding covers a multitude of techniques. We shall only give a flavor.

A trivial example: consider the 11×16 ‘T’ image in the Figure below. This started off as 176 bytes.

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0

```

```

0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

‘T’ image

1. Run-length encoding reduces to 54 bytes.
2. Simply quantizing to 1 bit reduces to $176/8 = 22$ bytes.
3. My guess is that the first 4 Fourier frequency components (out of 256) would produce an intelligible version of the ‘T’.

But, if we know we have a page of text, it can be reduced to 1 byte – the code for ‘T’ !

Or, if we know it is a graphics image with only rectangles, we can get away with 4 bytes per rectangle.

5.11 Differential and Predictive Coding

Consider the following sequence of data representing (say) 12-bit speech data:

s1	s2	s3	s4	s11
300,	305,	312,	320,	324, 326, 327, 327, 323, 319, 310, ...

This sequence is smoothly varying; compare with the rapidly varying:

300, 1092, 2935, 123, 4000,

If we take differences $d_i = s_{i+1} - s_i$, we have

d1	d2	d3	d4
5	7	8	4

Assume that the maximum difference is 15, and that we can arrange to we can transmit the starting value (300). We can then transmit the signal as the $\{d_i\}$ quantized to 4 bits, thus saving 8 bits per sample.

(In practice differential coding works slightly differently – it needs to guard against one error destroying all the signal, as the above simple method is prone to).

5.12 Dimensionality and Compression

Because of the importance of correlation in image and signal data compression (roughly, correlation means that a pixel is strongly related to – correlated with – some of its neighbors, there is redundancy) multiple dimensions in data offer greater opportunities for data compression. The greater the dimensionality the greater the number of neighbours and therefore potential for correlation, and thereby redundancy, along each dimension.

Thus, familiar examples are:

- Audio signal – one dimension,

$$f_0, f_1, \dots, f_{i-1}, f_i, f_{i+1}, \dots$$

f_i has 2 neighbors.

- Monochrome image – two dimensions,

$$f(0,0) \ f(0,1) \ \dots$$

$$\begin{array}{cccc} \dots & f(r-1,c-1) & f(r-1,c) & f(r-1,c+1) \\ \dots & f(r,c-1) & f(r,c) & f(r,c+1) \\ \dots & f(r+1,c-1) & f(r+1,c) & f(r+1,c+1) \end{array}$$

$f(r,c)$ has 8 neighbors.

- Sequence of mono images (e.g. a movie) – three dimensions.

each point has 26 neighbors.

The sequence is: 2, 8, 26, i.e. $(3^n - 1)$, where n is the dimensionality.

It is well known that the compression factor increases quite non-linearly as you go down the above list; the more neighbors, the more correlation that can be exploited by the compression algorithm.

Colour is a fourth dimension present in most image sequences – and this dimension offers its own correlation, and additional scope for compression.

Incidentally, I would consider a single datum to be zero-dimensional, and, consequently, a collection of data whose only relationship is that they are members of a set would be zero-dimensional.

5.13 Vector Quantization

The section above entitled “Quantization Coding” discussed quantization for single grey levels, i.e. assuming a monochrome image. What if we have a color image? In general a multispectral image where each pixel is represented by a vector of n bands.

We could quantize each band separately. However, it would make sense to take account of the bands together, i.e. divide the ‘space’ formed by the n bands into k quantization ‘classes’. So, we need to perform unsupervised classification (clustering) in k classes. k -means clustering (see Chapter 7) is one method of doing this.

Ex. 5.13-1 (a) Use the k -means clustering algorithm to segment the image in the Figure below into 2 regions (see Chapter 7), and hence explain how to compress it into 1 bit per pixel. Assuming 4 bit input data, what is the compression efficiency. Compute the root-mean-square error (see section 5.15).

```

1 2 3 1 3 2 3 1 2 3
2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 8 2 3 1 2
1 2 3 7 8 9 9 8 7 1
2 3 1 8 9 9 8 7 7 2
3 1 2 9 9 8 7 7 8 3
3 1 2 9 9 8 7 7 8 3
1 2 3 1 3 2 3 1 2 3
2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 1 2 3 1 2

```

(b) In addition to the one bit pixels, what data must you transmit?
Answer: lookup table giving (code: mean value).

(c) What changes if you have color (multispectral) data?

(d) Given the result of (a) what other compression mechanism could you bring into play? [Hint: won’t there be fairly long sequences of 0s and 1s?].

- (e) What is the compression efficiency/rate for the combination of (a) and (d). Verify that the root-mean-square-error (see section below on “Error Criteria for Lossy Compression”) remains the same as for (a).
- (f) Would it make any sense to use Huffman coding on the result of (a) ?

5.14 The JPEG Still Picture Compression Standard

See Wallace (1991), reference given below.

JPEG stands for Joint Picture Experts Group – a joint group formed by ISO and CCITT. ISO and CCITT realised that efficient compression is not enough – we need agreed standard methods. I.e. the decoder shouldn’t have to write a new program for every image.

Actually, there are three compression standards in the JPEG standard. One, the baseline (described below), is based on the Discrete Cosine Transform (DCT); this one is lossy. Another is also lossy but allows the user to specify the compression ratio required. The third is lossless.

We will just give a flavor of the principles behind the JPEG baseline standard; it uses a concatenation of some of the methods mentioned in the previous subsections:

1. the image is split into 8×8 subimages and these 8×8 subimages are transformed using the DCT,
2. The DCT coefficients are quantized: e.g. starting with 8 bits for the DC coefficient, down to 7 for the very low frequency components, down to 0 (i.e. the data are ignored) for the very high frequency terms,
3. The DCT components obtained from (2) are ‘differenced’ (see section above entitled “Differential and Predictive Coding”) with respect to the corresponding component of the previous subimage,
4. The difference values obtained from (3) are Huffman encoded (using a fixed preset Huffman code).

Hard on the heels of JPEG, comes MPEG standard – Moving Picture Experts Group; MPEG standard would be used for such applications as our video-conferencing link. High Definition TV (HDTV) also requires some form of moving picture compression.

5.15 Error Criteria for Lossy Compression

For lossy compression schemes it is necessary to specify how the loss/error is to be computed. Root-mean-square-error (RMSE) is a natural choice:

$$RMSE = \sqrt{[\Sigma(z - z_c)^2]}$$

where z_c is the value after compression,
 z is the (true) value before,
and the summation is over all pixels.

5.16 Additional References on Image and Data Compression

1. G.V. Wallace, "The JPEG Still Picture Compression Standard", *Comm. ACM*, Vol. 34, No. 4 April 1991.
Gives a good introduction to the current state of play in image compression. JPEG is Joint Picture Experts Group – a joint group formed by ISO and CCITT.
2. E. Edwards, *Information Transmission*, Chapman and Hall, 1969.
Very easy, readable introduction to information theory.
3. Gonzalez and Woods, Chap. 6.
4. M. Purser, *Data Communications for Programmers*, Addison-Wesley, 1986.

5.17 Additional Exercises

Ex. 5.17-1 Explain the use of the following methods by which the 16×16 image in the following figure may be compressed; give the compression factor (compressed bits / direct bits). Assume the source is $16 \times 16 \times 8$ levels (3 bits) = 768 bits.

1. Run-length encoding
2. Image model coding
3. Quantization.
4. Hadamard transform [Hint: see Gonzalez and Woods, Figure 3.26, p. 143, and shift and scale the pixel values so that the pixels are $-1, +1$ instead of $0,1$].

```

1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0

```

```

1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

```

Ex. 5.17-2 (a) Explain how the image in the figure below may be compressed using a Huffman code.

```

1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5
4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5
4 4 4 4 4 4 4 4 4 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 6 6 6 6 6 6 6 6

```

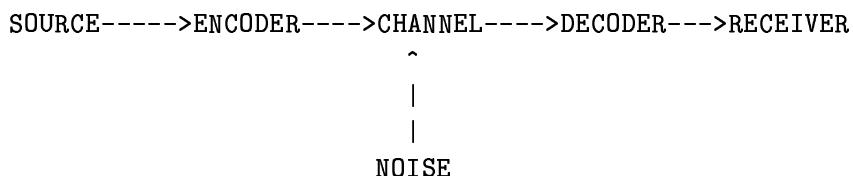
- (b) Assuming that the symbol probabilities are the same for all such images, show that the average entropy per symbol is 2.73.
- (c) Calculate the average entropy per symbol for your Huffman code.
- (d) Compare the compression achieved with the results obtained in Ex. 5.17-1.

Ex. 5.17-3 Identify, with explanations, applications that

- (a) can tolerate lossy compression,
- (b) cannot tolerate lossy compression.

5.18 Questions on Chapter 4 – Image Compression

1. (a) Define a quantitative measure of information; make sure to give intuitive explanations.
 (b) Explain, using appropriate numerical examples, why data compression is needed.
2. (a) Explain the role of entropy in data compression.
 (b) Explain the role of redundancy in data compression.
 (c) “A sequence is one-dimensional, an image two-dimensional, a ‘movie’ three-dimensional; as the dimensionality increases, so does the scope for exploiting redundancy for data compression”. Discuss.
3. (a) In the context of data compression, explain the components of the figure:



Information Transmission Model

- (b) Explain how channel capacity is limited.
4. (a) Explain, giving examples of each, *lossless* and *lossy* data compression.
 (b) Explain applications for which each is (i) definitely appropriate, (ii) definitely inappropriate.
 (c) Explain a ‘fidelity criterion’ for lossy data compression.
5. (a) Explain the terms *entropy*, *redundancy* in the context of image data compression [Remark: “in the context of...” – examples needed!].

- (b) Compute the probability density function of the 10×16 image given in the figure below. Hence, derive the average entropy per the average length of the Huffman codewords (averaged over the image). Assuming that the image is originally coded with 2 bit pixels, and, neglecting the space occupied by the code table, derive the compression performance of the Huffman code. Hence compare this compression performance with that of the theoretically optimum source code.

```

0 0 0 0 0 2 0 0 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 3 3 1 1 1 1 0 0 0
0 0 0 1 1 1 1 3 3 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 2 0 0 0 1 1 1 1 0 0 0 2 0 0

```

- (c) Hence, derive a compression code for the image in the following figure.

```

9 9 9 9 9 0 9 9 0 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 1 1 1 1 1 1 1 1 1 1 1 1 9 9 9
9 9 9 1 1 1 1 1 5 5 1 1 1 1 1 9 9 9
9 9 9 1 1 1 1 1 5 5 1 1 1 1 1 9 9 9
9 9 9 9 9 9 1 1 1 1 1 9 9 9 9 9 9
9 9 9 9 9 9 1 1 1 1 1 9 9 9 9 9 9
9 9 9 9 9 9 1 1 1 1 1 9 9 9 9 9 9
9 9 9 9 9 9 1 1 1 1 1 9 9 9 9 9 9
9 9 0 9 9 9 1 1 1 1 9 9 9 0 9 9

```

6. (a) Explain run-length encoding.
 (b) Apply run-length encoding to the figure below. Assuming ‘runs’ can be encoded as 4 bits [1..16], and grey level as two bits compute the coding efficiency (see Appendix provided in examinations for formulas).

```

0 0 0 0 0 2 0 0 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 3 3 1 1 1 1 0 0 0
0 0 0 1 1 1 1 3 3 1 1 1 1 0 0 0

```

0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	2	0	0	0	1	1	1	1	0	0	0	2	0	0	0

- (c) Obviously, run length encoding will not provide any compression for ‘busy’ images (i.e., where the pixel values are rapidly varying, and there is very little correlation between adjacent pixels). Considering the model of a checker-board (alternate black (0), white (1), squares, and a 16×16 image, and starting with (1) a pure white image, next, (2) an image with 4 squares

(white, black
black, white)

and so on, at what stage does run-length encoding start to *increase* rather than compress the data?

7. (a) Explain the principle underlying variable length source coding; give an appropriate example.

(b) Explain how variable length codes need to be uniquely decodable; give an appropriate example.

8. (a) Compute the histogram of the 10×10 image given below; (b) hence, compute the probability density; (c) hence, derive the average entropy per symbol / pixel; (d) derive a Huffman code for this image; (e) compute the average length of the Huffman code (averaged over all pixels in the image); (f) compare (e) with the theoretical optimum; (g) assuming the image is originally coded with 3 bit pixels, how many bits will the full image occupy? (h) neglecting the space occupied by the code table, how many bits will the Huffman code image occupy? (i) what saving will the Huffman code give?

- (b) Use the previous results to derive a Huffman code for the image below.

119	119	119	119	119	119	119	119	119	119	119
119	119	119	119	119	121	121	121	121	121	121
121	121	121	121	121	121	121	121	121	121	121
24	250	250	24	250	250	24	24	250	24	24
250	250	250	24	121	121	24	24	121	121	121
121	119	119	93	119	119	93	93	119	93	93
93	93	93	150	93	150	93	93	150	150	150
150	150	93	93	61	61	200	61	150	150	150
119	119	119	119	119	119	119	119	119	119	119
119	119	119	119	119	119	119	119	119	119	119

- (c) Compare the coding efficiencies of the original image (assume 8 bits) and coded image.
 (d) Apply quantization coding to the image above; compare the coding efficiency of the quantization coding with that of (1) the raw image, and (2) the Huffman code found in sub-question (k).

9. (a) The distribution of values in an image are:

z	0	1	2	3
$p(z)$	0.5	0.25	0.125	0.125

- (i) Compute average entropy per symbol.
 (ii) Derive a Huffman variable length code for this set of values.
 (iii) Compare the average length of the Huffman code with that of (1) the average entropy per symbol, and (2) the average codeword length in the raw image. Comment.

- (b) Repeat (a) for the following symbols and probabilities; comment on your result.

z	1	2	3	4
$p(z)$	0.5	0.25	0.125	0.125

- (c) Repeat (a) for the following symbols and probabilities; comment on your result.

z	1	120	122	250
$p(z)$	0.5	0.25	0.125	0.125

- (d) Repeat (a) for the following symbols and probabilities; comment on your result.

z	122	120	250	1
$p(z)$	0.5	0.25	0.125	0.125

10. (a) Explain *vector quantization* in the context of image data compression. [Answer: first explain optimum quantization (a histogram would help), then go to 2-dims.... feature space type diagram would help.]
- (b) (i) Use the k-means clustering algorithm to segment the image below into 2 classes, and hence explain how to compress it into 1 bit per pixel. (ii) Assuming 4 bit input data, what is the compression efficiency. (iii) Compute the root-mean-square error caused by the compression.

```

1 2 3 1 3 2 3 1 2 3
2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 8 2 3 1 2
1 2 3 7 8 9 9 8 7 1
2 3 1 8 9 9 8 7 7 2
3 1 2 9 9 8 7 7 8 3
3 1 2 9 9 8 7 7 8 3
1 2 3 1 3 2 3 1 2 3
2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 1 2 3 1 2

```

- (c) Comment on the changes if you have colour (multispectral) data.
- (d) Given the result of (b) what other compression mechanism could you bring into play – after the clustering compression?
11. Describe the principals, techniques and applications of transform image compression.

Chapter 6

From Image to Objects

6.1 Introduction

This chapter presents methods for segmentation; mathematical morphology operations; and multiresolution transforms, focusing on two wavelet transforms. There is a common theme in these different processing methods: open up an image, to reveal the objects represented within it. The objects of interest may well be closely associated with “segments” derived from the image. The objects of interest may exist on a number of resolution scales, – in the foreground or background, embedded, and so on. Mathematical morphology provides us with highly-effective tools for improving, often dramatically, the objects derived from our images.

6.2 Introduction to Segmentation

Image segmentation is a process which partitions an image into regions (or segments) based upon similarities within regions – and differences between regions.

Commonly, an image represents a scene in which there are different objects or, more generally, regions; it is a widely held view that the first stage, in human interpretation of a scene, is its division into regions; hence, there is great interest in image segmentation. However, although humans have little difficulty in separating the scene into regions, this process can be difficult to automate.

The following depiction of operations performed on an image shows a possible context for segmentation – this might represent the information flow in an automated inspection system that monitors that manufactured parts are the correct shape, or a character recognition system, etc. The following can be characterized as our *image analysis model*, or even *vision model*.

RAW IMAGE (pixel values are intensities, noise-corrupted)

→ *preprocessing* →

PREPROCESSED IMAGE (pixels represent physical attribute, e.g. thickness of absorber, greyness of scene)

→ *segmentation* →

SEGMENTED or SYMBOLIC IMAGE (each pixel labelled, e.g. into object and background)

→ *feature extraction* (e.g. line detection, moments) →

EXTRACTED FEATURES or RELATIONAL STRUCTURE

→ *shape detection and matching* (pattern recognition) →

OBJECT ANALYSIS

The major point to note in this depiction is that we start off with raw data (an array of grey levels) and we end up with information – the identification and position of an object. As we progress, the data and processing move from low-level to high-level.

But here we are mainly concerned with the process of labelling each pixel as either object or background, one major task of *segmentation*.

Examples of practical segmentation tasks are:

- segment an image of metal parts on a conveyer belt into metal (object), background non-metal, i.e. conveyer belt.
- segment a scanned image of a printed page into ink, white page (background). The result could be used for fax transmittal, or for further processing to recognise characters.
- segment an X-ray image of a printed circuit board into metal (object), or plastic (background). Again, further processing may be desired to identify individual tracks and other features.
- the same could apply to a medical X-ray: soft-tissue, bone.
- segment a multispectral satellite image of the earth into regions of different landuse. Although, in this case the pixels are vector valued (each pixel has many attributes – not just one grey level), the underlying principles are the same.
- segmentation for data compression: if we have an image with (say) only two ‘colours’ of interest in it; let us say metal, non-metal; assume the image is 256 grey-levels × 3 colours, i.e. 24 bits per pixel. Now if we can segment into metal, non-metal, we can get away with 1 bit!

- provided we transmit, before the 1 bit label data, what is called a code-book; the code-book gives the mean colour of the two regions or objects, from which the picture can be reconstructed.

Haralick and Shapiro (1985) give the following wish-list for segmentation: “What should a good image segmentation be? Regions of an image segmentation should be uniform and homogeneous with respect to some characteristic (property) such as grey tone or texture. Region interiors should be simple and without many small holes. Adjacent regions of a segmentation should have significantly different values with respect to the characteristic on which they (the regions themselves) are uniform. Boundaries of each segment should be simple, not ragged, and must be spatially accurate”.

In this chapter we will cover the three general approaches to image segmentation, namely, single pixel classification, boundary-based methods, and region growing methods. There are other methods – many of them. Segmentation is one of the areas of image processing where there is certainly no agreed theory, nor agreed set of methods.

Broadly speaking, single pixel classification methods label pixels on the basis of the pixel value alone, i.e. the process is concerned only with the position of the pixel in grey-level space, or colour space in the case of multi-valued images. The term *classification* is used because the different regions are considered to be populated by pixels of different *classes*.

Boundary-based methods detect boundaries of regions; subsequently pixels enclosed by a boundary can be labelled accordingly.

Finally, region growing methods are based on the identification of spatially connected groups of similarly valued pixels; often the grouping procedure is applied iteratively – in which case the term relaxation is used.

6.2.1 Single Pixel Classification

Introduction

Single pixel classification is reminiscent of the point enhancement methods covered in Chapter 4; each pixel is labelled according to its *own* value, and for that labelling, the values of its neighbours are irrelevant.

We start off with simple thresholding, where choice of label depends on which side of a threshold the pixel value lies; this is applied globally. Next, as in Chapter 4, we discuss how we can make the threshold adaptive to local variations in the image. Then we explain how thresholding can be used to determine boundaries, and hence regions. Then we generalize thresholding to “level-banding” for multilevel/multiclass problems. Next we generalize monochrome thresholding to multispectral classification, and finish off with unsupervised classification, which is also termed clustering.

Threshold Selection

Let the input image have a histogram $p(z)$. Let there be two peaks in $p()$, at z_i, z_j , i.e. $p(z_i)$ is a local maximum (local = the maximum of $p()$ within z_l grey levels, e.g. $z_l = 10$), and similarly $p(z_j)$ is a local peak.

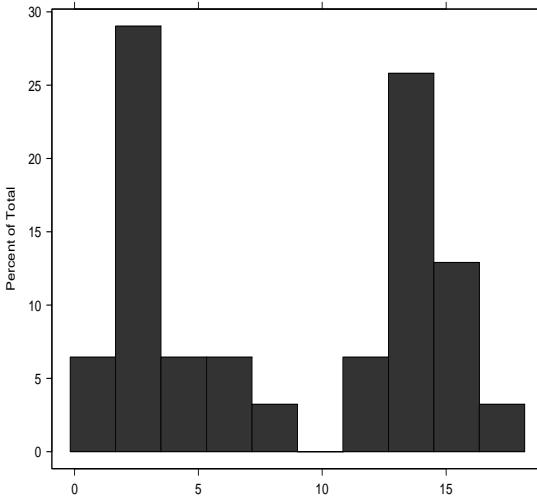
Let z_k be the mid-point between z_i and z_j , ($z_k = (z_i + z_j)/2$). Then if $p(z_k)$ is much smaller than either $p(z_i)$ or $p(z_j)$ (i.e. z_k is in a significant valley), z_k should be a useful threshold for segmenting the picture. The segmentation rule is:

```
label background if  z < T1
label object          otherwise
```

or in pseudo-code:

```
if z < T1 label = background
else label = object.
```

The following figure gives an example (it is reminiscent of the histogram of an object superimposed on a background, with noise added).



The values used in this histogram were 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 4, 5, 6, 7, 9, 11, 12, 14, 14, 14, 14, 14, 14, 14, 15, 16, 16, 16, 18. The mean of the lower “chunk” of data is $z_i = 3.25$. The mean of the larger “chunk” is $z_j = 14.4$. The mid-point between these two means is $p_k = 8.825$ (or nearly 9). Thus thresholding at 9 would do a reasonable job of separating the object from the background.

This method is highly intuitive. Often ad hoc changes have to be made to cope with particular distributions of grey levels. The clustering methods mentioned below are more general.

Visually, 10 would be even easier in our example, and would be a solution also in this case. Therefore we have proposed two solutions: (i) find the clusters, and their means, and use the mid-point of means as a threshold; and (ii) find the deepest valley(s) in the histogram of the data.

Local versus Global

If there is unevenness of illumination the histogram approach may run into difficulties – the histogram may tell more about the illumination distribution, than about the colours of the object(s) and background. In such cases it may be beneficial to perform segmentation first on small local areas, followed by some rationalisation of the different segmentation results.

Segmentation Based on Boundary Pixels

In some cases there may be two or more background levels (the same may be true for illumination effects – see (b) above), which may confuse the segmentation, e.g. there will be significant peaks in the histogram corresponding to each level. In such cases it may be better to work on a histogram of only those points on or near boundaries. This implies working on points with a large gradient value (cf. Chapter 4 on edge detection).

Multilevel Thresholding

We use here the same principle as was used for simple single level thresholding. For an image with three peaks, the lowest peak is noise on the dark background, the next corresponds to a grey object, and the third to a white object. Again, we look for local peaks in the histogram. We choose (multiple) thresholds in between. We then apply the rule:

```
label background if       $z < T_1$ 
label object1 if         $T_1 \leq z < T_2$ 
label object2 if         $T_2 < z$ 
```

or in pseudo-code:

```
if  $z < T_1$  then label = background
else if  $z < T_2$  then label = object1
else label = object2.
```

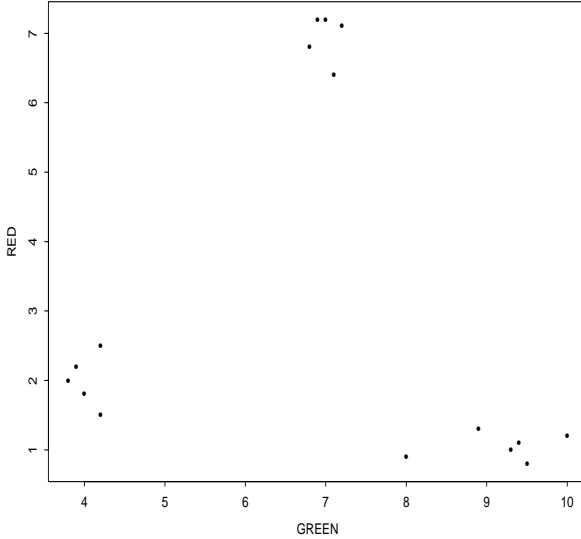
Multilevel Thresholding for Edge Detection

Consider an image that has two main regions (say grey level 3 ± 2 for region A, and grey level 10 ± 2 for region B). It is likely that at the boundary between A and B the grey level will gradually increase between (say) 3 and

10, i.e. the sequence of pixels along a row might be 3, 5, 6, 7, 9, 10. If we segment the image into $\{5..7\}$ (edge) and others, we will get a contour showing the edge between the two regions.

Multispectral Images

Consider a two-colour image. Its histogram can be represented as a two-dimensional scatter plot showing the numbers of pixels in slots. The following figure shows a scatter plot of (some of the pixels) of an image having Red and Green bands.



Question: could one use instead a two-dimensional histogram? How would you construct a two-dimensional histogram?

If an image corresponding to such a scatter diagram were to be segmented, it would be natural to segment it into three regions – corresponding to the clusters of points at, approximately, $(\text{Green}, \text{Red}) = (1,10)$, $(7,7)$, and $(4,2)$.

Obviously it would be possible to extend the histogram valley method (discussed above in the context of simple single-band thresholding) to two dimensions and beyond. However, a more general concept, that of clustering, makes this idea a more practical one.

Clustering

Clustering has been studied by statisticians for many decades. Their interest is in finding significant groupings of points according to some property or set of properties. In one dimension clustering is easy to visualize – a cluster is a “hill” shape in the histogram. In two dimensions, we are similarly

looking for hills. In three dimensions we seek a globule where the points are fairly dense (i.e. a cluster!). Beyond that, visualization is difficult, but the principle is the same – peaks in the density are good cluster centres.

k-Means Clustering

There is a vast array of clustering methods – see any book on image processing or pattern recognition. A very simple, but powerful method is k-means clustering (sometimes called ISODATA – though ISODATA is really a special algorithm for doing k-means clustering):

Procedure k-Means Clustering:

```

Inputs: nc = number of classes,
        xij data to be clustered; i=1..d; j=1..N
        d = dimensionality of data vectors (d colours)
        N = total number of pixels.
Outputs: labj, j=1..N, labels of pixels xj
         labj = {1..nc}

```

For the purposes of this procedure we are totally unconcerned with the spatial position of pixels – just their position in data space or *parameter space* or *feature space* – cf. the two-dimensional scatter plot above.

1. Partition the pixels arbitrarily into nc classes (labels), e.g. divide the image into nc regions, label all the pixels in the first region 1, second region, label 2, etc.
2. Compute the mean vectors of each class: `mic[i][c], i=1..d, c=1..nc`
3. Procedure: k-Means Classifier:

```

3.0 Change = False
3.1 for j = 1..N do
    3.2   reset Distmin = Bignum, class = 0
    3.3   Find nearest mean:
        3.3.1 for c = 1..nc do
            3.3.2 Compute Dist = Distance(xj to mean mc)
            3.3.3 if Dist < Distmin then Distmin = Dist
                  class = c
        3.3.4 end
    3.4   labj = class; if 'changed' set Change = True
3.5 end

```

4. if Change = True goto 2 (loop again)
5. end (we are finished)

```
Function Distance(x,m)
```

```
(*this calculates squared Euclidean distance - because there
is no need to take square root, since we are not interested in
actual values, just in finding the minimum*)

Dist = 0
for i = 1 .. d do
    Dist = Dist + (xi -mi)**2 (*Euclidean distance*)
return Dist
end
```

The above algorithm terminates when there are no class changes. It will generally iterate to the same result if the classes are well separated, but not necessarily so when the classes are confused and overlapping. In fact, this algorithm is representative of *suboptimal* algorithms – a good but not best result is produced. It is fast, – as an indication, it terminates in about nc^2 iterations, i.e. if there are 4 classes, about 15 to 20 iterations. Note in particular that this method works equally well for 1-dimensional data as for 50-dimensional.

The following distance may be quicker to calculate:

```
Dist = Dist + abs(xi-mi)
```

This is called the *city-block distance* due to the fact that it is reminiscent of a regularly planned city, where travel means going down some blocks in a given direction, then some blocks in another direction, and so on. On the other hand, the Euclidean distance is the distance “as the crow flies”.

We note in passing that the neural network method known as the Kohonen neural network, or self-organizing feature map, effectively implements a k-means clustering.

Example 1

The following 10×10 image is a small part of a satellite image of the River Foyle and its east bank at St. Columb's Park. We can consider segmenting the image using

- (a) k-means clustering, with $k = 2$,
- (b) thresholding based on histogram; we can use broad grey level slots for your histogram, e.g. 20 levels: 0-19, 20-39, ...

0	2	7	7	7	10	10	10	13	15
5	7	10	10	10	10	10	10	15	31
7	10	10	10	10	13	15	23	71	92
10	10	10	13	15	23	63	106	132	116
10	10	18	45	47	79	124	135	124	122
13	45	87	79	79	119	138	116	116	127
50	114	87	77	106	132	140	124	132	122
106	103	90	119	132	146	146	135	138	135
108	85	108	132	146	148	143	132	124	127
92	100	127	140	151	148	148	135	98	92

From the k-means clustering, we find class means of 15.79 and 118.07.
The segmented image can be represented as follows.

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	2	2	2	2
1	2	2	1	2	2	2	2	2	2
1	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2

Example 2

Verify that the 3-colour image below gives the same labelling as the one-colour image used in the previous example.

Dim: 0 (infrared)

0	2	7	7	7	10	10	10	13	15
5	7	10	10	10	10	10	10	15	31
7	10	10	10	10	13	15	23	71	92
10	10	10	13	15	23	63	106	132	116
10	10	18	45	47	79	124	135	124	122
13	45	87	79	79	119	138	116	116	127
50	114	87	77	106	132	140	124	132	122
106	103	90	119	132	146	146	135	138	135

108	85	108	132	146	148	143	132	124	127
92	100	127	140	151	148	148	135	98	92

Dim: 1 (red)

15	23	31	31	23	23	23	23	23	39
23	31	23	23	23	31	23	15	31	39
31	23	31	31	31	23	31	47	63	55
31	39	47	47	31	31	55	79	95	71
47	63	55	39	39	55	71	79	95	87
71	55	39	47	63	71	71	79	95	87
63	39	47	63	79	95	95	95	103	95
47	39	55	71	87	103	103	103	87	79
39	47	71	87	87	87	87	79	55	55
39	55	63	79	63	55	79	63	55	71

Dim: 2 (green)

30	30	37	45	37	45	37	30	37	37
30	30	30	37	37	37	30	22	45	45
37	30	45	52	30	30	37	30	52	52
45	45	45	45	30	22	52	67	67	60
60	45	37	37	30	37	52	67	75	60
52	30	30	45	45	52	60	67	82	60
37	22	30	45	60	67	67	67	75	67
30	22	37	60	60	67	75	67	60	60
22	30	52	67	67	60	60	52	45	52
22	30	45	52	45	45	52	45	45	67

The minimum and maximum values for infrared, red and green are, respectively, (0,151), (15,103) and (22, 82). We find the mean of class 1 to be (15.79, 34.35, 37.47). We find the mean of class 2 to be (118.07, 71.98, 53.04). In the single-band image, we see that we were actually using the infrared image. It *is* possible that the assignments of pixels based on the single band image would differ from the assignments of pixels based on the three-band image. This is because in the latter case, we are taking information into account from three different sources, and not just one. Our solution will be a good one based on these three different sources of information. Potentially, it will be an optimal solution.

When you hear the term “optimal”, you have every right to demand to know what the criterion of optimality happens to be. For us here, optimality is with respect to a least squared error criterion. Alternatively we may say that, for us, discrepancy between desired target and given data is measured

by the Euclidean distance.

We find the segmented image, just like before, to be:

The segmented image can be represented as follows.

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	2	2	2	2
1	2	2	1	2	2	2	2	2	2
1	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2

Discussion

With respect to the Haralick and Shapiro desiderata quoted in the introduction, the single pixel methods which we have described clearly perform well on the homogeneity criteria. But, since they are concerned only with single pixels, they may not perform well on the spatial criteria. We may well want to ignore small holes in the interiors of regions. We may also want to ignore very ragged boundaries.

It is possible to smooth the results of single pixel segmentations using “shrink-expand” or “expand-shrink” smoothing (see Niblack, 1986, p. 117). An approach for doing this will be looked at later in this chapter, using mathematical morphology.

A general remark is that one should be wary of ad hoc methods that work on one type of image. They may not transfer well to another type, nor (maybe) be robust to changes in (say) noise level.

6.2.2 Boundary-Based Methods

Boundary-based methods involving finding boundaries (which are connected and closed) between regions; pixels enclosed by the boundary are then taken as belonging to the region. (See Niblack, 1986, Chapter 5.2; Rosenfeld and Kak, 1982, Chapter 10.)

Chapter 4 has described methods of enhancing edge points, i.e. gradients. However, these merely produce a measure of edginess for each pixel and there are a few extra steps before we get a set of linked boundary points:

1. Compute the gradient, e.g. using the combination of Sobel horizontal and vertical operators (see Chapter 4).

2. Thin the edges. There are many methods; two examples:
 - (i) retain only points whose gradient is a local maximum in its gradient direction,
 - (ii) (simpler). Work on horizontal and vertical edges (e.g. Sobel, before they are added). For vertical edges, choose a threshold (T); eliminate all pixels with values less than T . Then scan the image vertically eliminating all but the maximum of groups (connected runs) of edge points. Ditto horizontal. Then add vertical and horizontal thinned edge points.
3. Determine possible edge neighbours. If edge points are to be linked, they must (a) be neighbours, (b) have edge values in the same direction (see gradient angle in Chapter 4)
4. Link edge points, thereby yielding edge chains.
5. Link edge chains.
6. Extend edge chains, e.g. to jump across a discontinuity in an otherwise continuous chain.
7. Eliminate edge chains:
 - i.e. those that are
 - too short,
 - too curvy,
 - or other such elimination criteria.
8. Eliminate non-closed chains.
9. The closed chains now form the boundaries of regions.

Region Growing Methods

A simple intuitive region growing method works as follows; note the similarity with clustering – but with a spatial criterion added.

1. Choose a seed point (perhaps a pixel having a value at the peak of the histogram).
2. Grow the region using the two criteria:
 - (a) the difference in grey level (or colour distance if multicolour – see k-means clustering algorithm above) between the seed and the candidate is less than D ; typically, D would be chosen as some percentage of the image grey level range (maximum grey level – minimum grey level).

(b) the candidate must be connected to another pixel which has already been included. Connected can be defined as touching, i.e. one of the eight pixels which border a central pixel.

6.2.3 To Read Further on Image Segmentation

1. Gonzales, R.C. and Woods, R.E. 1992. Digital Image Processing. Addison-Wesley.
2. Haralick, R.M. and Shapiro, L.G. 1985. Image segmentation techniques. Computer Vision, Graphics and Image Processing. Vol. 29, pp. 100–132.
3. Niblack, W. 1986. An Introduction to Digital Image Processing. Prentice-Hall.
4. Rosenfeld, A. and Kak, A.C. 1982. Digital Picture Processing. Volumes 1, 2. Academic Press.

6.2.4 Exercises on Image Segmentation

1. Use the histogram/threshold method to segment the following image into 2 regions.

```

1 2 3 1 3 2 3 1 2 3
2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 8 2 3 1 2
1 2 3 7 8 9 9 8 7 1
2 3 1 8 9 9 8 7 7 2
3 1 2 9 9 8 7 7 8 3
3 1 2 9 9 8 7 7 8 3
1 2 3 1 3 2 3 1 2 3
2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 1 2 3 1 2

```

2. Use the k-means clustering algorithm to segment the above image into 2 regions.

Initialize by labelling all the pixels rows 1 to 5 as class 1, all in rows 6 to 10 as class 2, then iterate... (Don't worry, it terminates very quickly).

3. It should be obvious from the foregoing exercise that a better initialization would make the process terminate even sooner, i.e. choose as

starting means the histogram peaks, and initialize all labels according to the closest of these means. Redo the k-means clustering using this method.

4. Run an edge detector on the above image. Choose an appropriate edge threshold. Thin the edges. Then link edge points. Find regions.
5. Analyze the problem of segmenting an image of a human face. Ideally you want to be able to extract the face from its background.
 - (a) Identify the major problems with respect to the simple models mentioned above.
 - (b) Will we need to segment into multiple classes? – As well as separating “face” from background we may need to segment within the face. Mention problems.
 - (c) Would colour help?
 - (d) How will lighting affect the problem?
 - (e) Suggest a layout for the subject (face), camera, and background.
6. Segmentation for data compression. Take the image given in the first exercise above. Assume the grey levels are coded in *four* bits [0..15].
 - (a) Now as in the first exercise, segment it into two regions – call them regions 0, 1;
 - (b) Calculate the means for each region, m_0, m_1 .
 - (c) code region 0 pixels as 0 and region 1 pixels as 1;
 - (d) Assuming 8 bits for the means (the code-book), how many bits are there in the coded image?
 - (e) reconstruct the image, using the code-book and coded 1-bit image.
 - (f) How would you estimate the loss or distortion caused by the compression?
7. Repeat this compression experiment for the 10×10 three-colour image used at the end of section 6.2.1.

6.3 Mathematical Morphology

6.3.1 Introduction to Mathematical Morphology

This section begins with some basic definitions, symbols and terminology to do morphological image processing. First binary morphological operations are treated. These are then extended to their corresponding grey-level operations. The discussion focusses on the application of these operations to

detection of high intensity peaks, low intensity valleys, and edges in grey-level images; the application is feature extraction in images of human faces.

We first present relevant definitions in terms of – the original basis of mathematical morphology – spatial set theory, and then look at the same topic using a hit-or-miss operator definition and scanned operator approach. Following this, we cover the extension of this approach to grey-level images, and finally composite operations (i.e. those that can be defined in terms of the basic operations of dilation and erosion). Examples are then used to illustrate the powerfulness of these operations.

Basic Morphological Operations

Morphological image processing operations take account of the spatial structure or correspondence between groups of neighbouring pixels. To some extent, therefore, there are similarities with convolution based spatial enhancement operations, e.g. smoothing and edge detection; these are essentially linear operations. On the other hand, morphological operations are based on set theory operations, i.e. union, intersection, or in terms of Boolean algebra: AND, OR; hence they are definitely non-linear.

Both convolution and morphological operations can be considered “geometric”, but, while many morphological operations are implemented by scanning “operator” masks across the image – in the same way as convolution the theory and background of the two are very different.

In the literature there are two distinct methods of introducing morphological operations: (i) using the original spatial set theory – traced back to Matheron (1975) and Minkowski (1903), and used as the basis of Serra's work (Serra, 1982, 1988), (ii) a practical approach, by describing the common operations via their implementation as scanned operators and by discussing their effects and applications. In deference to history, we will mention (i) but quickly move onto (ii) since that interpretation is easier to understand.

Set Theoretic Basis

Consider the binary images shown in the following. The first is called a *structuring element* for reasons soon to become clear. For pixels numbered 0, 1 and 2, it is a 3×3 image. The second is a 16×16 image.

	0	1	2
0	1	1	1
1	1	1	1
2	1	1	1

```

0 0 0 1 1 1 1 0 0 0 1 1 0 0 0
0 0 0 0 1 1 1 1 0 0 0 1 1 0 0 0
0 0 0 1 1 1 1 1 0 0 0 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 0 0 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 0 0 0 1 0 0 0
0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0
0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Let us call the structuring element H . We can represent image H as the set of points where 1 is present:

$$H = \{h\} = \{(0,0), (0,1), (0,2) \dots (2,1), (2,2)\}$$

H equals the set of elements denoted h , or again the set of elements consisting of couples $(0,0), (0,1)$ etc. Thus a binary image can be represented as a set. Our universal set is a subset of integer pairs $\in Z^2$,

$$\{(r,c), \quad 0 \leq r \leq N - 1, \quad 0 \leq c \leq M - 1\}$$

i.e. we are concerned with, at largest, $N \times M$ images; thus, $N = 16, M = 16$ in the case of the image represented above.

For scanning mask operations, which form the basis of mathematical morphological operations, we need to define the translation of H . Translating H to location (r, c) is denoted $H_{r,c}$ and is defined thus:

$$H_{r,c} = \{h_t \mid h_t = h + (r, c), h \in H\}$$

For example, we can translate H to $(5,1)$:

$$H_{5,1} = \{(5,1), (5,2), (5,3), (6,1)\}$$

We can define the complement of H , H_c , as

$$H_c = \{h_c \mid h_c = (r, c), (r, c) \notin H\}$$

Dilation

The dilation of F by H , $F \oplus H$, is defined as

$$F \oplus H = \{c \in Z^2, \quad c = b + h, \quad \forall h \in H, \quad b \in F\} \quad (6.1)$$

This is read: for each pixel couple denoted b in our given image, F , and for each pixel couple denoted h in the structuring element or mask image, H , we translate couple b by h , and we require that the result of doing this is a bona fide pixel couple.

Let us look at an example. First, in the interests of pedagogy, we rearrange H so that its elements are symmetrically placed about the origin. In a program implementation, we can use the same convention: the origin of the structuring element is its centre. We consider only odd-sized structuring elements. Hence,

$$H = \{(-1, -1), (-1, 0), \dots (1, 1)\}$$

The dilation of F by H can be expressed in English as: for all 1 pixels $f = (r, c)$ in image (set) F , the output set C is extended by adding elements $f + h$, for all the elements $h \in H$. The first 1 in F is at location (4,3). Shown below is the extension or expansion at this point: the additional locations are (2,2), (2,3), (2,4) etc.

Structuring element, 3×3 , for pixels numbered -1 , 0 and 1 :

	-1	0	1
-1	1	1	1
0	1	1	1
1	1	1	1

First five rows of input image, F :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0	0	0	1	1	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	0

Dilation, with $b = (3, 3)$:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0(1)	0(1)	0(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0(1)	1(1)	1(1)	1	1	1	0	0	0	0	1	1	0	0	0	0	0	0
0	0	0(1)	0(1)	1(1)	1	1	1	0	0	0	0	1	1	0	0	0	0	0	0

The overall result of this dilation by a 3×3 structuring element is shown next. We can see that dilation swells the shape in F , and fills in small “lakes” and “bays” – using a geographical metaphor, 1 = land, 0 = water.

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

From an implementation point of view, the following is a much more satisfactory definition of dilation:

$$F \oplus H = \bigcup_{h \in H} F_h \quad (6.2)$$

i.e. the union all translations of F by the elements of H . It is easy to verify that equations 6.1 and 6.2 are equivalent. This definition also lends itself to the “scanned operator” interpretation mentioned in the introduction. We cover this interpretation in detail in the section to follow. First we look at erosion and an interpretation in terms of the “hit or miss operator”.

Erosion

The erosion of F by H , $F \ominus H$, is defined as:

$$F \ominus H = \{e \mid e \in Z^2, \quad e + h \in F, \quad \forall h \in H\} \quad (6.3)$$

Expressed in English: a pixel $e = (r, c)$ is in the output set if and only if the translation of e by *all* elements of H are present in F .

The following figures show the input and output images for erosion by a 3×3 mask filled with 1s. The image is eroded by the structuring element

Thus, we can see the erosion of the boundaries of the shape in F , and deletion of small “islands” and “headlands” – using the geographical metaphor, 1 = land, 0 = water, mentioned before.

Recalling equation 6.2, there is a corresponding definition of erosion,

$$F \ominus H = \bigcap_{h \in H} F - h \quad (6.4)$$

where $F - h$ is the translation of F by the reflection of $H = -h = (-r, -c)$ i.e. the intersection of all translations of F by the elements of the reflection of H ; actually, in the case of square structuring elements the reflection of H is identical to H . It is easy to verify that equations 6.3 and 6.4 are equivalent.

Hit or Miss Operations

In hit or miss operations, a mask is scanned over the input image. At each point in the scanning – each pixel – a check is made, whether the pattern in the mask matches the pattern in the image, and the output pixel is set to some appropriate state. Trivially, single pixel “spot-noise” can be removed by scanning with

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

When this pattern is encountered, the output is set to 0, otherwise 1.

An eight-neighbour dilate in terms of hit or miss can be defined as follows (see Pratt, 1991).

$$d[r, c] = 1 \text{ iff at least it or one of its eight-neighbours is } 1$$

If we regard 1-valued pixels as Boolean true, and 0 as false, we can define eight-neighbour dilate as:

$$d[r, c] = f[r, c] \text{ OR neighbour}_1 \dots \text{ OR neighbour}_8$$

We can define a cumulative OR operator, COR, to represent this:

$$d[r, c] = \text{COR}_{j, k \in 8-n} f[j, k]$$

where $j, k \in 8 - n$ means that (j, k) is in the neighbourhood.

Similarly, Pratt (1991) defines eight-neighbour erosion,

$$e[r, c] = 0 \text{ iff it or } \text{any of its neighbours is } 0.$$

We can define a cumulative AND operator, CAND:

$$e[r, c] = \text{CAND}_{j, k \in 8-n} f[j, k]$$

6.3.2 Scanned Operators

We now describe the implementation of erosion and dilation in terms of equations 6.2 and 6.4. This is based on the treatment of Pratt (1991, chapter 15.4) who introduced the conceptually tidier generalized dilation and erosion operations.

Generalized Dilation

Recall equation 6.2:

$$F \oplus H = \bigcup_{h \in H} F_h$$

This can be expanded to make the scanning explicit

$$F \oplus H = d[r, c] = \bigcup_{r' \in H} \bigcup_{c' \in H} f[r, c] r', c' \quad (6.5)$$

The set operations defined here are not easy to visualize; also we need a definition whose program implementation is easy and direct. The following Boolean algebra representation, and based on hit or miss operations, satisfies both requirements.

Regard 1-valued pixels as Boolean true and 0 as false; then the above equation can be expressed as

$$d[r, c] = COR_{r'=r-qr}^{r+qr} COR_{c'=c-qc}^{c+qc}(f[r', c'] \text{ AND } h[r' - r, c' - c]) \quad (6.6)$$

This is depicted graphically below, which shows a mask containing H , placed on the image F , and centred at (r, c) . H extends vertically $-qr$ to $+qr$ on either side of its center, and horizontally, $-qc$ to $+qc$. Thus, the mask is a matrix of size $(2qr + 1) \times (2qc + 1)$; in the example shown, $qc = qr = 1$.

The corresponding elements $f(., .)$ and $h(., .)$ are ANDed, and the results then ORed, to produce the output at $(r, c), d[r, c]$. This is done for every (r, c) – hence the scanned operator view of the operation.

Here is the dilation mask centred on pixel (r, c) :

$h(1,1)$ $f(r-1,c-1)$	$h(1,0)$ $f(r-1,c)$	$h(1,-1)$ $f(r-1,c+1)$
$h(0,1)$ $f(r,c-1)$	$h(0,0)$ $f(r,c)$	$h(0,-1)$ $f(r,c+1)$
$h(-1,1)$ $f(r+1,c-1)$	$h(-1,0)$ $f(r+1,c)$	$h(-1,-1)$ $f(r+1,c+1)$

Analogy with Convolution

The analogy with two-dimensional discrete convolution should be obvious, – see equation 6.6. With convolution we have multiplication (instead of ANDing) of the corresponding elements under the convolution mask, followed by summation (instead of cumulative ORing):

$$d[r, c] = \sum_{r'=r-qr}^{r+qr} \sum_{c'=c-qc}^{c+qc} (f[r', c'] \text{ AND } h[r' - r, c' - c])$$

Generalized Erosion

By analogy, generalized erosion can be defined as,

$$e[r, c] = CAND_{r'=r-qr}^{r+qr} CAND_{c'=c-qc}^{c+qc}(f[r', c'] \text{ OR } h[r' - r, c' - c])$$

6.3.3 Grey-level Morphology

The operations described above work only on binary valued images and structuring elements. They can be extended (Pratt, 1991, chapter 15.6)

to grey-level images by generalizing the Boolean operations to extremum operations:

$$\text{OR}(a, b) \longrightarrow \text{MAX}(a, b)$$

$$\text{AND}(a, b) \longrightarrow \text{MIN}(a, b)$$

Generalized Grey-level Dilation

Hence, extending equation 6.5, (generalized) grey-level dilation can be defined as:

$$d[r, c] = \text{MAX}_{r'=r-q_r}^{r+q_r} \text{MAX}_{c'=c-q_c}^{c+q_c} \text{MIN}(f[r', c'], h[r' - r, c' - c])$$

We constrain H to contain only binary values.

The formulation above allows for arbitrarily-shaped structuring elements, i.e. the shape is “drawn” within the rectangularly shaped mask: $h(., .) = 1$ or 0.

Often, we deal only with rectangular structuring elements, so that generalized grey-level dilation can be defined as:

$$d[r, c] = \text{MAX}_{r'=r-q_r}^{r+q_r} \text{MAX}_{c'=c-q_c}^{c+q_c} f[r', c']$$

Generalized Grey-level Erosion

Likewise, we can define grey-level erosion as,

$$e[r, c] = \text{MIN}_{r'=r-q_r}^{r+q_r} \text{MIN}_{c'=c-q_c}^{c+q_c} \text{MAX}(f[r', c'], h[r' - r, c' - c])$$

and, for the special case of rectangular structuring elements:

$$e[r, c] = \text{MIN}_{r'=r-q_r}^{r+q_r} \text{MIN}_{c'=c-q_c}^{c+q_c} f[r', c']$$

Completeness of Extension

By substituting 1 for true and 0 for false, it is easy to verify that the extension is complete: i.e. grey-level operations applied to binary images give the same result as the equivalent binary operation.

6.3.4 Composite Operations – Open and Close

Morphological open is defined as erosion followed by dilation.

Likewise, close is dilation followed by erosion.

6.3.5 Program Implementation

In the following C code, *both* dilation and erosion are carried out, depending on the Boolean dilate (== FALSE for erosion). Moreover, it does open and close by using two passes.

As it is, the routine uses rectangular structuring elements. We have left evidence (commented-out) of the general shaped element.

```

for(r=r1;r<=rh;r++){
    for(c=c1;c<=ch;c++){
        if(r<sr||r>rh1||c<sc||c>ch1){ /*boundaries */
            rv=0.0; IMDput(&rv,d,r,c,pd); continue;
        }
        if(binary)val=dilate?0:1;
        else rv=dilate?min:max;

        rr1=MAX(r1,r-qr);rrh=MIN(rh,r+qr);
        ccl=MAX(c1,c-qc);cch=MIN(ch,c+qc);
        for(rr=rr1,hp=0;rr<=rrh;rr++){
            for(cc=ccl;cc<=cch;cc++,hp++){
                /*hh=h[hp];*/
                /*rectangular structuring element*/
                IMDget(&rv1,d,rr,cc,ps);
                if(binary){
                    val1=rv1<1.0?0:1;
                    if(dilate)val=val||(val1);/*prev. (val1&&hh)*/
                    else val=val&&(val1);/*prev. (val1||!hh)*/
                }
                else { /*grey-level*/
                    if(dilate)rv=MAX(rv,rv1); /*prev. dilate&&hh*/
                    else rv=MIN(rv,rv1);
                }
            }
            if(binary)rv=val;
            IMDput(&rv,d,r,c,pd);
        }
    }
}

```

6.3.6 Examples of Morphological Operations

In the following figures, we give examples of practical applications of morphological operations. In all cases, a 3×3 structuring element is used.

We use an image consisting of some touching blocks and some straight lines.

Next we see erosion. Erosions of edges and of protrusions takes place. The effect of this can be clearly seen when we subtract the eroded image from the original image.

Next we see dilation, where the swelling effect is clearly evident.

Finally we show (Figures 6.2, 6.3) the opening (dilate followed by erode), and closing (erode followed by dilate).

We can go further if we wish. Morphological edge highlighting, using the difference (dilate – erode), uses the fact that dilate swells the edge regions outwards and inwards, while erosion is the opposite, hence, the difference highlights regions that are subject to erosion and swelling – especially both, i.e. edges.

Peak (peak = localised high intensity) highlighting can be performed by subtracting, from the original, the open using a structuring element that is large compared to the peak's spatial size; this can be explained by the tendency of opening to erase peaks.

Valleys (valley = thin channel of low intensity) can be highlighted by subtracting, from the original, the close using a structuring element that is large compared to the valley's spatial size; this can be explained by the tendency of close to block up valley regions.

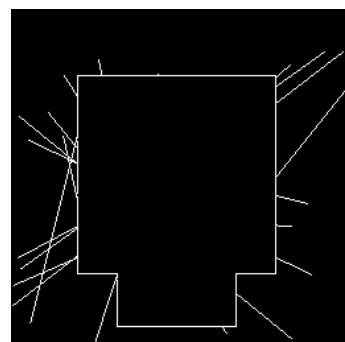
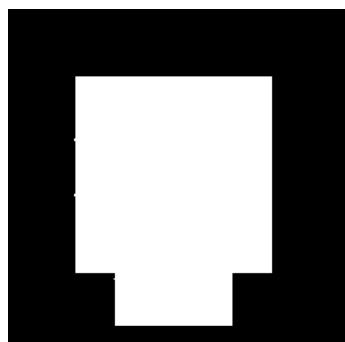
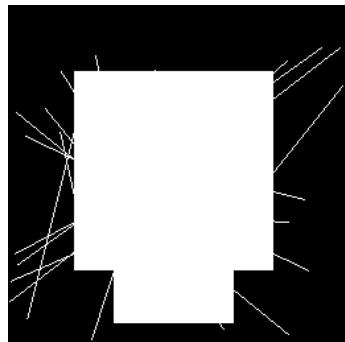


Figure 6.1: Upper left, original image. Lower left, erosion. Lower right, difference between original and erosion.

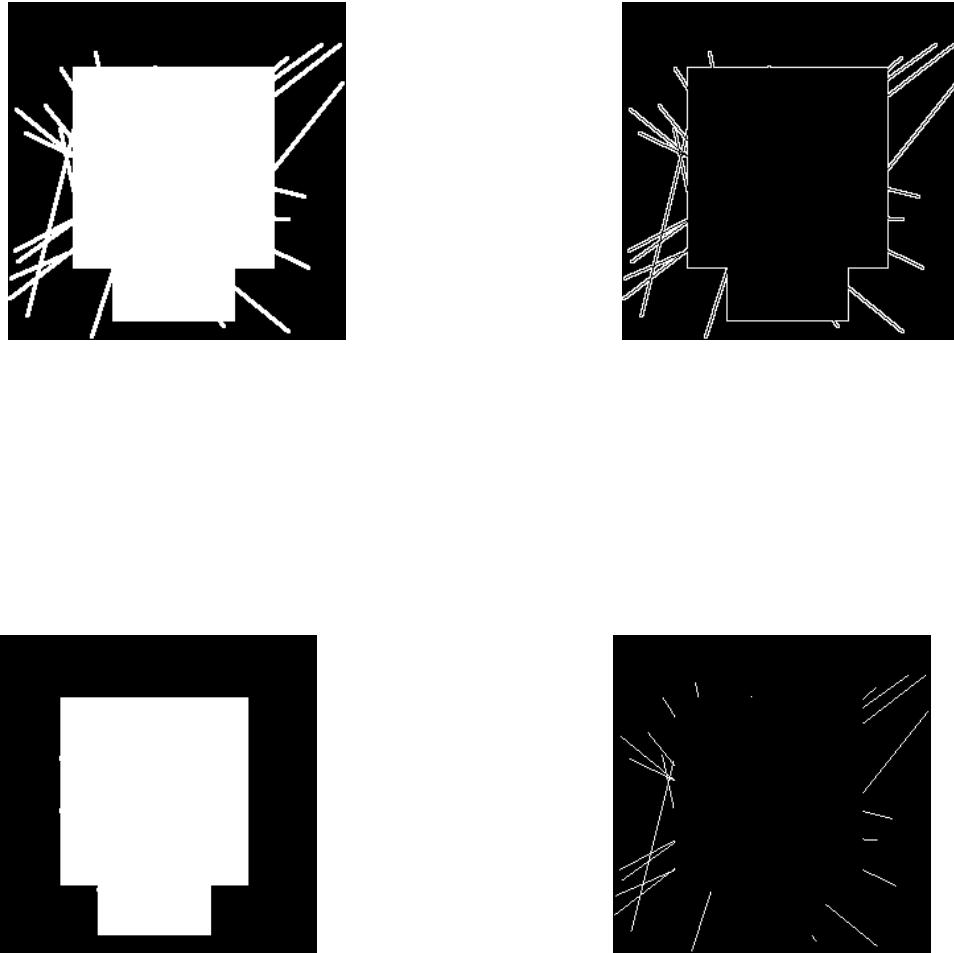


Figure 6.2: Upper left and right: dilation, and difference between original and dilation. Lower left and right: opening, and difference between original and opening.



Figure 6.3: Left, closing. Right, difference between closing and original.

6.3.7 To Read Further on Mathematical Morphology

1. Haralick, R.M., S.R. Sternberg, and X. Xhaung. 1987. Image analysis using mathematical morphology. *IEEE Trans. Pattern Analysis and Machine Intelligence*. Vol. PAMI-9, No. 4, July.
2. Jain, A.K. 1989. *Fundamentals of Digital Image Processing*. Prentice-Hall.
3. Low, A. 1991. *Introductory Computer Vision and Image Processing*. McGraw-Hill.
4. Matheron, G. 1975. *Random Sets and Integral Geometry*. Wiley.
5. Minkowski, H. 1903. *Volumen und Oberfläche*. *Math. Ann.* Vol. 57, pp. 447-495.
6. Pratt, W.K. 1991. *Digital Image Processing*, 2nd ed. Wiley.
7. Serra, J. 1982. *Image Analysis and Mathematical Morphology*. Academic Press.
8. Serra, J. 1987. *Image Analysis and Mathematical Morphology: Theoretical Advances*. Academic Press.

9. Shackleton, M.A. and W.J Welsh. 1991. Classification of facial features for recognition. Proc. IEEE 1991 Computer Vision and Pattern Recognition, pp. 573-579.

6.3.8 DataLab-J Demonstrations on Mathematical Morphology

Call the following script `mob.dlj` and run it with the command `java Dlj mob.jlj`. It constructs the various images shown as examples of application of morphological operations.

```
----- mob.dlj -----
//j.g.c. 18/6/98
//simple tests on binary morphological operators
//-----
//generate rectangle, top left (50,50), bottom right (200,200)
//put in image 0
grect
0
256,256, 50,50,200,200
//another rect, (180,80) (240,170)
grect
1
256,256, 180,80,240,170
//generate 20 random lines
grlines
2
256,256, 20
//now we add (actually binary OR) the two rects and the random
//lines an put the result in image 3
mobor
1,2
3
mobor
3,0
3
//save the combined image
savei
3
moborig,0,0,0
//erode original with a 3x3 structuring element
mobe
3
4
3
//and save in file "moberod.pgm"
savei
4
moberod,0,0,0
//dilate original with a 3x3 structuring element
mobd
3
5
3
```

```
savei
5
mobdil,0,0,0

//open original with a 3x3 structuring element
mobo
3
6
3
savei
6
mobopen,0,0,0
//close
mobc
3
7
3
savei
7
mobclos,0,0,0
//XOR original, eroded to show difference
mobxor
3,4
8
savei
8
mobderod,0,0,0
//original, dilated
mobxor
3,5
9
savei
9
mobddil,0,0,0
//original, opened
mobxor
3,6
10
savei
10
mobdopen,0,0,0
//original, closed
mobxor
3,7
```

```

11
savei
11
mobdclos,0,0,0
end

```

6.3.9 Exercises on Mathematical Morphology

1. Using the 3×3 structuring element

```

1   1   1
1   1   1
1   1   1

```

show (i) dilation and (ii) erosion on the following image:

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 1 1 0
0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

2. Morphological opening is an erosion followed by a dilation. Find the opening of the image given above.
3. Morphological closure is a dilation followed by an erosion. Find the closure of the image given above.
4. Find (i) the original image minus the erosion, and (ii) the original image minus the dilation. Comment on what you find in the case of both results.
5. Using the results found for erosion, dilation, open, and close, comment on the use of mathematical morphological operations for the purposes of (i) noise removal, and (ii) edge detection.
6. Indicate how and where morphological operators could be usefully employed in the application area of face recognition.

6.4 The Wavelet Transform

6.4.1 Introduction

We have looked at the Fourier transform in Chapter 3. We saw that it allows a frequency-based decomposition of our image or signal data. The data can be exactly reconstituted from the transform. As with any transform, we can carry out certain surgical interventions before reconstituting the data. We can, for instance, remove high frequencies, which amounts to saying that noisy and edgy parts of the data will be smoothed.

The wavelet transform shares some of the same objectives. Both transforms lead to a new representation of the data, which can serve a useful purpose such as allowing us to remove components (strata, if you like) which are a nuisance to our understanding of the data. Both transforms also allow us to *exactly* reconstitute our data if this is required. Different perspectives are provided by these two transforms on the high and low *frequencies* in our data, i.e. the relative local “activity” or variation.

High frequency data means rapidly changing data, which could be, but does not have to be, related to noise. Low frequency means slowly varying. Band-pass filtering our data means that we kill, say, high frequencies in our data, leaving behind a band of low pass frequencies and then reconstitute an approximation to our input data from this. If we do remove high frequencies, then this is the same as running a low-pass filter which is tantamount to smoothing or blurring our data. We can achieve such objectives with either the wavelet or Fourier transform.

Interfering in this way in transform space leads to denoising, or “data cleaning”, or generally data filtering.

The wavelet transform has become popular in recent years even though its roots go back many decades. The Haar wavelet transform goes back to the early 20th century.

Although lots of justifications for the wavelet transform could be given, we will just mention two major justifications. Firstly, it is a useful tool for finding faint features in an image – it is a mathematical and algorithmic “microscope”. This objective is favoured by the wavelet transform which we will look at in the next section. Secondly, the wavelet transform is found in practice to provide a sparse set of values (many zeros or many very small values) representing the original data. As a direct byproduct it is a useful preprocessing step towards the goal of compression. We will discuss this objective when we describe the Haar wavelet transform.

We will describe two wavelet transform methods. The first, the à trous method, is representative of so-called redundant methods. The second method, the Haar transform, is representative of a large family of orthonormal methods. We mention these facts in passing, to note that there are many wavelet transforms available, but that the ones we will deal with are useful

entry points into this rapidly growing and very popular field.

6.4.2 The à trous Wavelet Transform

We will perform a sequence of operations on our image which appears at first sight to be a strange thing to do. We smooth the image, f_0 , to form $f_1 = s(f_0)$. Smoothing uses convolution (Chapter 3). We smooth the image we thereby get to form $f_2 = s(f_1)$. We continue doing this for a user-specified number of times, often about $p = 4$ or 5.

That looks like a counterproductive thing to do. Now, we take the succession of coarser images, f_0, f_1, \dots, f_p , and form a set of *detail signal* images as follows: $f_0 - f_1, f_1 - f_2, \dots, f_{p-1} - f_p$. We have trivially

$$f_0 = (f_0 - f_1) + (f_1 - f_2) + \cdots + (f_{p-1} - f_p) + f_p$$

or

$$f_0 = d_1 + d_2 + \cdots + d_p + f_p \quad (6.7)$$

where the last image here is the coarsest image which we are considering.

These detail images contain structure which was present at one scale, and which did not survive the blurring in going to the next scale.

Equation 6.7 is an image expansion in the form of a sum of component images. These component images, $d_1 = (f_0 - f_1)$ etc., do not form an independent (i.e. orthogonal) system, whereas in the case of other wavelet transform approaches, including the Haar wavelet transform explored below, orthogonality of an analogous decomposition is considered a desirable property.

The above decomposition of f_0 gives us a fine-to-coarse transition of information. To understand what the detail images convey, take d_1 . This detail images provide information on features which “died” in the blurring at stage f_1 . Such faint aspects of the image f_0 did not survive the blurring, since they were too small and faint. We talk about the *scale* of such features. The succession of detail images, d_1, d_2, \dots , yields information which is scale-related.

The algorithm used can be stated as follows. Index k ranges over all pixels.

1. We initialize i to 0 and we start with an image $f_i(k)$.
2. We increment i , and we carry out a discrete convolution of the data $f_{i-1}(k)$ using the filter h . The distance between a central pixel and adjacent ones is 2^{i-1} .
3. From this smoothing we obtain the discrete wavelet transform, $d_i(k) = f_{i-1}(k) - f_i(k)$.

4. If i is less than the number p of resolutions we want to compute, then go to step 2.
5. The set $\mathcal{W} = \{d_0, d_1, \dots, d_p, f_p\}$ represents the wavelet transform of the data.

In one dimension, we take $h = (\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16})$. In two dimensions, we can use this same kernel sweeping out the horizontal (column) direction, followed by a sweep over the vertical direction (rows). The explanation for this choice is that it is associated with a B_3 spline, a function which is known for its good interpolation properties. The only messy aspect is handling the extremities of the signal. A few choices are open to us. We recommend reflection in the extremity, giving a “mirror” effect.

That is the general idea. Various other issues are also of importance. The image f_1 is most usually a half-resolution version of input image f_0 , and f_2 is a half-resolution version of f_1 , and so on. This is almost always the case, even if there is no necessity for doing things this way. To justify this *dyadic* decomposition, we can point to certain aspects of human perception being logarithmic in receptiveness, and to the computational advantages of doing things in this way.

In the case of the Haar or Daubechies methods (see Press et al., 1992, for an overview), such diminishing of resolution is made use of to halve the dimensionality of the image which is output. I.e. if f_2 is reduced in resolution by 2, then we can benefit by reducing its dimensionality (along each dimension) by 2 also. This is referred to as *decimation* or down-sampling and means that from a two-dimensional input image, we arrive at a transform which can be represented as a pyramid. Or, in the case of a one-dimensional transform consisting of $d_1, d_2, \dots, d_p, f_p$, the transform result can all be fit into an n -length vector of values, where n is the length of the f_0 signal.

The à trous method tackles the issue of halving resolution at each stage somewhat differently. Decimation is not used with this method, which means that d_1, d_2, \dots continue to have exactly the same dimensionality as the input data, f_0 . Therefore the à trous transform is a *redundant* one. This redundancy may be of great help to us in exactly demarcating features thrown up by the various detail images. We relate the features to what we can see, or not see, in the original image, and there is no *aliasing* introduced by the decimation.

In the à trous method the succession of smoothings or blurrings is carried out using a *scaling function*. Say that the smoothing is achieved by a length 5 scaling function, implying that at each image pixel value (we'll take a one-dimensional view for simplicity) the surrounding pixel values $(-2, -1, 0, +1, +2)$ enter into the calculation. Then at the following smoothing, we will use pixel values $(-4, -2, 0, +2, +4)$. At the following one again, we use $(-8, -4, 0, +4, +8)$. This is a versatile way to enforce two-fold diminishing of resolution. It also gives rise to the name à trous, “with holes”.

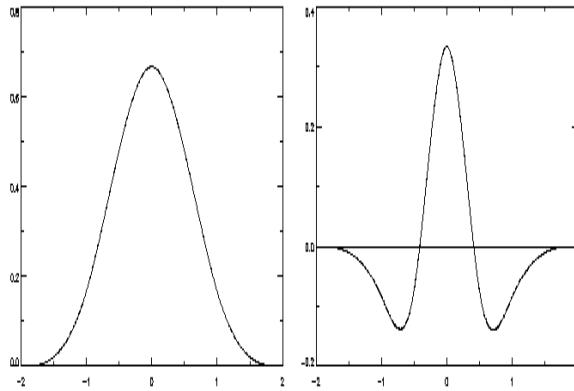


Figure 6.4: À trous wavelet transform. The scaling function here is defined by the B₃-spline function, which is a function with good interpolation properties.

The images f_1, f_2, \dots are formed by repeatedly smoothing. Consider these as being created instead directly from the original image, f_0 . The smoothing means that we convolve, which involves translating a function – the *scaling function*. At each successive scale, we *dilate* this function. One possible form for the scaling function which is a very practical one and also one that we have used very extensively is shown in Figure 6.4. We see that it is like a Gaussian, but of compact support (i.e. the wings are clipped).

We can consider similarly the direct determining of d_1, d_2, \dots from f_0 . The function needed to do this is a translated and dilated version of a *mother wavelet* function. The one which of necessity goes hand in hand with the scaling function described above is also shown in Figure 6.4. These dilated versions of the wavelet function constitute a new basis for the original data signal. We have already noted that in the case of the à trous method, they are not mutually orthogonal. We can also note a parallel with the Fourier transform: the latter defines a new basis in terms of sine and cosine functions.

The original data, f_0 , can be recreated exactly from the wavelet transform or decomposition. A property which follows from this is that the wavelet coefficient values in d_1 or d_2 etc. are of zero mean.

We have discussed the wavelet transform in terms of a discrete transform, and also in terms of a dyadic scale. A more general, *continuous wavelet transform* can also be considered. Mathematically, this is given by

$$W(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(x) \psi^*(\frac{x-b}{a}) dx$$

where W is the set of wavelet coefficients of the function $f(x)$; ψ is the

mother wavelet (and we take its complex conjugate above in the convolution); and a and b are, respectively, the scale and the position parameters. An admissibility condition defines the circumstances under which exact recreation of f from the transform values, W , is possible.

6.4.3 Examples of the À Trou Wavelet Transform

Figure 6.5 shows one important application, denoising. The denoising here is a data-driven, based on Gaussian noise at each wavelet transform scale (see Starck et al., 1998, for more background on filtering).

Figure 6.6 shows a succession of scales resulting from the à trous wavelet transform of the (noisy) image shown in Figure 6.5. By adding these images, we can reconstitute exactly the input image.

Figure 6.7 shows a different view of this, using a perspective plot. The plateaus on the peaks are for display purposes only, and show 3σ values (σ is the standard deviation of the given resolution level or scale). Again exact reconstitution of the input data can be carried out.

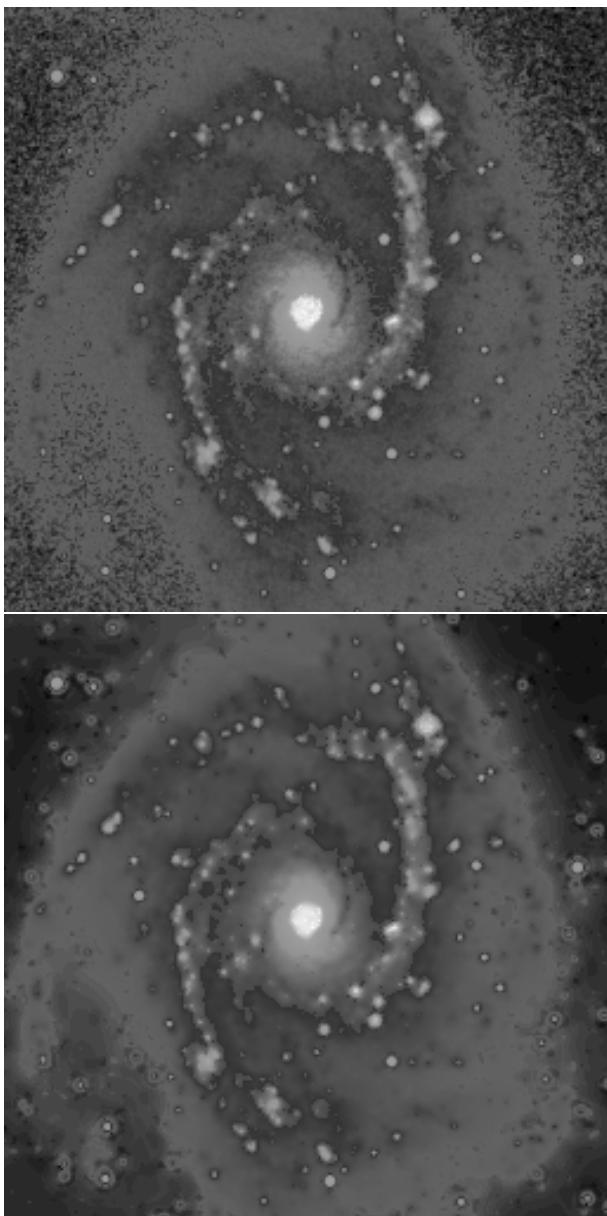


Figure 6.5: Top: galaxy NGC 2997 with stars, perhaps other objects, and background, noise, and detector faults. Bottom: a noise-filtered version of this image, where the denoising was carried out in wavelet space.

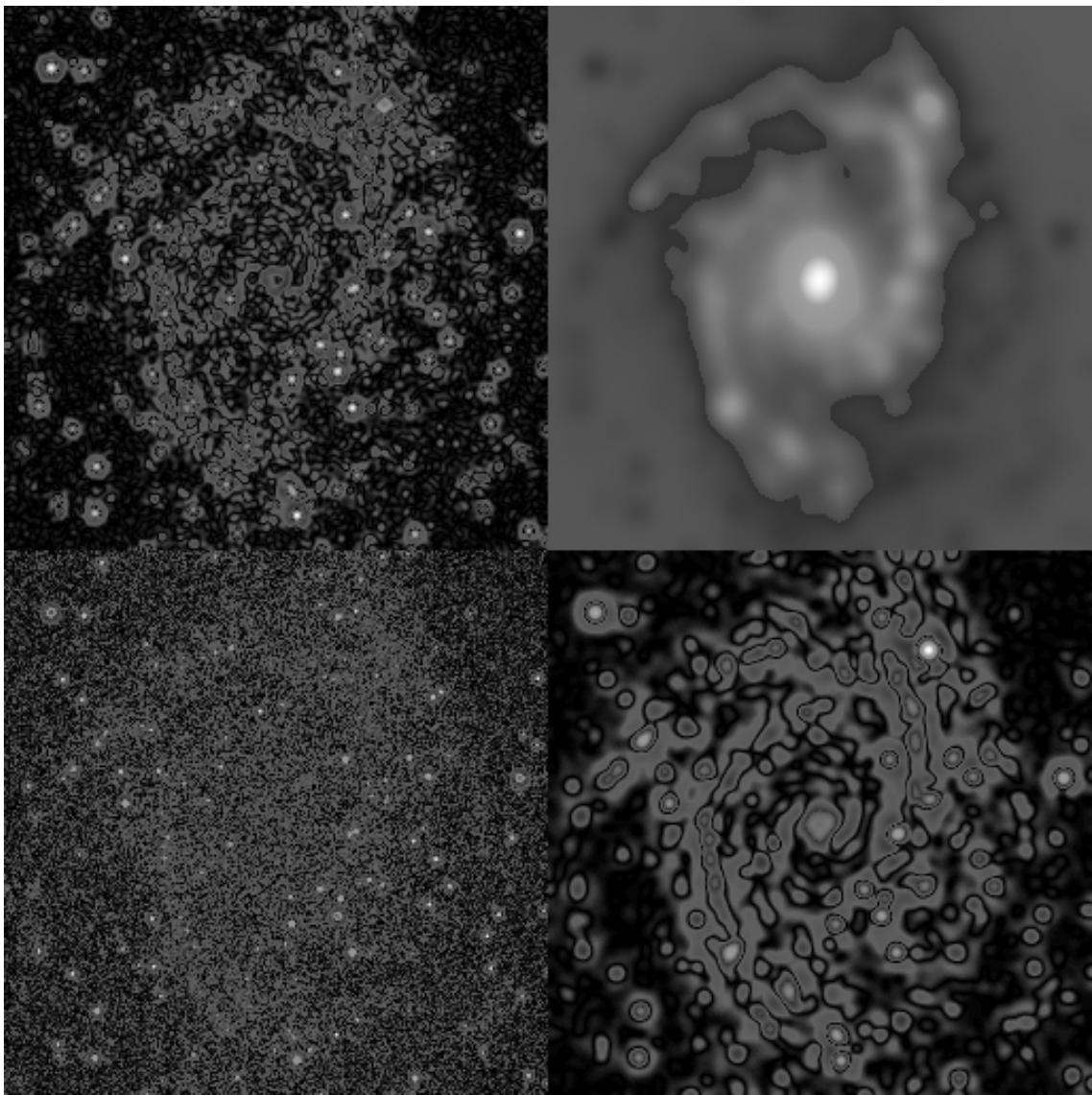


Figure 6.6: The à trous transform (3 resolution levels, together with the final smoothed version of the image) produces these images for the NGC 2997 image. Read in this order: lower left, upper left, lower right, upper right. The “shadow” effect (especially in the upper right) is due to the colour lookup data used in the display.

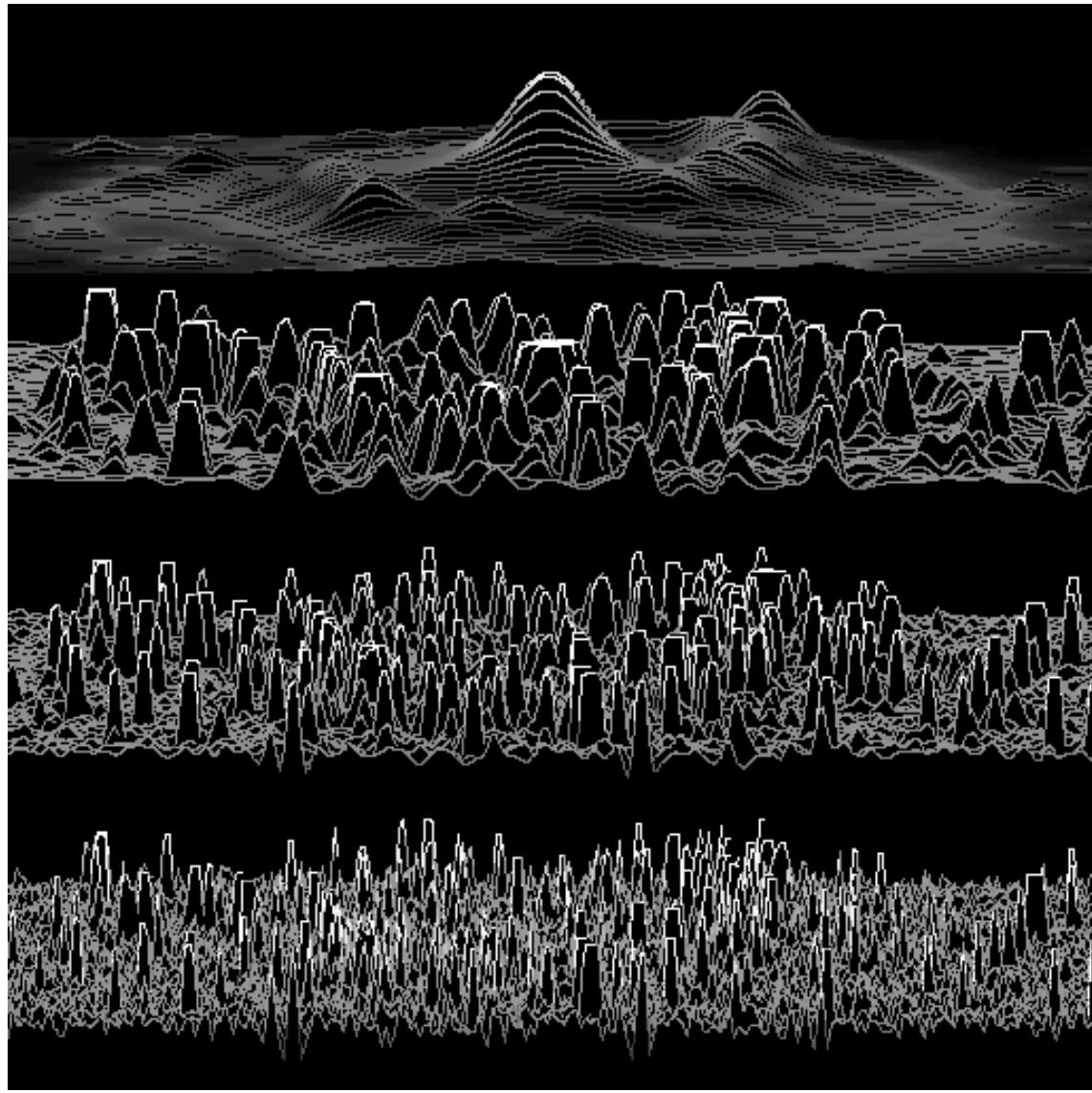


Figure 6.7: A perspective plot of the à trous transform of NGC 2997. Read from bottom to top.

6.4.4 The Haar Wavelet Transform

We consider the one-dimensional set (see Mulcahy, 1998, on which this description is based) 64, 48, 16, 32, 56, 56, 48, 24. We will use it, for the moment, as a one-dimensional image (or spectrum as it is called in certain fields).

We carry out a set of averagings and differencings on this string, to yield

64	48	16	32	56	56	48	24
56	24	56	36	8	-8	0	12
40	46	16	10	-	-	-	-
43	-3	-	-	-	-	-	-

Looking at row 2 here, 56 is the average of 64 and 48; 24 is the average of 16 and 32; etc. So we get 4 averaged values on row 2, from the 8 values on row 1. On row 3, we get 2 averaged values from the 4 values we have just constructed on row 2. And so on.

Now look at the detail values in bold font in the data above. The average value of 64 and 48 is 56. The first bold value on row 2, 8, indicates that $56 + 8$ and $56 - 8$ are needed to reconstruct the input values. Next, the average of 16 and 32 is 24. The bold value -8 (the second bold value on row 2, reading from left to right) indicates that $24 + (-8)$ and $24 - (-8)$ are needed to reconstruct our original values.

We have carried out just additions and subtractions, apart from division by 2. The ability to reconstruct our original data from the transform values is clear. In fact there is something more we can ask ourselves: do we really need to keep all the values arrived at in the table above? The answer to that is that we simply need the detail values (in bold) and the last averaged (smoothed) value (which, above, happens to be 43). That's all. Note how the situation is reminiscent of the à trous method, which decomposed the original data into the detail coefficients plus the last smoothed version of the data.

The wavelet transform of the input signal above, (64, 48, 16, 32, 56, 56, 48, 24), is then the detail values, together with the last smooth value: from left to right, (43, -3 , 16, 10, 8, -8 , 0, 12). This is all that is needed to exactly reconstruct the input data.

One major interest of the wavelet transform is that if detail coefficients are put to zero (or lowered in value in an adaptive or fuzzy way), then reconstruction of the original data is often very good, and we may be able to denoise the data in doing that.

Question: do this using the numeric data given above. Set -3 , a small detail coefficient in absolute value, to 0 and check the effect on the original data by plotting both original and filtered data. Then set detail coefficients of value -3 , -8 and $+8$ to zero and check the effect. How does the result compare to the original data? You will find that it is not exactly the same, but is close to it, getting progressively less close as more wavelet transform values are set to zero.

A matrix formulation for the numerical example considered above is as follows. Of course, it carries over to any such example, one-dimensional or higher dimensional. We form the first row of the transform by premultiplying our 8-valued vector by the following 8×8 matrix:

$$\begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

For the next row, we premultiply our 4-valued vector by the following 4×4 matrix:

$$\begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} \end{pmatrix}$$

Finally, the last row is formed by premultiplying the 2-valued vector by the 2×2 matrix:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$

We can easily verify orthonormality properties for these matrices. Reconstructing the input data at each stage is easy, using matrix inversion. Furthermore, if we wish, we can use redundant rows and columns with 1-values on the diagonal which carries over input values into the outputs produced at each stage. This has the advantage that we can keep 8-valued vectors at each stage – we will use 8×8 transform matrices at each stage. Then the entire transform can be made more compact, by just taking the product of the three 8×8 matrices, to produce – of course – an overall 8×8 matrix. In other words, the entire wavelet transform can be neatly expressed by a product with a particular orthonormal matrix.

We can do the same thing for the à trous method, but the result will be much more messy (due to handling of data extremities). The transform matrix will not be orthonormal. This is a big difference between the two wavelet transform methods we are dealing with in this chapter. Another difference is that the Haar transform involves half the amount of data at each resolution level.

The wavelet function, the mother wavelet, in the case of the Haar transform is defined by

$$1 \text{ when } 0 \leq x < \frac{1}{2}$$

$$-1 \text{ when } \frac{1}{2} \leq x < 1$$

$$0 \text{ otherwise}$$

and this provides the detail signal values. The scaling function, providing the smoothed or averaged values, is given by

$$1 \text{ for } 0 \leq x < 1$$

$$0 \text{ otherwise}$$

Figure 6.8 shows what this wavelet function looks like. Question: by plotting it, show that the Haar scaling function is a “box” function. Question: using a few numerical values, show how convolution with a box function is equivalent to averaging. Show how a dilated version of this box function can perform averaging for scales beyond the first one. Question: again using a small numerical example, show how convolution with the Haar wavelet can be used to carry out differencing. Show how a dilated version of this same function can perform the differencing at scales beyond the first one.

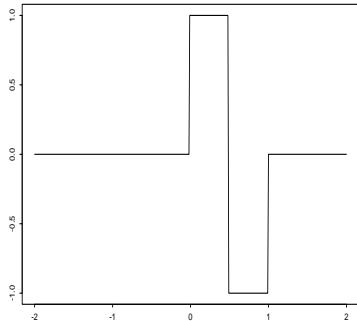


Figure 6.8: The Haar wavelet function.

Comprehensive and very readable background on the Haar wavelet transform is available in Mulcahy (1998). Two further topics are dealt with there, which the interested reader should check out. Firstly, the concept of *multiresolution analysis* which is closely associated with a seminal article by Mallat. It was Mallat’s description of the wavelet transform in terms of projections onto a set of embedded spaces which gave the wavelet transform a solid basis in image processing. Before that, it was more strictly a mathematical theory of signal processing. The roots, before Mallat, of the wavelet transform in modern times had been mainly to be found in the mathematical and geophysics areas.

6.4.5 Examples of the Haar Wavelet Transform

The figures shown in Figure 6.9 illustrate compression. Thus the input (upper left) is highly compressed, and then uncompressed. The compression method is a lossy one in all cases. The result of doing this is seen in the remaining parts of Figure 6.9. Given that there are 65536 values in the original image, and again 65536 values in the Haar wavelet transform (question: why is this?), we find in these cases just 1310 non-zero values retained for a fairly acceptable rebuilding of the data (upper right).

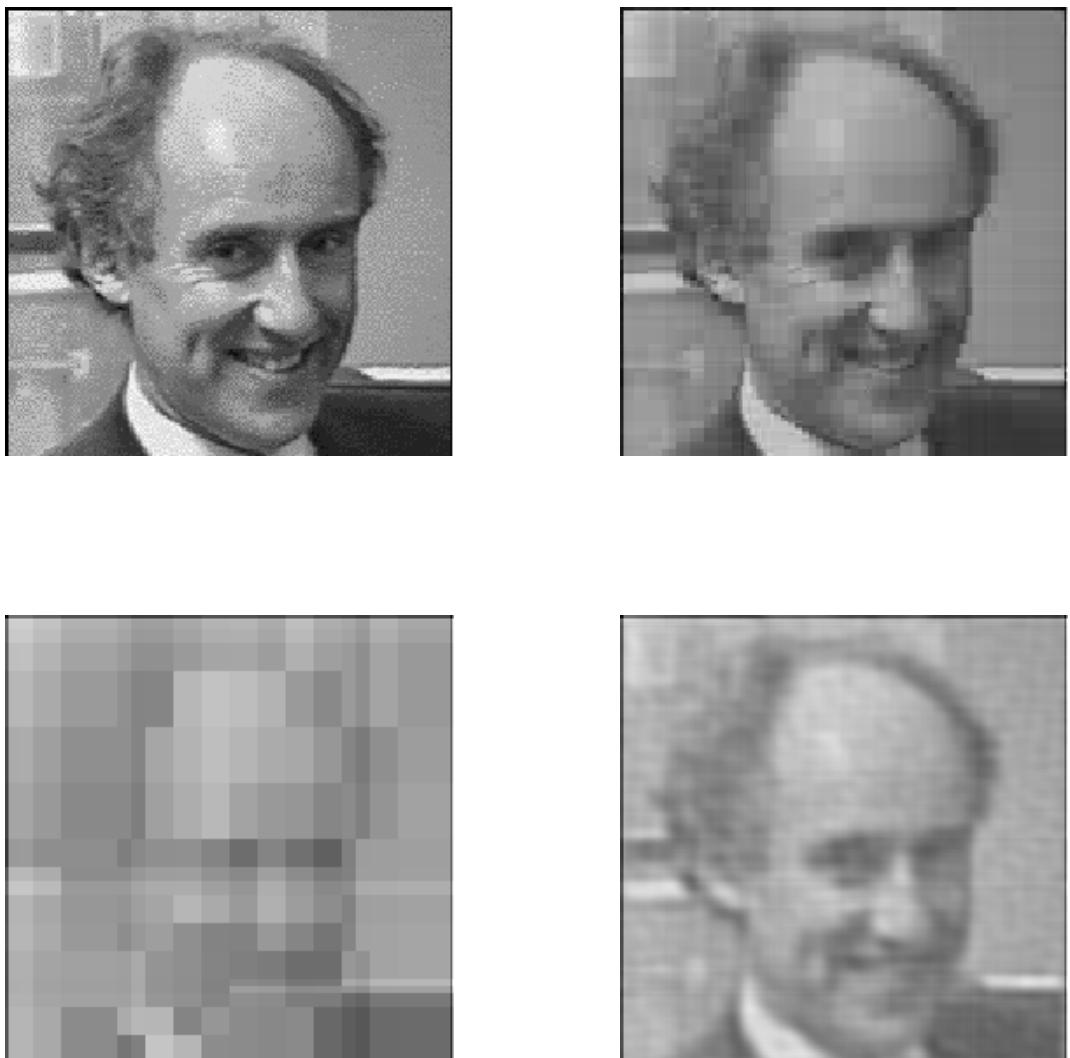


Figure 6.9: Upper left, original image of Jon Campbell, of dimensions 256×256 . Upper right: 2% of the Haar wavelet coefficients have been retained (or 1310 coefficient values in all). Lower left: 0.1% of the coefficients have been retained, or just 65 values. Some aspects of the image are perceptible (by squinting). Lower right: for comparison, 2% of the coefficients of a discrete Fourier Transform have been retained.

6.4.6 To Read Further on the Wavelet Transform

1. Mulcahy, C. 1998. Workshop on wavelets. Course notes 40 pp.
(This covers the Haar wavelet transform from the point of view of applications relating to compression. It goes on to look at the “lifting scheme” which allows a large family of wavelet transforms to be defined.)
2. Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. 1992. Numerical Recipes. 2nd edition. Cambridge University Press.
(C and Fortran code. The emphasis in Chapter 13 is on the Daubechies transform, and on the compression application.)
3. Starck, J.-L., Murtagh, F., and Bijaoui, A. 1998. Image and Data Analysis: The Multiscale Approach. Cambridge University Press.
(This covers the à trous wavelet transform with many applications. Details of an associated software package can be found at Web address <http://visitweb.com/multires>. Further examples from application domains such as astronomy, medicine, information science, cluster and pattern detection, financial engineering, telecommunications traffic modelling, geographic information systems, and other areas, can be found at Web address <http://hawk.infm.ulst.ac.uk:1998/multires>)
4. Wavelet Digest, <http://www.wavelet.org/wavelet>
(An important source of information on the latest developments in this field.)

6.4.7 DataLab-J Demonstrations of the Wavelet Transform

The following script, `haar.dlj`, carries out the examples shown above using the Haar wavelet transform. It uses image `jon.pgm` as input. To run it, do `java Dlj haar.dlj`.

```
//haar24.dlj j.g.c. 3/9/98
//tests on 2-d Haar transform
//data compression clipping smaller components
read
0
jon.pgm
haar2f
0
1
```

```
//coefficients, 2 pcent -- 1310 coeffs. retained
tftrunc
1
2
0, 2.
haar2i
2
3
savei
3
haar22pc,0,0,0
//components, 0.1 pcent -- 65 coeffs. retained
tftrunc
1
4
0, 0.1
haar2i
4
5
savei
5
haar201pc,0,0,0
//DFT for comparison
2ftrri
0
6
//components, 2 pcent
tftrunc
6
7
0, 2.
2ift
7
8
savei
8
dft22pc,0,0,0
end
```

The following script carries out an à trous wavelet transform decomposition. It uses image `jon.pgm` and to run it, do `java Dlj atrous.dlj`.

```
//atrous21.dlj j.g.c. 3/9/98
//tests on 2-d a-trous wavelet transform
read
0
jon.pgm
atrous2
0
1
4
savei
1
at20,0,0,0
savei
1
at21,1,0,0
savei
1
at22,2,0,0
savei
1
at23,3,0,0
savei
1
at24,4,0,0
end
```

6.4.8 Exercises on the Wavelet Transform

1. Take the one-dimensional signal (25, 33, 33, 17, 28, 14, 12, 6) and determine the Haar wavelet transform.

Proceed as follows. Take the first pair of values, and form their average. Take the next pair of values and form their average. Continue until all data values have been processed. Next write down the succession of difference terms which are needed by the average values to give back the values on which the average was calculated. (I.e., if a and b are values in the original data, then the average is $(a + b)/2$ and the difference term is $a - (a + b)/2$.) Continue for successive levels.

Answer: the Haar wavelet transform of this set of values is (21, 6, 2, 6, -4, 8, 7, 3), where we have written the final average term, followed by detail coefficients.

The implementation of the above algorithm gives:

25	33	33	17	28	14	12	6
29	25	21	9	-4	8	7	3
27	15	2	6	-	-	-	-
21	6	-	-	-	-	-	-

2. What are the requirements for the Haar wavelet transform algorithm, as described here? Comment on (i) length of the input signal, and (ii) integer versus real data types.
3. Given the input data signal (25, 33, 33, 17, 28, 14, 12, 6), the à trous wavelet transform with 2 scales, plus the final smooth of the data, is given by

scale 1: (-5, 3, 4.8125, -7.5625, 6.9375, -2.6875, 0, -4)

scale 2: (2.02344, 2.53125, 2.33984, 1.17969, 0.492188, -1.12891, -3.84375, -5.16406)

scale 3: (27.9766, 27.4688, 25.8477, 23.3828, 20.5703, 17.8164, 15.8438, 15.1641)

Graph these scales. Comment on why the values are so small in the case of scales 1 and 2, and much larger for scale 3.
4. The à trous wavelet transform, 3 levels, of the one-dimensional dataset (0, 0, 4, 4, 4, 8, 4, 4, 0, 0, 8, 0, 0, 4, 4, 0) produces the following result:

scale 1: (-0.5, -1.25, 1.25, 0.0, -1.0, 2.5, -0.75, 1.0, -1.75, -2.25, 5.0, -2.25, -1.75, 1.5, 1.25, -2.5)

scale 2: (-1.6875, -1.125, -0.125, 0.546875, 1.10938, 1.46875, 0.921875, -0.453125, -1.26563, -0.40625, 0.53125, -0.125, -0.625, 0.078125, 0.296875, 0.03125)

scale 3: (2.1875, 2.375, 2.875, 3.45313, 3.89063, 4.03125, 3.82813, 3.45313, 3.01563, 2.65625, 2.46875, 2.375, 2.375, 2.42188, 2.45313, 2.46875)

Plot these. Verify that scale 1 pinpoints the large isolated spike. Verify that scale 2 favours more the left block and spike. Finally, verify that the final smoothed version of the data again favours the stronger signal to the left of the data.

Comment on the objectives of an à trous wavelet transform, in regard to superimposed resolution scales in a dataset.

Chapter 7

Pattern Recognition

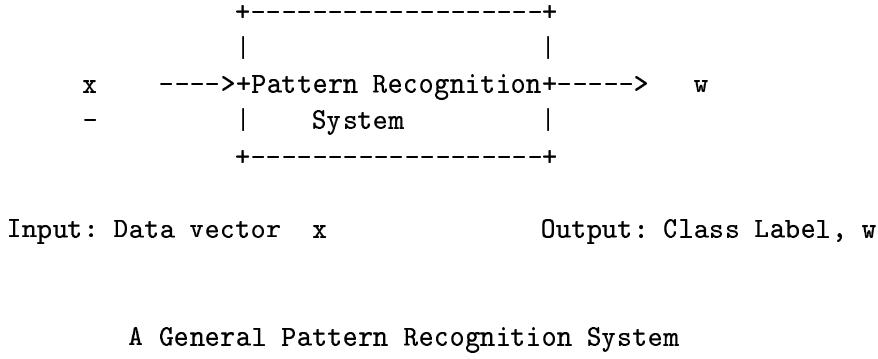
7.1 Introduction

Much of our interaction with our environment requires recognition of ‘things’: shapes in a scene (text characters, faces, animals, plants, etc.), sounds, smells, etc. If we are to automate certain tasks, clearly we must give machines some capability to recognise. And, obviously, from the point of view of this module we must pay attention to automatic recognition of patterns in images. Indeed, the term ‘pattern-recognition’ is strongly linked to image processing, particularly for historical reasons: the first studies in so-called pattern recognition were aimed at optical character recognition.

However, the allusion to human recognition of ‘things’, in the initial paragraph, is merely by way of motivation; usually, it will not be helpful to try to imitate the human recognition process – even if we claim to know how it is done! and no matter how much we would like to emulate its marvellous abilities.

On the other hand, pattern recognition encompasses a very broad area of study to do with automatic decision making. Typically, we have a collection of data about a situation; completely generally, we can assume that these data come as a set of p numbers, $\{x_1, x_2, \dots, x_p\}$; usually, they will be arranged as a tuple or vector, $x = (x_1, x_2, \dots, x_p)^T$. Extreme examples are: the decision whether a burgular alarm state is (intruder, no intruder) – based on a set of radar, acoustic, and electrical measurements, whether the state of a share is (sell, no sell, buy) – based on current market state, or, the decision by a bank on whether to loan money to an individual – based on certain data gathered in an interview. Hence pattern recognition principles are applicable to a wide range of problems, from electronic detection to decision support.

From the point of view of this chapter, and that of most pattern recognition theory, we define/summarize a pattern recognition system using the block diagram in the following figure.



The input to the system is a vector, $x = (x_1, x_2, \dots, x_p)^T$, the output is a label, w , taken from a set of possible labels $\{w_1, w_2, \dots, w_C\}$. In the case of OCR (optical character recognition) of upper-case alphanumeric characters used in post-codes, we have $w = \{w_1, w_2, \dots, w_{36}\}$, i.e. 36 classes, one corresponding to each of the possible text characters. In the case of a burgular-alarm, or of a credit-worthiness evaluation system, we have just two classes – alarm or quiet, loan or don't loan.

Because it is deciding/selecting to which of a number of classes the vector x belongs, a pattern recognition system is often called a *classifier* – or a pattern classification system.

For the purposes of most pattern recognition theory, a *pattern* is merely an ordered collection of numbers (just reiterating the comments of a few paragraphs back); this abstraction may seem a bit simple, but it is amazingly powerful and widely applicable.

In addition, we will focus strongly on binary classification (two classes); although this may seem limiting, it is not, for any problem involving, in general c classes, can, in fact, be reduced to a succession of two-class problems.

7.2 Features and Classifiers

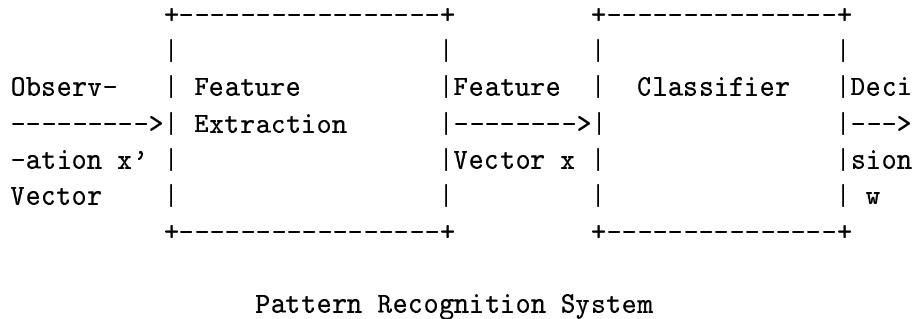
7.2.1 Features and Feature Extraction

In the scheme described above, our p numbers could be simply ‘raw’ measurements – from the burgular alarm sensors, or bank account numbers for the loan approval. Quite often it is useful to apply some ‘problem dependent’ processing to the ‘raw’ data – before submitting them to the decision mechanism; in fact, what we try to do is to derive some data (another vector) that are sufficient to discriminate (classify) patterns, but eliminate all superfluous and irrelevant details (e.g. noise). This process is called *feature extraction*.

The crucial principles behind feature extraction are:

1. Descriptive and discriminating feature(s).
2. As few as possible of them – leading to a simpler classifier.

The following figure gives an expanded summary of a pattern recognition system, showing the intermediate stage of feature extraction. Actually, an earlier ‘pre-processing’ step may precede feature extraction, e.g. filtering out noise.



The observation vector (x') is the input, the final output is the desision (w).

In this setting, the classifier can be made more general, and the feature extraction made to hide the problem specific matters. Hence, we can have a classifier algorithm that can be applied to many different problems – all we need to do is change the parameters.

7.2.2 Classifier

As already mentioned, is to identify, on the basis of the components of the feature vector (x), what class x belongs to. This might involve computing the ‘similarity measure’ between x and a number of stored ‘perfect’/prototype vectors one or more for each class w , and choosing the class with maximum similarity.

Another common scheme is to compute the mean (average) vector for each class, $m_i, i = 1 \dots c$, and to propose these as the single prototype vector for the classes. If we equate ‘closeness’ with similarity, we can derive a simple but effective classifier which measures the distance (see Chapter 3 and later in this chapter) between a pattern, x , and each of the m_i :

$$d_i = |x - m_i|$$

Note that x and m_i are vectors.

We decide on the class on the basis on *minimum* d_i . Such a classifier is call *nearest mean*, and though simple in concept, has remarkably wide applicability.

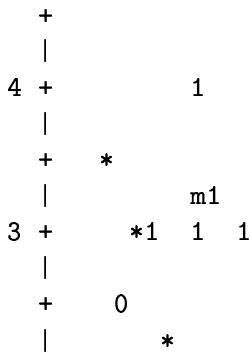
Mathematically speaking, the feature vectors form a *feature space*; and, formally, the job of the classifier is to partition this space into regions, each region corresponding to a class.

These concepts are best illustrated by a numerical example.

Assume that we are trying to discriminate between penny (class 0) and 20p coins (class 1); we have two measurements: weight, x_1 , and brightness, x_2 . Assume we have taken some measurements, and these are tabulated as follows.

class (y)	weight	brightness	
	x_1	x_2	
0	0.40	1.50	
0	1.00	0.50	
0	1.00	1.50	
0	1.00	2.50	
0	1.60	1.50	Measured Patterns
1	1.40	3.00	
1	2.00	2.00	
1	2.00	3.00	
1	2.00	4.00	
1	2.60	3.00	

To emphasise the idea of a feature space, the data are plotted below. Notice how the situation is clarified, compared to what the table; feature space diagrams are important for gaining insight to such problems. On the other hand, when the dimension of x , p , is greater than two, we have problems visualising it.



```

2 +           1   xa = (3,2)
|   m0   *
+ 0  0  0      0 = class 0, mean = m0
|           1 = class 1, mean = m1
1 +   xb   *   * * * = possible boundary
|           xa = (3, 2), pattern of unknown class
+   0   *   xb = (1, 1), pattern of unknown class
|
+-----+-----+-----+-----+-----> x1
0   1   2   3

```

Feature Space Diagram.

The 20ps are heavier than the 1ps, but errors (noise) in the weighing machine make this unreliable on its own, i.e. you cannot just apply a threshold:

$$x_1 > 1.5 \implies \text{class} = 1 \quad (20\text{p})$$

$$\leq 1.5 \implies \text{class} = 0 \quad (1\text{p})$$

There would be an overlap, and some errors. Likewise brightness, on its own, is not a reliable discriminator.

Now, we can test a number of classification rules, on two patterns whose classes are unknown, $x_a = (x_1 = 3, x_2 = 2)$, and $x_b = (1, 1)$.

(a) Nearest neighbour. Find the ‘training’/prototype pattern in the table which is closest to the pattern, and take its class; thus $x_b \rightarrow$ class 1, and $x_a \rightarrow$ class 0.

(b) Nearest mean. The class means are $m_0 = (1.0, 1.5)$, $m_1 = (2.0, 3.0)$. Find the class mean closest to the pattern, and take its class; thus $x_b \rightarrow$ class 1, and $x_a \rightarrow$ class 0.

(c) Linear Partition. Draw a straight line between the classes (in three dimensions it is a plane, in general p-dimensions it is a hyperplane); one side class 0, the other class 1. We will find that such a rule can be expressed as:

$$a_0 + a_1 x_1 + a_2 x_2 > 0 \text{ class 1}$$

$$a_0 + a_1 x_1 + a_2 x_2 \leq 0 \text{ class 0}$$

$$a_0 + a_1 x_1 + a_2 x_2 = 0, \text{ on the boundary}$$

7.2.3 Training and Supervised Classification

The classifier rules we have described above, are all *supervised* classifiers, and the data in the table correspond to *training* data.

Hence, we can have a subdivision of classifiers between supervised and *unsupervised*, i.e. whether or not the classifier rule is based on training data. The clustering/segmentation covered in Chapter 6 is unsupervised – the algorithm segments the image with no prior knowledge of the problem; i.e. in the example we did, the algorithm has not been ‘trained’ to recognise water pixels, and land pixels. On the other hand, if the algorithm is *supervised*, like all those covered in this chapter, it will have already been given example of water, and land, or 1ps and 20ps, i.e. *trained*. In this chapter, we will say no more about unsupervised – see Chapter 6.

7.2.4 Statistical Classification

In the example of the 1ps and 20ps, we have made the pattern vectors lie in a nice symmetric distribution centred on the means. Usually, in a practical situation, the training data will form a cloud of points. And, maybe, there will be an overlap, i.e. we have to decide a ‘best’ boundary, whose errors are a minimum, yet not zero. Here, we can usually develop a decision rules based on statistical/probabilistic criteria. Classification is based on some decision theoretic criterion such as maximum likelihood.

Note: statistical decision theory has a long history – from before that of computers, starting around the early 1900’s.

In statistical pattern recognition, we usually adopt the model: pattern vector = perfect-pattern + noise; thus, for each class we have a cloud of points centred on the ‘perfect-pattern’ vector. The class means are often good estimates of the class perfect-vectors.

The spread of each class ‘cloud’/cluster will depend on the noise amplitude. Obviously, the degree to which the classes are separable depends on two factors: (a) the distance between the ‘perfect-patterns’, and the cluster spread.

The major characteristic of the statistical pattern recognition approach is that it is ‘low-level’. The image (pattern) is merely an array of numbers; the only model applied is that of the random process, there would be no difference in the process, whether the data represented scanned text characters, or parts on a conveyer belt, or faces, or, indeed, if all three were mixed together. Only the data, and estimated parameters would change.

7.2.5 Feature Vector – Update

The components of a pattern vector are commonly called features, thus the term feature vector introduced above. Other terms are attribute, characteristic.

Often all patterns are called feature vectors, despite the literal unsuitability of the term if it is composed of raw data.

It can be useful to classify feature extractors according to whether they are high- or low-level.

A typical low-level feature extractor is a transformation $\mathbf{R}^{p'} \rightarrow \mathbf{R}^p$ which, presumably, either enhances the separability of the classes, or, at least, reduces the dimensionality of the data ($p < p'$) to the extent that the recognition task more computationally tractable, or simply to compress the data (see Chapter 5 – many data compression schemes are used as feature extractors, and vice-versa).

Examples of low-level feature extractors are:

- Fourier power spectrum of a signal – appropriate if frequency content is a good discriminator; additionally, it has the property of shift invariance – it doesn't matter what co-ordinates the objects appears at in the image,
- Karhunen-Loëve transform – transforms the data to a space in which the features are ordered according to their variance/information content, see Chapter 5.

At a higher-level, for example in image shape recognition, we could have a vector composed of: length, width, circumference. Such features are more in keeping with the everyday usage of the term feature.

A recent paper on face recognition has classified recognition techniques into:

- (a) geometric feature based, i.e. using high-level features such as distance between eyes, width of head at ears, length of face, etc.
- (b) template based, where we use the whole of the face image, possibly Fourier transformed, or such-like.

7.2.6 Distance

Statistical classifiers may use maximum likelihood (probability) as a criterion. In a wide range of cases, likelihood corresponds to ‘closeness’ to the class cluster, i.e. closeness to the center/mean, or closeness to individual points. Hence, *distance* is an important criterion/metric.

7.2.7 Other Classification Paradigms

Just to ensure that you are aware of the meanings of some terms that you may come across in textbooks, we will try to introduce some other paradigms – though, for the most part, you can ignore them.

Structural Pattern Recognition

Because in statistical pattern recognition, patterns are viewed as vectors, the information carried by the pattern is no more than its position in pattern space; nevertheless, there may exist structural relationships between pattern components arising from shape, for example. Thus, *structural* pattern recognition has the richer model of a pattern as a structured collection of symbols. Here the distance metric is no longer the sole basis for recognition (nor is it really suitable); simple methods such as string matching may be applied, or syntactic/linguistic methods which are based on the model of pattern generation by one of a family of grammars (classes); the recognition task usually involves parsing to find which grammar generated the unidentified pattern. In relational pattern matching, the symbols are stored in a relational graph structure and the recognition process involves graph matching.

Usually, there exists ‘structure’ in the pattern (e.g. in an image shapes formed by conjunctions of specific grey level values in adjacent regions, or, in a signal, peaks, troughs, valleys, etc.) which is not extracted when the pattern is treated as a mere p -dimensional vector; obviously, if the structure is the major discriminatory characteristic, and the position in p -space is relatively insignificant, this form of recognition will not work well.

As an example, the letter ‘A’ might be described, not as a vector containing invariant moments – or grey levels, but as a *sentence*:

stroke-diagonal1 stroke-horizontal stroke-diagonal2

i.e. a sentence constructed of three *primitives*.

Classification is then done by *parsing* the sentence.

Note: the primitives would probably be obtained by conventional pattern recognition.

Syntactic pattern recognition is difficult to understand and apply – this may explain the comparative lack of interest in it.

Knowledge-based Pattern Recognition

Even with the inclusion of structure, the knowledge (pattern) representation is may be low-level; the identification of the structure is probably done automatically and abstractly. On the other hand, if we bring human knowledge to bear, we can call the approach ‘*knowledge-based*’. A possible justification of this step is to say that we are performing interpretation of the pattern (image) rather than mere recognition of low-level features.

Knowledge-based systems form a branch of artificial intelligence; to some extend they represent a milder form of ‘expert-system’ – with, perhaps, the aims slightly lowered.

Knowledge-based systems try to automate the sort of complex decision task that confronts, for example, a medical doctor during diagnosis. No two cases are the same, different cases may carry different amounts of evidence. In essence, the doctor makes a decision based on a large number of variables, but some variables may be unknown for some patients, some may pale into insignificance given certain values for others, etc. This process is most difficult to codify. However, there are sufficient advantages for us to try: if we could codify the expertise of a specialist, waiting lists could be shortened, the expertise could be distributed more easily, the expertise would not die with the specialist, etc.

There are four major parts in a knowledge based system:

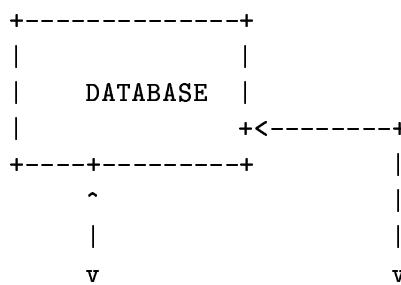
Knowledge elicitation: this is the extraction of knowledge from the expert; it may be done by person to person interview, by questionairre, or by specialised computer program,

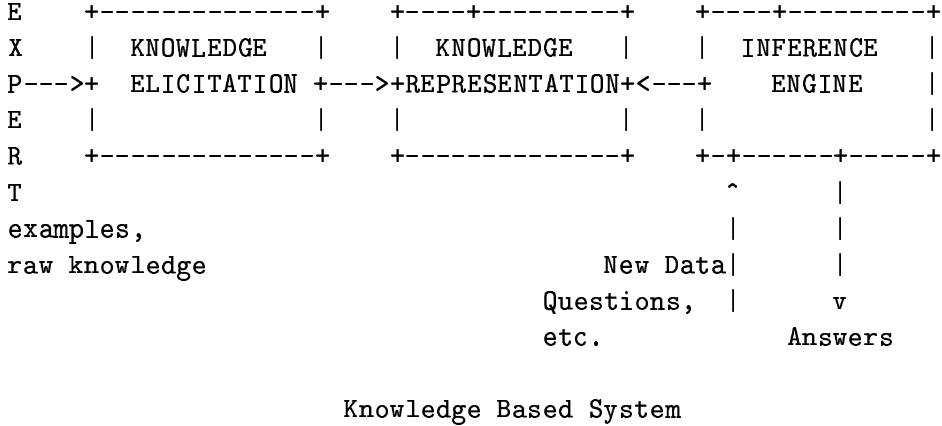
Knowledge representation: we must code the knowledge in a manner that allows it to be stored and retreived on a computer,

Knowledge database: where the knowledge is stored (using the ‘representation’ code mentioned above),

Inference engine: this takes data and questions from a user and provides answers, and/or updates the knowledge database.

The following figure depicts a possible organisation and operation of a knowledge based system. Actually, a well designed database with a good database management system, coupled with a query language that is usable by non-computing people, goes a long way to fulfilling the requirements of a knowledge based system. Also, some of the pattern recognition systems we mention, could be called knowledge based – after all, they store knowledge (the training data or some summary of it) and make inferences (based on the measured data or feature vector); feature extraction is very similar, in principle, to knwoledge representation. Furthermore, we note that neural networks show much promise as knowledge-based systems.





7.2.8 Neural Networks

Automated pattern recognition was the original motivation for *neural networks*. Here, we simply replace the ‘pattern recognition’ black box in the figure in section 7.1 with a neural network. In essence, neural network pattern recognition is closely related to statistical. Neural networks are covered in some detail in Chapter 8.

7.2.9 Summary on Features and Classifiers

A pattern is simply an ordered collection of numbers – usually expressed in the form of a pattern vector. The components of a pattern vector are commonly called features – even if the pattern consists merely of data straight from a sensor. However, the recognition problem can usually be simplified by extracting features from the raw data, usually reducing the size of the feature vector, and removing irrelevancies. Use of an appropriate and effective feature extractor can allow us to apply a quite general classifier; the feature extractor solves the application specific problems, hence we arrive at an abstract pattern/feature vector and the classifier changes from application to application only by its parameters. It is useful to view feature vectors as forming a p -dimensional space; obviously for p greater than two, this can be hard to visualize exactly, nevertheless the idea of points in space interpretation is strongly recommended. With the concept of feature space comes the concept of distance as a criterion for measuring the similarity between patterns.

7.3 A Simple but Practical Problem

7.3.1 Introduction

So what is a typical pattern recognition problem in computer-based picture processing? The following problem looks deceptively easy.

7.3.2 Naive Character Recognition

Below we have a letter ‘C’ in a small (3×3) part of a digital image (a digital picture). A digital picture is represented by brightness numbers (pixels) for each picture point. (It is possible, in a very simple model of visual sensing, to imagine each of these nine cells corresponding to a light sensitive cell in the retina at the back of your eye).

```

Pixels
 1,1  1,1  1,3
+---+---+---+
| ****| ****| ****|
| ****| ****| ****|
+---+---+---+
2,1 | ****| 2,2 | 2,3 |
| ****|      |      |
+---+---+---+
| ****| ****| ****|
| ****| ****| ****|
+---+---+---+
 3,1  3,2  3,3

```

A Letter C

Assume that the picture has been quantized – into ink (1) and background (0).

How can we mechanise the process of recognition of the ‘C’? Certainly if we can do it for one letter, we can do it for all the others – by looping over all the characters. The naive approach tackles the problem almost like that of a lookup table:

1. Pixel by pixel comparison: Compare the input (candidate) image pixel for pixel, with each perfect letter, and select the character that fits exactly:

```

for j = 1..nletters
    match = true
    for r= 0..NR-1 do
        for c= 0..NC-1 do
            if(f[r,c] != X[j][r,c]) /*r,cth pixel of jth letter
                match = false
                exit for r /*i.e. just ONE mismatch causes complete FAIL
            endif
        endfor c
    endfor r
    if(match==true) return j /* j is label
endfor j
return NULL

```

As we will see later, this naive approach has many shortcomings – especially, in general, pairwise comparisons of collections of numbers is usually fraught with difficulty: what if just one is wrong – by just a small amount? what if two dissimilar, by an even smaller amount? is one pixel more important than the others?

So, we're uncomfortable with comparison of individual elements of the pattern. Can we compare the full vectors. Yes, in fact much of classical pattern recognition is about the comparison of vectors, finding distances between them, etc.

Hence, to make this into a general pattern recognition problem, we represent the nine pixel values as elements of a vector (array), i.e. we collect them together in an array; assuming the character is white-on-black and that bright (filled in with '*') corresponds to '1', and dark to '0', the array corresponding to the 'C' is

```
x[0]=1, x[1]=1, x[2]=1, x[3]=1, x[4]=0, x[5]=0, x[6]=1, x[7]=1,
x[8]=1.
```

Note: from now on, in the usual fashion, we will index vector elements from 0 to p-1.

'Feature' number:

0	1	2	
+-----+	-----+	-----+	

	*****	*****	*****	
	*****	*****	***	
	+---+---+---+			
3	****	4	5	

	+---+---+---+			
	****	****	****	
	****	****	****	
	+---+---+---+			
6	7	8		

A Letter C

The letter ‘T’ would give a different observation vector:

```
'T': 1,1,1,      0,1,0,      0,1,0
'0': 1,1,1,      1,0,1,      1,1,1
'C': 1,1,1,      1,0,0,      1,1,1      etc...
```

So how is the recognition done? We could base decision-making on a set of vectors ($\mathbf{x}[j]$) corresponding to ‘perfect’ characters (‘T’, ‘C’ etc.); let these be denoted by $\mathbf{x}[1] = \mathbf{C}$, $\mathbf{x}[2] = \mathbf{T}$, etc.

Let the input (unknown) be \mathbf{x} , a 9-dimensional vector; and let us assume that the components can be real, i.e. no longer binary; let them be any real number between 0 and 1, and subject to noise. The recognition system needs to be invariant (tolerant) to noise.

What if there is a minor change in grey level? Grey Cs are the same as white Cs: the system needs to be amplitude invariant – tolerant to changes in amplitude.

2. Maximum correlation – template matching: Compute the correlation (match) of \mathbf{x} with each of the $\mathbf{x}[j]$ [.] and choose the character with maximum correlation:

```
maxcorr = 0
maxlett = NULL
for j = 1..nletters
```

```

corr = 0
for i= 0..p-1 do
    corr = corr + X[j][i]*x[i]
endfor
if(corr > maxcorr)
    maxcorr = corr
    maxlett = j
endifor
return j

```

That is, we choose letter with maximum corr(elation) or match; this is described mathematically as follows:

$$\text{corr}_j = x'X_j$$

i.e. the dot product of the transpose of x with X_j ,

$$= \sum_{i=0}^{p-1} x_i x_{ij}^p$$

the dot product of x with the j th ‘perfect’ letter.

This is called *template matching* because we are matching (correlating) each template (the ‘perfect’ $X[j]$ [.]s), and choosing the one that matches best.

Template matching is more immune to noise – we expect the ups and downs of the noise to balance one another.

Template matching can be made amplitude invariant by *normalizing* the correlation sum of products:

$$\text{corr} = x'X_j / \sqrt{x'x.X'_jX_j}$$

where the x and X_j are p -dimensional vectors.

This is tantamount to equalising the total ‘brightness’ in each character.

We are still far from having a perfect character-recogniser: what happens if the character moves slightly up, or down, left or right.

3. Shifting Correlation: Use two-dimensional correlation, as described in Chapter 4, i.e., move the character successively across and down, find the position of maximum correlation (match), then compare that maximum with the maxima for the other characters.

But now, what about rotation?

4. We could put a loop around the shifting correlation, each time rotating the character a little.

What happens if the size of the character changes – it's still the same character – the observation vector will change radically. We require scale invariance.

5. Now, we could loop over scale factor.

By now, I hope you have realised that things have fairly well got out of hand; and we haven't mentioned small variations in character shape.

The main point is: this sort of *ad hoc* adding to, and changing of algorithms rarely ever works. We need to build a general theory which is applicable to a wide variety of problems.

7.3.3 Invariance for Two-Dimensional Patterns

From the forgoing discussion, we can summarise some of the requirements of systems system for recognizing two-dimensional shapes in images:

- noise tolerance
- amplitude invariance
- shift invariance
- rotation invariance
- scale invariance

Note that, in the examples given above, 'C' differs from 'O' by only one element – so, naturally, 'C' is likely to be confused with 'O' and vice versa. There is very little we can do about that – except to change the character shapes! – this is why they use funny shaped characters on cheques.

7.3.4 Feature Extraction – Another Update

Let us say we have a simple two-class subset of the previous character recognition problem: 'C' versus 'O'. Clearly, the only attribute of discriminating value is the one in which they differ – number 5. Hence, a sensible feature extractor extracts this, call it y ; we end up with a scalar (one-dimensional) pattern, and the classifier becomes a simple thresholding:

$$y > 0.5 \implies \text{class O}$$

$$y \leq 0.5 \implies \text{class C}$$

7.4 Classification Rules

In this section we discuss the common classification rules used with feature-based classifiers.

In summary, roughly speaking, a supervised classifier involves:

Training: gathering and storing example feature vectors – or some summary of them,

Operation: extracting features, and classifying, i.e. by computing similarity measures, and either finding the maximum, or applying some sort of thresholding.

When developing a classifier, we distinguish between *training* data, and *test* data:

- training data are used to ‘train’ the classifier – i.e. set its parameters,
- test data are used to check if the trained classifier works, especially if it can generalise to new/unseen data.

Although it is normal enough to do an initial test of a classifier using the same training data, in general it is a dangerous practice – we must check if the classifier can generalise to data it has not seen before.

Example Data

The following ‘toy’ data will be used to exemplify some of the algorithms. It is two-dimensional, two-class. If you run DataLab batch file ‘ip2dcl’, that will set up the configuration and input the data for you, and show some histograms and scatter plots. The data are in file d1\dat\toy4.asc.

Here it is – DataLab output:

```
Type = REAL
Bounds = Rows: 0 - 0,  Cols: 0 - 9,  Dims: 0 - 1.
Number of classes: 2
```

```
The 2 labels are: 1 2
Class frequencies: 5 5
Prior probabilities: 0.500000 0.500000
```

```
Class means
-----
```

```
Class 1: 1.000000 1.500000
```

```
Class 2: 2.000000 3.000000
```

```
DataLab IImage Data
-----
[0, 0]

label cllab x0     x1

1   1   :  0.40 1.50
1   1   :  1.00 0.50
1   1   :  1.00 1.50
1   1   :  1.00 2.50
1   1   :  1.60 1.50
2   2   :  1.40 3.00
2   2   :  2.00 2.00
2   2   :  2.00 3.00
2   2   :  2.00 4.00
2   2   :  2.60 3.00
```

Here is another set of data, which would be typical of data from the same class, but with different noise. These data are in file d1\dat\toy41.asc. (These data are *test* data).

```
DataLab I-01
```

```
Type = REAL
Bounds = Rows: 0 - 0,  Cols: 0 - 9,  Dims: 0 - 1.
Number of classes: 2

The 2 labels are: 1 2
Class frequencies: 5 5
Prior probabilities: 0.500000 0.500000
```

```
[0, 0]

1   1   :  0.80 2.00
1   1   :  1.20 1.80
1   1   :  1.40 2.20
1   2   :  1.60 2.40
1   2   :  1.80 2.60
2   2   :  1.60 2.60
2   2   :  2.20 3.20
```

```

2   2   :  2.40 3.40
2   2   :  2.60 3.60
2   2   :  2.80 3.80

```

7.4.1 Similarity Measures Between Vectors

Two possibilities are *correlation*, and *distance*; these are closely related, and in most cases give identical results. See Chapter 3 for the mathematical foundation.

1. *Correlation.* Correlation is a very natural and obvious form of similarity measure. It is sometimes called ‘template matching’ because of the similarity with comparing a two-dimensional shape with a template.

The correlation between two vectors is given by the dot product of them. It is usually necessary to normalize the vectors, which is the purpose of the denominator in the following equation:

$$c_j = \frac{x' X_j}{\sqrt{x' x \cdot X_j' X_j}}$$

Here c_j gives the correlation (or match) between our unknown x and X_j , the j th ‘perfect’ vector. The transpose operations (denoted by prime) are necessary for the sole reason of making the vectors conformable for multiplication.

The dot product can be written as a summation:

$$x' X_j = \sum_{i=0}^{p-1} x_i X_{ji}$$

where x_i is the i th component of vector x and X_{ji} is the i th component of ‘perfect’ vector X_j (class j) for p -dimensional vectors.

Usually, classification would involve finding the X_j that has the *maximum* correlation value – the output from the classifier (the class) is then j .

Later, we shall see that correlation rules are at the basis of most neural networks.

2. *Distance.* Distance is another natural similarity measure. Points (feature vectors) that are close in feature space are similar, those far apart are dissimilar. The distance between two vectors is given by the following equations.

$$d(x, X_j) = | x - X_j |$$

$$d_e(x, X_j) = \sqrt{\sum_{i=0}^{p-1} (x_i - X_{ji})^2}$$

The latter gives the ‘true’ Euclidean distance, i.e. that would be obtained using a piece of string! Quite often when implementing the Euclidean distance in a program, we will be content with the squared Euclidean distance. This provides enough information e.g. to find the ‘nearest neighbour’ distance. There are other distances, the most notable being the Hamming (or Manhattan, or city-block) distance:

$$d_m(x, X_j) = \sum_{i=0}^{p-1} |x_i - X_{ji}|$$

Why is it called Manhattan/city-block? Answer: Think of getting from point a to point b in New York – you have to go up, then across, etc.

The Hamming distance requires no multiplication, and this explains its popularity in the days before hardware multipliers and arithmetic co-processors.

To impress friends at parties, you can use the following. The Hamming and Euclidean distances are referred to in mathematics as the L_1 and L_2 distances. The term ‘metric’ is often used as a synonym for distance. A metric space is a space with some definition of distance. In fact, scalar product is enough for us, since the scalar product of a vector with itself gives the norm, and with both of these we can define a metric. The L_1 and L_2 distances are just two from a class of distances called the Minkowski metrics.

Using distance for classification involves finding the *minimum* distance.

Actually, maximum normalised correlation can be shown to be *exactly equivalent* to minimum Euclidean distance.

Distance is perhaps used more often than correlation. However, even when minimum distance is used, most writers call it template matching; for those unfamiliar with the similarity mentioned in the previous paragraph, this use of terminology can cause confusion.

7.4.2 Nearest Mean Classifier

We have already encountered a nearest mean classifier in the k-means clustering algorithm in Chapter 6, and in this chapter in section 7.1.

No matter how ‘perfect’ a ‘pattern’ may seem, it is usually unadvisable to classify on the basis of a single archetype. It is better to use the average (mean) of a set of examples (sample). Thus the nearest mean classifier.

Training a nearest mean classifier involves collecting a set of training patterns, and computing the mean vector for each class:

$$m_{ji} = (1/n_j) \sum_{k=1}^{n_j} x_{jik}$$

where x_{jik} is the k th example for component i , and class j .

Classification then involves computing minimum distance, with m_{ji} substituted for X_{ji} :

$$d_e(x, m_j) = \sqrt{\sum_{i=0}^{p-1} (x_i - m_{ji})^2}$$

Ex. 7.3-1 Train a nearest mean classifier on data set ‘toy4.asc’, and test it on the same data; note: the class means are class 1 = (1.0,1.5), class 2 = (2.0,3.0). Here is the output, where label = true label, and cllab = classifier label. It gets them all correct, surprise, surprise!

label	cllab	x0	x1
1	1	: 0.40	1.50
1	1	: 1.00	0.50
1	1	: 1.00	1.50
1	1	: 1.00	2.50
1	1	: 1.60	1.50
2	2	: 1.40	3.00
2	2	: 2.00	2.00
2	2	: 2.00	3.00
2	2	: 2.00	4.00
2	2	: 2.60	3.00

Ex. 7.3-2 Train on the same data, but test on data set ‘toy41.asc’. The results are below. Note the two errors, marked (*). That is, point (1.6,2.4) is closer to (2.0,3.0) than it is to (1.0,1.5). So is (1.8,2.6).

[0, 0]
1 1 : 0.80 2.00

1	1	:	1.20	1.80
1	1	:	1.40	2.20
1	2*	:	1.60	2.40
1	2*	:	1.80	2.60
2	2	:	1.60	2.60
2	2	:	2.20	3.20
2	2	:	2.40	3.40
2	2	:	2.60	3.60
2	2	:	2.80	3.80

Ex. 7.3-3 Verify that the nearest mean classifier will give the results below.

- (a) 1 1 : 1.40 2.20
- (b) 1 2* : 1.60 2.40
- (c) 1 2* : 1.80 2.60

Answer:

Use:

$$d_e(x, m_j) = \sqrt{\sum_{i=0}^n (x_i - m_{ji})^2}$$

(a) Take (1.4, 2.2);

$$m_1 = (1.0, 1.5), m_2 = (2.0, 3.0)$$

Let us forget about the `sqrt()`, since minimum squared distance is the same as minimum distance. Call the squared distance `Ds()`.

$$\begin{aligned} Ds(x, m1) &= (1.4-1.0)^2 + (2.2-1.5)^2 \\ &= 0.4^2 + 0.7^2 \\ &= 0.16 + 0.49 \\ &= 0.65 \end{aligned}$$

$$\begin{aligned} Ds(x, m2) &= (1.4-2.0)^2 + (2.2-3.0)^2 \\ &= 0.6^2 + 0.8^2 \\ &= 0.36 + 0.64 \\ &= 1.0 \end{aligned}$$

Therefore, m_1 is closest, so the class is 1.

(b) (1.60, 2.40)

$$\begin{aligned} D_s(x, m_1) &= (1.6 - 1.0)^2 + (2.4 - 1.5)^2 \\ &= 0.6^2 + 0.9^2 \\ &= 0.36 + 0.81 \\ &= 1.17 \end{aligned}$$

$$\begin{aligned} D_s(x, m_2) &= (1.6 - 2.0)^2 + (2.4 - 3.0)^2 \\ &= 0.4^2 + 0.6^2 \\ &= 0.2 + 0.36 \\ &= 0.56 \end{aligned}$$

Therefore, m_2 is closest, so the class is 2.

(c) (1.80, 2.60)

(Left as exercise).

(d) Plot these data on a two-dimensional surface – and verify from this feature space diagram that the results are plausible.

7.4.3 Nearest Neighbour Classifier

If the classes occupy irregularly shaped regions in feature space, the nearest mean classifier may not perform well, i.e. the mean does not provide an adequate summary of the class. One solution is to store all the example patterns – no summarising – and in classification go through all examples, and choose the class of the closest example.

Although this sounds crude, the nearest neighbour classifier has very good theoretical credentials, and performs well in practice.

The major problem is the amount of processing. If the dimensionality of the feature vector is large ($p = 50$, say) then we may need a large amount of examples, and so a large amount of distance computations for each classification.

7.4.4 Condensed Nearest Neighbour Algorithm

(See Duda and Hart.) If the nearest neighbour rule requires excessive computation, it is possible to ‘condense’ the set of example vectors to those at

the boundaries of the class regions – these are the only ones that affect the final result.

7.4.5 k-Nearest Neighbour Classifier

The performance of the nearest neighbour rule can sometimes be improved (especially if examples are sparse) by finding k nearest neighbours (e.g. $k = 5$) and taking a vote over the classes of the k .

7.4.6 Box Classifier

Again, if the mean does not provide a good summary, it may be possible to describe the class region as a ‘box’ surrounding the mean. The classification rule then becomes:

```

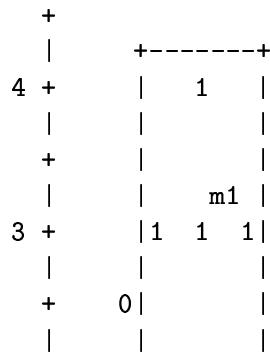
for each class j = 1..c
for each dimension i = 0..p-1
  if( $lX_{ji} < x_i < uX_{ji}$ ) then INj = true
      else INj = false

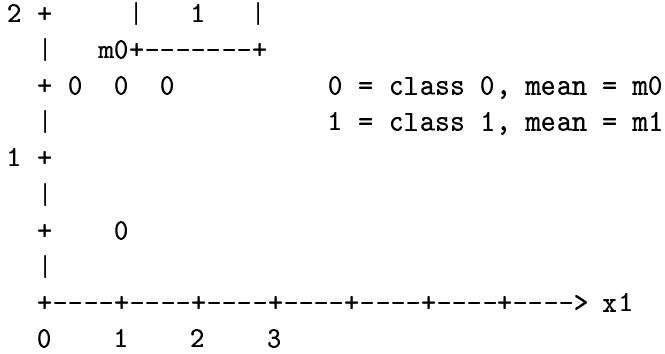
```

where lX_{ji} , uX_{ji} represent the lower and upper boundaries of the box.

This is precisely the algorithm mentioned in Ex. 1.8-4 (see Chapter 1), albeit using less glamorous terminology. The standard deviation for class j , in dimension i gives an objective method for determining the bounds.

Ex. 7.3-3 Here are the data given in section 7.3, with an appropriate ‘box’ drawn to enclose class 1; actually, for the case shown it just about works, but in general, these ‘box’ shapes can be improved on – see next section. Box classifier are easy to compute, however, and were very popular in the days of special-purpose hardware image processing and classification.





Feature Space Diagram.

7.4.7 Hypersphere Classifier

Instead of a ‘box’ – a hyperrectangle – we define a hypersphere, of radius r_j , surrounding the mean m_j . The classification rule then becomes: “if the distance between vector x and m_j is less than r_j , then the class is j ”.

7.4.8 Statistical Classifier

Consider the noisy image of (say) a mark on a page shown in the following figure. And assume that all we want to do is classify each pixel according to ink/no-ink (two classes).

```
0123456789012345678901234567890123456789012345678901
-----
0| -/, /, --. -,, -.-., -, -/-,, ---, -, ./., ---, --/-,,, |
1| -,/--, //-, /, --., -//, --, /-,, -, --., --., --/., /., |
2| , -./, /..---/.---. -., , --=,, , , , --, -, --., --, -.. |
3| //./,, -.. -., , , , --, , --, , , , --, ./, .., , , , |
4| , , -,-,, , -., , , , , , , -X=+X=X+---, , , , -./-, -, , --|
5| ---, .-,, ./, -, , , --, ., , +X=BX+-, --, , , -=-/-., , --.|
6| --/, ./, .=, /., -., --, -++=X=+, , -, -, --. -., /., -/., .|
7| , .--. .---, -., , , -., -B=/+XXX, ., -/-/, /-, -, , , /-/, , -|
8| -,-., /-., -., --, , , , -./++=XX.-, /-/, /-, -, , , --/., -|
9| -...//-, , /, -....-, , , , .=+X=BXX--, -, /., /--, --. -., -|
0| , -.-=XXX++=BXBX+X+BX++=+X+X=X+++X=XX++=++X+X++X, , --|
1| .-. -++X+X+X+XXBBB+X+X+BX+=+=XXXX+X+XXXXX++BXX++, -, , |
2| ----+X+X=+X+XX=+X/XX+XX=+BXX+XBX+X+=++++++X+X++X, , -|
3| , , -,+++B=X=+X+X+X+XXB+XXX=XXB+=+X+=MX++B==X+M+XX, /, , |
4| -,-+X++X=+B=XX+XX+XB+BB+BX++BB=B++X=++XXB=XXB+. , .-|
5| --, , +XXX+XXX==X++++++XXX+=++=X+B=X=++XXX++=XX, , , |
```

```

6|,,,--+XBXXX++++++B+=+BX+XXXXXB==X==+X++++++M++=.---|
7|-,-.,--,.-/-,-/,.-,,,,X+X+XBX,---,,,-,-,-.-..-..|
8|-,--/-,,,-/,,.,.,-.,---+=+XBXX- ,/-.-,--,-.,-.,.,--|
9|-,,-,-,-/,,/-./,-/,XX+XX+++,---,-,--,,--,,/,,,-|
0|.. ,,-.--,/,-/,-.--/.,++X+X+X,,--,,-- ,..--,,,-,/.,|
1|,/-,-.-.,---,-.,,,.,XBX+++==,,,-,---,,,,,-.,--.-|
2|,-,,.,.,.,,-/--,-,-.,-,-,,/---,,,-,,---=,,,-.,./|
3|,,--,,--,-.,--.,--,-/,-,-.-.-,,,-,-,,,-/,,,---|
4|,,,-,,.,.,-.,---,---,,/,-.,-/---./-/---,-,-,-,-|
5|.,,/,,/,-,-,-,-,-,-.,/,-,,.,-.,,,,-,-,,,-,-,-,-,-,|
+-----+
+0123456789012345678901234567890123456789012345678901
LISSP I-03:00
08/01/92 18:07

```

[11,0]

76	106	76	116	179	189	203	189	217	195
217	191	199	201	231	231	223	195	203	195
205	193	225	219	185	159	195	173	207	201
205	201	189	221	185	201	219	209	203	199
197	181	187	239	203	201	191	179	82	118
	86	84							

[12,0]

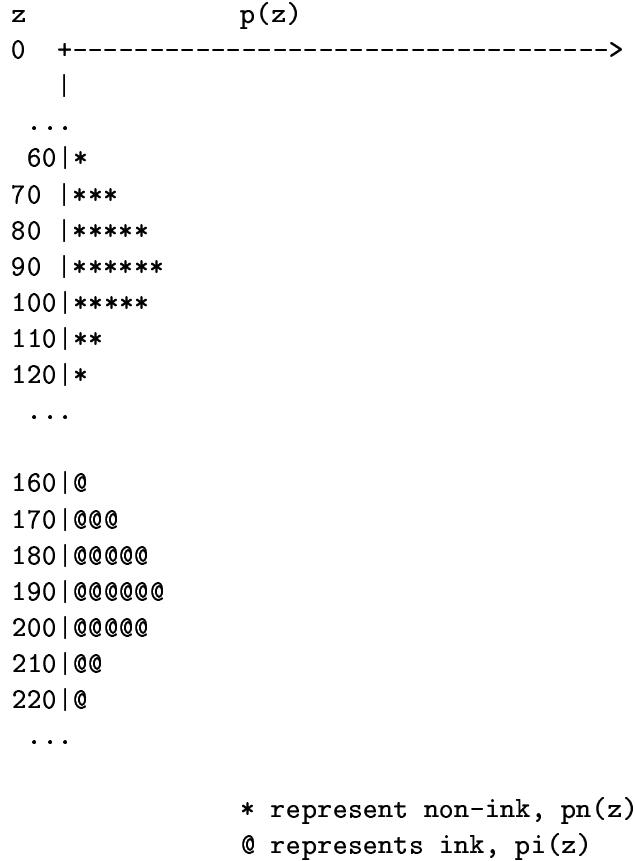
118	108	124	116	181	205	177	211	163	175
217	189	205	207	173	195	211	147	199	213
197	205	213	167	183	233	217	201	197	201
223	221	185	207	197	157	195	187	195	193
189	181	201	195	221	195	183	205	82	84
	116	94							

[13,0]

98	88	104	84	197	177	197	225	169	203
155	195	203	187	201	187	201	193	199	209
225	177	199	199	199	173	221	203	225	193
171	187	211	191	169	247	209	195	195	229
165	161	209	189	247	197	209	203	80	126
	96	90							

Noisy Image of Mark on a Page

A histogram/probability distribution of the image would look something like:



Histogram of Grey Levels

In this figure the $p_i(z)$, and $p_n(z)$ can be taken to represent the probability, at z , of ink, and non-ink, respectively. This suggests a classifier rule (z is the input pattern, i.e. one-dimensional feature):

```

if (pi(z) > pn(z)) then ink
else non-ink

```

I.e. maximum probability, or maximum likelihood.

This is the basis of all *statistical pattern recognition*. In the case under discussion, *training* the classifier simply involves histogram estimation.

Unfortunately, except for very small dimensionalities, histograms are hard to measure. Usually we must use *parametric* representations of probability density.

Quite often, a maximum likelihood classifier turns out to be remarkably similar in practice to a minimum distance (nearest mean) classifier; the

reason for this is that likelihood/probability is maximum at the mean, and tapers off as we move away from the mean.

7.4.9 Bayes Classifier

Assume two classes, w_0, w_1 . Assume we have the two probability densities $p_0(x), p_1(x)$; these may be denoted by

$$p(x | w_0), p(x | w_1)$$

the class conditional probability densities of x . Another piece of information is vital: what is the relative probability of occurrence of w_0 , and w_1 : these are the *prior* probabilities, P_0, P_1 – upper-case P s represent priors. In this case the ‘knowledge’ of the classifier is represented by the $p(x | w_j), P_j$; $j = 0, 1$.

Now if we receive a feature vector x , we want to know what is the probability (likelihood) of each class. In other words, what is the probability of w_j given x – the *posterior* probability.

Bayes’ Law gives a method of computing the posterior probabilities:

$$p(w_j | x) = P_j p(x | w_j) / (\sum_{j=0} P_j p(x | w_j))$$

(each of the quantities on the right-hand side of this equation is known – through training.)

In Bayes’ equation the denominator of the right hand side is merely a normalising factor – to ensure that $p(w_j | x)$ is a proper probability – and so can be neglected in cases where we just want maximum probability.

Now, classification becomes a matter of computing Bayes’ equation, and choosing the class, j , with maximum $p(w_j | x)$.

The classifiers mentioned in the previous sections were all based on intuitive criteria. The Bayes’ classifier is *optimal* based on an objective criterion – the class chosen is the most probable, with the consequence that the Bayes’ rule is also *minimum error*; i.e. in the long run it will make fewer errors than *any* other classifier: this is what maximum likelihood means.

7.5 Linear Transformations in Pattern Recognition and Estimation

A general ‘pattern’ or multidimensional ‘observed signal’ may be represented by a p -dimensional vector x . In signal estimation it is commonly required to transform x to some scalar attribute y – as, for example, in regression analysis. In pattern recognition, where we are required to decide the class to which x belongs, a possible solution is to transform x , again, to a scalar y , so that the decision can be expressed as a threshold – or multiple thresholds

for multiple classes. Another fairly universal requirement is to map from p -dimensions to q dimensions where $q \ll p$, based on some optimality criterion, e.g. maximum information retention in the reduced dimensionality space – data compression, maximisation of discrimination between classes – feature extraction.

For each of the above problems, it is possible (at least theoretically) to obtain solutions based on linear transformations of the form $y = Ax$, where x is a $p \times 1$ vector, y a $q \times 1$ vector, and A , a $q \times p$ matrix. Most of the transformations are based on statistical criteria, in particular least-square-error or maximum likelihood.

If the data are in image format, i.e. in the form of a N row $\times M$ column image, the above form is not directly applicable. However such an image easily can be rearranged as a single $NM \times 1$ vector, so the transformations are still valid – though the size of NM may make the matrix A difficult to handle, and worse still, estimate.

The section identifies some special solutions for image data and concludes with a focus on the application to human face recognition. Each principal transformation is summarized according to its criterion/premise (e.g. maximises separation of classes), enough theory to enable understanding of the mechanism, formula/algorithm or reference to existing software, and some evaluative commentary.

7.5.1 Linear Partitions of Feature Space

This section forms the basis of linking of classifiers with neural networks.

Consider template matching in the case of two classes:

Actually, to be perfectly correct, we must subtract the mean of the data, call it m , and $m_i = i$ th component of mean. Note – this is the overall mean, *not* class means.

$$x'X_j = \sum_{i=0}^{p-1} (x_i - m_i)'(X_{ji} - m_i)$$

where x_i is the i th component of vector x and X_{ji} the i th component of X_j for p -dimensional vectors. Prime denotes transpose.

Therefore finding maximum correlation reduces to:

$$\sum_{i=0}^{p-1} (x_i - m_i)(X_{0i} - m_i) > \sum_{i=0}^{p-1} (x_i - m_i)(X_{1i} - m_i) \implies \text{class} = 0$$

and otherwise $\implies \text{class} = 1$

Or:

$$\sum_{i=0}^{p-1} (x_i - m_i)(X_{0i} - m_i) - \sum_{i=0}^{p-1} (x_i - m_i)(X_{1i} - m_i) > 0 \implies \text{class} = 0$$

and otherwise $\implies \text{class} = 1$.

The left hand side is:

$$\sum (x_i X_{0i} + m_i m_i - m_i X_{0i} - m_i x_i - (x_i X_{1i} + m_i m_i - m_i X_{1i} - m_i x_i))$$

$$= \sum (x_i X_{0i} + m_i m_i - m_i X_{0i} - m_i x_i - x_i X_{1i} - m_i m_i + m_i X_{1i} + m_i x_i)$$

$$= \sum (x_i X_{0i} - m_i X_{0i} - x_i X_{1i} + m_i X_{1i})$$

$$= \sum (x_i (X_{0i} - X_{1i}) - m_i X_{0i} + m_i X_{1i})$$

$$= \sum_{i=0}^{p-1} (x_i a_i) + a_0 > 0 \implies \text{class} = 0$$

else $\implies \text{class} = 1$

where

$$a_0 = \sum_{i=0}^{p-1} (-m_i X_{0i} + m_i X_{1i}) = \text{constant}$$

and $a_i = (X_{0i} - X_{1i})$.

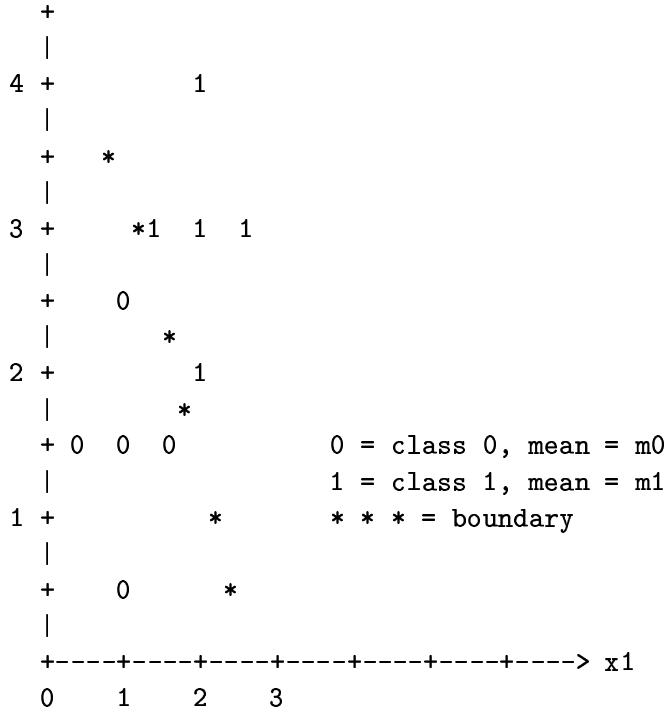
Geometrically, the decision is defined by the line

$$\sum_{i=0}^{p-1} (x_i a_i) + a_0 = 0$$

which is an equation of a straight line through the origin; points above the line are judged class 0, those below are class 1.

Usually X_0, X_1 would be m_0, m_1 the mean vectors of the classes; in this case the line given by $\sum a_i x_i$ bisects and is perpendicular to the line joining the two means.

Example. Look again at the data from section 7.3. The * * * shows a linear partition for the two classes.



7.5.2 Discriminants

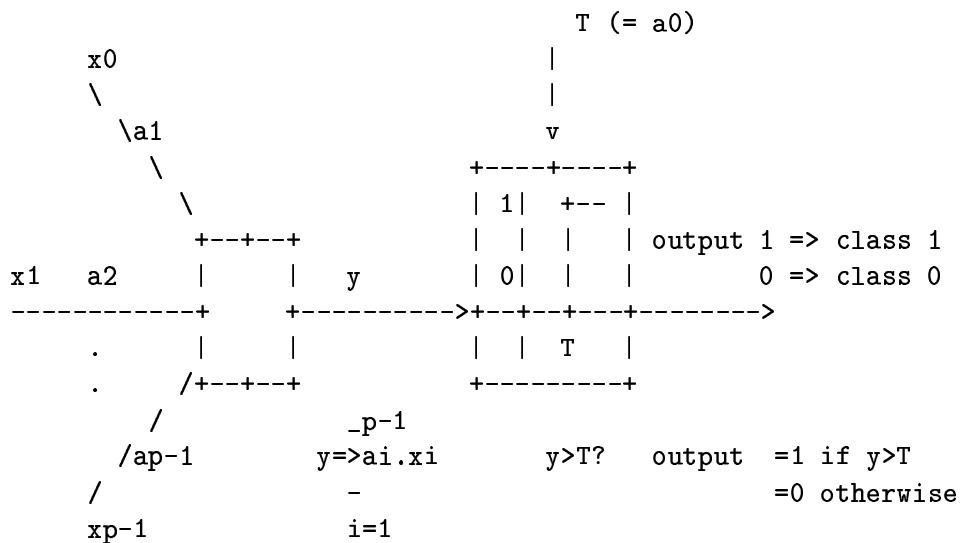
We can look at the classification decision equations derived above in yet another way:

$$\begin{aligned} y &= \sum_{i=0}^{p-1} x_i a_i \\ &= x' a \end{aligned}$$

the dot product of vectors x and a

$$\begin{aligned} y > a_0 &\implies \text{class1} \\ < a_0 &\implies \text{class0} \\ = a_0 &\implies \text{arbitrary} \end{aligned}$$

These equations form what is called a linear discriminant. A diagrammatic form is shown as follows:



Linear Discriminant

Readers familiar with neural networks will notice the similarity between this figure and a single neuron using a hard-limiting activation function.

A general discriminant function, g , is a function of the form:

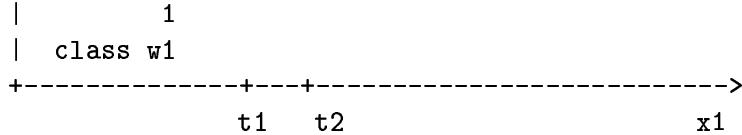
$$y = g(x)$$

$y > T \Rightarrow$ class1
 $< T \Rightarrow$ class2
 $= T \Rightarrow$ arbitrary

7.5.3 Linear Discriminant as Projection

The following figure shows feature vectors (points, data) for a two-class, two-dimensional feature space.

		1	2	2	2	2
x2		1	1	1	2	2
		1	1	1	1	2
		1	1	1	1	2
		1	1	1	1	2
		1	1	1	1	class w2

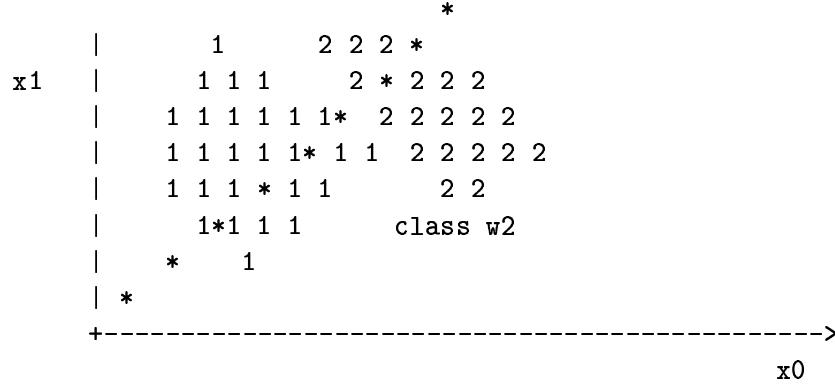


Feature space, two-class, two-dimensional.

We can now examine the utility of a simple linear discriminant, $y = a'x$.

Projecting the data onto the x_0 axis would result in a little overlap of the classes (between points t_1 , and t_2 – see the figure).

Projecting data onto axis x_2 , discriminant vector = (0.0,1.0) would result in much overlap. However, projecting the vectors onto the line shown as follows would be close to optimal – no class overlap.



* * * projection line - y_0

Linear Discriminant as Projection.

Projecting the data onto different axes is equivalent to rotating the axes – i.e. in the case above, we would rotate to a new set of axes y_0 , y_1 , where y_0 is the axis shown as * * * ; obviously, in the case above, we would ignore the second, y_1 dimension, since it would contribute nothing to the discrimination of the classes.

7.5.4 The Connection with Neural Networks

Recall the following equation – we will call the expression s – which interprets correlation as a linear partitioning (assume we are using means as ‘perfect’ vectors). Note: many treatments of pattern recognition deal with only with

two-class cases – the math is usually easier, and it is usually easy to extend to the multiclass case without any loss of generality.

$$s = \sum_{i=0}^{p-1} x_i a_i + a_0 = 0$$

As noted above, this is a straight line. It bisects the line joining the two means, and is perpendicular to them.

In neural networks, a_0 is called the *bias* term. Alternatively, for mathematical tidiness, we can include a_0 in the summation, and insist that the zeroth element of each feature is always 1 – the so-called *bias* input in neural networks.

We then subject s to a threshold function,

$$\begin{aligned} s > 0 &\implies \text{class 1} \\ s \leq 0 &\implies \text{class 0} \end{aligned}$$

as seen before.

In neural networks nowadays, this thresholding is usually done with the so-called sigmoid function, see Chapter 8.

7.5.5 Fisher Linear Discriminant

See Fisher (1936), Duda and Hart (1973), Fukunaga (1990).

Linear Discriminant Analysis, first proposed in Fisher's 1936 article, relates only to the two-class case. The Fisher discriminant is one of the best known of all pattern recognition algorithms. It projects the data onto a single axis, see section 3.3, defined by the Fisher discriminant vector a :

$$y = a^T x$$

The Fisher discriminant simultaneously optimises two measures/criteria:

- (a) maximum between-class separation, expressed as separation of the class means m_1, m_2 ,
- (b) minimum within-class scatter, expressed as the within class variances, v_1, v_2 .

The Fisher criterion combines (a), and (b) as:

$$J = (m_1 - m_2)/(v_1 + v_2)$$

where the transformed means and variances are:

$$m_j = a^T m_j$$

$$v_j = a S_j a^T$$

where S_j = covariance for class j , and m_j = mean vector for class j .

The discriminant is computed using:

$$a = W^{-1}(m_1 - m_2)$$

where W is the pooled (overall, class independent) covariance matrix, $W = p_1S_1 + p_2S_2$, p_1, p_2 are the prior probabilities. – see Appendix.

Procedure:

1. Estimate class means m_j and covariance matrices S_j , and prior probabilities, p_j .
2. Compute pooled covariance matrix, W (see definition above).
3. Invert matrix W (using some standard matrix inversion subprogram).
4. Compute the discriminant vector, a (see above).
5. Apply the discriminant using eqn. 3.4-1.

See DataLab function 'da' (Campbell, 1993).

7.5.6 Karhunen-Loëve Transform

See Fukunaga (1990). Also called Principal Components Analysis, Factor Analysis, Hotelling Transform.

The Karhunen-Loëve (KL) transform rotates the axes such that the covariance matrix is diagonalised:

$$y = Ux$$

where (see Appendix) U is the eigenvector matrix of S the covariance matrix (over all classes), i.e.

$$U^T S U = L$$

where

$$\begin{aligned} L = & \begin{vmatrix} 11 & 0 & 0 & \dots & 0 \\ 0 & 12 & 0 & & 0 \\ & & & \ddots & \\ & & & & 1p \end{vmatrix} \end{aligned}$$

L is a diagonal matrix containing the variances in the transformed space, and,

$$U = \begin{vmatrix} u_1 \\ u_2 \\ u_2 \\ \dots \\ u_i \\ \dots \\ u_p \end{vmatrix}$$

is the matrix formed by the **eigenvectors**, u_i , of S .

There is an eigenvector, u_i , corresponding to each eigenvalue λ_i ; If we order the eigenvectors, u_i , according to decreasing size of the corresponding eigenvalue, we arrive at a transform (eqn. 3.5-1) in which the variance in dimension y_0 is largest, y_1 , the next largest, etc.

If we equate variance and information, then the KL transform gives a method of mapping to a lower dimensionality space, m , say, $m < p$, with maximum retention of information; thus its theoretical appeal for data compression. Let U' be the $m \times p$ matrix containing the first m rows of U , i.e. the m eigenvectors/projections corresponding to the m largest eigenvalues. We can rewrite eqn. (3.5-1) as

$$y' = U'x$$

This corresponds to the coding part of the compression; we have reduced the data from p to m numbers. We can decode using,

$$x' = U'^T y'$$

Now, x' will be a maximally faithful (in a least-square-error sense) recreation of the original vector x , i.e. it minimizes the expected square error between the original vector, and the decoded vector; the minimum-square-error criterion is expressed as:

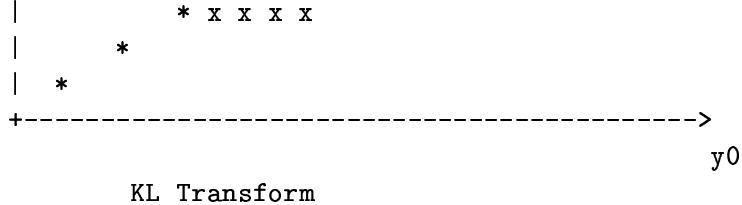
$$J = E\{(x - x')^T(x - x')\}$$

The following figure gives a geometrical representation of a KL transform; the cluster of data is elliptically shaped, with the major axis of the ellipse about 45 degrees from the horizontal axis; the minor axis is perpendicular to that; the line denoted with the '*' corresponds to the first eigenvector, i.e. the eigenvector corresponding to maximum variance.

```

| * first
y1 |           x x x * x eigenvector.
|           x x x x * x x
|           x x x x*x x x x
|           x x x*x x x x x

```



Frequently, the eigenvector/eigenvalue equation is expressed as:

$$R_{uk} = \lambda_k u_k$$

where, as above, λ_k is the k th eigenvalue, and u_k is the associated k th eigenvector.

Procedure:

1. Estimate the overall covariance matrix S .
2. Compute the eigenvalues, L , and eigenvectors, U , of S , using some standard subprogram.
3. Order the rows of U according to the corresponding eigenvalues, see above.
4. Apply the transform using eqn. 3.5-1; the components of y are ordered according to ‘information’ content, thus, we can retain the first, m , say, of them as the most ‘significant’ features.

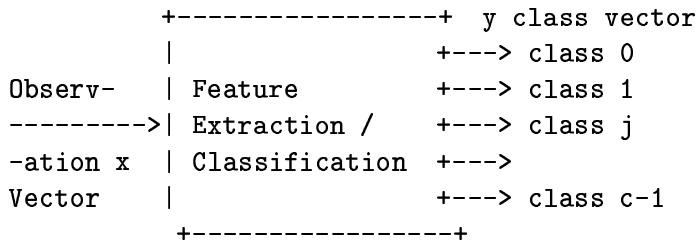
See DataLab function ‘klt’ (Campbell, 1993).

The KL transform U is computed without any reference to class occupancy, – the class labels are ignored when S is computed. Therefore, is contrast with, for example, the Fisher discriminant, KL has no particular discriminating capability. The KL transform belongs to the class of **unsupervised** methods.

7.5.7 Least-Square Error Linear Discriminant

The Fisher discriminant works only for two classes – though it is possible to tackle multi-class problems using pairwise discriminants for each of the C_2^c dichotomies, and subsection x below describes a fully multiclass generalization of the Fisher discriminant.

A alternative (but actually similar to the Fisher discriminant) approach is to express the pattern recognition problem as in the Figure below, which shows a mapping from the $p \times 1$ observation vector x directly to a $c \times 1$ class vector, y .



Multi-Class Least-Square-Error Linear Discriminant

The class vector is a binary vector, with only one bit set at any time, in which a bit j set ($= 1$) denotes class j ; i.e. $y = (1, 0, 0, \dots, 0)^T$ denotes class 0, and $y = (0, 1, 0, \dots, 0)^T$ denotes class 1, etc.

The problem can be set up as one of multiple linear regression, in which the independent variables are the components of x , and the dependent variables – the c class ‘bits’ – are the components of y .

For, initially, just one component of y , y_j , the regression problem can be expressed compactly as:

$$y_{ji} = x_i^T b + e_i$$

where $b = (b_0, b_1, \dots, b_{p-1})^T$ is a $p \times 1$ vector of coefficients for class j , $x_i = (1, x_{i1}, \dots, x_{ip-1})^T$ is the pattern vector, and e_i = error,

We have $y_{ji} = 1$ if the pattern, x , belongs to class j , and = 0 otherwise.

Formulation in terms of the augmented vector x , which contains the bias element ‘1’ is important; without it we would effectively be fitting a straight line through the origin – the bias (b_0) corresponds a non-zero intercept of the y -axis; compared to using a separate bias element, the analysis is greatly simplified.

The complete set of n observation equations can be expressed as:

$$y = Xb + e$$

where $e = (e_1, e_2, \dots, e_n)^T$, and $y = (y_1, y_2, \dots, y_n)^T$, the $n \times 1$ vector of observations of the class variable (bit). X is the $n \times p$ matrix formed by n rows of p pattern components.

The least-square-error fitting is given by (Beck and Arnold, p. 235),

$$b' = (X^T X)^{-1} X^T y$$

Note: The jk th element of the $p \times p$ matrix $X^T X$ is $\sum_j x_{ij} x_{jk}$, and the j th row of the $p \times 1$ vector $X^T y$ is $\sum_i x_{ij} y_i$. Thus, $X^T X$ differs from the autocorrelation matrix of x only by a multiplicative factor, n , so that

the major requirement for eqn. (3.6-3) to provide a valid result is that the autocorrelation matrix of x is non-singular.

We can express the complete problem, where the vector y has c components, by replacing the vector y in equation (3.6-3) with the matrix Y : the $n \times c$ matrix formed by n rows of c observations.

Thus, eqn. (3.6-3) extends to the complete least-square-error linear discriminant:

$$B' = (X^T X)^{-1} X^T Y$$

$X^T Y$ is now a $p \times c$ matrix, and B' is a $p \times c$ matrix of parameters, i.e. one column of p parameters for each dependent variable.

Applying the discriminant/transformation is simply a matter of premultiplying the (augmented) vector x by B' :

$$y' = B' x$$

7.5.8 Computational Considerations

The transforms given in the foregoing sections are, in general, directly applicable to image patterns. By an ‘image’ pattern we mean that the components of the pattern are derived from the pixels of a two-dimensional image. Let f be an image pattern, where the general pixel at row r , and column c is $f[r, c]$ – or f_{rc} ; there are N rows, $r = 0 \dots N - 1$ and M columns, $c = 0 \dots M - 1$.

$$\begin{array}{c|c} & \\ & \\ f = & \begin{array}{c|c} & \\ |f_{00} f_{01} & f_{0M-1}| \\ |f_{10} f_{11} & f_{1M-1}| \\ | & | \\ | & \dots f_{rc} \dots & | \\ | & & | \\ |f_{N-10} f_{N-11} & f_{N-1M-1}| & | \\ & & / \end{array} \end{array}$$

Trivially, f can be represented as an $NM \times 1$ vector as follows:

$$x = (f_{00}, f_{01}, \dots, f_{0M-1}, f_{10}, \dots, f_{rc}, \dots, f_{N-10}, \dots, f_{N-1M-1})$$

Pratt (1991), p. 130 gives an analytic form of this expression for x , which is sometimes useful in formal mathematical treatment of image patterns as vectors.

Usually, the major problem with extending the results of the foregoing sections to images is in the difficulty of estimating statistics. For example, if the images f are $N \times M$, the vectors x are $NM \times 1$; therefore the autocorrelation and covariance matrices for x (and f) are $NM \times NM$. If the images

are modestly sized at 128×128 ($N = 128, M = 128$) we have an autocorrelation matrix of $16,384 \times 16,384$ which means that there are 269×106 components. Such a matrix is simply not estimable, nor handleable.

7.5.9 Eigenimages

(See Turk and Pentland, 1991) Note: the treatment of the Karhunen-Loëve transform (eigenvector expansion) in section 3.5 uses the autocorrelation matrix R as the basis of the eigenvectors. Frequently, the covariance matrix S is used; this is the case in the Turk and Pentland (1991) work on ‘eigenfaces’; in this section, we will use the covariance matrix.

Naively, the Karhunen-Loëve transform given by eqn. (3.5-1) ($y = Ux$) can be easily extended to apply to image patterns, using eqn. 4.1-2 to express the sample images as vectors. However, the problem mentioned in the previous section, of a huge dimensionality covariance matrix, is immediately obvious; the dimensionality of x is $p = NM$; assume for the remainder that $N = M$, i.e. we have a square image.

If we express the image patterns f_i as vectors x_i using eqn. 4.1-2; let there be n of them in the sample, $x_i, i = 1 \dots n$.

If, see section 2.2, eqn. 2.2-11, $x' = (x - m)$, i.e. the pattern vector reduced to zero mean, and

$$X' = [x'_1 x'_2 \dots x'_i \dots x'_n]$$

which is of dimensions $p \times n$, $p = N^2$, the sample covariance matrix can be expressed as can be rewritten as

$$S = \frac{1}{n} X' X'^T$$

which is $N^2 \times N^2$. However, see Appendix A.12, the rank of S is only n , and therefore there are only n non-zero eigenvalues when S is diagonalized. Based on this, Turk and Pentland (1991) give a method of finding these n eigenvalues/eigenvectors based on a reduced dimensionality $n \times n$ matrix – in their paper $n = 16$.

If we express the eigenvalue/eigenvector equation as in eqn. (3.5-8), for the $n \times n$ matrix $T = X'^T X'$,

$$T_{vk} = \lambda_k v_k$$

i.e.

$$X'^T X' v_k = \lambda_k v_k$$

Premultiply each side by X' ,

$$X' X'^T X' u_k = \lambda_k X_{vk}$$

Now, see eqn. 4.2-2, eqn. 4.2-4 is an eigenvalue/eigenvector equation for the matrix $X'X'^T$ which, see eqn. 4.2-1, $= nS$. Therefore, the $(p \times 1)$ vectors $Xv_k = u_k$ are the eigenvectors of nS ; note the dimensionalities: $(p \times 1)$, and $(p \times n) \times (n \times 1)$.

$$u_k = Xv_k$$

for $k = 1 \dots n$. To clarify what is happening we need to show eqn. (4.2-4) fully expanded:

$$\begin{array}{c|ccccc|c} |u_{0k}| & |x_{01} & x_{02} & \dots & x_{0i} & \dots & x_{0n}| & |v_{k1}| \\ |u_{1k}| & |x_{11} & x_{12} & \dots & x_{1i} & \dots & x_{1n}| & |v_{k2}| \\ |. . .| & | & & \dots & & & |. . .| \\ |. . .| & | & & \dots & & & |. . .| \\ |. . .| & | & & \dots & & & |. . .| \\ |u_{p-1k}| & |x_{p-11} & x_{p-12} & \dots & x_{p-1i} & \dots & x_{p-1n}| & |v_{kn}| \end{array}$$

i.e. the k th eigenvector of S (an eigenimage) is formed by a linear combination of all the n training images; the coefficient/weight for image i being the i th component of v_k .

Procedure:

Assume we have n training images $f_i, i = 1 \dots n$, and they are of size $N \times N$; in fact, it is more convenient to deal with vectors x_i and $x'_i = (x_i - m)$ where m is the average over the n training vectors/images.

1. Compute the average image m :

$$m = \frac{1}{n} \sum_{i=1}^n x_i$$

2. Form the $n \times n$ matrix $T (= X^T X)$ where the r th component of T , $T_{rc} = x_r T_{xc}$ is the dot product of the r th and c th training vectors/images.
3. Find the n eigenvectors of T , v_k .
4. Order the v_k according to decreasing eigenvalue (see section 3.5).
5. Use eqn. 4.2-4 to compute the $n' (< n)$ most significant eigenimages, $u_k, k = 1 \dots n'$.

7.5.10 Other Connections and Discussion

In the early days of pattern recognition (late 1950s, early 60s) there were two major influences:

1. communication and radar engineering, for which the decision expression, s above, is a *matched filter*, i.e. maximum correlation device,
2. those who wished to model human pattern recognition. To these, s above is a single neuron. See chapter 8.

In all the pattern recognition approaches discussed in this section, the classification rules end up as minimum distance, or very similar. It is comforting to know that correlation and minimum distance boil down to the same thing, and that these are related to maximum likelihood, *and*, very closely, to neural networks. Furthermore, it can be shown that the nearest neighbour classifier – which is based on intuition – is very closely related to the Bayes' classifier. Also, many practical implementations of the Bayes' classifier end up using minimum distance (e.g. nearest mean).

Usually, it is the choice of features that governs the effectiveness of the system, not the classifier rule.

We now consider two practical pattern recognition problems related to image processing.

7.6 Shape and Other Features

7.6.1 Two-dimensional Shape Recognition

We now return to the character recognition problem. Of course, this is representative of many two-dimensional shape recognition problems, e.g. text-character recognition. The major problem is locating the character/shape, and extracting invariant features.

A possible solution is given by the following steps:

1. Segmentation: label ink/object pixels, versus background/non-ink pixels. See Chapter 6. This removes the effects of amplitude/brightness; also makes step (2) easier.
2. Isolate the object: e.g. starting off at the top-left corner of the object, move right and down, simultaneously scanning each column and each row; when there is a break – no ink – this marks the bottom-right corner of the object.
3. Feature Extraction: we need invariant features, e.g. 2-d. central moments, see G&W section 8.3.4 (p. 514) and the next section of these notes.

4. Gather Training Data: obtain a representative set of feature vectors for each class of object,
5. Test data: same requirement as training.
6. Train the classifier: train on the training set for each class, e.g. learn statistics (mean, standard deviation), or somehow find out the occupancy of the classes in measurement/feature space.
7. Classifier: see section 7.3:
 - Statistical: Bayes' Rule.
 - Geometric: simple nearest mean classifier, or 'box' classifier
 - nearest neighbor, or k-nn.

7.6.2 Two-dimensional Invariant Moments for Planar Shape Recognition

Assume we have isolated the object in the image (see chapter 7.4): its bounds are $x1..xh$, $y1..yh$. Two-dimensional moments are given by:

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y)$$

for $p, q = 0, 1, 2, \dots$

These are not invariant to anything, yet.

$$\tilde{x} = m_{10}/m_{00}$$

gives the x -centre of gravity of the object,
and

$$\tilde{y} = m_{01}/m_{00}$$

gives the y -centre of gravity.

Now we can obtain shift invariant features by referring all coordinates to the centre of gravity (\tilde{x}, \tilde{y}) . These are the so-called central moments:

$$m'_{pq} = \sum_x \sum_y (x - \tilde{x})^p (y - \tilde{y})^q f(x, y)$$

The first few m' can be interpreted as follows:

$m'_{00} = m_{00}$ = sum of the grey levels in the object,

$m'_{10} = m'_{01} = 0$, always, i.e. center of gravity is $(0,0)$ with respect to itself.

m'_{20} = measure of width along x -axis

m'_{02} = measure of width along y -axis.

From the m'_{pq} can be derived a set of normalized moments:

$$\mu_{pq} = m'_{pq}/((m'_{00})^g)$$

where $g = (p + q)/2 + 1$

Finally, a set of seven fully shift, rotation, and scale invariant moments can be defined:

$$\begin{aligned} p_1 &= n_{20} + n_{02} \\ p_2 &= (n_{20} - n_{02})^2 + 4n_{11}^2 \\ &\quad \text{Etc.} \end{aligned}$$

See G&W for further details.

Ex. 7.5-1 An apple pie manufacturer is setting up an automatic conveyer belt inspection system for the pastry tops for his pies. He requires:

1. the tops must be circular (to within some defined tolerance),
2. there must be no holes in the top.

The inspection can be carried out using the model given in section 7.4 and Chapter 2. Assuming the background grey level sufficiently contrasts with that of the pastry, location and segmentation should be simple.

After segmentation, inspection (2) can be carried out by searching for any background pixels in the interior of the pastry.

Inspection (1) can be done using invariant moments; we demand feature vectors within a certain small distance (the tolerance) of the perfect shape.

Note: if the input camera is always the same distance from the conveyer belt, we don't need scale invariance. And, because the pie tops are circular, we don't need rotation invariance. Thus, simple features might be (say) four, or eight diameters sampled at equally spaced angles around the pie; of course, \tilde{x} and \tilde{y} give the center of the pie top, through which the diameters must pass.

An alternative would be to detect the edges of the pie top, and thereby compute the circumference, c . Compute the diameter as above, d . Then, c/d must be close to π , otherwise it's a bad pi(e)!

Ex. 7.5-2 At Easter the pie manufacturer branches into hot-cross buns. In this case, he is interested, not only in the roundness, but in the perfection of the cross on top of the bun. Suggest possible features, and classifier techniques.

7.6.3 Classification Based on Spectral Features

For example, we require a system to automate the process of land-use mapping using multispectral (i.e. multicolour) satellite images. Or, maybe, to discriminate rain clouds from non-rain; or, to detect pollution in water.

Feature Extraction:

In this case the features are easier to obtain; in fact, the pixel values (in each colour) will do; i.e. the feature vector is the same as the observation vector – the plain radiance value in the spectral band. It might be necessary to do some preprocessing, e.g. correction for atmospheric effects.

Training data:

Obtain representative (note: *representative*) samples of each land-use class (this is called ground data, or ground-truth). This requires field work, or use existing maps.

Test data:

Same requirement as training; again – note – we need fully representative data.

Train classifier:

Train on samples for each class, learn statistics (mean, standard deviation), or somehow find out the occupancy of the classes in measurement/feature space.

Classifier:

See above. The output is an image containing class labels, these can be appropriately colour coded for map production.

Test/Evaluation:

Run the classifier on the test data. Compare ‘true’ classes, and ‘classifier’ classes. Produce error matrix, or simply percentage of pixels incorrectly classified.

NB We must use different data for testing – *not* training data.

Geometric Correction:

Scale, shift, rotate the image to correspond to map coordinates. Sometimes called geometric calibration.

Pitfalls and difficulties:

Poor features. The features just do not discriminate the classes. If the raw spectral bands do not discriminate, we may need to take the ratio of colors/bands. Or, we may need to use texture. Or, we may need to combine images from different parts of season – this will require image registration (see Chapter 3 for rotation and scaling, etc.).

Poor training data. Not representative – maybe contains only one subset of the real class – and so is not representative of the true class variability. Or, we have pixels of another class mixed in.

Mixed pixels. I.e. pixels that ‘straddle’ boundaries between class regions; bad for training data. Also difficult to classify correctly.

Overlapping classes. Some of our classes actually may have same ‘colour’; to solve this problem we need additional ‘features’ e.g. texture. Or use context, or try multiseason data – see above.

Testing on training data. This is a common error, somewhat akin to the same person specifying, writing, and testing a piece of software: the results of the tests usually give good news – whatever the reality!

7.6.4 Some Common Problems in Pattern Recognition

- Bad features. No amount of ‘clever’ classification can remedy the situation if the features are not properly descriptive.
- Large dimensionality. The larger the dimensionality of the feature vectors, the more training data required. Typically four or more samples per class, per dimension. E.g. dimensionality, $d = 50$, no. of samples = 250 per class.
- Mismatch in range of features. E.g. feature 1 ranges 0 to 1, feature 2 ranges 0 to 100: variations in feature 2 will swamp variations in feature 1, in for example, distance or correlation measures. We need to normalise the components of the feature vector, to equal ranges of (say) 0 to 1.
- Testing using training data.
- Multimodal distributions, i.e. really two classes combined. May fool statistical algorithms.

7.6.5 Problems Solvable by Pattern Recognition Techniques

Engine Monitoring

Observation vector: sound signal.

Feature vector: Fourier Spectrum of sound signal.

Classes: Healthy, not healthy – needs maintenance.

Medical Diagnosis

Observation vector: Binary – list of symptoms, e.g.

x₁ = pain in hamstring,
x₂ = history of back trouble,
x₃ = pain intermittent
x₄ = pain constant
x₅ = recent athletic activity

Feature vector: = observation vector

Classes:

w₁ = Sciatica,
w₂ = pulled muscle,
w₃ = stiffness,
w₄ = don't know – refer to specialist.

Information Retrieval

Features: presence of keywords.

Classes: subject areas.

Burgular Alarm

Features: sound level in microphone, current level in loop sensors, readings from infrared temperature sensors.

Classes: Intruder – ring bell, no intruder.

7.6.6 For Further Reading

1. S. Aeberhard, D. Coomans, and O. de Vel. 1994. Comparative Analysis of Statistical Pattern Recognition Methods in High Dimensional Settings. *Pattern Recognition*, Vol. 27, No. 8.
2. A.K. Agrawala (ed.). 1976. *Machine Recognition of Patterns*. IEEE Press. [Collection of Key Papers + tutorial].
3. J.V. Beck, and K.J. Arnold. 1977. *Parameter Estimation in Engineering and Science*. John Wiley and Sons.
4. R.O. Duda, and P.E. Hart. 1973. *Pattern Classification and Scene Analysis*. Wiley-Interscience.
5. R.A. Fisher. 1936. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*. [contained in (Agrawala, 1976)]
6. D.H. Foley, and J.W. Sammon. 1975. An Optimal Set of Discriminant Vectors. *IEEE Trans. Comp.* March.

7. K. Fukunaga. 1990. Introduction to Statistical Pattern Recognition. 2nd. ed. Academic Press.
8. K. Fukunaga, and W. Koontz. 1970. A Criterion and an Algorithm for Grouping Data. IEEE. Trans. Comp. October.
9. W.K. Pratt. 1991. Digital Image Processing. 2nd. ed. Wiley- Interscience.
10. C.W. Therrien. 1989. Decision Estimation and Classification. John Wiley and Sons.
11. M. Turk and A. Pentland. 1991. Eigenfaces for Recognition. J. Cognitive Neuroscience. Vol. 3, No. 1.

7.7 Exercises

1. Draw a histogram for the following data (one-dimensional features) from two classes:

```
class 0, w0:  
  
1.21 3.11 3.97 6.21  
1.32 3.12 4.12 6.58  
1.40 3.21 4.30 7.00  
1.56 3.31 4.70  
2.07 3.37 4.86  
2.21 3.45 4.92  
2.22 3.50 4.97  
2.73 3.78 5.10  
3.00 3.90 5.70  
  
class 1, w1:  
  
6.89 10.03 11.23 11.71 12.37  
8.01 10.31 11.25 11.82 13.01  
8.76 10.45 11.34 11.99 13.50  
9.25 10.56 11.37 12.22 13.57  
9.33 10.72 11.45 12.32 14.60  
9.76 10.80 11.60 12.33
```

Determine a decision boundary (threshold) that classifies the points with minimum error.

2. Determine the means of each class. What is the effective decision boundary for the nearest mean classifier.
3. If you wanted to use a nearest neighbour classifier, but decided to ‘condense’ the points, which points are significant and must be retained, in order to give minimum error.
4. Pick 10 men and 10 women, at random (more if you have time). Find their height – inches. Plot a histogram. Design a histogram based (statistical) classifier that distinguishes men from women.
5. Design a nearest mean classifier using the data from 4. Compare the result.
6. Add an additional, second, feature to Ex. 4, i.e. shoe size. Plot a two dimensional scatter plot (or histogram, if you wish). Plot the optimum boundary between the classes – based on the scatter plot.
7. Using the data from 6 work out the means vectors of the classes. Plot the linear boundary that would be formed by a nearest mean classifier.
8. Design a pattern recognition system for coins (money). What are the:
 - observations,
 - features.

Suggest simple classifier rules.

9. In the character recognition example given in section 7.1 the feature space is nine dimensional. Thus, visualisation of the data in feature space is difficult. The following example is easier to visualise.

Consider an imaging system which has just two pixels – or, an animal which has just two light sensitive cells in its retina, see following figure. Call the outputs of these x_1 , and x_2 , therefore they form a two-dimensional vector $x = (x_1, x_2)$.

x_1	x_2
+-----+	-----+

Two-Pixel Image

(a) If the components are binary (0 or 1) we can consider a problem which wishes to distinguish ‘bright’ objects – class 1, from dark – class 0. For now we will define class 1 as ‘both pixels light’. I.e. we have (where ‘*’ denotes light, or class 1):

x1	x2	
*****	*****	class 1
*****	*****	

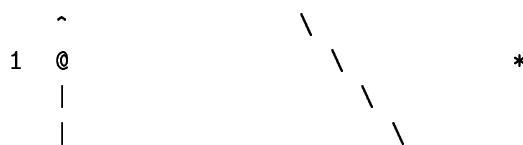
x1	x2	
*****		class 0

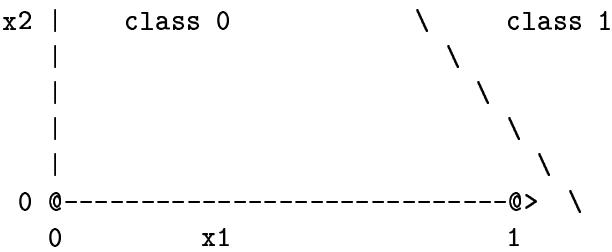
x1	x2	
	*****	class 0

x1	x2	
		class 0

Note the similarity with the Boolean *and* function.

The feature space representation of these classes are shown in the next figure; ‘@’ represents class 0, ‘*’ represents class 1. We have shown a linear boundary which segregates the classes.





Two-Dimensional Scatter Diagram - Feature Space

- (b) Let us change to a problem which wishes to distinguish striped objects (class 0, say) from plain (class 1). I.e. we have ('*' denotes light, class 1):

```

x1   x2
+-----+
|*****|*****|  class 1
|*****|*****|
+-----+
x1   x2
+-----+
|*****|      |  class 0
|*****|      |
+-----+
x1   x2
+-----+
|      |*****|  class 0
|      |*****|
+-----+
x1   x2
+-----+
|      |      |  class 1
|      |      |
+-----+

```

Draw the feature space diagram. Draw appropriate decision boundary line(s) – note the difficulty compared to (a).

Note the similarity with the Boolean XOR function.

(c) Let us change to a problem which wishes to distinguish left-handed objects (class 1, say) from right-handed (class 2), with neither left- or right-handed as reject, class 0. I.e. we have (here '*' denotes light, class 1):

x1	x2	
*****	*****	class 0
*****	*****	
-----	-----	
x1	x2	
*****		class 1

-----	-----	
x1	x2	
	*****	class 2

-----	-----	
x1	x2	
		class 0
-----	-----	

Draw the feature space diagram. Show the linear boundaries.

(d) (See (a)) Describe a state of affairs that corresponds to Boolean *or*; draw the diagrams. Will a single linear boundary do? [Yes].

10. Change the system in Ex. 9 to allow non-binary data. Allow the data to extend from 0 to +1 and assume Real values (e.g. 0.995, 0.0256). Now extend 9(a) to (d) assuming that there are small amounts of noise on the pixels, e.g. we have values spread over the range 0.9 to 1.0 for light, and 0.0 to 0.1 for dark.

Draw the feature space diagrams for each case (a) to (d).

Draw suggested linear boundaries.

11. Now let the noise in Ex. 10 increase. Now, we have values spread over the range 0.55 to 1.0 for light, and 0.0 to 0.45 for dark.

- (i) Draw the feature space diagrams for each case (a) to (d).
(ii) Draw suggested linear boundaries.
12. Now let the noise in Ex. 11 increase further. Now, we have values spread over the range 0.4 to 1.0 for light, and 0.0 to 0.6 for dark.
(i) Draw the feature space diagrams for each case (a) to (d).
(ii) Draw suggested linear boundaries.
(iii) Suggest alternatives to the ‘linear’ classifiers.
13. In section 7.31 we mentioned the problem of mixed pixels. Research ‘fuzzy sets’ and suggest how they could be applied to this problem.
[Note: a class can be considered to be defined as a *binary* membership function in feature space; i.e. it is a *set* – a feature value is either a member (1) or not (0). A fuzzy set is one which can take on continuous membership values in the range 0 to 1, e.g. 0.0, 0.1, low membership; 0.9, 0.95, 1.0 high membership].
How would fuzzy classes change our ability to define decision boundaries?
14. Iris data. [R.A. Fisher. 1936. Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics, Vol. 7, pp. 179-188.]
The following data (in files ‘ih1.asc’) contain a subset of the famous iris (flowers) data. The data are four-dimensional: x0 = sepal length, x1 = sepal width, x2 = petal length, x3 = petal width. There are two classes – corresponding to two families of iris.

```

/*dh    rh    ch
3,    0,   49
cl  x0    x1    x2    x3
1    5.1,3.5,1.4,0.2
1    4.9,3.0,1.4,0.2
1    4.7,3.2,1.3,0.2
1    4.6,3.1,1.5,0.2
1    5.0,3.6,1.4,0.2
1    5.4,3.9,1.7,0.4
1    4.6,3.4,1.4,0.3
1    5.0,3.4,1.5,0.2
1    4.4,2.9,1.4,0.2
1    4.9,3.1,1.5,0.1
1    5.4,3.7,1.5,0.2
1    4.8,3.4,1.6,0.2

```

1	4.8,3.0,1.4,0.1
1	4.3,3.0,1.1,0.1
1	5.8,4.0,1.2,0.2
1	5.7,4.4,1.5,0.4
1	5.4,3.9,1.3,0.4
1	5.1,3.5,1.4,0.3
1	5.7,3.8,1.7,0.3
1	5.1,3.8,1.5,0.3
1	5.4,3.4,1.7,0.2
1	5.1,3.7,1.5,0.4
1	4.6,3.6,1.0,0.2
1	5.1,3.3,1.7,0.5
1	4.8,3.4,1.9,0.2
2	7.0,3.2,4.7,1.4
2	6.4,3.2,4.5,1.5
2	6.9,3.1,4.9,1.5
2	5.5,2.3,4.0,1.3
2	6.5,2.8,4.6,1.5
2	5.7,2.8,4.5,1.3
2	6.3,3.3,4.7,1.6
2	4.9,2.4,3.3,1.0
2	6.6,2.9,4.6,1.3
2	5.2,2.7,3.9,1.4
2	5.0,2.0,3.5,1.0
2	5.9,3.0,4.2,1.5
2	6.0,2.2,4.0,1.0
2	6.1,2.9,4.7,1.4
2	5.6,2.9,3.6,1.3
2	6.7,3.1,4.4,1.4
2	5.6,3.0,4.5,1.5
2	5.8,2.7,4.1,1.0
2	6.2,2.2,4.5,1.5
2	5.6,2.5,3.9,1.1
2	5.9,3.2,4.8,1.8
2	6.1,2.8,4.0,1.3
2	6.3,2.5,4.9,1.5
2	6.1,2.8,4.7,1.2
2	6.4,2.9,4.3,1.3

I have split the original data into two halves; the other half is in ‘ih2.asc’.

- (a) Run DataLab batch file ‘ipiris’, which will configure and read in

ih1.asc – to image 0, and ih2.asc – to image 1. It will also show some scatter plots and histograms.

- (b) (b-1) run clnm (nearest mean classifier) on image 0 (source – training data) and image 1 (destination – test data).
- (b-2) Run ‘clcfus’ on 1 to see the result.
- (b-3) Check by examining image 1 using ‘tpsv’.
- (c) (c-1) run clnn (nearest neighbour classifier) on image 0 (source – training data) and image 1 (destination – test data).
- (c-2) Run ‘clcfus’ on 1 to see the result.
- (c-3) Check by examining image 1 using ‘tpsv’.
- (d) (d-1) run clbpnn (backpropogation neural network classifier) on image 0 (source – training data) and image 1 (destination- - test data).
- (d-2) Run ‘clcfus’ on 1 to see the result.
- (d-3) Check by examining image 1 using ‘tpsv’.

15. Here are the data introduced in section 7.3.

label	cllab	x0	x1
1	1	: 0.40	1.50
1	1	: 1.00	0.50
1	1	: 1.00	1.50
1	1	: 1.00	2.50
1	1	: 1.60	1.50
2	2	: 1.40	3.00
2	2	: 2.00	2.00
2	2	: 2.00	3.00
2	2	: 2.00	4.00
2	2	: 2.60	3.00

- (a) Draw a scatter plot for the data [do this first, it will make the rest of the question easier],
- (b) Work out the class means,
- (c) Hence, apply a nearest mean classifier to the patterns:

x0	x1
1.0	, 1.0

2.0,3.5
5.0,5.0

- (d) Compare the error results that you would obtain by applying (i) a nearest mean classifier, and (ii) a nearest neighbour classifier to the training data (i.e. the training data are the data in the tables above, and you are also using these data for testing).
- (e) Illustrate a linear boundary classifier that would suit these training data.
- (f) Design and demonstrate a neural network classifier for these data; comment on the relationship between your result and that obtained in (e).
- (g) Design and demonstrate a maximum likelihood classifier for these data.

Chapter 8

Neural Networks: From the Perceptron to the Multilayer Perceptron

8.1 Introduction

In the search for efficient computing structures for artificial intelligence and knowledge-based systems, one natural and reasonable approach is to attempt to model the workings of mammalian brains.

The term ‘artificial neural network’ or simply ‘neural network’ refers to computing architectures which are supposedly based on the networks present in brains.

There are two common motivations for the study of neural networks:

- to research computational models of human/mammalian mental activity.
- as novel computational structures and algorithms.

We will focus on the latter (algorithmic), and show that neural networks perform well at a range of computational tasks, and, moreover, that they are strongly related to many well known traditional algorithms.

In addition, our view is that, while human cognitive and computational processes are surely of interest, so little that is known seems to be practically implementable that psychology seems mostly irrelevant to those who are attempting to develop artificially intelligent systems. Indeed, there is good reason to question the validity of the term ‘artificial intelligence’, *per se*; it is significant that research in this area is often now called ‘knowledge engineering’. Furthermore, perhaps the history of mechanical intelligence may parallel that of mechanical flight: the real progress was made when the obsession with feathers and flapping was removed!

However, we will initially accept the claim that, at a mechanical level, much of what goes on in human brains can be expressed in terms of (1) pattern recognition, and (2) computation of functions – either logical functions or numerical functions. We will show that artificial neural networks are capable of performing simple versions of these tasks.

As discussed in Chapter 7, pattern recognition is concerned with ‘making sense’ of what we see, hear, smell, touch. When you see a face that you have seen before, you recognise it: learning, perception, recognition. In the example developed below, we show a very simple model of text character recognition.

Very roughly, in the context of artificial intelligence and knowledge-based systems, ‘computation of functions’ is to do with creating some new information out of information you already have, e.g. you recognise the character ‘2’ followed by ‘+’ followed by ‘5’ and you can infer ‘7’, i.e. arithmetic.

For engineers and computer scientists (forgetting now psychologists and neurobiologists), interest in neural networks is twofold:

- the fact that the algorithms associated with them seem to be efficient at certain tasks, e.g. pattern recognition; they can ‘learn’ from examples, and are ‘model-free’, unlike some competing statistical algorithms (e.g. multiple linear regression).
- they are implementable in parallel and special purpose hardware, i.e. they can be made to work *fast*.

Additional impetus arises from the possibility of implementing neural networks using optical components – fast, and use little power.

In this lecture we will explain some of the applications of neural networks to image processing – specifically pattern recognition.

First, in section 9.2, we give a historical background, and discuss the early motivation for neural network research. Then, in section 9.3, we describe the basics of artificial neurons, their relationship with ‘real’ neurons, and go on to describe perceptrons and multilayer networks. Next, section 9.4, gives a very brief introduction to the implementation of neural networks in software and hardware. Section 9.5 introduces training. Since most of the chapter is on backpropagation trained multilayer feedforward networks, section 9.6 mentions some other architectures. Section 9.7 is conclusions and summary. Finally, Section 9.8 gives a bibliography and references.

8.2 Historical Background

See Nagy (1991), Widrow and Lehr (1990), Hecht-Nielsen (1991). The initial studies of neural networks started in the 1940s by psychologists trying to come up with a mechanical/mathematical model of human thought

(MacCarthy 1955). It was then taken up in the 1950s by the AI (artificial-intelligence) community, especially those interested in pattern recognition, – who, not unreasonably, reckoned that the best way to produce artificial intelligence was to produce artificial brains. And, since brains were believed to be made of networks of ‘processing units’ that we call neurons, how better to produce artificial brains than use artificial neural networks.

Significant work on neurophysiology started about 1850, however, the earliest notable paper is McCullough and Pitts (1943), which started by identifying the equivalence of the ON/OFF response of a neuron with a (logical) proposition, i.e. has value true or false.

They then went on to show how simple one- and two-input neurons could implement the NOT, AND, and OR Boolean functions. Consequently, of course, more complex Boolean functions are only a matter of connecting up a network.

Some early work in machine pattern recognition focussed on human pattern recognition from a psychological perspective (e.g. Deutsch 1955), and other contributions in the collection Uhr (1966). Others focussed on the physiological structure of mammalian brains and vision and auditory systems, e.g. (Hubel and Wiesel 1962). Work on the brains of frogs, culminating in Barlow (1953), identified evidence for a ‘matched detector’ for small dark objects (e.g. a fly) – a ‘fly detector’ neuron that ‘fires’ when a fly-like object enters the frogs visual field; this was an inspiration for the ‘perceptron’.

Starting around 1956, Frank Rosenblatt, at Cornell University, invented and built the ‘Perceptron’, which, amongst other things, was used to model the processing that happens in the visual cortex – the part of the brain that does initial processing on signals sent from the retina (sensitive part of the eye). It was shown that a perceptron could recognise (differentiate between) different patterns (see next section for definition of a pattern). More importantly, it was proved mathematically, and demonstrated that a perceptron could ‘learn’; by giving it example patterns along with what each pattern ‘represents’ you can get it to self-organise (learn) such that if a pattern similar to one of the learned patterns is encountered, the machine can recognise what it represents. See, for example, Duda and Hart (1973), Block (1962), Rosenblatt (1960), Hecht-Nielsen (1991).

There was also an active group at Stanford led by Bernard Widrow, (Widrow and Lehr, 1990), which developed perceptron-like structures: Adaline, and Madaline. They developed the Widrow-Hoff or ‘delta-rule’ training algorithm for Adaline (effectively a perceptron).

However, at the same time, another school of artificial intelligence was being set up – mostly based on ‘symbolic processing’; this is what you will read in most current books on AI (see Minsky, 1960). The language LISP was developed for this sort of AI. PROLOG (PROgramming LOGic) (Bratko, 1991) is another (more advanced) example of a symbolic processing approach to AI and knowledge-based systems.

Also, much of the pattern recognition work going on was based on statistical theory – statistical decision theory; statistical pattern recognition was probably born in Chow (1957), but statistical decision theory has been around since the early 1900s.

Actually, statistics and statistical decision theory are remarkably good examples of knowledge-based systems: knowledge capture may be easy (estimate the statistical parameters), representation is easy (the parameters), and statistical inference and decision theory are well understood.

Much work was done on perceptron-like structures until 1969, when Marvin Minsky and Seymour Papert (Massachusetts Institute of Technology) published a book called ‘Perceptrons’, Minsky and Papert (1969), which showed that a simple, single layer perceptron could not differentiate between certain quite dissimilar patterns (see next section); in addition, they showed that a perceptron could not compute a very simple function – the XOR function. (Minsky had studied computational aspects of neural networks in his PhD. thesis, 1953 – but I don’t know of any publications arising from that time). This Minsky-Papert attack proved a great setback to neural network research; it caused what neural net researchers call the ‘Dark Ages’ – government funding almost completely dried up.

Hecht-Nielsen (1991) attributes a conspiratorial motive to Minsky and Papert, namely, that the MIT AI Laboratory had just been set up and was focussing on LISP based AI, and needed to spike other consumers of grants. A good story, whatever the truth, and given extra spice by the coincidence that Minsky and Rosenblatt attended the same high-school, and the same class. Moreover, any bitterness is probably justified because neural network researchers spent the best part of 20 years in the wilderness.

Work did not stop however, and the current upsurge of interest began in 1986 with the famous PDP books which announced the invention of a viable training algorithm (backpropagation) for multilayer networks (Rumelhart and McClelland, 1986).

8.3 Neural Networks Basics

8.3.1 Introduction

This section motivates the study of neural networks by demonstrating how neural nets do pattern recognition and compute functions. First we give a simplified account of ‘real’ neurons. Then we show how a very simple neuron can perform a limited pattern recognition task. We introduce the ‘perceptron’, and its limitations. Then we show how these limitations can be mitigated by combining neurons in ‘layers’.

8.3.2 Brain Cells

Here is a very brief mention of how mammalian brains are thought to be constructed; and, at a small scale, one theory of how they ‘work’. These brains are composed from networks of interconnected neurons.

The following figure shows a *neuron* and its surroundings. A neuron has a *cell body* with one *axon* stretching out from the cell body, which also has many *dendrites* protruding from it.

A ‘Real’ Neuron.

The *axon* is the channel by which the neuron sends signals to other neurons. These signals are in the form of a series of electrical pulses – the more frequent the pulses, the stronger the effect of the signal. *Axons* only transmit – away from their neuron.

Dendrites receive signals from the axons of other neurons. Between the axons of the transmitter neuron and the dendrites of receiver neurons are *synapses*. Synapses are narrow regions of conductive material; the strength of the excitation signal received by the receiver neuron depends on how well the pulses are conducted from the axon (sender) to dentrite (receiver).

Neurons operate by sending signals between one another, and each neuron fires (sends pulses) only if it receives a minimum number of excitatory pulses via its dendrites. The number of excitatory pulses received obviously depends on (a) the amount of pulses injected by the connected axons (from other neurons), and (b) the amount of those pulses that are conducted by the synapses.

There are a great many neurons in a typical human brain, perhaps 10^{11} . There are even more synapses – perhaps 10^{16} . There is good reason to believe that synapses form what we understand as memory.

Artificial neural networks were initially studied in an attempt to model the brain. In fact, the similarity – actual to artificial – may be relatively slight; however, artificial neural networks are now used as computing structures in their own right.

In the *retina* of an eye there are light sensitive cells – like neurons; when a cell is illuminated it will transmit pulses via its *axon* to one (or more than one) other neuron(s).

8.3.3 Artificial Neurons

The feedforward neural network shown in the following figure is a parallel network of neurons (processing units, sometimes called nodes).

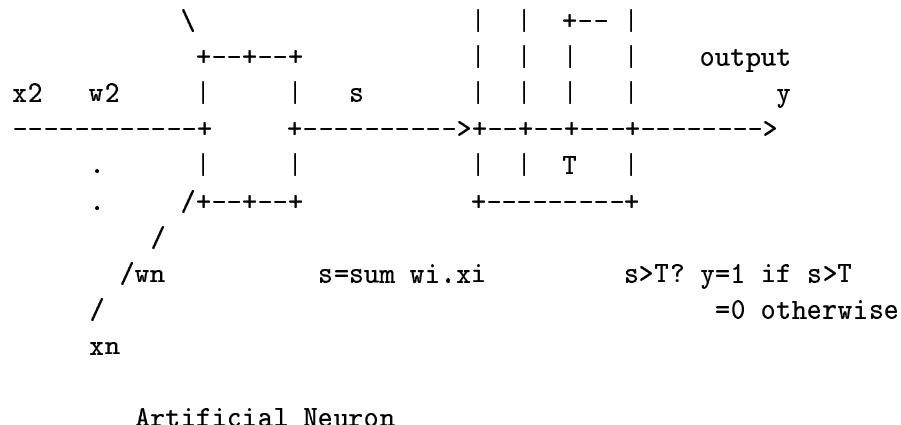
Each circle in the ANN figure represents a neuron of the form shown; each performs the weighted-sum given by the first of the two following equations, followed by application of the threshold given by the second equation and shown in the figure.

$$\text{sum} = \sum_{i=1,n} w_i x_i$$

$$\text{output} = y = 1 \text{ if sum } > T, \text{ and } 0 \text{ otherwise}$$

This equation can be written in the vector notation, $s = x'w$, – we can consider a single neuron as a ‘template matcher’.





Artificial Neuron

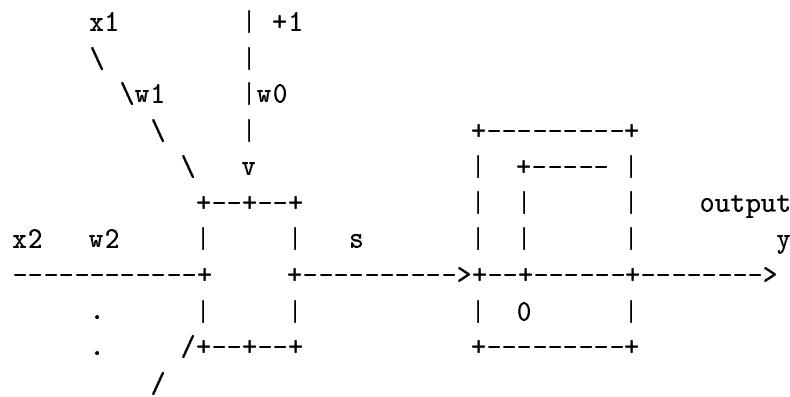
In fact, the two equations can be expressed much more neatly by bringing in the negative of the threshold (T) as a weight, w_0 , which is always tied to a +1 signal:

$$\text{sum} = \sum_{i=0..n} w_i x_i$$

where $w_0 = -T$ and the summation is now over $0..n$. Now, the thresholding simplifies to:

$$\text{output} = 1 \text{ if sum} > 0, \text{ and } 0 \text{ otherwise}$$

The weight w_0 tied to +1 represents the ‘so-called’ *bias* input – you could think of it as an ‘inhibitory’ input, since in the normal sense, w_0 will be negative, whereas the other inputs are all ‘excitory’. These equations are shown in the following two figures. The second of these two figures makes the threshold function implicit; this is by far the most common way of representing neurons.

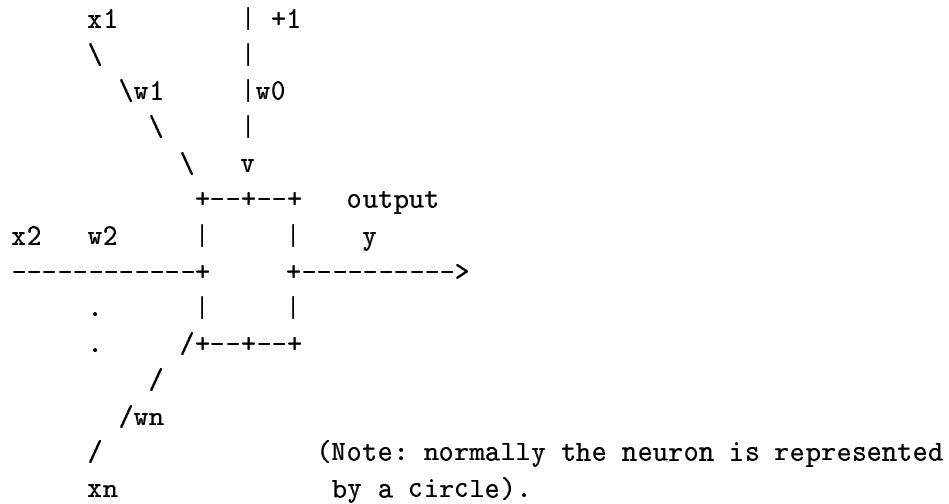


```

/wn           s=sum wi.xi    s>0?      y=1 if s>0
/             |                   =0 otherwise
xn

```

Artificial Neuron - with Bias.



Artificial Neuron - Threshold Implicit

Neural networks are trained rather than programmed: that is, the weights are adjusted to provide a best fit representation for a (training) set of examples of pairs (x,y) . A training rule called ‘back-propagation’, which can effectively train multi-layer networks, has made multilayer networks a practical reality; see section 8.5.

The weights correspond to the synapses mentioned in section 8.3.1, and it is these weights that represent the memory/knowledge-base of the system.

8.3.4 Neural Networks and Knowledge Based Systems

Knowledge-based systems form a branch of artificial intelligence; to some extend they represent a milder form of ‘expert-system’ – with, perhaps, the aims slightly lowered.

Knowledge-based systems try to automate the sort of complex decision task that confronts, for example, a medical doctor during diagnosis. No two cases are the same, different cases may carry different amounts of evidence. In essence, the doctor makes a decision based on a large number of variables,

but some variables may be unknown for some patients, some may pale into insignificance given certain values for others, etc. This process is most difficult to codify. However, there are sufficient advantages for us to try: if we could codify the expertise of a specialist, waiting lists could be shortened, the expertise could be distributed more easily, the expertise would not die with the specialist, etc.

There are four major parts in a knowledge based system:

Knowledge elicitation: this is the extraction of knowledge from the expert; it may be done by person to person interview, by questionairre, or by specialised computer program,

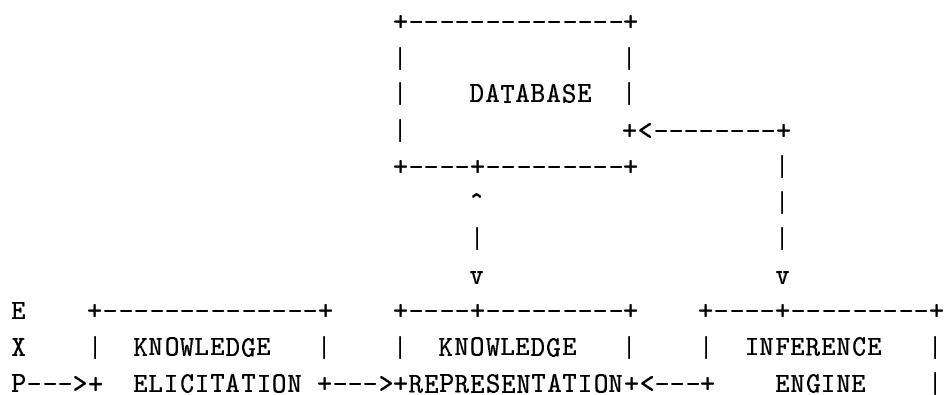
Knowledge representation: we must code the knowledge in a manner that allows it to be stored and retrieved on a computer,

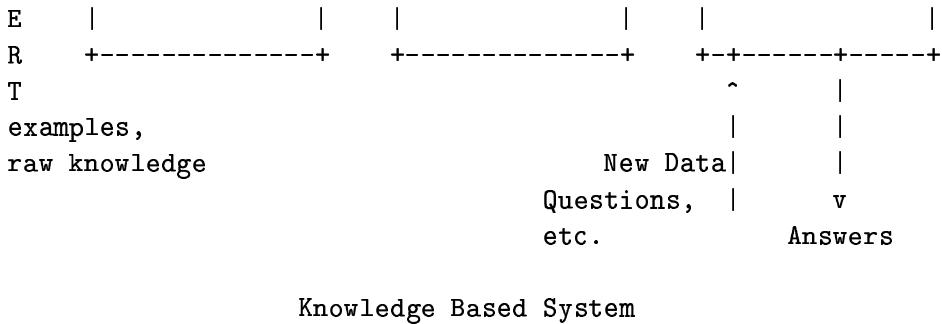
Knowledge database: where the knowledge is stored (using the ‘representation’ code mentioned above),

Inference engine: this takes data and questions from a user and provides answers, and/or updates the knowledge database.

The following figure depicts a possible organisation and operation of a knowledge based system.

Actually, a well designed database with a good database management system, coupled with a query language that is usable by non-computing people, goes a long way to fulfilling the requirements of a knowledge based system. Also, some of the pattern recognition systems we mention, could be called knowledge based – after all, they store knowledge (the training data or some summary of it) and make inferences (based on the measured data or feature vector); feature extraction is very similar, in principle, to knwoledge representation. Furthermore, we note that neural networks show much promise as knowledge-based systems.





Where do neural networks fit in here?

First, the inference engine. This is the neural network. The questions/new data are the input data – input at the nodes at the left of the network the xs, the answers are the outputs – from the right hand side of the network, the ys. You may worry that the inputs and outputs of the network are numerical, but is not too difficult to code non-numerical inputs to numerical, and to translate numerical outputs into some appropriate form – e.g. voice synthesis.

Second, the knowledge database. Well, this is in the network too – the weights represent all the knowledge in the system.

Third, knowledge elicitation and knowledge representation. These are done by presenting to the network training algorithm, representative examples (inputs and outputs) of what expertise the network must eventually emulate.

One big advantage, and also a disadvantage, with neural networks is that the knowledge about a particular topic may be distributed over many of the weights. This (advantage) makes them robust to damage to parts of the system.

On the other hand, it makes them difficult to understand/debug, i.e. they are ‘opaque’ – it is not easy to get an explanation from them (a requirement in many expert systems/KBS). In a rule based system (even a fuzzy rule based system), it is possible to identify which rules have fired, and hence, for example, identify the cause of a bad or peculiar decision.

8.3.5 Neurons for Recognising Patterns

[This is meant to be only a motivating example, so do not take it too literally; in fact, some of the examples in section 8.3.12 may be more appropriate and easier understood.]

Imagine that we have nine light sensitive neurons, one corresponding to each of the nine cells shown below, the letter ‘C’ etc. And imagine that there are nine axons from each of the receptor neurons, and all nine of them connected to the dendrites of a single ‘C’ recognising neuron. Assuming the

character is white-on-black and that bright (filled in with '*') corresponds to '1', and dark to '0', the array corresponding to the 'C' is

```
x[1]=1, x[2]=1, x[3]=1, x[4]=1, x[5]=0, x[6]=0, x[7]=1, x[8]=1,
x[9]=1.
```

Pixel number:

1	2	3
+---+---+---+		
**** **** ****		
**** **** ****		
+---+---+---+		
4 **** 5 6		

+---+---+---+		
**** **** ****		
**** **** ****		
+---+---+---+		
7 8 9		

A Letter 'C'

The letter 'T' would give a different observation vector:

'T': 1,1,1, 0,1,0, 0,1,0

'O': 1,1,1, 1,0,1, 1,1,1

'C': 1,1,1, 1,0,0, 1,1,1

etc.

So how is the recognition done?

- (1) Pixel-by-pixel comparison: Compare the input (candidate) image pixel for pixel; we could code this up in a rule-based system:

```
if (x[1] == 1) and (x[2] == 1) and (x[3] == 1) and
(x[4] == 1) and (x[5] == 0) and (x[6] == 0) and
(x[7] == 1) and (x[8] == 1) and (x[9] == 1)
```

then letter is 'C'.

But we haven't really solved anything: any minor difference in any pixel would fool the system, e.g. addition of a small amount of noise.

(a) The recognition system needs to be invariant (tolerant) to noise.

(b) What if there is a minor change in grey level? grey Cs are the same as white Cs: the system needs to be amplitude invariant – tolerant to changes in amplitude.

(2) Maximum Correlation or Template Matching: Compute the correlation (match) of x with each of the templates for each of the potential letters, and choose the character with maximum correlation. That is, we choose letter with maximum corr(elation); this is described mathematically as follows:

$$x'X_j = \sum_{i=0}^{p-1} x_i X_{ij}$$

i.e. the dot product of x and X_j where the latter is the j th template letter.

This is called *template matching* because we are matching (correlating) each template (the X_j s), and choosing the one that matches best.

Template matching is more immune to noise – we expect the ups and downs of the noise to balance one another.

Returning to artificial neurons, we want to make the 'C' neuron emit a '1' when a 'C' appears, and '0' otherwise.

If we set up the following vector of weights, w_C , for the 'C' recognising neuron:

```
set the threshold = w0 = - 6.5

vector element 0      1 2 3      4 5 6      7 8 9

wC     = [-6.5,  1,1,1,      1,-1,-1,  1,1,1]
```

then for a 'C' input we get:

$$\begin{aligned} \text{sumC} &= -6.5 + 1+1+1+ \quad +1-0-0, \quad +1+1+1 \\ &= 0.5 \end{aligned}$$

which is > 0 , so the neuron will fire for a 'C'.

For a 'T' input we get:

$$\begin{aligned} \text{sumT} &= -6.5 + 1+1+1 + 0-1-0 + 0+1+0 \\ &= -3.5 \end{aligned}$$

which is < 0 and so the neuron does not fire.

The foregoing discussion is grossly simplified; for example, we have not mentioned ‘shift-invariance’: a ‘C’ is a ‘C’ no matter where it appears in the visual field.

[Actually, we would get better discrimination if we substituted for 0, -1 in the above, but that is fine detail which will not concern us for now]

8.3.6 Perceptrons

The neuron presented in section 8.3.3 is a single *perceptron*; actually, it is a ‘straight-through’ perceptron. The distinction is necessary because Rosenblatt’s perceptron had an additional layer of so-called ‘associator’ units between the retina and the inputs to the variable weights, see the following figure.

Each input of an associator unit is connected via a weight to some, relatively randomly positioned, cell in the retina; as well as random positioning, the weights are randomly distributed in $\{-1, 0, +1\}$.

Perceptron with Associator Units.

8.3.7 Neural Network Training

As stated earlier, neural networks are trained – not programmed. Therefore, the simple ‘C’ neuron would be trained presenting it with a number of examples of ‘Cs’ and of the other letters, and by adjusting the nine weights until it reliably gave ‘1’ for ‘Cs’ and ‘0’ for the others.

8.3.8 Limitations of Perceptrons

For ease of explanation we reduce now the input vector to two dimensions.

Recall the following equations:

$$\text{sum} = \sum_{i=0..n} w_i x_i$$

where $w_0 = -T$ and the summation is now over $0, 1, 2 (n = 2)$;
and the neuron fires if $\text{sum} > 0$ implying, when writing out the equation in full:

$$\text{sum} = w_0(+1) + w_1.x_1 + w_2.x_2 \quad (= f(x, w))$$

$$\begin{aligned} &> 0 \text{ for fire i.e. 1 output} \\ &\leq 0 \text{ for 0 output.} \end{aligned}$$

Thus, the (sharp) boundary between the $f_t(x, w) = 1$, and $f_t(x, w) = 0$ (f_t is thresholded function) is given by:

$$f(x, w) = 0$$

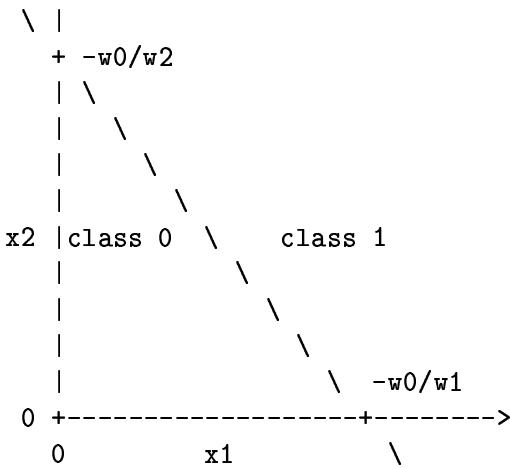
i.e.

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

i.e.

$$w_1 x_1 + w_2 x_2 = -w_0$$

This is a straight line which cuts the x_1 axis at $-(w_0/w_1)$ and cuts the x_2 axis at $-(w_0/w_2)$. See the following figure.



Perceptron Linear Decision Boundary

Thus, the perceptron can only discriminate between patterns which can be separated by a (single) straight line. Likewise functions, see section 8.3.8: OR and AND can be computed, BUT XOR cannot; see exercises in section 8.3.12. This was one of the achilles heels that Minsky and Papert successfully attacked.

8.3.9 Neurons for Computing Functions

The neuron in the figure to follow can compute the AND function. The AND function is:

x_1	x_2	$AND(x_1, x_2)$	Neuron summation	Hard-limit ($>0?$)
0	0	0	$sum = -1x0.5 + 0.35x0 + 0.35x2 = -0.5 \Rightarrow output=0$	
1	0	0	$sum = -1x0.5 + 0.35x0 + 0.35x2 = -0.15 \Rightarrow output=0$	
0	1	0	$sum = -1x0.5 + 0.35x0 + 0.35x2 = -0.15 \Rightarrow output=0$	
1	1	1	$sum = -1x0.5 + 0.35x0 + 0.35x2 = +0.2 \Rightarrow output=1$	


```

    x1      +1
    \       |
    \0.35   |
    \       |-0.5
    \       |
    +----+
    x2      0.35  |
    -----+      +-----> F
                    |
                    +----+
  
```

AND Function via Neural Network

Ex. 8.3-1 Work out the weights required for an OR function.

Ex. 8.3-2 (a) Are the weights for any function unique? *Answer:* No. (b) Rationalise why this is the case (non-uniqueness of weights).

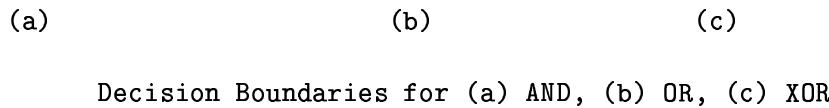
The example in the next figure shows how a *two-layer* network of two interconnected neurons can be used to compute the XOR function; of course, this is obvious from our knowledge of Boolean algebra:

$$A \text{ XOR } B = A \text{ AND } B' \text{ OR } A' \text{ AND } B$$

where \neg denotes complement.

XOR Function via Neural Network.

The following figure shows the AND, OR and XOR functions plotted in the (x_1, x_2) plane, together with appropriate boundaries: linear for AND, OR, while XOR needs the ORing of two decision regions.



Decision Boundaries for (a) AND, (b) OR, (c) XOR

Even though we have analyses only these simple functions, it should be obvious that combinations of neurons – in a network – can implement arbitrarily complex functions, with many inputs.

8.3.10 Complex Boundaries via Multiple Layer Nets

We have shown in the previous section how two-layers can implement a non-linear decision boundary, now we give qualitative arguments to show that two- and three-layer networks can implement more complex boundaries/decision regions; actually, three-layers can implement arbitrarily complex boundaries.

The following figures show two input neurons fed into a second-layer neuron that implements the AND function as given in the previous section. Each of the input neurons implements a linear boundary, and ANDing the boundaries produces the decision region shown. Now, it is easy to argue that

N input neurons, ANDed together, can yield *any* arbitrary open or closed convex decision region as shown.

Finally, if we add another, third, layer that effectively ORs the convex regions produced by the second layer, we can obtain completely arbitrary decision regions.

Feeding perceptron output to another perceptron

- (a) ANDing two linear boundaries
- (b) (c) ANDing many linear boundaries.
- (d) Third layer

Complex Decision Regions via Multiple Layers.

8.3.11 ‘Soft’ Threshold Functions

Up to now we have used the hard-limit (McCullough-Pitts ‘all-or-nothing’) neuron activation function:

$$\begin{aligned} \text{output} &= 1 \text{ if sum } > 0 \\ &= 0 \text{ otherwise} \end{aligned}$$

For reasons mostly to do with training, most neural networks now use a ‘softer’ activation function, namely the sigmoid function (also called the logistic function):

$$\text{output} = 1/(1 + \exp(-a \text{ sum}))$$

represented in the following figure. Usually, the ‘gain’ factor is set to 1.0 (obviously, setting a to a very high value yields the simple threshold function (> 0)).

Sigmoid Function

8.3.12 Multilayer Feedforward Neural Network

The generalised feedforward neural network is a parallel network of neurons. Although there is no intrinsic reason against a general topology, so long as the data flow forward, the layered structure, in which outputs from layer n flow only to inputs in layer $n + 1$, is preferred in most practical hardware and software implementations.

Early neural network designs, e.g. Nilsson (1965), used so-called threshold (hard-limit) activation functions mentioned earlier; however, the sigmoid is preferred where the network is required to provide other than ‘hard’ decision outputs, and, particularly, for ease of training – see below.

As indicated earlier, the bias inputs are important in that they give a neuron freedom to shift the position of its threshold.

The neural network architecture is defined by: number of input nodes, number of output nodes, number of processing layers, number of nodes in each processing layer; its ‘memory’ is the matrix of weights for each processing layer. In the literature, there is often confusion as to what constitutes a layer; we adopt and recommend the convention that networks are named according to the number of processing layers; thus, the figure shown earlier in this Chapter is two-layer; the input ‘layer’ does not count because it does no processing, but the output layer does. By convention, processing layers whose outputs are not available outside the network are called ‘hidden’.

Clearly, the number of nodes in the input and output layers are fixed by the problem; but, the number and content of the hidden layers are free. One layer networks are rather trivial, in that they are simply one neuron

per output. Two layers are common but three layers are more general - see the previous section. After the number of processing layers is specified, it remains to specify the number of nodes in the hidden layers. For a two-layer net (one hidden layer) Eberhart and Dobbins (1990) suggest

```
numberhidden = sqrt(numberin+numberout+2)
```

8.3.13 Exercises

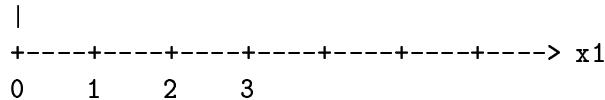
Ex. 1 (a) Plot the following data on a two-dimensional surface; note: the class means are class 0 = (1.0,1.5), class 1 = (2.0,3.0).

class (y)	x1	x2
0	0.40	1.50
0	1.00	0.50
0	1.00	1.50
0	1.00	2.50
0	1.60	1.50
1	1.40	3.00
1	2.00	2.00
1	2.00	3.00
1	2.00	4.00
1	2.60	3.00


```

+
|
4 +      1
|
+
|
3 +      1  1  1
|
+
0
|
2 +      1
|
+ 0  0  0
|
1 +
|
+      0

```



Feature Space Diagram

- (b) Verify that a single neuron neural network with the following weights:

```
w0 = 38.0    (bias)
w1 = -13.6   weight on x1
w2 = -8.0    weight on x2
```

will discriminate between the two classes.

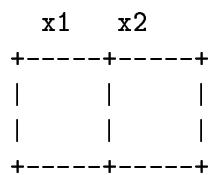
- (c) Recall the two-class scatter plot figure earlier in the chapter, and draw the class boundary. Hint:

```
intercepts y-axis at -w0/w2 = 4.5
x-axis at -w0/w1 = 2.8
```

- (d) Verify that this boundary line approximately bisects the line joining the two means.

Ex. 2 In the character recognition example given in section 8.3.4 the feature space is nine dimensional. Thus, visualization of the data in feature space is difficult. The following example is easier to visualize.

Consider an imaging system which has just two pixels – or, a simple organism which has just two light sensitive cells in its retina, see following figure. Call the outputs of these x_1 , and x_2 , therefore they form a two-dimensional vector $x = (x_1, x_2)$.



Two Pixel Image

(a) If the components are binary (0 or 1) we can consider a problem which wishes to distinguish ‘bright’ objects – class 1, from dark – class 0. For now we will define class 1 as ‘both pixels light’. I.e. ‘*’ denotes light and class 1:

x1	x2	
*****	*****	class 1
*****	*****	

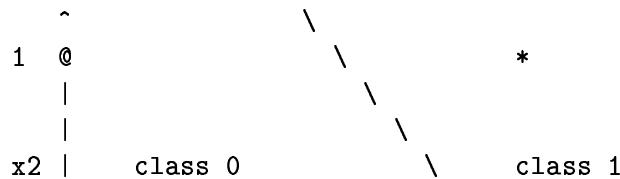
x1	x2	
*****		class 0

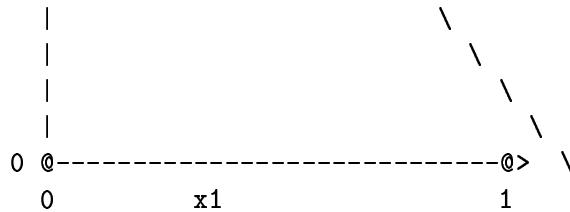
x1	x2	
	*****	class 0

x1	x2	
		class 0

Note the similarity with the Boolean AND function.

The feature space representation of these classes are shown in the figure below; ‘@’ represents class 0, ‘*’ represents class 1. We have shown a linear boundary which segregates the classes.





Two-dimensional Scatter Diagram – Feature Space

- (b) Let us change to a problem which wishes to distinguish striped objects (class 0, say) from plain (class 1). I.e. we have (where '*' denotes light, class 1):

x1	x2	
*****	*****	class 1
*****	*****	
-----+-----+		
x1	x2	
-----+-----+		
*****		class 0

-----+-----+		
x1	x2	
-----+-----+		
	*****	class 0

-----+-----+		
x1	x2	
-----+-----+		
		class 1
-----+-----+		

Draw the feature space diagram. Draw appropriate decision boundary line(s) - note the difficulty compared to (a).

Note the similarity with the Boolean XOR function.

(c) Let us change to a problem which wishes to distinguish left-handed objects (class 1, say) from right-handed (class 2), with neither left- or right-handed as reject, class 0. I.e. we have (where '*' denotes light, class 1):

x1	x2	
*****	*****	class 0
*****	*****	
-----	-----	
x1	x2	
-----	-----	
*****		class 1

-----	-----	
x1	x2	
-----	-----	
	*****	class 2

-----	-----	
x1	x2	
-----	-----	
		class 0
-----	-----	

Draw the feature space diagram. Show the linear boundaries.

(d) (See (a)) Describe a state of affairs that corresponds to Boolean OR; draw the diagrams. Will a single linear boundary do? [Yes].

Ex. 3 Change the system in Ex. 2 to allow non-binary data. Allow the data to extend from 0 to +1 and assume Real values (e.g. 0.995, 0.0256). Now extend 2(a) to (d) assuming that there are small amounts of noise on the pixels, e.g. we have values spread over the range 0.9 to 1.0 for light, and 0.0 to 0.1 for dark.

Draw the feature space diagrams for each case (a) to (d).

Draw suggested linear boundaries.

Ex. 4 Now let the noise in Ex. 3 increase. Now, we have values spread over the range 0.55 to 1.0 for light, and 0.0 to 0.45 for dark.

- (i) Draw the feature space diagrams for each case (a) to (d).
- (ii) Draw suggested linear boundaries.

Ex. 5 Consider the following trivialised credit-worthiness expert system (see Luger and Stubblefield, p. 484 for a less trivial example of the same thing). Let x_1 represent age of the client, let x_2 denote collateral. Code age as > 25 : $x_1 = 1$, 0 otherwise; has collateral: $x_2 = 1$, has not $x_2 = 0$. Consider the following set of examples:

age	collat.	
x_1	x_2	loan? y
<hr/>		
1	1	yes (code as 1)
0	1	no (code as 0)
1	0	no 0
0	0	no 0
<hr/>		

Now this knowledge can be represented by a single neuron network – it is the AND function encountered in section 9.3.8.

Obviously, it is possible, more realistically, as in exs. 3 and 4 to allow x_1 and x_2 to assume non-binary values.

Ex. 6 Consider another loan-assessemnt system. The expert system must capture the following examples. x_1 is annual salary (for convenience of this example) divided by 100,000, salary 20,000, $x_1 = 0.2$. x_2 is years owning own residence (again for convenience of the example) divided by 2, i.e. 1 year, $x_2 = 0.5$.

salary	resid.	x_1	x_2	loan
<hr/>				
5,000	0	0.05	0	no
5,000	0.5	0.05	0.25	no
25,000	0	0.25	0	no
10,000	0.2	0.1	0.1	no
50,000	0	0.5	0	yes
40,000	0.2	0.4	0.1	yes
30,000	0.4	0.3	0.2	yes
20,000	0.6	0.2	0.3	yes
10,000	0.8	0.1	0.4	yes

5,000	1.0	0.05	0.5

- (a) Explain how such ‘knowledge’ can be captured in a single neuron.

Hint: recall OR function.

- (b) Plot the data on a scatter plot, and show that the ‘yes’ and the ‘no’ can be separated by a linear boundary. Answer: a line joining ($x=0, y=0.4$) to ($x=0.4, y=0$) does the trick.

- (c) Verify that the following weights implement an appropriate boundary:

```
w0 = -0.4
w1 = 1.0
w2 = 1.0
```

Answer 1. $-w_0/w_2 = 0.4$, y-axis intercept
 $-w_0/w_1 = 0.4$, x-axis intercept
and see (c)

Answer 2. Fill in some values from the table and see, recall the equation of neural node:

```
sum = 1.w0 + x1.w1 + x2.w2
      (bias) (input 1) (inp. 2)

      if sum > 0, then output = 1
                  else output = 0
```

- (d) How could this arrangement be extended to include more input variables?
(e) How could this arrangement be extended to include more output variables?

8.4 Implementation

8.4.1 Software

Currently, most neural networks are implemented in software, for example, the neuron in eqns. 9.3-3 and 9.3-4 could be implemented as:

```

sum: FLOAT;
w : ARRAY [0..9] of FLOAT;
x : ARRAY [0..9] of FLOAT;
(*NB element 0 = bias*)
output,i : INTEGER;

...initialise x

...initialise w

sum:=0;
FOR i=0 TO 9 DO
    sum:=sum+w[i]*x[i];
END;

IF (sum > 0) output =1;
ELSE output =0;

```

8.4.2 Hardware

If we are to operate neural networks in real-time, we must implement them in parallel or some sort of special purpose fast hardware, so, there are plenty of hardware implemented (digital) neural network boards as add-ons for PCs; usually these are based on DSP chips such as the Texas Instruments TMS32040, or the Intel i860.

Also, there are a number of analogue integrated circuit neural network chips, see e.g. Brauch et al (1992), and IEEE (1992), IEEE (1993) for special issues on neural network hardware.

In the past variable weights were a problem; Rosenblatt's Perceptron used variable resistors driven by motors. Bernard Widrow formed a company which produced (profitably, by all accounts) a device called a 'memistor' (memory-resistor). The memistor was sort-of 'liquid-state'! it used a copper wire (the variable resistor) immersed in a copper sulphate solution; the copper wire was a cathode and there was a copper plate anode; appropriate voltage level and polarity deposited or removed copper from the wire, thus reducing its cross-section area and hence its resistance.

8.4.3 Optical Implementations

Obviously fast, and low power.

- optical multipliers are easy in principle, light \times transmissivity,
- summer, just use the summing effect of a sensor.

8.5 Training Neural Networks

8.5.1 Introduction

Up until 1986 (Rumelhart and McClelland, 1986) training was the big problem. Neural networks are trained rather than programmed: that is, the weights are adjusted to provide a best fit representation for a (training) set of examples of pairs (x,y) , i.e. (input vector, output).

It was clear enough (and pointed out by Minsky and Papert (1969)) that multilayer nets could possibly overcome some of the problems of the single layer – but multiple layer couldn't be trained.

A training rule called ‘back-propagation’, which can effectively train multi-layer networks, has made multilayer networks a practical reality.

8.5.2 Hebbian Learning Algorithm

D.O. Hebb in 1949 proposed a neural learning algorithm that has been highly influential. Hebb (see Wasserman (1989), p. 212) proposed the following deceptively simple training rule: a synapse (weight) connecting two neurons is strengthened (weight increased) whenever both neurons fire, i.e.

$$w_{ij}(t+1) = w_{ij}(t) + \text{out}_i(t) \cdot \text{out}_j(t)$$

where $w_{ij}(t)$ is the weight connecting neuron i and neuron j , at time t
 $\text{out}_i(t)$ and $\text{out}_j(t)$ are the respective outputs at time t ,
 $w_{ij}(t+1)$ is the (updated) weight at time $t+1$.

8.5.3 The Perceptron Training Rule

This is a supervised training rule: the weights are adjusted to provide a best fit representation for a (training) set of examples of pairs (x,y) , i.e. (input vector, output).

Algorithm:

Initialise: Set all weights to small random numbers.

Loop:

1. Apply an input pattern to the net;
compute the output y'
2. Compare y' with y , the target output
- 3.1 If $y' = y$ (correct) goto 1.
- 3.2 If incorrect and $y' = 0$; add each input x_i to its corresponding weight w_i ;
- 3.3 If incorrect and $y' = 1$; subtract each input x_i from its corresponding weight w_i ;

Until overall result satisfactory.

Rosenblatt proved that *if there was a solution* (complete separation) the perceptron would find it. However, suboptimal solutions? when to stop? etc.

8.5.4 Widrow-Hoff Rule

Widrow's Adaline (Widrow and Lehr, 1990) was just a continuous valued version of the perceptron (as well as binary output, the perceptron has binary inputs). The Widrow-Hoff training rule is a steepest descent algorithm – it adjusts the weights (and biases) to minimise the sum-of-squares-error (sum of $(\text{target} - \text{output})^2$).

Actually, it is only a little different from the perceptron rule: step 3 needs modification to cope with continuous values:

Algorithm:

Initialise: Set all weights to small random numbers.

Loop:

1. Apply an input pattern to the net;
compute the output y'
2. Compare y' with y , the target output
- 3.1 $\text{error}_j = y_j - y'_j$; i.e. target - output, for neuron j
- 3.2 modify weight w_{ij} according to:

$$w_{ij}(n+1) = w_{ij}(n) + a \cdot x_i \cdot \text{error}_j$$

Until overall result satisfactory.

8.5.5 Statistical Training

Actually, the Adaline (continuous valued perceptron) training problem is identical to linear regression, and so, where the data are suitable, the Moore-Penrose pseudo-inverse yields an appropriate solution. See Duda and Hart (1973).

8.5.6 Backpropogation

Backpropogation is another iterative descent rule. Back-propogation training proceeds in stages:

Initialise:

The weights are initialized to small random values in the range [-.3, .3]

Train:

Repeat until total error small enough:

Repeat for all training data:

(1) a training input vector is applied to the input layer of the network and the outputs of the hidden layers, and finally, the output layer are computed,

(2) then the weights of the output neurons (layer n) are adjusted according to gradient descent on the error between the target outputs and the actual outputs;

(3) the weights of the previous layer are adjusted according to the same criterion (NB. the adjustments at layer n-1 are still optimising the overall output -- layer n);

The theory is remarkably concise and simple, depending only on chain rule derivatives; however, the continuity of the sigmoid function, and the simplicity of its derivative are crucial. Training continues iteratively: at each iteration all the example inputs and outputs are presented and they all contribute towards the estimation of the error gradient. Lippmann (1987) gives a concise (half-page) and complete specification of the algorithm; van Camp (1993), and Winston (1992) also give good explanations.

There are two big problems with backpropagation:

- long training time (usually), and this is not easy to parallelise,
- the algorithm may get stuck in local minima; the only thing to do (provided you can determine when it has got stuck, is to reinitialise with fresh random weights and start again). It is said that XOR will only train correctly 90% of the time; I've had no problems with XOR (two-bit input, one bit out), but I've had problems with the equivalent four-bit problem. Simulated annealing provides a possible solution to this problem.
- it is not easy to interpret the weights.

8.5.7 Simulated Annealing

– see Boltzmann training (Wasserman, 1989), p. 81...

8.5.8 Genetic Algorithms

- see Luger and Stubblefield.

8.6 Other Neural Networks

[This is very short, so see any of the textbooks, but especially Wasserman (1989).]

- Hopfield: acts as content-addressable memory that can tolerate imperfect inputs,
- Kohonen: self-organising (i.e. unsupervised training); for the most part, acts as a k-means clustering algorithm,
- Neocognitron: (Fukushima, 1983) shift- and scale-invariant neural network that is supposed to more closely model the human visual system, than any other network,
- WISARD (see Boyle and Thomas (1988)); remarkably simple in concept
- based on RAM memory; a cross between straight look-up table and a perceptron; training simply consists of writing to the RAM, application, readout.
- recurrent: layer n+1 outputs fed back to layer n; some promise for prediction.

8.7 Conclusion

The objective of this chapter has been just to give you a flavour of the sorts of processing tasks that neural networks can do. We have made some simplifications, however, the major principles of neural networks are present.

Nevertheless, we have shown how neurons – or very simple networks of them can:

- recognise simple patterns,
- compute functions,
- store and recall knowledge.

And, of course, combining together these two capabilities will allow them to recognise arbitrarily complex patterns.

One thing that we have covered only sparsely is the ability of neural networks to ‘generalise’ – i.e. you can apply them to (input) data which does not appear in the training data, and get a sensible result; of course, this ‘unknown’ data must be somewhat similar to what the network has been trained on. Thus, neural networks are not just lookup tables, or rule-bases, as might have appeared from the very simple examples given.

In general there is much frothy talk about neural networks, and a certain ‘magic’ attributed to them. As in everything, neural networks are no panacea – if, for example, your example (training) data are contradictory, or you have very little training data, then neural networks will not help you, and nor will any KBS for that matter (garbage in, garbage out, still applies!).

8.7.1 Exercises

Included here are some exam questions that have been used in the past. See also section 9.3.12

1. Explain how a neural network may be used as a component of a knowledge-based system. [Hint: how are the following implemented: knowledge-base, knowledge elicitation, inference].
2. Explain some advantages and disadvantages of neural networks compared to other knowledge-based system techniques.
3. (a) What is meant by the statement, “neural networks are trained, not programmed”, and explain one major advantage, and one major disadvantage of this fact. [10 marks]
(b) What is the significance of the XOR function in the history and theory of neural networks. [6 marks]
(c) The figure below shows four two-pixel images and their associated classes (class 0 or class 1); '*' denotes bright, value 1, blank denotes dark, value 0; describe a neural network that will distinguish class 1 objects from class 0. [9 marks]

x1	x2	
*****	*****	class 1
*****	*****	
x1	x2	
*****		class 0

x1	x2	
	*****	class 0

x1	x2	
		class 0

4. (a) Describe the operation of a two-input, single layer neural network, and discuss the difficulty of implementing an XOR function using such a network. [10 marks]

(b) The figure below shows four two-pixel images and their associated classes (class 0 or class 1); '*' denotes bright, value 1, blank denotes dark, value 0; describe a neural network that will distinguish class 1 objects from class 0. [10 marks]

x1	x2	
*****	*****	class 1
*****	*****	
x1	x2	
*****		class 0

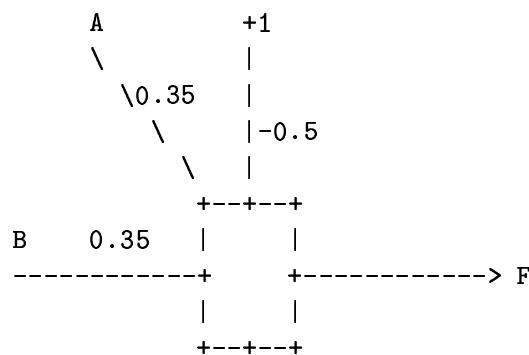
x1	x2	
	*****	class 0

x1	x2	
		class 0

x1	x2	
	*****	class 0

+-----+-----+		
x1	x2	
+	-	class 0
+-----+-----+		

5. (a) Describe the activities carried out by a single (neuron) processing unit; explain what components of the network represent its ‘memory’, and explain what is meant by the statement ‘neural networks are trained, not programmed’. [10 marks] [10 marks]
- (b) Explain how the neuron shown in the following figure computes the AND function ($F = A \text{ AND } B$); show how an alternative choice of weights can implement an ‘OR’ neuron. [4 marks]

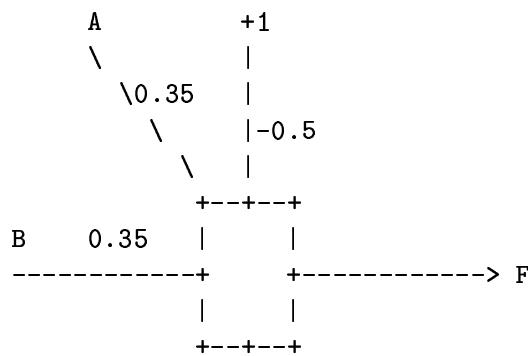


- (c) Explain how you would apply a neural network to pattern recognition. Identify a major weakness of single layer neural network and explain how a multilayer network can overcome this weakness. [6 marks]
6. (a) Explain the similarity between the activity of an artificial neuron and template matching.
- (b) Explain how a neuron can implement an AND function. Hence, explain how a neural network may implement any Boolean function.

(c) Explain the limitations of a single layer neural network for pattern recognition and show how multiple layers can remedy this problem.

7. (a) Explain what components of a neural network represent its ‘memory’, and explain what is meant by the statement ‘neural networks are trained, not programmed’.

(b) Explain how the neuron shown in the following figure computes the AND function ($F = A \text{ AND } B$); show how an alternative choice of weights can implement an ‘OR’ neuron.



(c) Explain how you would apply a single neuron to pattern recognition.

(d) Sketch the software implementation of a single neuron.

(e) Explain a weakness of a single neuron (or single layer of neurons) and show how multiple layers can remedy this problem.

8. (a) Give an intuitive explanation of neural network training.

(b) Explain ‘sigmoid activation function’.

(c) Explain how to use a multiple layer neural network for pattern recognition. What problem does the ‘soft’ sigmoid activation cause, and how is it solved?

(d) what is meant by the statement: “neural networks are model free”.

9. (a) Give feature space explanation of the similarity of pattern recognition and Boolean function computation.

(b) Discuss the difficulty of implementing XOR using a single layer of neurons.

(c) Relate the XOR problem to pattern recognition. [Answer: linear boundaries].

(d) Discuss two weakness of neural networks.

8.8 Recommended Reading

[This is included mainly for someone who would like to pursue the topic further, e.g. as part of a project/dissertation].

The current *best* book on neural networks is Haykin (1994).

Wasserman (1989) is particularly easy and complete for teach-yourself; it covers all the major architectures, and training algorithms; gets to the essence of the matter often much better than most of the original papers.

From an Artificial Intelligence / Expert Systems point of view, Luger and Stubblefield (1993), and the other popular AI textbook, Winston (1992), both give good introductions to neural networks.

Lippmann (1987) has been the traditional teach-yourself guide – but, in my opinion, is not easy going. Rumelhart and McClelland (1986) was obviously influential but, being an edited collection, seems uneven and not easy to read. There is a companion volume (Vol. 3) that contains software.

Nagy (1991) gives a good brief account of the history; Hecht-Nielsen (1991) gives his own colourful version of the story – but is not a good teach-yourself book; ditto Kosko (Fuzzy Sets and Neural Networks): very clever, crusading and inspiring – but the book would have benefited from better editing.

Duda and Hart (1973) is still the classic on pattern recognition; Agrawala (1976) gives many classic pattern recognition papers. Schalkoff (1990) is a modern pretender – and gives a modern coverage of neural nets (but only feedforward backprop.) Uhr (1966) is a collection that includes much of the influential early work on human perception and pattern recognition.

IEEE publish a bi-monthly Transactions on Neural Networks. IEEE Trans. on Systems Man and Cybernetics often contains NN applications. Other journals include: Neurocomputing, Neural Computation, Connection Science, Neural Networks, Neural Network World, and many others.

Applications are covered well in Gonzalez and Woods (1992), Winston (1992), Luger and Stubblefiled (1993); the Rosenfeld Image Processing survey papers (Rosenfeld, 1993, 1992, 1991, etc.) have sections on neural network applications to image processing.

Mathematics packages like MATLAB now have promising neural net additions. There are dedicated NN software packages: most of them seem expensive for what they offer – don't buy one without a good recommendation, trial, and/or review. A considerable amount of public domain code is available – do a Web search. The expected take-off of neural network hardware does not appear to have been significant. Neural network chips were available from HNC (Hecht-Nielsen Corp.), Siemens, and others, but are do not have any significant market share.

8.9 References and Bibliography

1. Agrawala, A.K. 1976. Machine Recognition of Patterns. IEEE Press.
2. Aleksander, I. and H. Morton. 1990. An Introduction to Neural Computing. London: Chapman and Hall.
3. Arbib, M.A., and J. Buhmann. 1990. Neural Networks Encyclopedia of AI. In Shapiro, 1990.
4. Barlow, H.B. 1953. Summation and Inhibition in the Frog's Retina. *J. Physiology*, Vol. 119, pp. 69-88.
5. Beck, J.V., and Arnold, K.J., 1977, Parameter Estimation in Engineering and Science, John Wiley.
6. Block, H.D. 1962. The Perceptron: A Model for Brain Functioning. *Reviews of Modern Physics*, Vol. 34, No. 1, January.
7. Bratko, I. 1991. PROLOG: Programming for Artificial Intelligence. Addison-Wesley.
8. Boyle, R.D. and R.C. Thomas. 1988. Computer Vision: A First Course. Blackwell Scientific.
Includes simple easy to grasp description of WISARD (more than its inventors ever managed!).
9. Brauch, J., Tam, S.M., Holler, M.A., and Shmurun, A.L., 1992, Analog VLSI Neural Networks for Impact Signal Processing, *IEEE Micro*, Vol. 12, No. 6, December.
10. Brookshear, J.G. 1991. Computer Science: An Overview. Benjamin/Cummings.
See section 10.2, p. 366, for a very accessible and simple introduction to NNs.
11. Campbell, J.G. and A.A. Hashim. 1992. Fuzzy Sets, Pattern Recognition, Linear Estimation, and Neural Networks – a Unification of the Theory with Relevance to Remote Sensing. in Cracknell, A.P. and R.A. Vaughan, eds. Proc. 18th Annual Conf. of the Remote Sensing Society. University of Dundee, September, pp. 508-517.
12. Chow, C.K. 1957. An Optimum Character Recognition System Using Decision Functions. *IRE Trans. Electron. Comput.*, Vol. EC-6, Dec. 1957.
13. Davalo, E. and P. Naim. 1991. Neural Networks, Macmillan Press.

14. Deutsch, J.A. 1955. A Theory of Shape Recognition. *British Journal of Psychology*, Vol. 46, pp. 30-37. Reprinted in (Uhr 1966).
15. Devijver, P.A., and J. Kittler. 1982. *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, NJ: Prentice-Hall.
16. Duda, R.O. and Hart, P.E., 1973, *Pattern Classification and Scene Analysis*, Wiley-Interscience.
17. Eberhart, R.C. and Dobbins, R.W., eds., 1990, *Neural Network PC Tools*, Academic Press.
18. Feller, W., 1966, *An Introduction to Probability Theory and its Applications*, Volume II, John Wiley and Sons.
19. Fisher, R.A. 1936. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*. Vol. 7. pp. 179-188. (in Agrawala, 1976).
20. Fix, E. and J.L. Hodges. 1951. Discriminatory Analysis, Nonparametric Discrimination: Consistency Properties. USAF School of Aviation Medicine, Randolph AFB, TX. Project 21-49- 004, Report No. 4, February.
21. Fix, E. and J.L. Hodges. 1952. Discriminatory Analysis, Nonparametric Discrimination: Small Sample Performance. USAF School of Aviation Medicine, Randolph AFB, TX. Project 21-49- 004, Report No. 11, August.
22. Funahashi, K-I. 1989. On the Approximate Realisation of Continuous Mappings by Neural Networks *Neural Networks*, Vol. 2, pp. 183-192, 1989.
23. Fukushima, K., S. Miyake, and T. Ito. 1983. Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition. *IEEE Trans. Systems, Man, and Cybernetics*. Vol. SMC-13, No. 5.
24. Fukunaga, K. 1992. *Introduction to Statistical Pattern Recognition* 2nd ed. Academic Press 1992.
25. Gonzalez, R.C. and R.E. Woods. 1992. *Digital Image Processing* 3rd ed. Addison-Wesley 1992.
26. Hammerstrom, D. 1993a. Neural Networks at Work *IEEE Spectrum* June 1993, pp. 26-32
27. Hammerstrom, D. 1993b. Working with Neural Networks. *IEEE Spectrum* July 1993, pp. 46-53

28. Haykin, S. 1994. Neural Networks. Macmillan.
29. Hecht-Nielsen, R. 1987. Kolmogorov's Mapping Neural Network existence Theorem Proc IEEE 1st International Conference on Neural Network, Vol. III pp. 11-14
30. Hecht-Nielsen, R., 1990, Neurocomputing, Addison-Wesley.
31. Hinton, G.E. 1992. How Neural Networks Learn from Experience Scientific American Sept. 1992.
32. Hopfield, J.J. 1982. Neural Networks and Physical Systems with emergent collective computational abilities Proc. Natl. Acad. Sci. USA Vol. 79. pp. 2554-2558 April 1982
33. Hubel, D.H. and T.N. Weisel. 1962. Receptive Fields, Binocular Interaction, and Functional Architecture in the Cat's Visual Cortex. Journal of Physiology. Vol. 160, pp. 106-123. Reprinted in (Uhr 1966).
34. IEEE. 1993. Special Issue on Neural Network Hardware. IEEE Trans. Neural Networks. Vol. 4 No. 3. May.
35. IEEE. 1992. Special Issue on Neural Network Hardware. IEEE Trans. Neural Networks. Vol. 3 No. 3. May.
36. IEEE. 1990a. Special Issue on Neural Networks I: Theory and Modelling. Proceedings of the IEEE, 78, No. 9, Sept. 1990.
37. IEEE. 1990b. Special Issue on Neural Networks I: Analysis, Techniques, and Applications. Proceedings of the IEEE, 78, No. 10, Oct. 1990.
38. IEEE. 1983. Special Issue on Neural and Sensory Information Processing. IEEE Trans. Systems, Man, and Cybernetics. Vol. SMC-13, No. 5.
39. Karnofsky, K. 1993. Neural Networks and Character Recognition Dr. Dobb's Journal, June 1993.
40. Kosko, B. 1992. Neural Networks and Fuzzy Systems, Prentice-Hall Int.
41. Kosko, B. 1991. Neural Networks for Signal Processing Prentice-Hall 1991
42. Lippmann, R.P., 1987, An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, April.

43. Luger, G.F. and W.A. Stubblefield. 1993. Artificial Intelligence 2nd ed. Benjamin/Cummings.
44. MacCarthy, R.A. 1955. Electronic Principles in Brain Design J. Irish Medical Association. Vol. 37, No. 221. November.
45. McCulloch, W.S., and W. Pitts. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics, Vol. 5, 1943.
46. Mehra, P. and B.W. Wah (eds). 1992. Artificial Neural Networks: Concepts and Theory IEEE Press 1992. (ordered for library 10/8/93).
47. Minsky, M. 1961. Steps Towards Artificial Intelligence. Proc. IRE, Vol. 49, No.1 Jan. 1961.
48. Minsky, M.L., and Papert, S.A., 1969, Perceptrons, MIT Press. Expanded/Reprinted edition, 1988, MIT Press.
49. Nagy, G. 1991. Neural Networks – Then and Now. IEEE Trans. Neural Networks. Vol. 2 No. 2.
50. Nilsson, N.J. 1965. Learning Machines: Foundations of Trainable Pattern-Classifying Systems. New York: McGraw-Hill.
51. Rao Vemuri, V. (ed.). 1992. Artificial Neural Networks: Concepts and Control Applications IEEE Press 1992. (Ordered for library 10/8/93)
52. Rosenblatt, F. 1961. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Washington D.C.: Spartan Books.
53. Rosenblatt, F. 1960. Perceptron Simulation Experiments. Proc. IRE. Vol. 48, pp. 301-309. March.
54. Rosenfeld, A. and A.C. Kak. 1982a. Digital Picture Processing. Vol. 1 Academic Press.
55. Rosenfeld, A. and A.C. Kak. 1982b. Digital Picture Processing. Vol. 2 Academic Press.
56. Rosenfeld, A. 1992. Survey, Image Analysis and Computer Vision: 1991 CVGIP:Image Understanding, Vol. 55, No. 3, May pp. 349-380.
57. Rosenfeld, A. 1993. Survey, Image Analysis and Computer Vision: 1992 CVGIP:Image Understanding, Vol. 58, No. 1, July pp. 85-135.
58. Rosenfeld, A. 1990. Survey, Image Analysis and Computer Vision: 1989 Computer Vision, Graphics, and Image Processing, Vol. 50, pp. 188-240, 1990.

59. Rosenfeld, A. 1989. Survey, *Image Analysis and Computer Vision: 1988 Computer Vision, Graphics, and Image Processing*, Vol. 46, pp. 196-264, 1989.
60. Rosenfeld, A. 1988. Survey, *Image Analysis and Computer Vision: 1987 Computer Vision, Graphics, and Image Processing*, Vol. 42, pp. 234-293, 1988.
61. Schalkoff, R. 1992. *Pattern Recognition: Statistical, Structural and Neural Approaches* Wiley 1992
62. Shapiro, S. (ed.) 1990 *Encyclopedia of Artificial Intelligence*. ?? (in reference section of Magee library).
63. Therrien, C.W. 1989. *Decision Estimation and Classification*. New York:John Wiley.
64. Uhr, L. 1966. *Pattern Recognition: Theory, Experiment, Computer Simulations, and Dynamic Models of Form Perception and Discovery*. New York: John Wiley.
65. van Camp, D. 1992. Neurons for Computers *Scientific American* Sept. 1992.
66. Wasserman, P.D. 1989. *Neural Computing – Theory and Practice*. New York: Van Nostrand Reinhold.
67. Widrow, B., and Lehr, M.A., 1990, 30 Years of Adaptive Neural Networks. *Proceedings of the IEEE*, 78, No. 9, Sept. 1990.
68. Winston, P.H. 1992. *Artificial Intelligence* 3rd ed Addison-Wesley.

8.10 Questions on Chapters 7, 8 and 9 – Segmentation, Pattern Recognition and Neural Networks

1. (a) “What should a good image segmentation be? Regions of an image segmentation should be uniform and homogeneous with respect to some characteristic (property) such as grey tone or texture. Region interiors should be simple and without many small holes. Adjacent regions of a segmentation should have significantly different values with respect to the characteristic on which they (the regions themselves) are uniform. Boundaries of each segment should be simple, not ragged, and must be spatially accurate.”, Haralick and Shapiro (1992). Discuss, paying particular attention to the criteria ‘uniform and homogeneous’.

8.10. QUESTIONS ON CHAPTERS 7, 8 AND 9 – SEGMENTATION, PATTERN RECOGNITION AND NE

- (b) Explain the role of DISTANCE as a similarity measure.
- (c) Describe one segmentation technique.
2. (a) Explain, employing appropriate illustrations of techniques and applications, the three image segmentation categories:
- single pixel classification,
 - boundary based methods,
 - region growing methods.
- (b) Briefly, explain the application of TWO image segmentation techniques to the image below.

```
1 2 3 1 3 2 3 1 2 3  
2 3 1 3 2 3 1 2 3 1  
3 1 3 2 3 8 2 3 1 2  
1 2 3 7 8 9 9 8 7 1  
2 3 1 8 9 9 8 7 7 2  
3 1 2 9 9 8 7 7 8 3  
3 1 2 9 9 8 7 7 8 3  
1 2 3 1 3 2 3 1 2 3  
2 3 1 3 2 3 1 2 3 1  
3 1 3 2 3 1 2 3 1 2
```

3. (a) Compare and contrast SUPERVISED classification and UNSUPERVISED.
- (b) Identify and compare and contrast TWO similarity measures for patterns.
- (c) Use k-means clustering to segment, into two classes / regions, the 10 x 10 image given in Figure 3-1. The image in Figure 3-1 is monochrome, explain what changes you would make to the algorithm, for a multi-colour image. Suggest an improved / alternative segmentation scheme that may, in general, give improved results.

```
1 2 3 1 3 2 3 1 2 3  
2 3 1 3 2 3 1 2 3 1  
3 1 3 2 3 8 2 3 1 2  
1 2 3 7 8 9 9 8 7 1  
2 3 1 8 9 9 8 7 7 2  
3 1 2 9 9 8 7 7 8 3  
3 1 2 9 9 8 7 7 8 3  
1 2 3 1 3 2 3 1 2 3
```

```

2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 1 2 3 1 2

```

Figure 3-1

4. (a) Explain, employing appropriate illustrations of techniques and applications, the three image segmentation categories:

- single pixel classification,
- boundary based methods,
- region growing methods.

- (b) Use a boundary based technique to segment the image given in Figure 4-1.

[Answer: Run an edge detector on Figure 4-1. Choose an appropriate 'edge' threshold. Thin the edges. Then link edge points. Find regions.]

```

1 2 3 1 3 2 3 1 2 3
2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 8 2 3 1 2
1 2 3 7 8 9 9 8 7 1
2 3 1 8 9 9 8 7 7 2
3 1 2 9 9 8 7 7 8 3
3 1 2 9 9 8 7 7 8 3
1 2 3 1 3 2 3 1 2 3
2 3 1 3 2 3 1 2 3 1
3 1 3 2 3 1 2 3 1 2

```

Figure 4-1

5. Analyse the problem of segmenting an image of a human face. (Ideally you want to be able to extract the face from its background.)

- (a) Identify the major problems with respect to the simple models of segmentation.
- (b) Will you need to segment into multiple classes? - As well as separating 'face' from background we may need to segment within the face? Mention problems.
- (c) Would colour help? How will it affect algorithms?
- (d) How will lighting affect the problem?
- (e) Suggest a layout for the subject (face), camera, and background.

8.10. QUESTIONS ON CHAPTERS 7, 8 AND 9 – SEGMENTATION, PATTERN RECOGNITION AND NE

6. (a) Use a simple example to describe pattern recognition.
(b) In two dimensional shape recognition explain the requirements for:
– shift invariance,
– amplitude invariance,
– scale invariance,
– rotation invariance,
– noise invariance.
(c) Define features / classification techniques that are invariant to any THREE.
7. (a) Using appropriate pictorial or numerical illustrations, explain the roles of FEATURE VECTOR and FEATURE EXTRACTION in pattern recognition.
[Answer: raw obs -> feat extract -> feat vect -> class; diagram of feat. space helpful,]
(b) Define two features (ie. use a two-dimensional feature vector) that will distinguish the shapes given below; plot each shape on a feature space diagram.
(c) Illustrate an appropriate classification technique.

*****	*****	*
*****	****	**
*****	****	***
*****	****	****
*****	****	****
*****	****	*****
*****	****	*****
*****	****	*****
*****	****	*****

8. (a) Explain the role of template matching in pattern recognition.
(b) Illustrate template matching on the three shape classes given below.

*****	*****	*
*****	****	**
*****	****	***
*****	****	****

*****	*****	*****
*****	*****	*****
*****	*****	*****
*****	*****	*****

9. (a) In the context of pattern recognition explain NEAREST NEIGHBOUR and NEAREST MEAN classifiers. Use numerical or pictorial (feature space) examples to explain their operation.
 (b) Identify one case (data distribution) each in which one of them would be preferred against the other.
10. (a) Give an intuitive explanation of maximum likelihood pattern recognition. [Hint: use a one-dimensional histogram].
 (b) Illustrate a case (data distribution) in which maximum likelihood and nearest mean classification will give the same results.
11. (a) In the context of pattern recognition explain NEAREST NEIGHBOUR and k-nearest classifiers. Use numerical or pictorial (feature space) examples to explain their operation.
 (b) Identify one case (data distribution) each in which k-NN would be preferred against simple NN.
12. (a) In the context of pattern recognition explain NEAREST MEAN and HYPERSPHERE classifiers. Use numerical or pictorial (feature space) examples to explain their operation.
 (b) Identify one case (data distribution) each in which one of them would be preferred against the other.
13. Describe a possible set of steps to be carried out to perform planar shape recognition in digital images. Illustrate your answer with appropriate pictures and numbers. Explain two pitfalls.
14. (a) Draw a histogram for the following data (one-dimensional features) from two classes:

```

class 0, w0:
1.21 3.11 3.97 6.21
1.32 3.12 4.12 6.58
1.40 3.21 4.30 7.00
1.56 3.31 4.70
2.07 3.37 4.86
2.21 3.45 4.92
2.22 3.50 4.97

```

8.10. QUESTIONS ON CHAPTERS 7, 8 AND 9 – SEGMENTATION, PATTERN RECOGNITION AND NE

2.73 3.78 5.10
3.00 3.90 5.70

```
class 1, w1:  
  
6.89 10.03 11.23 11.71 12.37  
8.01 10.31 11.25 11.82 13.01  
8.76 10.45 11.34 11.99 13.50  
9.25 10.56 11.37 12.22 13.57  
9.33 10.72 11.45 12.32 14.60  
9.76 10.80 11.60 12.33
```

- (b) Hence, determine a decision boundary (threshold) that classifies the points with minimum error.
- (c) Determine the means of each class. What is the effective decision boundary for the nearest mean classifier.
- (d) If you wanted to use a nearest neighbour classifier, but decided to 'condense' the points, which points are significant and must be retained, in order to give minimum error.
15. Draw a two-dimensional scatter plot showing the following data:

Class 0:

1,1
1,2
1,3
2,1
2,2
2,3
3,1
3,2
3,3
3.5,3.5

```
class 1:  
3,3  
3,4  
3,5  
4,3  
4,4
```

4,5
5,3
5,4
5,5
2.5,2.5

- (b) Work out the mean vectors of each class.
 (c) Hence, apply a nearest mean classifier to the patterns:

2,3.5
2,5

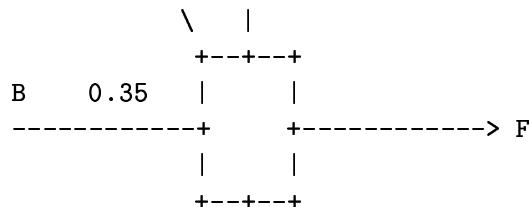
(d) Compare the error results that you would obtain by applying (i) a nearest mean classifier, and (ii) a nearest neighbour classifier to the training data (ie. the training data are the data in the tables above, and you are also using these data for testing).

(e) Illustrate a linear boundary classifier that would suit these training data.

16. Look at Section 8.10, problems 9, 10, 11, 12.
17. (a) Explain the similarity between the activity of an artificial neuron and template matching.
 (b) Explain how a neuron can implement an AND function. Hence, explain how a neural network may implement any Boolean function.
 (c) Explain the limitations of a single layer neural network for pattern recognition and show how multiple layers can remedy this problem.
18. (a) Explain what components of a neural network represent its 'memory', and explain what is meant by the statement 'neural networks are trained, not programmed'.
 (b) Explain how the neuron shown in Figure 18-1 computes the AND function ($F = A \text{ AND } B$); show how an alternative choice of weights can implement an 'OR' neuron.

$$\begin{array}{ccc}
 A & & +1 \\
 \backslash & & | \\
 & \backslash 0.35 & | \\
 & \backslash & | -0.5
 \end{array}$$

8.10. QUESTIONS ON CHAPTERS 7, 8 AND 9 – SEGMENTATION, PATTERN RECOGNITION AND NEURAL NETWORKS



- (c) Explain how you would apply a single neuron to pattern recognition.
 - (d) Sketch the software implementation of a single neuron.
 - (e) Explain a weakness of a single neuron (or single layer of neurons) and show how multiple layers can remedy this problem.
19. (a) Give an intuitive explanation of neural network training.
 (b) Explain 'sigmoid activation function'.
 (c) Explain how to use a multiple layer neural network for pattern recognition. What problem does the 'soft' sigmoid activation cause, and how is it solved?
 (d) what is meant by the statement: "neural networks are model free".
20. (a) Give feature space explanation of the similarity of pattern recognition and Boolean function computation.
 (b) Discuss the difficulty of implementing XOR using a single layer of neurons.
 (c) Relate the XOR problem to pattern recognition. [Ans: linear boundaries].
 (d) Discuss two weakness of neural networks.
21. (a) In pattern recognition, what is meant by a discriminant. Explain two common forms of discriminant.

8.11 Recommended Texts and Indicative Reading

1. D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice-Hall 1982
2. H. Bassmann and P.W. Besslich, *Ad Oculos – Digital Image Processing*, International Thompson 1995. Bassman and Besslich is down to earth and practical. It has plenty of software examples and comes with a disk.
3. B. Batchelor and P. Whelan, *Intelligent Vision Systems for Industry*, Springer, 1997
4. T.C. Bell, J.G. Cleary, I.H. Witten, *Text Compression*, Prentice-Hall, 1989
5. C.M. Bishop,
Neural Networks for Pattern Recognition, Oxford University Press, 1995
6. R.D. Boyle and R.C. Thomas, *Computer Vision: A First Course*, Blackwell Scientific 1988
7. S.M. Bozic, *Digital and Kalman Filtering*, 2nd ed., Edward Arnold, 1994
8. R.N. Bracewell, *Two-Dimensional Imaging*, Prentice Hall 1995
9. K.R. Castleman, *Digital Image Processing*, Prentice Hall, 1996
10. S. Chang, *Principles of Pictorial Information Systems*, Prentice-Hall 1989
11. R. Chellappa, *Digital Image Processing*, 2nd ed. IEEE Press 1991
12. E.O. Doebelin, *Measurement Systems*, 4th ed., McGraw-Hill
13. R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley 1973. A classic and still very relevant in spite of its age.
14. K.S. Fu, R.C. Gonzalez, and C.S.G. Lee, *Robotics – Control, Sensing, Vision, and Intelligence*, McGraw-Hill 1987
15. K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed., Academic Press 1992
16. R.C. Gonzalez and P. Wintz, *Digital Image Processing*, 2nd Ed., Addison-Wesley 1987

17. R.C. Gonzalez and R.E. Woods, *Digital Image Processing*, Addison-Wesley 1992 (effectively 3rd ed. of Gonzalez and Wintz). Gonzalez and Woods is probably the closest treatment to these notes.
18. U. Grenander, *Elements of Pattern Theory*, The Johns Hopkins University Press, 1996
19. R.W. Hamming, *Digital Filters*, 3rd ed., Prentice-Hall 1989
20. R.M. Haralick and L.G. Shapiro, *Computer and Robot Vision, Volume 1*, Addison-Wesley 1992
21. R.M. Haralick and L.G. Shapiro, *Computer and Robot Vision, Volume 2*, Addison-Wesley 1993
22. D. Harel, *Algorithmics – the Spirit of Computing*, 2nd ed., Addison-Wesley 1992
23. R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley 1992
24. C.W. Helstrom, *Elements of Signal Detection and Estimation*, Prentice Hall 1995
25. J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, 1991
26. A.K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall 1989
27. R. Jain, R. Kasturi and B. Schunck, *Machine Vision*, McGraw-Hill, 1995
28. R. Kasturi and R. Jain, *Computer Vision: Principles*, IEEE 1991. A collection of some key papers.
29. R. Kasturi and R. Jain, *Computer Vision: Advances and Applications*, IEEE 1991. Another collection of papers.
30. A.D. Kulkarni, *Artificial Neural Networks for Image Understanding*, Van Nostrand Reinhold, 1994
31. J. Lesurf, “Inside Science – Information”, *New Scientist*, 7 November 1992. A simple introduction to information theory – important for data compression.
32. T.M. Lillesand and R.W. Kiefer, *Remote Sensing and Image Interpretation*, 2nd. ed., John Wiley 1987
33. J.S. Lim, *Two-Dimensional Signal and Image Processing*, Prentice-Hall 1990

34. A. Low, *Introductory Computer Vision and Image Processing* McGraw-Hill 1991
35. G.F. Luger and W.A. Stubblefield, *Artificial Intelligence*, 2nd ed., Benjamin/Cummings 1993
36. P.A. Lynn and W. Fuerst, *Introductory Digital Signal Processing with Computer Applications*, John Wiley and Sons 1989
37. David J. Maguire et al. (eds.), *Geographical Information Systems, Vol. 1: Principles*, Longman 1991
38. David J. Maguire et al. (eds.), *Geographical Information Systems, Vol. 2: Applications*, Longman 1991
39. T. Masters, *Practical Neural Network Recipes in C++*, London: Academic Press, 1993
40. T. Masters, *Signal and Image Processing with Neural Networks: A C++ Sourcebook*, John Wiley, 1994
41. T. Masters, *Advanced Algorithms for Neural Networks: a C++ sourcebook*, John Wiley 1995
42. T. Masters, *Neural, Novel and Hybrid Algorithms for Time Series Prediction*, John Wiley 1995
43. J.M. Mendel, *Lessons in estimation Theory for Signal Processing Communications and Control*, Prentice-Hall 1995
44. M. Nelson, *The Data Compression Book*, MIS Press, 2nd edn., 1996
45. W. Niblack, *An Introduction to Digital Image Processing*, 2nd Ed. Prentice-Hall 1986. This is by far the easiest introduction to the material which we cover.
46. A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall 1975
47. A.V. Oppenheim and R.W. Schafer, *Discrete Time Signal Processing*, Prentice-Hall 1989
48. J.R. Parker, *Practical Computer Vision Using C*, Wiley 1994. May be consulted for software.
49. J.R. Parker, *Algorithms for Image Processing and Computer Vision*, John Wiley, 1997
50. I. Pitas, *Digital Image Processing Algorithms*, Prentice Hall 1993 This is very complete but difficult (mathematical) in places.

51. W.K. Pratt, *Digital Image Processing*, 2nd ed., Wiley 1991
52. W.K. Pratt, *PIKS Foundation – C Programmer's Guide*, Prentice-Hall 1995
53. B.D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996
54. A. Rosenfeld and A.C. Kak, *Digital Picture Processing – Vol. 1*, Academic Press 1982.
55. A. Rosenfeld and A.C. Kak, *Digital Picture Processing – Vol. 2*, Academic Press 1982. Rosenfeld and Kak, in two volumes, is still the best overall treatment of image processing – the most complete and correct.
56. R.J. Schalkoff, *Digital Image Processing and Computer Vision*, Wiley 1989
57. R. Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches*, Wiley 1992
58. M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis and Machine Vision*, Chapman and Hall, 1993
59. J.L. Starck, F. Murtagh, and A. Bijaoui, *Image and Data Analysis: the Multiscale Approach*, Cambridge University Press, 1998
60. J.E. Szymanski, *Basic Mathematics for Electronic Engineers*, Van Nostrand Reinhold 1989
61. J. Teuber, *Digital Image Processing*, Prentice-Hall 1993
62. C.W. Therrien, *Decision Estimation and Classification*, John Wiley 1989
63. F. van der Heijden, *Image Based Measurement Systems*, Wiley 1994. Contains some good practical material.
64. D. Vernon, *Machine Vision*, Prentice-Hall 1991
65. P.D. Wasserman, *Neural Computing: Theory and Practice*, Van Nostrand 1989
66. P.D. Wasserman, *Advanced Methods in Neural Computing*, Van Nostrand 1993
67. B. Widrow and S Stearns, *Adaptive Signal Processing*, Prentice-Hall 1985
68. P.H. Winston, *Artificial Intelligence*, 3rd ed., Addison-Wesley 1992

Appendix A

Appendix: Essential Mathematics

A.1 Introduction

This Appendix firstly gathers together some basic definitions, symbols and terminology to do with random variables, random processes, and random fields; the topics are chosen according to their applicability to pattern recognition, signal processing, image processing and data compression. We resent some fundamental theorems and definitions related to estimation, prediction, and general analysis of data that are generated by random processes.

Secondly, basic definitions of linear algebra are covered. This mathematical language, and tool-set, is invaluable for topics such as pattern recognition and neural networks, image segmentation, and many other areas besides.

A.2 Random Variables, Random Signals and Random Fields

We start by presenting relevant definitions and theorems, and progress to identifying the types of (theoretical) processes that will be appropriate models for our data. We identify properties of these processes that are relevant to pattern recognition, signal processing, image processing and data compression.

A.2.1 Basic Probability and Random Variables

Events and Probability

See Rosenfeld and Kak (1982), Mortensen (1987).

Let there be a set of **outcomes** to an experiment:

$$\{\omega_1, \omega_2, \omega_3, \dots\} = \Omega$$

For each ω_i there is a probability p_i :

$$p_i \geq 0 \quad (\text{A.1})$$

$$\sum p_i = 1$$

The simple definition of probability over outcomes is satisfactory for simple applications, but for many applications we need to extend it to apply to subsets of Ω , called **events**.

Let there be subsets of Ω called events: a general event is a , the set of all a is A . We define a probability measure P on A ; P is a number. P satisfies the following axioms:

$$P(a) \geq 0 \quad (\text{A.2})$$

$$P(\Omega) = 1 \quad (\text{certain event})$$

$$a_i \cap a_j = \emptyset \forall i, j$$

a_1, a_2, \dots are **disjoint** members of A and \emptyset is the **empty set**

$$P(\cup_{k=1}^{\infty} a_k) = \sum_{k=1}^{\infty} P(a_k)$$

Put simply, if a_i and a_j are disjoint,

$$P(a_i \cup a_j) = P(a_i) + P(a_j)$$

A fourth axiom, which is really a corollary of these is sometimes included:

$$P(\emptyset) = 0 \quad (\text{the impossible event})$$

Some Comments on Events and the Probability Measure

This subsection discusses some limitations on events and probabilities which are theoretical and, in practice of little restriction. This subsection primarily introduces some further terminology related to the previous subsection, and may be skipped by the reader who prefers to continue with more central themes.

Some papers and texts, though of an applied nature, feel obliged to use the terminology of rigorous probability theory; the purpose of this note is to dispel some of the mystique of that terminology.

As in earlier and subsequent sections, Ω is the set of (elementary) outcomes; and A is the set of subsets of Ω to which probabilities can be allocated.

Theoretically, it is not possible to assign probabilities to **all** subsets of Ω , but only to a class of these subsets that form a type of **field** called a **Borel field**. This is of no practical consequence, but is an analytical necessity arising (*inter alia*) from the impossibility of assigning probabilities in the case where there are infinitely many subsets of Ω , and still have these probabilities satisfy the axioms of probability (eqns. 2.1-2a, b and c). In general, the powerset of Ω is infinite, and the Borel field A which defines the restricted collection of subsets which can be allocated a probability is not.

The term ‘probability **measure**’ was used deliberately; roughly speaking, measure is a method of associating a **number** with a subset. Measure is a general form of integration: for example, in many practical applications, the definition of probability involves integration over a domain, e.g. integral between x_1 and x_2 on the real line. The measurability of a function depends on its values, and its domain.

Example: A simple function, $f(x)$ that is 1 between 0 and 1, and 0 elsewhere: $f(x) = 1 : 0 \leq x \leq 1, f(x) = 0 : \text{otherwise}$. Obviously,

$$\int_0^1 f(x)dx = 1$$

i.e. the area is 1.

What then if the function drops to 0 at 0.5, but only at 0.5? Clearly the integral (area) is still 1, and so on for many such zero values; i.e. the integral is still defined, even though the function is not ‘well behaved’ according to our ordinary understanding. However, at some stage, when the number of zeros become infinite, the area must decrease: at this stage (still roughly speaking) the function becomes unmeasurable.

Incidentally, Borel measurability is a general criterion applied to functions; it is claimed that the class of multilayer neural networks with three layers, feedforward, and using a sigmoid activation function can be trained to represent any Borel measurable function (see Hecht-Nielsen 1990, p. 122 for a discussion).

Random Variable

If, to every outcome, ω , of an experiment, we assign a number, $X(\omega)$, X is called a **random variable** (r.v.). X is a function over the set $\Omega = \{\omega_1, \omega_2, \dots\}$ of outcomes; if the range of X is the real numbers or some

subset of them, X is a **continuous** r.v.; if the range of X is some integer set, then X is a **discrete** r.v.

Distribution Function

Also called cumulative distribution function.

1. Of a continuous r.v.

$$FX(x) = P\{-\infty < X \leq x\}$$

Note: $\{\dots\}$ is an event, so that, although X is a function over outcomes (Ω) , FX is a function over events.

2. Of a discrete r.v.

If X can assume only a finite number of possible values x_1, x_2, \dots, x_n , the probability of the matter can be adequately described by defining a corresponding list of probabilities, p_1, p_2, \dots, p_n . In this case FX is shaped like a staircase. And, there is little need for the probability density function (of a discrete r.v.) described below.

Probability Density Function

If FX (continuous) is differentiable,

$$fX(x) = d/dx FX(x)$$

is called the **probability density function** of the r.v. X . Note that the values of fX are **not** probabilities; the values of FX are. fX must be integrated to get useful probability values, e.g.

$$P\{x_1 \leq X \leq x_2\} = \int_{x_1}^{x_2} fX(x) dx$$

Expected Value of a Random Variable

1. Continuous.

$$mX = E\{X\} = \int_{-\infty}^{+\infty} xfX(x) dx$$

2. Discrete.

$$mX = EX = \sum_i x_i p_i$$

General interpretation of expected value: average over range of x , weighted by probability of individual values.

In practice, it is usual that neither $fX(x)$ nor p are known, and **estimates** must be used: e.g. for the mean of a discrete random variable:

$$mX = \sum_{j=1}^N x_j/N$$

where the x_j are representative examples of the r.v., and where N is large enough that the sample, $\{x_1, \dots, x_N\}$ is large enough that the frequency of occurrence of x values properly represents the probabilities.

Random Vector

If the value assigned to an outcome, ω , is vector valued,

$$X = [X_1, \dots, X_n]$$

then X is called a **random vectors**, i.e. $X \in R^n$, rather than $X \in R$, as in subsection 2 above.

Note: there is temptation to think of x as a sequence of random variables, i.e. each x_i generated by a separate outcome, ω . Strictly, this is incorrect: in a random vector each $x_i, i = 1 \dots n$, is generated by the same outcome. However, I am unaware of the consequences of making such a mistake. And, on reflection, I suppose it is possible to ‘manufacture’ a composite experiment whose outcome is a set containing a number of ω s.

A.2.2 Random Processes

Definition

A remark first: the adjective ‘stochastic’ is entirely equivalent to ‘random’; thus the frequently encountered term ‘stochastic process’ is equivalent to ‘random process’.

Recalling subsection 2.2, a r.v. is a rule for assigning a number, X , to each outcome, ω . Correspondingly, a **random process** is a rule for assigning, to each outcome, a **function**, $X(s, \omega)$. s is called the parameter, or the index, of the random process. Sometimes $X(s, \omega)$ is written $X_s(\omega)$.

S is the set of admissible parameter values, s . In general, S is a subset of the real line, R , or of R^n . Commonly, S is the time axis, and X is a function of time: $X(t, \omega)$, or $X_t(\omega)$; this is why t replaces s in most of the literature.

If S is one-dimensional, and discrete (i.e. the parameters can take only a finite set of values), X is just a random vector, which is equivalent to a discrete parameter random process. Note: a discrete **state** random process refers to one in which the values of X are discrete.

Interpretations of a random process (one-dimensional) (see Papoulis, 1991, p. 285):

1. A family (an **ensemble** in the literature) of functions, $X(s, \omega)$.
2. A single function, $X(s)$, i.e. ω is fixed.
3. If s is fixed and ω is variable, then $X(s)$ is a random variable equal to the **state** of the process at ‘time’ s .
4. If s and ω are fixed, $X(s)$ is a (plain) number.

Although we have used s in this section, we will bow to the preponderance of usage, and use t (for time) in later sections.

Random Fields

If S is two-dimensional or of greater dimensionality then X is called a **random field**. Thus, if $S = \{x, y : x_0 \leq x \leq x_1, y_0 \leq y \leq y_1\}$, i.e. the domain commonly associated with a (continuous) image function, we have a random field (as in e.g., Rosenfeld and Kak, p.38). Likewise for a discrete image, $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$.

First Order Statistics of Random Processes

Since $x(t)$ is a random variable for a given t , we can extend the definitions of section 2.3, 2.4 and 2.5 by including an extra parameter, s (or t).

1. Distribution function.

For a given t ,

$$FX(x, t) = P - \infty < X(t) \leq x$$

2. Expected Value (Mean).

Continuous case.

$$mX(t) = E\{X(t)\} = \int_{-\infty}^{+\infty} x(t) fX(x, t) dx$$

Likewise, discrete, see eqn 2.5-2.

3. Probability density function, pdf,

$$fX(x, t) = \frac{\partial}{\partial x} FX(x, t)$$

(This is the partial derivative of $FX(x, t)$ with respect to x .)

Second Order Statistics of Random Processes

1. Distribution function.

The second order distribution of the random process $x(t)$ is the joint distribution

$$FXX(x_1, x_2; t_1, t_2) = P\{x(t_1) \leq x_1, x(t_2) \leq x_2\}$$

2. Probability density function, pdf,

$$fXX(x, t) = \frac{\partial^2}{\partial x_1 \partial x_2} FXX(x_1, x_2; t_1, t_2)$$

And, of course, to fully describe the probability distribution of a random process it is necessary to extend to n th-order:

$$FXt_1, \dots, Xt_n(x_1, \dots, x_n; t_1, \dots, t_n) = P\{x(t_1) \leq x_1, \dots, x(t_n) \leq x_n\}$$

FXt_1, \dots, Xt_n in the foregoing equation completely determines the statistical properties of $x(t)$. The joint density function fXt_1, \dots, Xt_n is defined by analogy with the above definition of pdf.

Autocorrelation

The autocorrelation of a random process $x(t)$ is the expected value of the product of the two random variables $x(t_1), x(t_2)$:

Continuous.

$$RXX(t_1, t_2) = E\{x(t_1)x(t_2)\}$$

$$= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x_1 x_2 fXX(x_1, x_2; t_1, t_2) dx_1 dx_2$$

Discrete.

$$RXX(t_i, t_j) = \sum_{j=1}^n \sum_{i=1}^n x_i x_j p_{ij}$$

From its definition,

$$RXX(t_1, t_2) = RXX(t_2, t_1)$$

i.e. RXX is symmetric.

Autocovariance.

$$\begin{aligned} CXX(t_1, t_2) &= E\{(x(t_1) - mX(t_1)).(x(t_2) - mX(t_2))\} \\ &= RXX(t_1, t_2) - mX(t_1).mX(t_2) \end{aligned}$$

Stationary Process

1. Strict-sense stationary (see Papoulis, 1991, p. 297).

Let t_1, \dots, t_n be a set of points in T , likewise $t_1 + t_0, \dots, t_n + t_0$; the corresponding r.v.s are characterised by n th-order joint pdfs:

$$f_{Xt_1, \dots, Xt_n}(x(t_1), \dots, x(t_n); t_1, t_2, \dots, t_n)$$

, and

$$f_{Xt_1+t_0, \dots, Xt_n+t_0}(x(t_1+t_0), \dots, x(t_n+t_0); t_1+t_0, t_2+t_0, \dots, t_n+t_0)$$

When these two functions are equal $\forall t_0$, then the process is **strict-sense stationary**, i.e. all statistical properties are invariant to shifts in the origin; there is an analogous definition for random fields (e.g. images).

2. Wide-sense stationary.

Also called ‘second-order stationary’.

If the following two conditions are met, the process is called wide-sense stationary:

$$mX(t) = mX$$

i.e. m is constant for all t , and,

$$RXX(t_1, t_2) = RXX(t_1 - t_2)$$

i.e. the autocorrelation depends only on $(t_1 - t_2)$ (the displacement).

In the discrete case, wide-sense stationarity has the following important consequence, when RXX is expressed as a matrix:

$$\begin{array}{ccccccccc} RXX = & r_{00} & r_{01} & r_{02} & & \dots & & r_0 & n-1 \\ & r_{10} & r_{11} & & & & & r_1 & n-1 \\ & & & & & & & & \\ & r_{n-1} & 0 & r_{n-1} & 1 & & \dots & & r_{n-1} & n-1 \end{array}$$

Applying eqn. 3.6-2

$$\begin{array}{ccccccccc} RXX = & r_0 & r_1 & r_2 & & \dots & & r_{n-1} \\ & r_1 & r_0 & r_1 & & & & r_{n-2} \\ & & & & & & & \\ & r_{n-1} & r_{n-2} & & & & r_1 & r_0 \end{array}$$

i.e. RXX is a circulant matrix – each row is merely the previous row rotated by one position. Sets of linear simultaneous equations that are characterised by a circulant matrix (or more generally a Toeplitz matrix) can be solved using ‘fast’ and recursive algorithms – such solution is required in prediction.

Another property of Toeplitz matrices is that they are diagonalised by the Discrete Fourier Transform – an important consequence of this fact is that the DFT is equivalent to the Karhunen-Loëve Transform for such data (the KL transform is an important transform in lossy data compression).

Incidentally, discrete convolution can be expressed as multiplication by a circulant matrix – the delayed impulse response weights form the rows; therefore the matrix may be diagonalised by the DFT, and this is why convolution decomposes into multiplication in the Discrete Fourier domain.

To read further, see Jain (1989), Pratt (1991), and Press et al. (1992).

Ergodic Processes

The statistics defined in the previous sections on random processes are defined by taking the expectation, $E\{\cdot\}$, over the **ensemble** of x s; now, as hinted in eqn. 2.5-3, we rarely have access to a sufficiently large ensemble of x s that we can estimate pdfs; indeed, practically, we rarely have more than **one sample**, x . If the process is **ergodic**, i.e. roughly speaking, we can replace expectations over the ensemble with **time averages** over one sample, we have a practical method by which to estimate statistics.

Of course, a process first has to be stationary, so that these time averages do not vary with time.

Thus, the mean of a (discrete) ergodic process:

$$mX_i = E\{x_i\}$$

can be estimated

$$m\hat{X} = 1/N \sum_{i=1}^N x_i$$

Another example is the estimate of autocorrelation:

$$RX\hat{X}(k) = 1/N \sum_{i=1}^N x(i+k)x(i)$$

Both eqns. 3.7-1 and 3.7-2 are normally used **without** qualification.

As with stationarity (section 3.6), ergodicity can be wide-sense (e.g. ergodic in the mean (first-order), ergodic in autocorrelation (second-order) etc., or strict-sense, in which the ergodicity is defined in terms of the probability functions.

Markov Process

A Markov process is a random process in which the probability of achieving any future state depends only on the present, and not on the past. As with stationarity and ergodicity (see section 3.7), Markov processes can be defined as strict sense or wide sense.

The definition of a strict-sense Markov process can be given more formally, in terms of conditional pdfs as follows:

$$\begin{aligned} fXt_{m+1}, Xt_{m+2} \dots, Xt_n | Xt_1, \dots, Xt_m(x_{m+1}, xt_{m+2} \dots, x_n | x_1, \dots, x_m) \\ = fXt_m, \dots, Xt_n | Xt_m(x_{m+1}, \dots, x_n | x_m) \end{aligned}$$

where t_m = present time, and $|$ denotes ‘conditional’ (probability).

In a strict-sense Markov process, the probabilities of states at $t_{m+1} \dots$ depend only on the state at t_m (present) and not on t_{m-1}, t_{m-2}, \dots and backwards.

Clearly this has strong implications for the usage of ‘past’ states in predicting the future.

In an image context (see Rosenfeld and Kak p. 312), we have Markov random fields; in that context Markov means that the probability of the state (greylevel) of a pixel depends only on the states of its eight neighbours.

Gaussian Process

The random process, X_t , is called a Gaussian process, provided that for any finite collection of ‘times’ (t_1, t_2, \dots, t_n) the vector of states $x = (xt_1, xt_2, \dots, xt_n)$ has the joint pdf:

$$fx(x) = (\frac{1}{2}\pi n/2)(| Cxx | \frac{1}{2}) \exp\{-\frac{1}{2}(x - m)TCxx^{-1}(x - m)\}$$

where C_{xx} is the autocovariance (see eqn 3.5-4), $m = E\{x_t\}$ and $| \cdot |$ denotes determinant.

Gaussian processes are good models for many natural random processes. In addition, they are analytically convenient, And, see eqn. 3.9-1, their pdf is totally determined by first and second order statistics (mean and autocovariance, respectively).

A.2.3 Further Background Reading

1. Chung, K.L. 1968. A Course in Probability Theory. New York: Harcourt, Brace and World.
2. Feller, W. 1966. An Introduction to Probability Theory and its Applications. Vol II. New York: John Wiley and Sons.
3. Hecht-Nielsen R. 1990. Neurocomputing. Reading, Mass: Addison-Wesley.
4. A.K. Jain. 1989. Fundamentals of Digital Image Processing. Englewood Cliffs, NJ: Prentice-Hall Int.
5. Kosko, B. 1992. Neural Networks and Fuzzy Systems. Englewood Cliffs, NJ: Prentice-Hall Int., 1992
6. Mortensen R.E. 1987. Random Signals and Systems. New York: John Wiley and Sons.
7. Papoulis A. 1991. Probability, Random Variables and Stochastic Processes. 3rd ed. New York: McGraw-Hill.
8. W.K. Pratt. 1991. Digital Image Processing. New York: Wiley-Interscience.
9. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. 1992. Numerical Recipes in C. Cambridge, U.K. : Cambridge University Press.
10. Proakis J.G. 1989. Digital Communications. 2nd ed. New York: McGraw-Hill.
11. Rosenfeld A. and A.C. Kak. 1982. Digital Picture Processing. 2nd ed. London: Academic Press. (2 Volumes).
12. Thomasian, A.J. 1969. The Structure of Probability Theory with Applications. New York: McGraw-Hill.
13. Widrow, B., and Lehr, M.A. 1990. 30 Years of Adaptive Neural Networks. Proceedings of the IEEE. 78, No. 9.

A.3 Linear Algebra

A.3.1 Basic Definitions

Vectors and Matrices

Pattern or measurement vector before any processing:

$$x = (x_0, x_1, \dots, x_i, \dots, x_{p-1})^T$$

a $p \times 1$ column vector.

After transformation:

$$y = (y_0, y_1, \dots, y_i, \dots, y_{p-1})^T$$

a $q \times 1$ column vector.

A matrix transformation is defined by an equation of the form:

$$y = Ax$$

with respective dimensionalities: $q \times 1$, $q \times p$, and $p \times 1$.

Multivariate Statistics

$$x = (x_0, x_1, \dots, x_i, \dots, x_{p-1})^T$$

a $p \times 1$ feature vector.

Mean vector:

$$m = (m_0, m_1, \dots, m_i, \dots, m_{p-1})^T$$

$$= E\{x\}$$

where $E\{\cdot\}$ denotes expectation.

Normally we estimate expected values using the sample average, e.g. mean m is estimated using the average of x computed over a sample of n values, generically called x_i :

$$\hat{m} = \frac{1}{n} \sum_{i=1}^n x_i$$

Autocorrelation matrix:

$$\begin{aligned} R &= E\{xx^T\} \\ &= [r_{ij}] \end{aligned}$$

where

$$r_{ij} = E\{x_i x_j\}$$

the expected value of the product of the i th and j th components of x .

Normally we will use the sample autocorrelation matrix, i.e. \hat{R} estimated from a sample of n vectors, $x_i, i = 1 \dots n$,

$$\hat{R} = \frac{1}{n} \sum_{i=1}^n x_i x_i$$

It is sometimes convenient to write (2.2-5) completely in matrix notation; let X be the $p \times n$ matrix formed by arranging the n x_i values as columns

$$X = [x_1 \ x_2 \ \dots \ x_i \ \dots \ x_n]$$

a $p \times n$ matrix, now eqn. 2.2-5 can be rewritten as

$$\hat{R} = \frac{1}{n} X X^T$$

Covariance matrix:

$$S = E\{(x - m)(x - m)^T\}$$

$$= [s_{ij}]$$

where

$$s_{ij} = E\{(x_i - m_i)(x_j - m_j)\}$$

the expected value of the product of the i th and j th components of the deviation of x from its mean.

The diagonal elements s_{ii} of S are the variances of the element x_i .

It is easy to verify that:

$$S = R - mm^T$$

Also, there is a matrix representation, analogous to eqn. (2.2-7), see the discussion of autocorrelation above.

If $x' = (x - m)$, i.e. the pattern vector reduced to zero mean, and

$$X' = [x'_1 \ x'_2 \ \dots \ x'_i \ \dots \ x'_n]$$

of dimensions $p \times n$, then the sample covariance can be rewritten as

$$\hat{S} = \frac{1}{n} X' X'^T$$

Multivariate Gaussian Random Vectors:

Multivariate vectors generated by a multivariate random process follow the probability density function (pdf):

$$f_x(x) = \left(\frac{1}{2}\pi\frac{n}{2}\right)\left(\mid S \mid \frac{1}{2}\right) \exp\left\{-\frac{1}{2}(x - m)^T S^{-1}(x - m)\right\}$$

where S is the covariance matrix (see (2.2-8), $m = E\{x\}$ (see eqn. 2.2-1), and $\mid . \mid$ denotes determinant.

Gaussian processes are good models for many natural random processes. In addition, they are analytically convenient, since their pdf is totally determined by first and second order statistics (mean and covariance, respectively).

Statistics after Transformation:

If we transform into a feature space using eqn. (2.1-1), $y = Ax$, the mean, autocorrelation, and covariance statistics are transformed as follows:

Mean: $m' = Am$

Autocorrelation: $R' = ARA^T$

Covariance: $S' = ASA^T$

Prior Probabilities:

Many pattern recognition algorithms use (estimates of) the relative frequency of occurrence of the various classes in the population; these are called **prior**, or **a priori**, probabilities, because these probabilities are known before any measurement is made.

Prior probabilities are denoted P_1, P_2 etc. for $P(\text{class 1})$ etc.

Contrast **posterior**, or **a posteriori** probabilities, which are the probabilities of the classes after the measurements have been taken into account.

A.3.2 Linear Simultaneous Equations

Eqn. 3.1 is a system of linear (simultaneous) equations.

$$\begin{aligned} y_1 &= 3x_1 + 1x_2 \\ y_2 &= 2x_1 + 4x_2 \end{aligned} \tag{A.3}$$

Ex. 3.2.1-1 Practically, Eqn. 3.1 could express the following:

Price of an apple = x_1 , price of an orange = x_2 (both unknown). Person A buys 3 oranges, and 1 apple and the total bill is 5p (y_1). Person B buys 2 oranges, 4 apples and the total bill is 10p (y_2).

Question: What is x_1 , the price of apples, and x_2 , the price of oranges?

$$(1) \quad 5 = 1x_1 + 3x_2$$

$$(2) \quad 10 = 4x_1 + 2x_2$$

$$(1) \implies x_2 = -5 + 3x_1$$

Substitute into (2):

$$\begin{aligned} 10 &= 4x_1 + 2(-5 + 3x_1) \\ 10 &= 10x_1 - 10 \\ 20 &= 10x_1 \\ 2 &= x_1 \end{aligned}$$

Now, substitute $x_1 = 2$ into (1):

$$\begin{aligned} 5 &= 2 + 3x_2 \\ 3 &= 3x_2 \\ x_2 &= 1 \end{aligned}$$

The simultaneous equations 3.1 can be written in matrix form as follows:

$$y = Ax$$

where y is a two-dimensional vector, $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ x is a two-dimensional vector, $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ and A is a 2 row \times 2 column matrix, $A = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$

Note: a *vector* is simply an array of numbers. A vector with n numbers is called an n -dimensional vector; such a vector represents a point in n -dimensional space. *Don't* try to visualise $n > 3$! Just think of the n numbers grouped together.

In two- or three-dimensions it is possible to visualise a vector as a line with an arrow-head – the arrow indicates the path between the origin $(0, 0)$ and the point (x, y) that the vector represents; again, for our purposes this view has limited use.

Generally, a system of m equations, in n variables,

$$\begin{aligned} y_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ y_2 &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ &\vdots \\ y_r &= a_{r1}x_1 + a_{r2}x_2 + \cdots + a_{rc}x_c + \cdots + a_{rn}x_n \\ &\vdots \\ y_m &= a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{aligned}$$

can be written in matrix form as,

$$y = Ax \tag{A.4}$$

where y is an m -dimensional vector

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

x is an n -dimensional vector,

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

and A is an m -row \times n -column matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \ddots & \ddots & \ddots & \ddots \\ \ddots & a_{rc} & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

That is, the matrix A is a rectangular array of numbers whose element in row r , column c is a_{rc} . The matrix A is said to be $m \times n$, i.e. m rows, n columns.

Vectors can be considered as specialisations of matrices, i.e. matrices with only one column. Thus x is $m \times 1$, and y is $n \times 1$.

Eqns. 3.1 or 3.2 can be interpreted as the definition of a function which takes n arguments $(x_1, x_2 \dots x_n)$ and returns m variables $(y_1, y_2 \dots y_m)$. Such a function is also called a *transformation*: it transforms n -tuples of real numbers to m -tuples of real numbers.

Eqn. 3.2 is a *linear* transformation because there are no terms in x_r^2 or higher, only in x_r .

A.3.3 Basic Matrix Operations

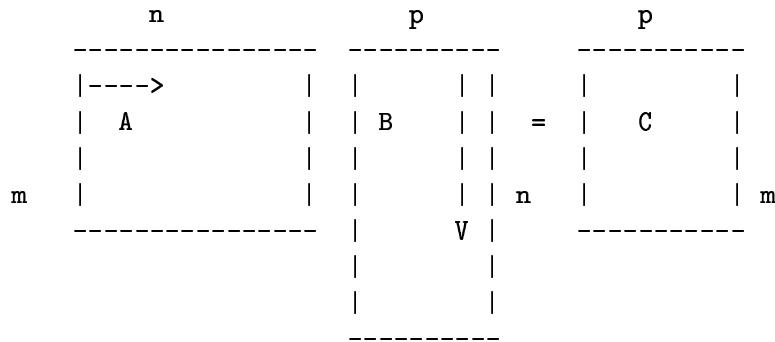
Matrix Multiplication

We may multiply two matrices $A, m \times n$, and $B, q \times p$, as long as $n = q$. Such a multiplication produces an $m \times p$ result. Thus,

$$\begin{matrix} C \\ m \times p \end{matrix} = \begin{matrix} A \\ m \times n \end{matrix} \begin{matrix} B \\ n \times p \end{matrix} \quad (\text{A.5})$$

Method: The element at the r th row and c th column of C is the product (*dot* or *inner* or *scalar* product) of the r th row vector of A with the c th column vector of B .

Pictorially:



Thus,

$$C = AB$$

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$C = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Ex. 3.2.2-1 Consider Eqn. 3.2, $y = Ax$. Thus the product of $A(m \times n)$ and $x(n \times 1)$

$$y_1 = a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n$$

(row1 col1 = product of 1st row of A with 1st column of x) etc.

The product is $(m \times n) \times (n \times 1)$ so the result is $(m \times 1)$, i.e. y .

Ex. 3.2.2-2 Apply the transformation given by

$$\begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix}$$

to

$$x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Scaling and Rotation

Consider scaling and rotation in computer graphics. Here the vectors are:

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

and the output, transformed, vector is:

$$\begin{pmatrix} x' \\ y' \end{pmatrix}$$

The scaling transformation takes the form:

$$\begin{aligned} x' &= xS_x \\ y' &= yS_y \end{aligned}$$

That is, x is expanded ($S_x > 1$) or contracted ($S_x < 1$) to give x' ; ditto y -axis. S_x and S_y are called scaling factors. The matrix is

$$\begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$

The following transformation rotates (x, y) a clockwise angle B about the origin $(0,0)$:

$$\begin{aligned} x' &= x \cos B + y \sin B \\ y' &= -x \sin B + y \cos B \end{aligned}$$

The matrix is

$$R(B) = \begin{pmatrix} \cos B & \sin B \\ -\sin B & \cos B \end{pmatrix} \quad (\text{A.6})$$

Ex. 3.2.3-1 (a) Rotate the point

$$\begin{pmatrix} 0.707 \\ 0.707 \end{pmatrix}$$

clockwise by 45 degrees.

(Note: $\sin 45 = 0.707 = 1/\sqrt{2} = \cos 45$; $0.707 \times 0.707 = 0.5$.)

(b) Rotate the point

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

clockwise by 90 degrees.

Ex. 3.2.3-2 What is the effect of applying the rotation matrix twice? That is,
what is

$$R(B)R(B) \begin{pmatrix} x \\ y \end{pmatrix}$$

The following formulae may be useful:

$$\begin{aligned}\sin(a+b) &= \sin(a)\cos(b) + \cos(a)\sin(b) \\ \cos(a+b) &= \cos(a)\cos(b) - \sin(a)\sin(b)\end{aligned}$$

Ex. 3.2.3-3 What is the effect of applying the negative rotation, $-B$, to a point that has already been rotated by $+B$. That is, what is

$$R(-B)R(B) \begin{pmatrix} x \\ y \end{pmatrix}$$

Multiplication by a Scalar

$$c \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} ca_{11} & ca_{12} \\ ca_{21} & ca_{22} \end{pmatrix}$$

Addition of Matrices

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

The matrices must be the same size (dimensions).

Inverses of Matrices

Only for square matrices ($m = n$).

Consider Eqn. 3.1:

$$\begin{aligned}y_1 &= 3x_1 + 1x_2 \\ y_2 &= 2x_1 + 4x_2\end{aligned}$$

i.e. $y = Ax$ where

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$A = \begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix}$$

Apply this to

$$x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Get:

$$y_1 = 3 \cdot 1 + 1 \cdot 2 = 5$$

$$y_2 = 2 \cdot 1 + 4 \cdot 2 = 10$$

What if you know $y = (5, 10)$ and want to retrieve $x = (x_1, x_2)$?

Answer: Apply the inverse transformation to y . That is, multiply y by the inverse of the matrix.

$$x = A^{-1}y$$

In the case of a 2×2 matrix

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$A^{-1} = \frac{1}{|A|} \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} \quad (\text{A.7})$$

where the determinant of the array, A , is $|A| = a_{11}a_{22} - a_{12}a_{21}$

If $|A|$ is zero, then A is not invertible, it is *singular*.

Thus for

$$A = \begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix}$$

we have $|A| = 3 \times 4 - 2 \times 1 = 10$ so

$$A^{-1} = (1/10) \begin{pmatrix} 4 & -1 \\ -2 & 3 \end{pmatrix} = \begin{pmatrix} 0.4 & -0.1 \\ -0.2 & 0.3 \end{pmatrix}$$

Therefore, apply A^{-1} to $\begin{pmatrix} 5 \\ 10 \end{pmatrix}$

We find: $A^{-1}y =$

$$\begin{pmatrix} 0.4 & -0.1 \\ -0.2 & 0.3 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 10 \end{pmatrix} = \begin{pmatrix} 5 \times 0.4 + 10 \times -0.1 \\ 5 \times -0.2 + 10 \times 0.3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

which is what we started off with, i.e.

$$x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Note: in Ex. 3.2.1-1 (this was solving the linear equation system for the price of apples and oranges), we were actually doing something that is very similar to inverting the matrix

$$A = \begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix}$$

A.3.4 Particular Matrices

Diagonal Matrices

The scaling matrix mentioned in section 3.2.3

$$A = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix}$$

is *diagonal*, i.e. the only non-zero elements are on the diagonal.

The inverse of a diagonal matrix

$$\begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix}$$

is

$$\begin{pmatrix} 1/a_{11} & 0 \\ 0 & 1/a_{22} \end{pmatrix}$$

Transpose of a Matrix (A^t)

Only for square matrices. If

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

then

$$A^t = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix}$$

i.e. replace column 1 with row 1 etc.

The transpose is often written as A^t or A^T or A' . It is pronounced ‘A-transpose’.

The Identity Matrix

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

i.e. produces no transformation effect. Thus, $IA = A$

Note: If $AB = I$ then $B = A^{-1}$.

Orthogonal Matrix

A matrix which satisfies the property:

$$AA^t = I$$

i.e. the transpose of the matrix is its inverse.

Another way of viewing this is:

For each row of the matrix $(a_{r1} a_{r2} \dots a_{rn})$, the dot product with itself is 1, and with all other rows 0 (see section 3.2.13). I.e.

$$\begin{aligned} \sum_{c=1}^n a_{rc} a_{pc} &= 1 \text{ for } r = p \\ &= 0 \text{ otherwise} \end{aligned}$$

A.3.5 Complex Numbers

A complex number is simply a convenient way of representing the pair of numbers that represent the coordinates (x, y) of points in a plane,

$$z = x + jy$$

where $j = \sqrt{-1}$.

In many ways, a complex number is like a two-dimensional vector.

The *modulus* of the complex number (which may be interpreted as the distance between the origin and (x, y)) is given by:

$$|z| = |x + jy| = \sqrt{(x \times x + y \times y)}$$

i.e. using Pythagoras' Theorem

The angle, or *argument*, which may be interpreted as the angle between the line $(0, 0)$ to (x, y) and the x-axis, is given by:

$$\arg z = \arctan y/x$$

i.e. the angle whose tangent (opposite/adjacent) is y/x .

Addition of complex numbers is as follows: If

$$z = x + jy, \quad w = u + jv$$

then

$$z + w = x + u + j(y + v)$$

A graphical interpretation of addition of complex numbers is,

- draw a line from $(0,0)$ to (x,y) ,
- using (x,y) as the origin, draw a line to (u,v) ,
- the point reached is $(x+u, y+v)$.

I.e. *vector* addition.

Multiplication of complex numbers is as follows:

If

$$z = x + jy, \quad w = u + jv$$

then

$$z.w = (x + jy).(u + jv) = x.u + j.j.y.v + j.(y.u + x.v)$$

We use $j.j = -1$ (i.e. $\sqrt{-1}\sqrt{-1} = -1$) This gives:

$$z.w = (x.u - y.v) + j.(y.u + x.v)$$

Note: if complex numbers have zero imaginary parts, the rules given here collapse to the rules of normal arithmetic for real numbers.

Ex. 3.2.11-1 Verify the last statement, i.e. set $y, v = 0$ in addition and multiplication of complex numbers.

The *complex conjugate* of a complex number, $c = a + jb$ is

$$c^* = a - jb$$

Complex Numbers and Matrices

Matrices and vectors can contain complex numbers. The rules for matrix addition, multiplication, given above, all apply; we just replace the normal addition, multiplication with the complex versions given in the previous section dealing with “Complex Numbers”.

A.3.6 Further Matrix and Vector Operations

Vector Inner (Dot) Product

Let vector

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

i.e. $x = (x_1, x_2, \dots, x_n)^t$ (t denotes transpose), and vector $y = (y_1, y_2, \dots, y_n)^t$

The *inner product* (*dot product*, *scalar product*) of x and y is the matrix product (see section 3.2.2)

$$x^t y$$

Dimensions: 1×1 , $1 \times n$, $n \times 1$

This is the same as:

$$xy^t = \sum_{i=1}^n x_i y_i$$

If the dot product of two vectors is 0, they are said to be *orthogonal*, see section 3.2.10 above, and 3.2.16, 17 and 18 below.

Vector Addition

Three $n \times 1$ vectors x, y, z :

$$z = x + y$$

with

$$\begin{pmatrix} z_1 \\ z_2 \\ \cdot \\ \cdot \\ z_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \cdot \\ \cdot \\ x_n + y_n \end{pmatrix}$$

Distance between Vectors

Considering n -dimensional vectors as points in n -dimensional space, we can talk about the distance, d , between vectors x and y . The following is the squared distance:

$$d^2(x, y) = (x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2$$

or

$$d^2(x, y) = \sum_{i=1}^n (x_i - y_i)^2 \quad (\text{A.8})$$

Ex. 3.2.15-1 Determine the Euclidean distance between the points $(1,1)$, $(1,3)$.

```

    |
    3 + * (1,3)
    |
    2 +
    |
dim 2 1 + * (1,1)

```

```

  |
  +---+---+---+---+---+
  1   2   3   4       dim 1

```

$$d = \sqrt{(1-1)^2 + (1-3)^2} = \sqrt{0+4} = 2$$

Length or Magnitude of a Vector

Considering now an n -dimensional vector as the line joining the origin to its ‘point’ in n -dimensional space, we can talk about the *length* – or, more usually, the *magnitude* – of a vector as the distance between the vectors x and the origin $(0,0,0\dots)$:

$$|x| = \sqrt{\left(\sum_{i=1}^n (x_i)^2\right)}$$

Ex. 3.2.16-1 Verify that the magnitude of the vector $(1,0)$ is 1.

Ex. 3.2.16-2 Verify that the magnitude of the vector $(0,2)$ is 2.

Ex. 3.2.16-3 Verify that the magnitude of the vector $(1,1)$ is 1.414; i.e. $\sqrt{2}$. Note that it is *not* 1, as you can easily verify by sketching.

Normalized or Unit Length Vectors

Quite often we are just interested in the relative *directions* of vectors (see Chapters 8 and 9) and, for easier comparison, we would like to reduce all vectors to unity magnitude – this is called *normalization*.

Normalization is performed by the following (scaling – see section 3.2.3) transformation:

$$x_{i'} = x_i / |x|$$

where $x_{i'}$ is the i th component of the normalised vector, x_i is the i th component of the original vector, and $|x|$ is the magnitude of the original vector.

Ex. 3.2.17-1 Normalize the vector $(0,1)$; *Answer:* $(0,1)$.

Ex. 3.2.17-2 (a) Normalize the vector $(0,2)$; *Answer:* $(0,1)$ (b) Verify, with a diagram, that normalization has retained the ‘direction’ of the original vector.

Ex. 3.2.17-3 (a) Normalize the vector (1,1); *Answer:* (0.707,0.707).

(b) Verify.

Answer:

$$0.707 = 1/\sqrt{2}$$

$$\text{magnitude} = \sqrt{x_1^2 + x_2^2} = \sqrt{1/\sqrt{2}^2 + 1/\sqrt{2}^2} = \sqrt{1/2 + 1/2} = 1$$

Ex. 3.2.17-4 Verify that, in two dimensions, all unit vectors lie on the unit circle (a circle with centre at the origin (0,0) and with radius 1).

Template Matching of Unit Vectors

Quite often we wish to compare two vectors, x and y (assume they have already been normalized).

One way is to compute the distance between them (see section 3.2.15).

$$d = \sqrt{\left(\sum_{i=1}^n (x_i - y_i)^2\right)}$$

Then we can use the common sense rule:

Small distance \implies *similar*

Big distance \implies *different*

Alternatively, we can compute how well the corresponding components correlate, i.e. perform a ‘template’ matching by multiplying corresponding components and summing (i.e. a dot product – see section 3.2.13):

$$c = \sum_{i=1}^n x_i y_i$$

Then we can use the rule:

Big correlation value (c) \implies *similar*

Small \implies *different*

(See section 3.2.13: if $c = 0$ the two vectors are orthogonal.)

Maximizing correlation is equivalent to minimizing distance.

Intuitive proof (two-dimensions):

Expand Eqn. 3.6 for the case of 2-dimensions:

$$d = ((x_1 - y_1)^2 + (x_2 - y_2)^2) = (x_1^2 + y_1^2 - 2x_1y_1 + x_2^2 + y_2^2 - 2x_2y_2)$$

$$= (x_1^2 + x_2^2 + y_1^2 + y_2^2 - 2x_1y_1 - 2x_2y_2)$$

$$d = \sum_{i=1}^2 x_i^2 + \sum_{i=1}^2 y_i^2 - 2 \sum_{i=1}^2 x_i y_i$$

Since x and y are normalized to unit magnitude:

$$\sum_{i=1}^2 x_i^2 = \sum_{i=1}^2 y_i^2 = 1$$

Thus,

$$d = -2(c - 1)$$

When the vectors are the same, $x = y$, so that

$$c = \sum x_i y_i = \sum x_i x_i = 1$$

since x is unit magnitude. This is the highest possible value that c can attain (i.e. when the vectors are the *same*, the components are completely matched/correlated).

A.3.7 Vector Spaces

Vectors in Neural Networks and Pattern Recognition

Many approaches to automatic ‘pattern recognition’ (especially neural networks) use representations of patterns as arrays of numbers (vectors).

The recognition process often involves finding the ‘most similar’, of a set of stored patterns, to an unknown pattern.

Obviously, the distance is clearly a good measure of similarity: small distance large similarity; clearly also, ‘template-matching’ or correlation is intuitively appealing. We have shown that they both give the same result.

Neural Networks:

The computation performed by a single neuron, as used in artificial neural networks, is simply the dot product between the input excitations x_i and the weights, w_i ,

$$\text{sum} = \sum_{i=1}^n x_i w_i$$

followed by passing ‘sum’ through some threshold function, such as:

$$\begin{aligned} \text{output} &= 1 \text{ if sum } > T \\ &= 0 \text{ otherwise} \end{aligned}$$

Note the similarity with ‘template-matching’.

Ex. 3.2.19-1 The Figure below shows a letter ‘C’ in a small (3×3) part of a digital image (a digital picture). A digital picture is represented by brightness numbers (pixels) for each picture point.

Now, represent the nine pixel values as elements of a vector. Assuming the character is white-on-black and that bright (filled in with ‘*’) corresponds to ‘1’, and dark to ‘0’, the components of the vector corresponding to the ‘C’ are:

$$x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 0, x_6 = 0, x_7 = 1, x_8 = 1, x_9 = 1$$

Pixel representation:

1	2	3
+-----+-----+-----+		
**** **** ****		
**** **** ****		
+-----+-----+-----+		
4 **** 5 6		

+-----+-----+-----+		
**** **** ****		
**** **** ****		
+-----+-----+-----+		
7 8 9		

A Letter ‘C’

The letter ‘T’ would give a different observation vector:

‘T’: 1,1,1,	0,1,0,	0,1,0
‘O’: 1,1,1,	1,0,1,	1,1,1
‘C’: 1,1,1,	1,0,0,	1,1,1
etc.		

Linear Independence of Vectors

Two vectors a_i, a_j ,

$$a_i = (a_{i1}, a_{i2}, \dots, a_{ip})$$

$$a_j = (a_{j1}, a_{j2}, \dots, a_{jp})$$

are linearly **dependent** if one can be written as a scalar product of the other,

$$a_i = ca_j = (ca_{j1}, ca_{j2}, \dots, ca_{jp})$$

i.e. the vectors differ only by a scale factor, c , that is applied to all elements.

In such a case, the directions (see A10. above) of the vectors are the same; only their length differs by the scaling factor c .

If we have

$$b = \sum_{j=1}^n c_j a_j$$

then b is a linear combination of the a_j s and is linearly dependent on them.

Normally, as in the next section (rank) we are interested in the linear independence of vectors formed by rows of a matrix. If we have one row of a matrix that is linearly dependent on (some – or all) the others, this means that the simultaneous equation associated with that row contributes no new information.

Rank of a Matrix

Given a q -row $\times p$ -column matrix, A

$$A = \begin{vmatrix} / & & \backslash \\ | a_{11} & a_{12} & & a_{1p} \\ | a_{21} & a_{22} & & a_{2p} \\ | & & \dots & arc \dots \\ | & & & | \\ | a_{q1} & a_{q2} & & a_{qp} \\ \backslash & & & / \end{vmatrix}$$

the rank of A is the number of linearly independent rows in it (see A.11).

If $p = q$ the matrix is square, and we may need to invert it, it will only invert if all the rows are linearly independent; otherwise, the matrix is **singular** – non-invertable. One simple way of viewing this problem is that, for a system of simultaneous equations to be solvable, we need p equations in p unknowns; if one, or more, of the equations is linearly dependent on the others, this equation contributes no new information, i.e. we effectively

have only $p - 1$ ‘useful’ equations, and the system is unsolvable – its rank is $p - 1$.

In pattern recognition and estimation, the incidence of singular or nearly singular matrices is insidious and common, e.g. a common source is taking reading for a dependent variable, y , say, for the same, or nearly the same values of the independent variable, x . It can lead to nonsense results – analogous to what happens close to divide by 0 in floating point arithmetic.

Eigenvalues and Eigenvectors

For any positive-definite matrix R , there exists a unitary matrix U that satisfies the following equation:

$$U^T R U = L$$

where

$$\begin{aligned} L = & \begin{vmatrix} 11 & 0 & 0 & \dots & 0 \\ 0 & 12 & 0 & & 0 \\ & & \ddots & & \\ & & & 1p & \end{vmatrix} \end{aligned}$$

L is a diagonal matrix containing the **eigenvalues** of R , and

$$\begin{aligned} U = & \begin{vmatrix} u_1 \\ u_2 \\ \dots \\ u_i \\ \dots \\ u_p \end{vmatrix} \end{aligned}$$

is the matrix formed by the **eigenvectors**, u_i , of R .

Another equation governing eigenvalues and eigenvectors is

$$R u_i = \lambda_i u_i$$

or, in matrix form, showing all the eigenvectors and eigenvalues:

$$R U = U L$$

The equivalence of (A.13-4) and (A.13-1) is easily verified by pre-multiplying each side of (A.13.4) by U^T .

Appendix B

Appendix: Image Analysis and Pattern Recognition in Java