Martin H. Trauth

MATLAB® Recipes for Earth Sciences

Fourth Edition

Springer

www.it-ebooks.info

MATLAB® Recipes for Earth Sciences

www.it-ebooks.info

Martin H. Trauth

MATLAB® Recipes for Earth Sciences

Fourth Edition

Springer

www.it-ebooks.info

Martin H. Trauth
Institute for Earth and Environmental Science
University of Potsdam
Karl-Liebknecht-Str. 24-25
14476 Potsdam
Germany

E-Mail: trauth@geo.uni-potsdam.de

Additional material to this book can be downloaded from http://extras.springer.com.

ISBN 978-3-662-46243-0 ISBN 978-3-662-46244-7 (eBook)
DOI 10.1007/978-3-662-46244-7

Library of Congress Control Number: 2015932246

Springer Heidelberg New York Dordrecht London
© Springer-Verlag Berlin Heidelberg 2006, 2007, 2010, 2015
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

Springer-Verlag GmbH Berlin Heidelberg is part of Springer Science+Business Media (www.springer.com)

www.it-ebooks.info

Preface to the 4th Edition

The book *MATLAB Recipes for Earth Sciences* is designed to help undergraduate and postgraduate students, doctoral students, post-doctoral researchers, and professionals find quick solutions for common data analysis problems in earth sciences. It provides a minimal amount of theoretical background and demonstrates the application of all described methods through the use of examples. The MATLAB software is used since it not only provides numerous ready-to-use algorithms for most methods of data analysis but also allows the existing routines to be modified and expanded, or new software to be developed. The book contains MATLAB scripts, or *M-files*, that can be used to solve typical problems in earth sciences by simple statistics, time-series analysis, geostatistics, and image processing, and also demonstrates the application of selected advanced techniques of data analysis such as nonlinear time-series analysis, adaptive filtering, bootstrapping, and terrain analysis. The book's supplementary electronic material (available online through Springer Extras) includes recipes with all the MATLAB commands featured in the book and the example data. The MATLAB codes can be easily modified for application to the reader's own data and projects.

This revised and updated Fourth Edition includes new sections on creating graphical user interfaces with MATLAB, on hypothesis testing (including the Kolmogorov-Smirnov test, the Mann-Whitney test and the Ansari-Bradley test), on detecting abrupt transitions within time series, on exporting 3D graphics to create interactive documents, on processing LANDSAT and HYPERION satellite images, on shape-based object detection in images, on discriminant analysis, and on multiple linear regression analysis. Many of the pre-existing sections have also been expanded and rewritten, and numerous new examples have been included.

In order to derive the maximum benefit from this book the reader will need to have access to the MATLAB software and be able to execute the recipes while reading the book. The MATLAB recipes display various graphs on the screen that are not shown in the printed book. The tutorial-style book does, however, contain numerous figures making it possible to go through the text without actually running MATLAB on a computer. I have developed the recipes using MATLAB 8 Release R2014b, but most of them will also work with earlier software releases. While undergraduates participating in a course on data analysis might go through the entire book, more experienced readers may choose to refer to only one particular method in order to solve a specific problem. The concept of the book and the contents of its chapters

www.it-ebooks.info

are therefore outlined below, in order to make it easier to use for readers with a variety of different requirements.

- *Chapter 1* – This chapter introduces some fundamental concepts of samples and populations. It also links the various types of data, and questions to be answered from the data, to the methods described in the succeeding chapters.
- *Chapter 2* – A tutorial-style introduction to MATLAB designed for earth scientists. Readers already familiar with the software are advised to proceed directly to the succeeding chapters. The Fourth Edition now includes new sections on matrix manipulation, control flow, and creating graphical user interfaces. Many passages in the older sections have been expanded and rewritten.
- *Chapters 3 and 4* – Fundamentals in univariate and bivariate statistics. These two chapters contain basic concepts in statistics and also introduce advanced topics such as resampling schemes and cross validation. Readers who are already familiar with basic statistics might want to skip these chapters. The Fourth Edition now also includes an introduction to hypothesis testing as well as new sections on the Kolmogorov-Smirnov test, the Mann-Whitney test, and the Ansari-Bradley test. Furthermore, the section on correlation coefficients has been expanded to include introductions to Spearman's correlation coefficient and Kendall's correlation coefficient.
- *Chapters 5 and 6* – Readers who wish to work with time series are recommended to read both of these chapters. Time-series analysis and signal processing are closely linked. A good knowledge of statistics is required to work successfully with these methods. These two chapters are independent of the preceding chapters. The Fourth Edition now also includes a section on detecting abrupt transitions within time series.
- *Chapters 7 and 8* – I recommend reading through both of these chapters since the processing methods used for spatial data and for images have much in common. Moreover, spatial data and images are often combined in earth sciences, for instance when projecting satellite images onto digital elevation models. The Fourth Edition now includes new sections on exporting 3D graphs to create interactive documents. The introduction to remote sensing using MATLAB has been expanded and now includes sections on importing, processing and exporting LANDSAT, ASTER and HYPERION satellite images. Chapter 8 now also includes a comprehensive discussion of methods for image enhancement, correction and rectification as well as a new section on shape-based object detection in images.

www.it-ebooks.info

- *Chapter 9* – Data sets in earth sciences often have many variables and many data points. Multivariate methods are applied to a great variety of large data sets, including satellite imagery. Any reader particularly interested in multivariate methods is advised to read Chapters 3 and 4 before proceeding to this chapter. The Fourth Edition now includes new sections on discriminant analysis and multiple linear regression, and the older sections have been substantially rewritten.
- *Chapter 10* – Methods to analyze circular and spherical data are widely used in earth sciences. Structural geologists measure and analyze the orientation of slickensides (or striae) on a fault plane. The statistical analysis of circular data is also used in paleomagnetic applications. Microstructural investigations include the analysis of grain shapes and quartz c-axis orientations in thin sections.

The book has benefited from the comments of many people, in particular my contributing authors Robin Gebbers and Norbert Marwan, and my colleagues Sven Borchardt, Zuze Dulanya, Thomas Ebert, Joaquim Góis, Lydia Olaka, Eduardo Santamaría-Del-Angel, Marco Tedesco, Frederik Tilmann and many others. I very much appreciate the expertise and patience of Elisabeth Sillmann at *blaetterwaldDesign* who created the graphics and the complete page designs of the book. I am much obliged to Ed Manning for professional proofreading of the text. I also acknowledge Naomi Fernandes, Kate Fiore and Dana Wright from the *Book Program* at *The MathWorks, Inc.,* Tanja Dorfmann, Claudia Olrogge and Annegret Schumann at *The MathWorks GmbH Deutschland*, Annett Büttner, Chris Bendall and Christian Witschel and their team at *Springer*, and Andreas Bohlen, Brunhilde Schulz and their team at *UP Transfer GmbH*. I also thank Jan Danek at *HUMUSOFT*, Stacey Gage and Pradeep Ramamoorthy at *The MathWorks, Inc.* for their help in creating the animated 3D objects using *Simulink 3D Animation* and with writing the corresponding section of this book. Many thanks also to Steve Eddins for dealing with my many questions about the history of MATLAB and for helping in the use of the Image Processing Toolbox. I also acknowledge Pietro Cavallo, Pantelis Hadjipantelis, Giacomo Mazzi, and Christoph Stockhammer at The MathWorks, Inc. for their patience in dealing with the large number of questions that I have sent to the Support section of The MathWorks, Inc. over the last few months. I think the excellent support provided by the company more than justifies any expenditure that I have made on MATLAB licenses over the last 20 years or more. I also thank the *NASA/GSFC/METI/ERSDAC/JAROS* and *U.S./Japan ASTER Science Team*, and the director Mike Abrams, for allowing me to include the ASTER

www.it-ebooks.info

images in this book. I am grateful to Stuart W. Frye at the NASA Goddard Space Flight Center (GSFC) and his team for allowing me to include EO-1 data in the book, as well as for fruitful discussions while working on the section about Hyperion images.

Potsdam, December 2014
Martin H. Trauth

www.it-ebooks.info

Preface to the Interactive 4th Edition

With this fourth edition the book makes its first appearance as an interactive ebook. This format allows the reader to follow the contents in much the same way as they are presented in the courses that I teach at the University of Potsdam and elsewhere. During such courses the participants interact with the various tools by changing the input parameters of the algorithms and exploring the subsequent modifications introduced to the graphics. The spectral analysis of time series is demonstrated not only by examining graphics but also by listening to audio examples of signals and noise. Using MATLAB to visualize three-dimensional digital terrain models that can be rotated interactively helps to understand complex topography much better than two dimensional graphics.

This ebook allows the reader to interact with the book in a very similar manner. It comes in two different formats, with the first being for Apple iPads and Macs (in the form of an *.ibooks* file created with Apple's iBooks Author software) and the second being for other platforms (as a PDF file created with Adobe InDesign). The ebook has been designed to be read in landscape mode; it includes movies, galleries, audios and interactive 3D displays, as well as reviews at the end of each chapter. The *movies* demonstrate the use of graphical user interface tools; they also help to explore the effect that changing input parameters has on the output of a function and to visualize mathematical operations. *Galleries* are generally used to present a series of linked graphics, such as those displaying the various measures of the dispersion and shape of a distribution. *Audios* are used to provide a striking representation of signals and noise, as well as the effect of filters. *Interactive 3D displays* allow interactive rotating of digital terrain models and other three-dimensional objects. Each chapter ends with a *Review* (or quiz), which tests the reader's understanding of its content. These interactive media (or widgets) can be explored, observed, or listened to (on Macs) using a multi-touch trackpad or a mouse, or (on iPads) using your finger(s), by clicking the Play button (to watch movies and to listen to audios), by clicking the arrows or swiping left or right (to navigate through a gallery), or by clicking and dragging a 3D object to rotate it. Movies, galleries and 3D objects can be zoomed to full screen by toggling the full-screen mode in the lower right corner (with movies), or by simply clicking on the image (with galleries and 3D objects).

The interactive ebook for other platforms comes as a *.pdf* file, which can be accessed with the free Adobe Reader software. The reader of the PDF version of the book receives a complete package of files that includes the

www.it-ebooks.info

actual ebook as well as a series of directories containing the interactive objects, the recipes with all the MATLAB commands featured in the book and the example data. It is important to note that the specified directory tree, especially the names of the directories, should not be changed. The layout

of the *.pdf* file is identical to that of the printed book with icons outside the type area that refer to the interactive objects. The interactive objects are stored outside the PDF file and can be accessed from the *MRES4-Index.html* file that can be viewed in a web browser. The PDF version of the ebook contains all interactive objects included in the ebook for iPads and Macs, except for the reviews. The interactive PDF version was planned as an ebook with embedded interactive objects such as movies, galleries, audios and reviews. During the production process, however, it soon became clear that it is not yet possible to offer such a product that is compatible with all platforms.

The first edition of *MATLAB Recipes of Earth Sciences*, which came out in 2006, contained many small, and some (unfortunately) not so small, errors. These errors made it necessary to produce an errata file that was made available for download on my webpage. The ebook now allows the contents to be kept up-to-date through regular updates that is possible for the printed book. It is thus a living book that thrives on the comments of its readers, which I expect to receive in due course!

I hope that you enjoy using the animated objects while reading the interactive ebook! Comments, corrections and suggestions are very welcome.

Potsdam, December 2014
Martin H. Trauth

www.it-ebooks.info

Contents

Preface to the 4th Edition V
Preface to the Interactive 4th Edition IX

1 Data Analysis in Earth Sciences 1

1.1 Introduction 1
1.2 Data Collection 2
1.3 Types of Data 3
1.4 Methods of Data Analysis 7
Recommended Reading 9

2 Introduction to MATLAB 11

2.1 MATLAB in Earth Sciences 11
2.2 Getting Started 12
2.3 The Syntax 14
2.4 Array Manipulation 18
2.5 Data Structures and Classes of Objects 24
2.6 Data Storage and Handling 31
2.7 Control Flow 37
2.8 Scripts and Functions 41
2.9 Basic Visualization Tools 45
2.10 Generating Code to Recreate Graphics 48
2.11 Publishing M-Files 50
2.12 Creating Graphical User Interfaces 51
Recommended Reading 55

3 Univariate Statistics 57

3.1 Introduction 57
3.2 Empirical Distributions 58
3.3 Examples of Empirical Distributions 64
3.4 Theoretical Distributions 74
3.5 Examples of Theoretical Distributions 82
3.6 Hypothesis Testing 88
3.7 The t-Test 89
3.8 The F-Test 93
3.9 The χ^2-Test 97
3.10 The Kolmogorov-Smirnov Test 100

www.it-ebooks.info

3.11 Mann-Whitney Test 103
3.12 The Ansari-Bradley Test 109
3.13 Distribution Fitting 115
Recommended Reading 119

4 Bivariate Statistics 121

4.1 Introduction 121
4.2 Correlation Coefficients 122
4.3 Classical Linear Regression Analysis 131
4.4 Analyzing the Residuals 135
4.5 Bootstrap Estimates of the Regression Coefficients 137
4.6 Jackknife Estimates of the Regression Coefficients 138
4.7 Cross Validation 140
4.8 Reduced Major Axis Regression 141
4.9 Curvilinear Regression 143
4.10 Nonlinear and Weighted Regression 145
Recommended Reading 148

5 Time-Series Analysis 151

5.1 Introduction 151
5.2 Generating Signals 152
5.3 Auto-Spectral and Cross-Spectral Analysis 157
5.4 Examples of Auto-Spectral and Cross-Spectral Analysis 161
5.5 Interpolating and Analyzing Unevenly-Spaced Data 170
5.6 Evolutionary Power Spectrum 175
5.7 Lomb-Scargle Power Spectrum 179
5.8 Wavelet Power Spectrum 184
5.9 Detecting Abrupt Transitions in Time Series 192
5.10 Nonlinear Time-Series Analysis (by N. Marwan) 195
Recommended Reading 211

6 Signal Processing 215

6.1 Introduction 215
6.2 Generating Signals 217
6.3 Linear Time-Invariant Systems 218
6.4 Convolution, Deconvolution and Filtering 220
6.5 Comparing Functions for Filtering Data Series 224
6.6 Recursive and Nonrecursive Filters 226

www.it-ebooks.info

6.7 Impulse Response 228
6.8 Frequency Response 231
6.9 Filter Design 237
6.10 Adaptive Filtering 240
Recommended Reading 248

7 Spatial Data 249

7.1 Types of Spatial Data 249
7.2 The Global Geography Database GSHHG 250
7.3 The 1-Minute Gridded Global Relief Data ETOPO1 252
7.4 The 30-Arc Seconds Elevation Model GTOPO30 255
7.5 The Shuttle Radar Topography Mission SRTM 257
7.6 Exporting 3D Graphics to Create Interactive Documents 260
7.7 Gridding and Contouring 264
7.8 Comparison of Methods and Potential Artifacts 271
7.9 Statistics of Point Distributions 278
7.10 Analysis of Digital Elevation Models (by R. Gebbers) 285
7.11 Geostatistics and Kriging (by R. Gebbers) 295
Recommended Reading 313

8 Image Processing 315

8.1 Introduction 315
8.2 Data Storage 316
8.3 Importing, Processing and Exporting Images 322
8.4 Importing, Processing and Exporting LANDSAT Images 326
8.5 Importing and Georeferencing TERRA ASTER Images 331
8.6 Processing and Exporting EO-1 Hyperion Images 338
8.7 Digitizing from the Screen 343
8.8 Image Enhancement, Correction and Rectification 345
8.9 Color-Intensity Transects Across Varved Sediments 352
8.10 Grain Size Analysis from Microscope Images 357
8.11 Quantifying Charcoal in Microscope Images 364
8.12 Shape-Based Object Detection in Images 367
Recommended Reading 373

9 Multivariate Statistics 375

9.1 Introduction 375
9.2 Principal Component Analysis 377

www.it-ebooks.info

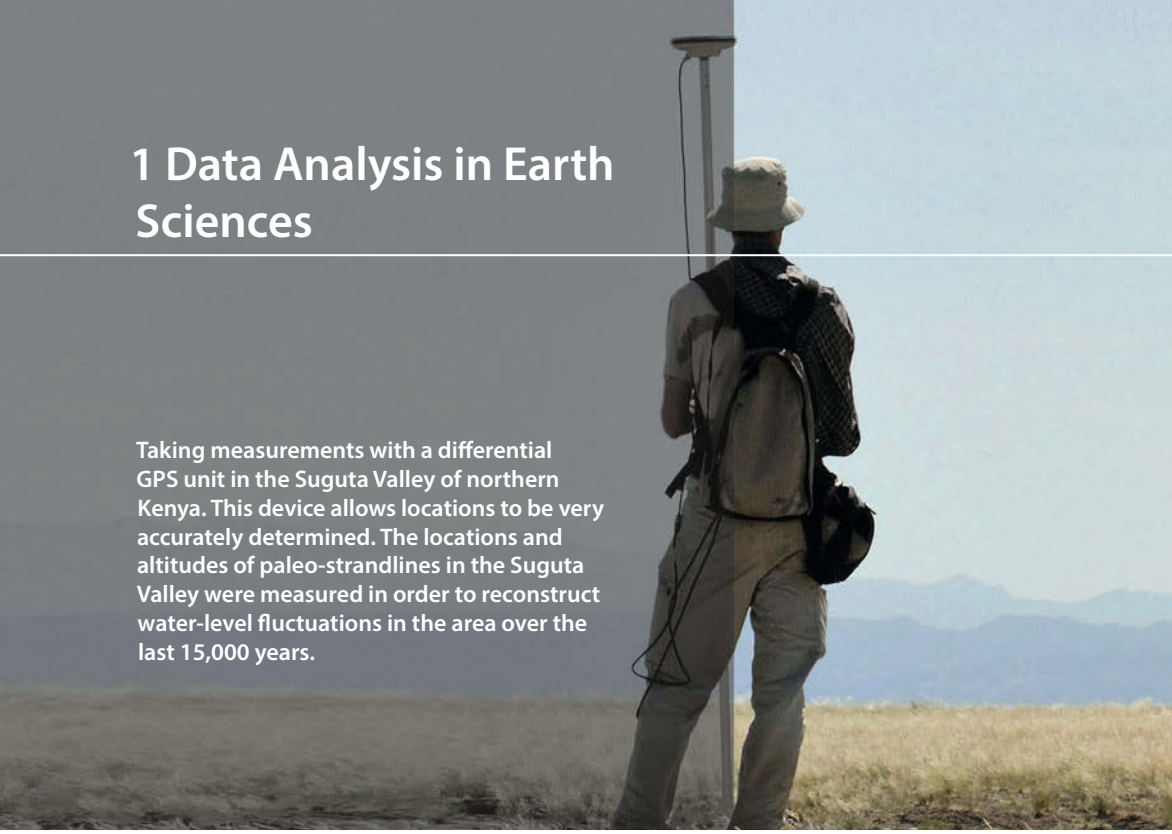
9.3 Independent Component Analysis (by N. Marwan) 386
9.4 Discriminant Analysis 392
9.5 Cluster Analysis 398
9.6 Multiple Linear Regression 402
Recommended Reading 411

10 Directional Data 413

10.1 Introduction 413
10.2 Graphical Representation 415
10.3 Empirical Distributions 417
10.4 Theoretical Distributions 419
10.5 Test for Randomness of Directional Data 421
10.6 Test for the Significance of a Mean Direction 422
10.7 Test for the Difference between Two Sets of Directions 424
Recommended Reading 427

www.it-ebooks.info

1 Data Analysis in Earth Sciences



Taking measurements with a differential GPS unit in the Suguta Valley of northern Kenya. This device allows locations to be very accurately determined. The locations and altitudes of paleo-strandlines in the Suguta Valley were measured in order to reconstruct water-level fluctuations in the area over the last 15,000 years.

1.1 Introduction

Earth scientists make observations and gather data about the natural processes that operate on planet Earth. They formulate and test hypotheses on the forces that have acted on a particular region to create its structure and also make predictions about future changes to the planet. All of these steps in exploring the Earth involve the acquisition and analysis of numerical data. An earth scientist therefore needs to have a firm understanding of statistical and numerical methods as well as the ability to utilize relevant computer software packages, in order to be able to analyze the acquired data.

This book introduces some of the most important methods of data analysis employed in earth sciences and illustrates their use through examples using the MATLAB® software package. These examples can then be used as recipes for the analysis of the reader's own data, after having learned their application with synthetic data. This introductory chapter deals with data acquisition (Section 1.2), the various types of data (Section 1.3) and the appropriate methods for analyzing earth science data (Section 1.4). We therefore first explore the characteristics of typical data sets and subsequently investigate the various ways of analyzing data using MATLAB.

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_1

www.it-ebooks.info

1.2 Data Collection

Most data sets in earth sciences have a very limited sample size and also contain a significant number of uncertainties. Such data sets are typically used to describe rather large natural phenomena, such as a granite body, a large landslide or a widespread sedimentary unit. The methods described in this book aim to find a way of predicting the characteristics of a larger *population* from a much smaller *sample* (Fig. 1.1). An appropriate sampling strategy is the first step towards obtaining a good data set. The development of a successful strategy for field sampling requires decisions on the *sample size* and the *spatial sampling scheme*.

The sample size includes the sample volume, the sample weight and the number of samples collected in the field. The sample weights or volumes can be critical factors if the samples are later analyzed in a laboratory and most statistical methods also have a minimum requirement for the sample size. The sample size also affects the number of subsamples that can be collected from a single sample. If the population is heterogeneous then the sample needs to be large enough to represent the population's variability, but on the other hand samples should be as small as possible in order to minimize the time and costs involved in their analysis. The collection of smaller pilot

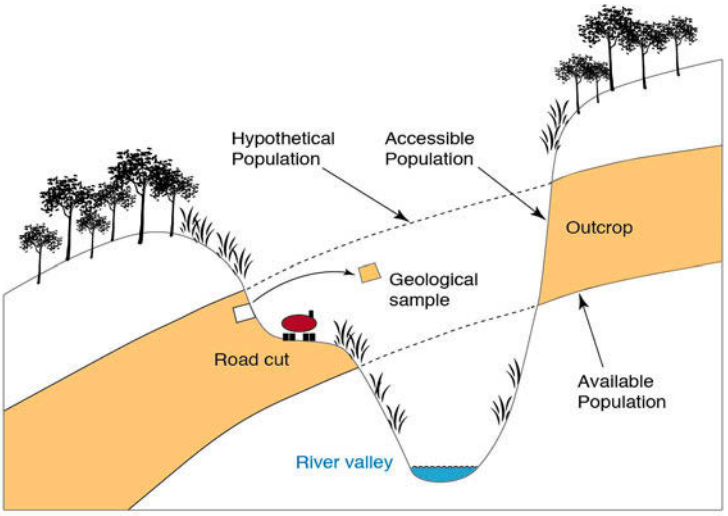


Fig. 1.1 Samples and populations. Deep valley incision has eroded parts of a sandstone unit (*hypothetical population*). The remaining sandstone (*available population*) can only be sampled from outcrops, i.e., road cuts and quarries (*accessible population*). Note the difference between a statistical sample as a representative of a population and a geological sample as a piece of rock.

www.it-ebooks.info

samples is recommended prior to defining a suitable sample size.

The design of the spatial sampling scheme is dependent on the availability of outcrops or other material suitable for sampling. Sampling in quarries typically leads to clustered data, whereas sampling along road cuts, shoreline cliffs or steep gorges results in one-dimensional traverse sampling schemes. A more uniform sampling pattern can be designed where there is 100% exposure or if there are no financial limitations. A regular sampling scheme results in a gridded distribution of sample locations, whereas a uniform sampling strategy includes the random location of a sampling point within a grid square. Although these sampling schemes might be expected to provide superior methods for sampling collection, evenly-spaced sampling locations tend to miss small-scale variations in the area, such as thin mafic dykes within a granite body or the spatially-restricted occurrence of a particular type of fossil (Fig. 1.2).

The correct sampling strategy will depend on the objectives of the investigation, the type of analysis required and the desired level of confidence in the results. Having chosen a suitable sampling strategy, the quality of the sample can be influenced by a number of factors resulting in the samples not being truly representative of the larger population. Chemical or physical alteration, contamination by other material or displacement by natural and anthropogenic processes may all result in erroneous results and interpretations. It is therefore recommended that the quality of the samples, the method of data analysis employed and the validity of the conclusions drawn from the analysis be checked at each stage of the investigation.

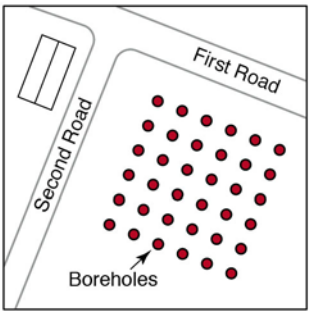
1.3 Types of Data

Most earth science data sets consist of numerical measurements, although some information can also be represented by a list of names such as fossils and minerals (Fig. 1.3). The available methods for data analysis may require certain types of data in earth sciences. These are

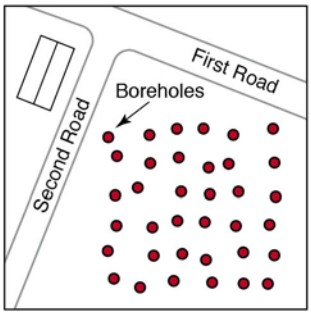
- *nominal data* – Information in earth sciences is sometimes presented as a list of names, e.g., the various fossil species collected from a limestone bed or the minerals identified in a thin section. In some studies, these data are converted into a binary representation, i.e., *one* for present and *zero* for absent. Special statistical methods are available for the analysis of such data sets.

- *ordinal data* – These are numerical data representing observations that can be ranked, but in which the intervals along the scale are irregularly

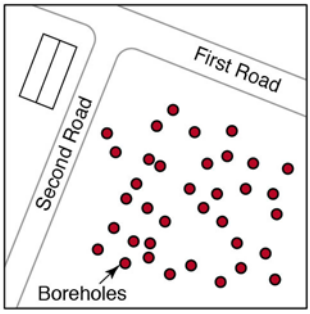
www.it-ebooks.info



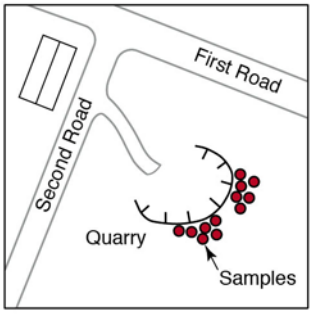
a



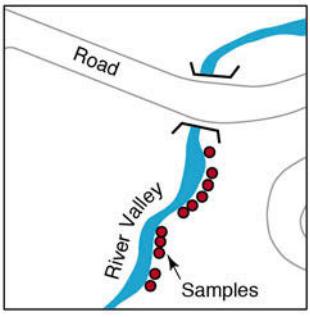
b



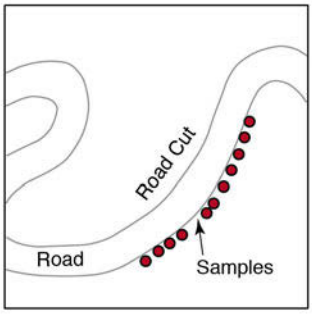
c



d



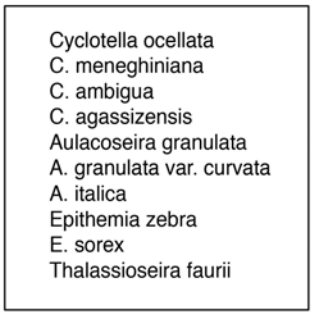
e



f

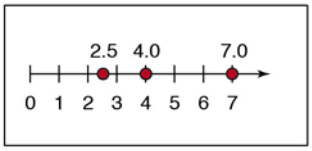
Fig. 1.2 Sampling schemes. **a** *Regular sampling* on an evenly-spaced rectangular grid, **b** *uniform sampling* by obtaining samples randomly located within regular grid squares, **c** *random sampling* using uniformly-distributed *xy* coordinates, **d** *clustered sampling* constrained by limited access in a quarry, and *traverse sampling* along **e** river valleys and **f** road cuts.

www.it-ebooks.info

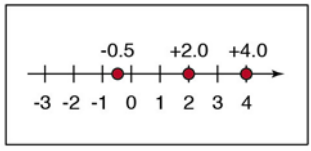


a

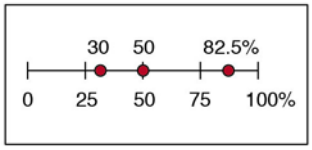
b



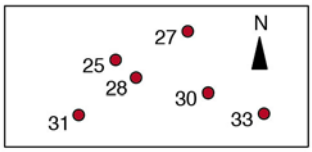
c



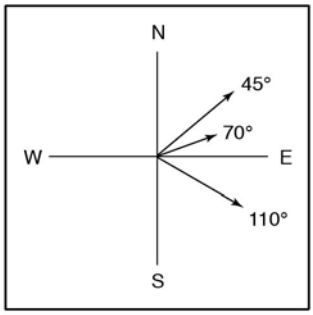
d



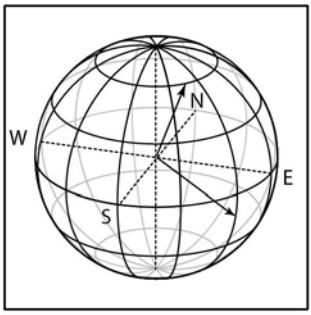
e



f



g



h

Fig. 1.3 Types of earth science data. **a** *Nominal data*, **b** *ordinal data*, **c** *ratio data*, **d** *interval data*, **e** *closed data*, **f** *spatial data*, and **g-h** *directional data*. All of these data types are described in this book.

www.it-ebooks.info

spaced. The hardness scale of German mineralogist Friedrich Mohs (1773–1839) is one example of an ordinal scale. The hardness value indicates the material's resistance to scratching. Diamond has a hardness of 10, whereas the value for talc is 1, but in terms of absolute hardness diamond (hardness 10) is four times harder than corundum (hardness 9) and six times harder than topaz (hardness 8). The Modified Mercalli Scale, which attempts to categorize the effects of earthquakes, is another example of an ordinal scale; it ranks earthquakes from intensity I (barely felt) to XII (total destruction) (Richter 1958).

- *ratio data* – These data are characterized by a constant length of successive intervals, therefore offering a great advantage over ordinal data. The zero point is the natural termination of the data scale, and this type of data allows for either discrete or continuous data sampling. Examples of such data sets include length or weight data.

- *interval data* – These are ordered data that have a constant length of successive intervals, but in which the data scale is not terminated by zero. Temperatures C and F represent an example of this data type even though arbitrary zero points exist for both scales. This type of data may be sampled continuously or in discrete intervals.

In addition to these standard data types, earth scientists frequently encounter special kinds of data such as

- *closed data* – These data are expressed as proportions and add up to a fixed total such as 100 percent. Compositional data represent the majority of closed data, such as element compositions of rock samples.

- *spatial data* – These are collected in a 2D or 3D study area. The spatial distribution of a certain fossil species, the spatial variation in thickness of a sandstone bed and the distribution of tracer concentrations in groundwater are examples of this type of data, which is likely to be the most important data type in earth sciences.

- *directional data* – These data are expressed in angles. Examples include the strike and dip of bedding, the orientation of elongated fossils or the flow direction of lava. This is another very common type of data in earth sciences.

www.it-ebooks.info

Most of these different types of data require specialized methods of analysis, some of which are outlined in the next section.

1.4 Methods of Data Analysis

Data analysis uses precise characteristics of small samples to hypothesize about the general phenomenon of interest. Which particular method is used to analyze the data depends on the data type and the project requirements. The various methods available include:

- *Univariate methods* – Each variable is assumed to be independent of the others and is explored individually. The data are presented as a list of numbers representing a series of points on a scaled line. Univariate statistical methods include the collection of information about the variable, such as the minimum and maximum values, the average, and the dispersion about the average. This information is then used to attempt to infer the underlying processes responsible for the variations in the data. Examples are the effects of chemical weathering on the sodium content of volcanic glass shards, or the influence of specific environmental factors on the sizes of snail shells within a sediment layer.

- *Bivariate methods* – Two variables are investigated together to detect relationships between these two parameters. For example, the correlation coefficient may be calculated to investigate whether there is a linear relationship between two variables. Alternatively, the bivariate regression analysis may be used to find an equation that describes the relationship between the two variables. An example of a bivariate plot is the *Harker Diagram*, which is one of the oldest methods of visualizing geochemical data from igneous rocks and simply plots oxides of elements against SiO_2 (Harker 1909).

- *Time-series analysis* – These methods investigate data sequences as a function of time. The time series is decomposed into a long-term trend, a systematic (periodic, cyclic, rhythmic) component and an irregular (random, stochastic) component. A widely used technique to describe cyclic components of a time series is that of spectral analysis. Examples of the application of these techniques include the investigation of cyclic climatic variations in sedimentary rocks, or the analysis of seismic data.

- *Signal processing* – This includes all techniques for manipulating a signal

www.it-ebooks.info

to minimize the effects of noise in order to correct all kinds of unwanted distortions or to separate various components of interest. It includes the design and realization of filters, and their application to the data. These methods are widely used in combination with time-series analysis, e.g., to increase the signal-to-noise ratio in climate time series, digital images or geophysical data.

- *Spatial analysis* – This is the analysis of parameters in 2D or 3D space and hence two or three of the required parameters are coordinate numbers. These methods include descriptive tools to investigate the spatial pattern of geographically distributed data. Other techniques involve spatial regression analysis to detect spatial trends. Also included are 2D and 3D interpolation techniques, which help to estimate surfaces representing the predicted continuous distribution of the variable throughout the area. Examples are drainage-system analysis, the identification of old landscape forms and lineament analysis in tectonically active regions.

- *Image processing* – The processing and analysis of images has become increasingly important in earth sciences. These methods involve importing and exporting, compressing and decompressing, and displaying images. Image processing also aims to enhance images for improved intelligibility, and to manipulate images in order to increase the signal-to-noise ratio. Advanced techniques are used to extract specific features or analyze shapes and textures, such as for counting mineral grains or fossils in microscope images. Another important application of image processing is in the use of satellite remote sensing to map certain types of rocks, soils and vegetation, as well as other parameters such as soil moisture, rock weathering and erosion.

- *Multivariate analysis* – These methods involve the observation and analysis of more than one statistical variable at a time. Since the graphical representation of multidimensional data sets is difficult, most of these methods include dimension reduction. Multivariate methods are widely used on geochemical data, for instance in tephrochronology where volcanic ash layers are correlated by geochemical fingerprinting of glass shards. Another important usage is in the comparison of species assemblages in ocean sediments for the reconstruction of paleoenvironments.

- *Analysis of directional data* – Methods to analyze circular and spherical data are widely used in earth sciences. Structural geologists measure

www.it-ebooks.info

and analyze the orientation of slickensides (or striae) on a fault plane, circular statistical methods are common in paleomagnetic studies, and microstructural investigations include the analysis of grain shapes and quartz c-axis orientations in thin sections.

Some of these methods of data analysis require the application of numerical methods such as interpolation techniques. While the following text deals mainly with statistical techniques it also introduces several numerical methods commonly used in earth sciences.

Recommended Reading

Araghinejad S (2014) Data-Driven Modeling: Using MATLAB® in Water Resources and Environmental Engineering. Springer, Berlin Heidelberg New York

Borradaile G (2003) Statistics of Earth Science Data – Their Distribution in Time, Space and Orientation. Springer, Berlin Heidelberg New York

Carr JR (1994) Numerical Analysis for the Geological Sciences. Prentice Hall, Englewood Cliffs, New Jersey

Davis JC (2002) Statistics and Data Analysis in Geology, Third Edition. John Wiley and Sons, New York

Gonzalez RC, Woods RE, Eddins SL (2009) Digital Image Processing Using MATLAB – 2nd Edition. Gatesmark Publishing, LLC

Haneberg WC (2004) Computational Geosciences with Mathematica. Springer, Berlin Heidelberg New York

Harker A (1909) The Natural History of Igneous Rocks, Macmillan, New York

Holzbecher E (2012) Environmental Modeling using MATLAB – 2nd Edition. Springer, Berlin Heidelberg New York

Menke W, Menke J (2012) Environmental Data Analysis with MATLAB. Springer, Berlin Heidelberg New York

Middleton GV (1999) Data Analysis in the Earth Sciences Using MATLAB. Prentice Hall, New Jersey

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007) Numerical Recipes: The Art of Scientific Computing – 3rd Edition. Cambridge University Press, Cambridge

Quarteroni A, Saleri F, Gervasio P (2014) Scientific Computing with MATLAB and Octave – 4th Edition. Springer, Berlin Heidelberg New York

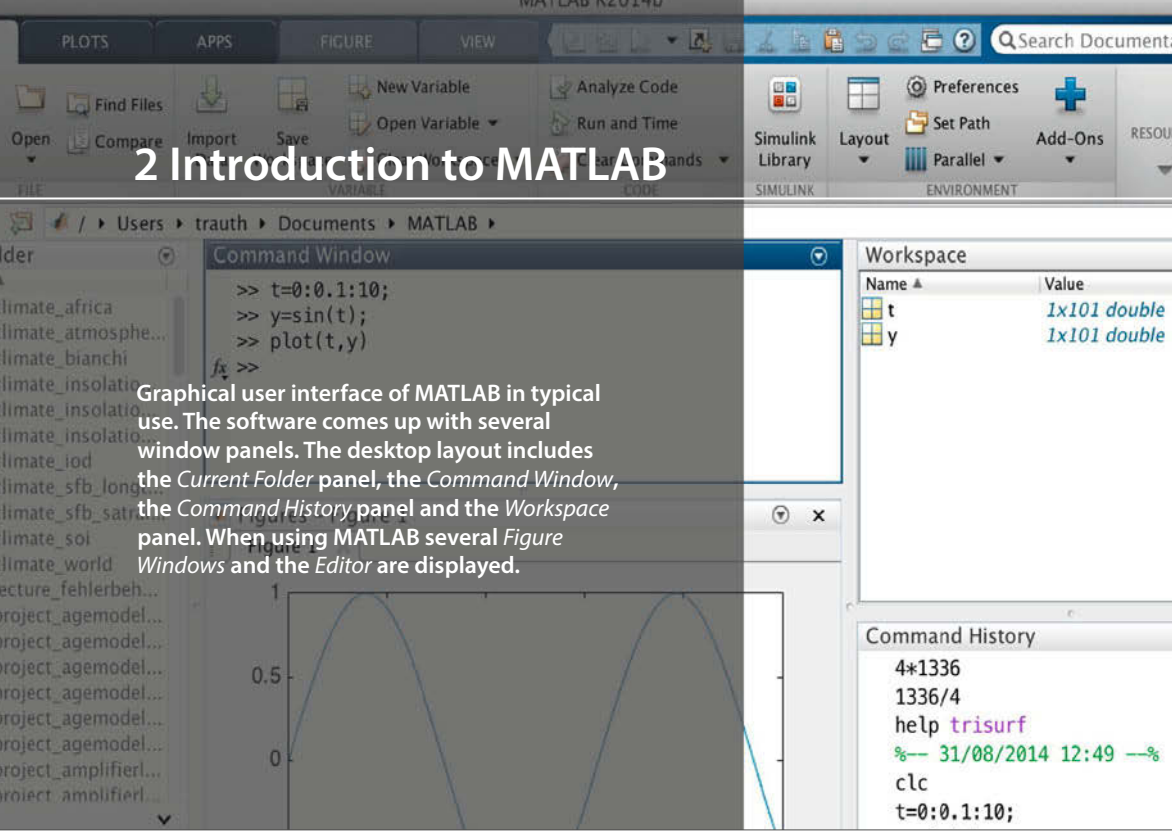
Richter CF (1958) Elementary Seismology. WH Freeman and Company, San Francisco

Swan ARH, Sandilands M (1995) Introduction to Geological Data Analysis. Blackwell Sciences, Oxford

Wilks DS (2011) Statistical methods in the atmospheric sciences – 3rd Edition. Academic Press, Elsevier, Amsterdam

www.it-ebooks.info

2 Introduction to MATLAB



Graphical user interface of MATLAB in typical use. The software comes up with several window panels. The desktop layout includes the *Current Folder* panel, the *Command Window*, the *Command History* panel and the *Workspace* panel. When using MATLAB several *Figure Windows* and the *Editor* are displayed.

2.1 MATLAB in Earth Sciences

MATLAB® is a software package developed by The MathWorks, Inc., founded by Cleve Moler, Jack Little and Steve Bangert in 1984, which has its headquarters in Natick, Massachusetts (http://www.mathworks.com). MATLAB was designed to perform mathematical calculations, to analyze and visualize data, and to facilitate the writing of new software programs. The advantage of this software is that it combines comprehensive math and graphics functions with a powerful high-level language. Since MATLAB contains a large library of ready-to-use routines for a wide range of applications, the user can solve technical computing problems much more quickly than with traditional programming languages, such as C++ and FORTRAN. The standard library of functions can be significantly expanded by add-on toolboxes, which are collections of functions for special purposes such as image processing, creating map displays, performing geospatial data analysis or solving partial differential equations.

During the last few years MATLAB has become an increasingly popular tool in earth sciences. It has been used for finite element modeling, processing of seismic data, analyzing satellite imagery, and for the generation

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_2

www.it-ebooks.info

of digital elevation models from satellite data. The continuing popularity of the software is also apparent in published scientific literature, and many conference presentations have also made reference to MATLAB. Universities and research institutions have recognized the need for MATLAB training for staff and students, and many earth science departments across the world now offer MATLAB courses for undergraduates. The MathWorks, Inc. provides classroom kits for teachers at a reasonable price, and it is also possible for students to purchase a low-cost edition of the software. This student version provides an inexpensive way for students to improve their MATLAB skills.

The following sections contain a tutorial-style introduction to MATLAB, covering the setup on the computer (Section 2.2), the MATLAB syntax (Sections 2.3 and 2.4), data input and output (Sections 2.5 and 2.6), programming (Sections 2.7 and 2.8), and visualization (Section 2.9). Advanced sections are also included on generating M-files to recreate graphics (Section 2.10), on publishing M-files (Section 2.11), and on creating graphical user interfaces (Section 2.12). The reader is recommended to go through the entire chapter in order to obtain a good knowledge of the software before proceeding to the following chapters of the book. A more detailed introduction can be found in the MATLAB *Primer* (MathWorks 2014a) which is available in print form, online and as PDF file.

In this book we use MATLAB Version 8 (Release 2014b), the Image Processing Toolbox Version 9.1, the Mapping Toolbox Version 4.0.2, the Signal Processing Toolbox Version 6.22, the Statistics Toolbox Version 9.1, the Wavelet Toolbox Version 4.14, and the Simulink 3D Animation Toolbox Version 7.2.

2.2 Getting Started

The software package comes with extensive documentation, tutorials and examples. The first three chapters of the book *MATLAB Primer* (MathWorks 2014a) are directed at beginners. The chapters on programming, creating graphical user interfaces (GUIs) and development environments are aimed at more advanced users. Since *MATLAB Primer* provides all the information required to use the software, this introduction concentrates on the most relevant software components and tools used in the following chapters of this book.

After the installation of MATLAB, the software is launched either by clicking the shortcut icon on the desktop or by typing

```
matlab
```

www.it-ebooks.info

in the operating system prompt. The software then comes up with several window panels (Fig. 2.1). The default desktop layout includes the *Current Folder* panel that lists the files in the directory currently being used. The *Command Window* presents the interface between the software and the user, i.e., it accepts MATLAB commands typed after the prompt, `>>`. The *Workspace* panel lists the variables in the MATLAB workspace, which is empty when starting a new software session. In this book we mainly use the Command Window and the built-in *Editor*, which can be launched by typing

```
edit
```

By default, the software stores all of your MATLAB-related files in the startup folder named *MATLAB*. Alternatively, you can create a personal working directory in which to store your MATLAB-related files. You should then make this new directory the working directory using the *Current Folder* panel or the *Folder Browser* at the top of the MATLAB desktop. The software uses a *Search Path* to find MATLAB-related files, which are

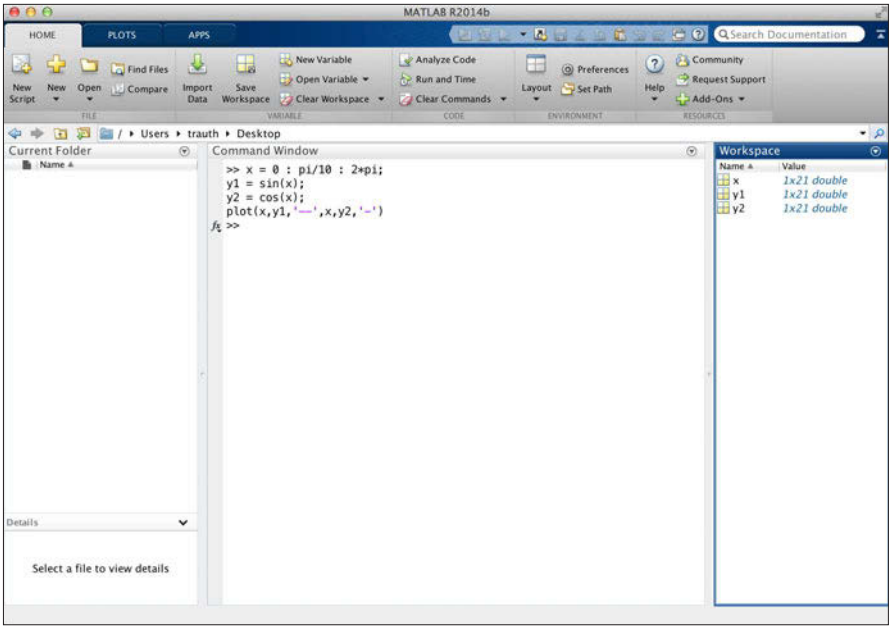


Fig. 2.1 Screenshot of the MATLAB default desktop layout including the *Current Folder* (left in the figure), the *Command Window* (center), the *Workspace* (right) panels. This book uses only the Command Window and the built-in *Editor*, which can be called up by typing `edit` after the prompt. All information provided by the other panels can also be accessed through the Command Window.

www.it-ebooks.info

organized in directories on the hard disk. The default search path includes only the *MATLAB_R2014b* directory that has been created by the installer in the applications folder and the default working directory *MATLAB*. To see which directories are in the search path or to add new directories, select *Set Path* from the *Home* toolstrip of the MATLAB desktop, and use the *Set Path* dialog box. The modified search path is saved in a file *pathdef.m* on your hard disk. The software will then in future read the contents of this file and direct MATLAB to use your custom path list.

2.3 The Syntax

The name MATLAB stands for *matrix laboratory*. The classic object handled by MATLAB is a *matrix*, i.e., a rectangular two-dimensional *array* of numbers. A simple 1-by-1 array is a *scalar*. Arrays with one column or row are *vectors*, time series or other one-dimensional data fields. An *m*-by-*n* array can be used for a digital elevation model or a grayscale image. Red, green and blue (RGB) color images are usually stored as three-dimensional arrays, i.e., the colors red, green and blue are represented by an *m*-by-*n*-by-3 array.

Before proceeding, we need to clear the workspace by typing

```
clear
```

after the prompt in the Command Window. Clearing the workspace is always recommended before working on a new MATLAB project to avoid name conflicts with previous projects. We can also go a step further, close all Figure Windows using `close all` and clear the content of the Command Window using `clc`. It is therefore recommended that a new MATLAB project should always start with the line

```
clear, close all, clc
```

Entering matrices or arrays in MATLAB is easy. To enter an arbitrary matrix, type

```
A = [2 4 3 7; 9 3 -1 2; 1 9 3 7; 6 6 3 -2]
```

which first defines a variable `A`, then lists the elements of the array in square brackets. The rows of `A` are separated by semicolons, whereas the elements of a row are separated by blank spaces, or alternatively, by commas. After pressing *return*, MATLAB displays the array

```
A =
     2     4     3     7
     9     3    -1     2
```

www.it-ebooks.info

```
    1     9     3     7
    6     6     3    -2
```

Displaying the elements of `A` could be problematic for very large arrays such as digital elevation models consisting of thousands or millions of elements. To suppress the display of an array or the result of an operation in general, the line should be ended with a semicolon.

```
A = [2 4 3 7; 9 3 -1 2; 1 9 3 7; 6 6 3 -2];
```

The array `A` is now stored in the workspace and we can carry out some basic operations with it, such as computing the sum of elements,

```
sum(A)
```

which results in the display

```
ans =
    18    22     8    14
```

Since we did not specify an output variable, such as `A` for the array entered above, MATLAB uses a default variable `ans`, short for *answer* or *most recent answer*, to store the results of the calculation. In general, we should define variables since the next computation without a new variable name will overwrite the contents of `ans`.

The above example illustrates an important point about MATLAB: the software prefers to work with the columns of arrays. The four results of `sum(A)` are obviously the sums of the elements in each of the four columns of `A`. To sum all elements of `A` and store the result in a scalar `b`, we simply need to type

Movie 2.1

```
b = sum(sum(A));
```

which first sums the columns of the array and then the elements of the resulting vector. We now have two variables, `A` and `b`, stored in the workspace. We can easily check this by typing

```
whos
```

which is one the most frequently-used MATLAB commands. The software then lists all variables in the workspace, together with information about their sizes or dimensions, number of bytes, classes and attributes (see Section 2.5 for details about classes and attributes of objects).

```
Name      Size            Bytes  Class     Attributes
A         4x4               128  double
```

www.it-ebooks.info

```
ans       1x4                32  double
b         1x1                 8  double
```

Movie 2.2

Note that by default MATLAB is case sensitive, i.e., `A` and `a` can define two different variables. In this context, it is recommended that capital letters be used for arrays that have two dimensions or more and lower-case letters for one-dimensional arrays (or vectors) and for scalars. However, it is also common to use variables with mixed large and small letters. This is particularly important when using descriptive variable names, i.e., variables whose names contain information concerning their meaning or purpose, such as the variable `CatchmentSize`, rather than a single-character variable `a`. We could now delete the contents of the variable `ans` by typing

```
clear ans
```

Next, we will learn how specific array elements can be accessed or exchanged. Typing

```
A(3,2)
```

simply yields the array element located in the third row and second column, which is `9`. The array indexing therefore follows the rule *(row, column)*. We can use this to replace single or multiple array elements. As an example we type

```
A(3,2) = 30
```

to replace the element `A(3,2)` by `30` and to display the entire array.

```
A =
     2     4     3     7
     9     3    -1     2
     1    30     3     7
     6     6     3    -2
```

If we wish to replace several elements at one time, we can use the *colon operator*. Typing

```
A(3,1:4) = [1 3 3 5]
```

or

```
A(3,:) = [1 3 3 5]
```

replaces all elements of the third row of the array `A`. The colon operator also has several other uses in MATLAB, for instance as a shortcut for entering array elements such as

www.it-ebooks.info

```
c = 0 : 10
```

which creates a vector, or a one-dimensional array with a single row, containing all integers from 0 to 10. The resultant MATLAB response is

```
c =
     0   1   2   3   4   5   6   7   8   9   10
```

Note that this statement creates 11 elements, i.e., the integers from 1 to 10 and the zero. A common error when indexing arrays is to ignore the zero and therefore expect 10 elements instead of 11 in our example. We can check this from the output of `whos`.

```
Name        Size              Bytes  Class     Attributes
A           4x4                 128  double
ans         1x1                   8  double
b           1x1                   8  double
c           1x11                 88  double
```

The above command creates only integers, i.e., the interval between the array elements is one unit. However, an arbitrary interval can be defined, for example 0.5 units. This is later used to create evenly-spaced time vectors for time series analysis. Typing

```
c = 1 : 0.5 : 10
```

results in the display

```
c =
  Columns 1 through 6
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000
  Columns 7 through 12
    4.0000    4.5000    5.0000    5.5000    6.0000    6.5000
  Columns 13 through 18
    7.0000    7.5000    8.0000    8.5000    9.0000    9.5000
  Column 19
   10.0000
```

which autowraps the lines that are longer than the width of the Command Window. The display of the values of a variable can be interrupted by pressing *Ctrl*+*C* (*Control*+*C*) on the keyboard. This interruption affects only the output in the Command Window, whereas the actual command is processed before displaying the result.

MATLAB provides standard arithmetic operators for addition, `+`, and subtraction, `-`. The asterisk, `*`, denotes matrix multiplication involving inner products between rows and columns. For instance, we multiply the matrix `A` with a new matrix `B`

www.it-ebooks.info

```
B = [4 2 6 5; 7 8 5 6; 2 1 -8 -9; 3 1 2 3];
```

the matrix multiplication is then

```
C = A * B'
```

where `'` is the complex conjugate transpose, which turns rows into columns and columns into rows. This generates the output

```
C =
    69   103   -79    37
    46    94    11    34
    53    76   -64    27
    44    93    12    24
```

In linear algebra, matrices are used to keep track of the coefficients of linear transformations. The multiplication of two matrices represents the combination of two linear transformations into a single transformation. Matrix multiplication is not commutative, i.e., `A*B'` and `B*A'` yield different results in most cases. Similarly, MATLAB allows matrix divisions representing different transformations, with `/` as the operator for right-hand matrix division and `\` as the operator for left-hand division. Finally, the software also allows powers of matrices, `^`.

In earth sciences, however, matrices are often simply used as two-dimensional arrays of numerical data rather than a matrix *sensu stricto* representing a linear transformation. Arithmetic operations on such arrays are carried out element-by-element. While this does not make any difference in addition and subtraction, it does affect multiplicative operations. MATLAB uses a dot, `.`, as part of the notation for these operations.

As an example multiplying `A` and `B` element-by-element is performed by typing

```
C = A .* B
```

which generates the output

```
C =
     8     8    18    35
    63    24    -5    12
     2     3   -24   -45
    18     6     6    -6
```

2.4 Array Manipulation

MATLAB provides a wide range of functions with which to manipulate arrays (or matrices). This section introduces the most important functions

www.it-ebooks.info

for array manipulation, which are used later in the book. We first clear the workspace and create two arrays, `A` and `B`, by typing

```
clear

A = [2 4 3; 9 3 -1]
B = [1 9 3; 6 6 3]
```

which yields

```
A =
     2     4     3
     9     3    -1

B =
     1     9     3
     6     6     3
```

When we work with arrays, we sometimes need to concatenate two or more arrays into a single array. We can use either `cat(dim,A,B)` with `dim=1` to concatenate the arrays `A` and `B` along the first dimension (i.e., along the rows). Alternatively, we can use the function `vertcat` to concatenate the arrays `A` and `B` vertically. By typing either

```
C = cat(1,A,B)
```

or

```
C = vertcat(A,B)
```

we obtain (in both cases)

```
C =
     2     4     3
     9     3    -1
     1     9     3
     6     6     3
```

Similarly, we can concatenate arrays horizontally, i.e., concatenate the arrays along the second dimension (i.e., along the columns) by typing

```
D = cat(2,A,B)
```

or using the function `horzcat` instead

```
D = horzcat(A,B)
```

which both yield

```
D =
```

www.it-ebooks.info

```
     2     4     3     1     9     3
     9     3    -1     6     6     3
```

When working with satellite images we often concatenate three spectral bands of satellite images into three-dimensional arrays of the colors red, green and blue (RGB) (Sections 2.5 and 8.4). We again use `cat(dim,A,B)` with `dim=3` to concatenate the arrays `A` and `B` along the third dimension by typing

```
E = cat(3,A,B)
```

which yields

```
E(:,:,1) =
     2     4     3
     9     3    -1

E(:,:,2) =
     1     9     3
     6     6     3
```

Typing

```
whos
```

yields

```
  Name      Size             Bytes  Class     Attributes

  A         2x3                 48  double
  B         2x3                 48  double
  C         4x3                 96  double
  D         2x6                 96  double
  E         2x3x2               96  double
```

indicating that we have now created a three-dimensional array, as the size 2-by-3-by-2 suggests. Alternatively, we can use

```
size(E)
```

which yields

```
ans =
     2     3     2
```

to see that the array has `2` rows, `3` columns, and `2` layers in the third dimension. Using `length` instead of `size`,

```
length(A)
```

yields

www.it-ebooks.info

```
ans =
     3
```

which tells us the dimension of the largest array only. Hence `length` is normally used to determine the length of a one-dimensional array (or vector), such the evenly-spaced time axis `c` that was created in Section 2.3.

MATLAB uses a matrix-style indexing of arrays with the (1,1) element being located in the upper-left corner of arrays. Other types of data that are to be imported into MATLAB may follow a different indexing convention. As an example, digital terrain models (introduced in Chapter 7.3 to 7.5) often have a different way of indexing and therefore need to be flipped in an up-down direction or, in other words, about a horizontal axis. Alternatively, we can flip arrays in a left-right direction (i.e., about a vertical axis). We can do this by using `flipud` for flipping in an up-down direction and `fliplr` for flipping in a left-right direction

```
F = flipud(A)
F = fliplr(A)
```

yielding

```
F =
     9     3    -1
     2     4     3

F =
     3     4     2
    -1     3     9
```

In more complex examples we can use `circshift(A,K,dim)` to circularly shift (i.e., rotate) arrays by `K` positions along the dimension `dim`. As an example we can shift the array `A` by 1 position along the 2nd dimension (i.e., along the rows) by typing

```
G = circshift(A,1,2)
```

which yields

```
G =
     3     2     4
    -1     9     3
```

We can also use `reshape(A,[m n])` to completely reshape the array. The result is an `m`-by-`n` array `H` whose elements are taken column-wise from `A`. As an example we create a 3-by-2 array from `A` by typing

```
H = reshape(A,[3 2])
```

www.it-ebooks.info

which yields

```
H =
     2     3
     9     3
     4    -1
```

Another important way to manipulate arrays is to sort their elements. As an example we can use `sort(C,dim,mode)` with `dim=1` and `mode='ascend'` to sort the elements of `C` in ascending order along the first array dimension (i.e., the rows). Typing

```
I = sort(C,1,'ascend')
```

yields

```
I =
     1     3    -1
     2     4     3
     6     6     3
     9     9     3
```

The function `sortrows(C,column)` with `column=2` sorts the rows of `C` according to the second column. Typing

```
J = sortrows(C,2)
```

yields

```
J =
     9     3    -1
     2     4     3
     6     6     3
     1     9     3
```

Array manipulation also includes the comparison of arrays, for example by checking whether elements in `A(i,j)` are also found in `B` using `ismember`. Typing

```
A, B

K = ismember(A,B)
```

yields

```
A =
     2     4     3
     9     3    -1

B =
     1     9     3
     6     6     3
```

www.it-ebooks.info

```
K =
     0     0     1
     1     1     0
```

The array `L(i,j)` is zero if `A(i,j)` is not in `B`, and one if `A(i,j)` is in `B`. We can also locate elements within `A` for which a statement is true. For example we can locate elements with values less than zero and replace them with `NaN`s by typing

```
L = A;
L(find(L<0)) = NaN
```

or, more briefly

```
L(L<0) = NaN
```

which yields

```
L =
     2     4     3
     9     3   NaN
```

This is very useful when working with digital elevation models, where values below sea level are not relevant. Alternatively, we can replace data voids other than `NaN`s such as `-32768`, which are often used with digital terrain models (Section 7.3 to 7.5). We can then determine which elements of an array are `NaN`s by typing

```
M = isnan(L)
```

which yields

```
M =
     0     0     0
     0     0     1
```

where `NaN`s are indicated by ones and non-`NaN` values are indicated by zeros. Which of the elements in array `A` are unique can be determined by typing

```
N = unique(A)
```

which yields

```
N =
    -1
     2
     3
     4
     9
```

www.it-ebooks.info

The value of `3` occurs twice in `A` and the number of elements in `N` is therefore one less than in `A`.

2.5 Data Structures and Classes of Objects

The default data type or *class* in MATLAB is *double precision* or `double`, which stores data in a 64-bit array of floating-point numbers. Such floating-point numbers are approximations of real numbers that allow a maximum range of values in a limited numbers of bits. A double-precision array allows the sign of a number to be stored (bit 63), together with the exponent (bits 62 to 52), and roughly 16 significant decimal digits (bits 51 to 0). Typing

```
clear

realmin('double')
realmax('double')
```

yields the smallest and largest positive floating-point number in double precision

```
ans =
  2.2251e-308

ans =
  1.7977e+308
```

The actual number of floating point numbers is therefore limited by the number of bits available, in contrast to real numbers. The difference between 1.0 and the next largest double-precision number can be calculated using the floating-point relative accuracy `eps` by typing

```
eps(1.0)
```

which yields

```
ans =
   2.2204e-16
```

The round-off error depends on the value of the real number; it is, for example, different for 5.0, as we can see by typing

```
eps(5.0)
```

which yields

```
ans =
   8.8818e-16
```

www.it-ebooks.info

For real numbers there is, by definition, no such gap between consecutive numbers. The use of a finite number of floating-point numbers is limited by the number of available bits due to the finite precision arithmetic of a computer. There are countless examples available with which to demonstrate this, but we will restrict ourselves to the simple example of the sine of π. Typing

```
sin(pi)
```

yields

```
ans =
   1.2246e-16
```

and not, as would be expected, zero. Since `pi` is only the nearest floating-point value to π, the sine of `pi` is not exactly zero but a value very close to zero.

Let us now look at some examples of arrays in order to familiarize ourselves with the different data types in MATLAB. For the first example we create a 3-by-4 array of random numbers with double precision by typing

```
clear

rng(0)
A = rand(3,4)
```

We use the function `rand` that generates uniformly distributed pseudorandom numbers within the open interval [0,1]. To obtain identical data values, we use `rng(0)` to reset the random number generator by using the integer 0 as *seed* (see Chapter 3 for more details on random number generators and types of distributions). Since we did not use a semicolon here we get the output

```
A =
    0.8147    0.9134    0.2785    0.9649
    0.9058    0.6324    0.5469    0.1576
    0.1270    0.0975    0.9575    0.9706
```

By default, the output is in a scaled fixed point format with 5 digits, e.g., 0.8147 for the `(1,1)` element of `A`. Typing

```
format long
```

switches to a fixed point format with 16 digits for double precision. Recalling `A` by typing

```
A
```

www.it-ebooks.info

yields the output

```
A =
  Columns 1 through 2
   0.814723686393179   0.913375856139019
   0.905791937075619   0.632359246225410
   0.126986816293506   0.097540404999410

  Columns 3 through 4
   0.278498218867048   0.964888535199277
   0.546881519204984   0.157613081677548
   0.957506835434298   0.970592781760616
```

which autowraps those lines that are longer than the width of the Command Window. The command `format` does not affect how the computations are carried out, i.e., the precision of the computation results remains unchanged. The precision is, however, affected by converting the data type from *double* to 32-bit *single precision*. Typing

```
B = single(A)
```

yields

```
B =
   0.8147237   0.9133759   0.2784982   0.9648885
   0.9057919   0.6323593   0.5468815   0.1576131
   0.1269868   0.0975404   0.9575068   0.9705928
```

Although we have switched to `format long`, only 8 digits are displayed. The command `whos` lists the variables `A` and `B` with information on their sizes or dimensions, number of bytes, and classes

```
Name      Size            Bytes  Class     Attributes
A         3x4                96  double
B         3x4                48  single
```

The default class `double` is used in all MATLAB operations in which the physical memory of the computer is not a limiting factor, whereas `single` is used when working with large data sets. The double-precision variable `A`, whose size is 3-by-4 elements, requires 3·4·64=768 bits or 768/8=96 bytes of memory, whereas `B` requires only 48 bytes and so has half the memory requirement of `A`. Introducing at least one complex number to `A` doubles the memory requirement since both real and imaginary parts are double precision, by default. Switching back to `format short` and typing

```
format short
A(1,3) = 4i + 3
```

yields

www.it-ebooks.info

```
A =
  Columns 1 through 2
   0.8147 + 0.0000i   0.9134 + 0.0000i
   0.9058 + 0.0000i   0.6324 + 0.0000i
   0.1270 + 0.0000i   0.0975 + 0.0000i

  Columns 3 through 4
   3.0000 + 4.0000i   0.9649 + 0.0000i
   0.5469 + 0.0000i   0.1576 + 0.0000i
   0.9575 + 0.0000i   0.9706 + 0.0000i
```

and the variable listing is now

```
Name      Size            Bytes  Class     Attributes
A         3x4               192  double    complex
B         3x4                48  single
```

indicating the class `double` and the attribute `complex`.

MATLAB also works with even smaller data types such as 1-bit, 8-bit and 16-bit data, in order to save memory. These data types are used to store digital elevation models or images (see Chapters 7 and 8). For example *m*-by-*n* pixel RGB true color images are usually stored as three-dimensional arrays, i.e., the three colors are represented by an *m*-by-*n*-by-3 array (see Chapter 8 for more details on RGB composites and true color images). Such multi-dimensional arrays can be generated by concatenating three two-dimensional arrays representing the *m*-by-*n* pixels of an image. First, we generate a 100-by-100 array of uniformly distributed random numbers in the range [0,1]. We then multiply the random numbers by 255 to get values between 0 and 255.

```
clear

rng(0)
I1 = 255 * rand(100,100);
I2 = 255 * rand(100,100);
I3 = 255 * rand(100,100);
```

The command `cat` concatenates the three two-dimensional arrays (8 bits each) into a three-dimensional array (3·8 bits=24 bits).

```
I = cat(3,I1,I2,I3);
```

Since RGB images are represented by integer values between 0 and 255 for each color, we convert the 64-bit double-precision values to unsigned 8-bit integers using `uint8` (Section 8.2). The function `uint8` rounds the values in `I` to the nearest integer. Any values that are outside the range [0,255] are assigned to the nearest endpoint (0 or 255).

www.it-ebooks.info

```
I = uint8(I);
```

Typing `whos` then yields

```
Name            Size                Bytes  Class     Attributes
I             100x100x3             30000  uint8
I1            100x100               80000  double
I2            100x100               80000  double
I3            100x100               80000  double
```

Since 8 bits can be used to store 256 different values, this data type can be used to store integer values between 0 and 255, whereas using `int8` to create signed 8-bit integers generates values between −128 and +127. The value of zero requires one bit and there is therefore no space left in which to store +128. Finally, `imshow` can be used to display the three-dimensional array as a true color image.

```
imshow(I)
```

We next introduce *structure arrays* as a MATLAB data type. Structure arrays are multi-dimensional arrays with elements accessed by textual field designators. These arrays are data containers that are particularly helpful in storing any kind of information about a sample in a single variable. As an example we can generate a structure array `sample_1` that includes the image array `I` defined in the previous example as well as other types of information about a sample, such as the name of the sampling location, the date of sampling, and geochemical measurements, stored in a 10-by-10 array.

```
sample_1.location = 'Plougasnou';
sample_1.date = date;
sample_1.image = I;
sample_1.geochemistry = rand(10,10);
```

The first layer of the structure array `sample_1` contains a character array, i.e., a two-dimensional array of the data type `char` containing a character string. We can create such an array by typing

```
location = 'Plougasnou';
```

We can list the size, class and attributes of a single variable such as `location` by typing

```
whos location
```

and learn from

```
Name          Size            Bytes  Class    Attributes
location      1x10               20  char
```

www.it-ebooks.info

that the size of this character array `location` corresponds to the number of characters in the word *Plougasnou*. Character arrays are 16-bit arrays, i.e., 2^{16}=65,536 different characters can be stored in such arrays. The character string `location` therefore requires 10·16=160 bits or 160/8=20 bytes of memory. In addition, the second layer `datum` in the structure array `sample_1` contains a character string generated by `date` that yields a string containing the current date in `dd-mm-yyyy` format. We access this particular layer in `sample_1` by typing

```
sample_1.date
```

which yields

```
ans =
   27-Jun-2014
```

as an example. The third layer of `sample_1` contains the image created in the previous example, while the fourth layer contains a 10-by-10 array of uniformly-distributed pseudorandom numbers. All layers of `sample_1` can be listed by typing

```
sample_1
```

resulting in the output

```
sample_1 =
        location: 'Plougasnou'
            date: '06-Oct-2009'
           image: [100x100x3 uint8]
    geochemistry: [10x10 double]
```

This represents a list of the layers `location`, `date`, `image` and `geochemistry` within the structure array `sample_1`. Some variables are listed in full, whereas larger data arrays are only represented by their size. In the list of the layers within the structure array `sample_1`, the array `image` is characterized by its size `100x100x3` and the class `uint8`. The variable `geochemistry` in the last layer of the structure array contains a 10-by-10 array of double-precision numbers. The command

```
whos sample_1
```

does not list the layers in `sample_1` but the name of the variable, the bytes and the class `struct` of the variable.

```
  Name            Size              Bytes  Class     Attributes
  sample_1        1x1               31546  struct
```

www.it-ebooks.info

MATLAB also has *cell arrays* as an alternative to structure arrays. Both classes or data types are very similar and are containers of different types and sizes of data. The most important difference between the two is that the containers of a structure array are *named fields*, whereas a cell array uses *numerically-indexed cells*. Structure arrays are often used in applications where the organization of the data is particularly important. Cell arrays are often used when processing large data sets in count-controlled loops (Section 2.7).

As an example of cell arrays we use the same data collection as in structure arrays, with the layers of the structure array as the cells in the cell array. The cell array is created by enclosing the location name *Plougasnou*, the date, the image `I` and the 10-by-10 array of uniformly-distributed pseudorandom numbers in curly brackets.

```
C = {'Plougasnou' date I rand(10,10)}
```

Typing

```
C
```

lists the contents of the cell array

```
C =
  Columns 1 through 2
    'Plougasnou'    '27-Jun-2014'
  Columns 3 through 4
    [100x100x3 uint8]    [10x10 double]
```

which contains the location name and date. The image and the array of random numbers are too large to be displayed in the Command Window, but the dimensions and class of the data are displayed instead. We access a particular cell in `C`, e.g., the cell 2, by typing

```
C{2}
```

which yields

```
ans =
27-Jun-2014
```

We can also access the other cells of the cell array in a similar manner.

2.6 Data Storage and Handling

This section deals with how to store, import, and export data with MATLAB. Many of the data formats typically used in earth sciences have to be converted

www.it-ebooks.info

before being analyzed with MATLAB. Alternatively, the software provides several import routines to read many binary data formats in earth sciences, such as those used to store digital elevation models and satellite data.

A computer generally stores data as *binary digits* or *bits*. A bit is analogous to a two-way switch with two states, on = 1 and off = 0. The bits are joined together to form larger groups, such as bytes consisting of 8 bits, in order to store more complex types of data. Such groups of bits are then used to encode data, e.g., numbers or characters. Unfortunately, different computer systems and software use different schemes for encoding data. For instance, the characters in the widely-used text processing software Microsoft Word differ from those in Apple Pages. Exchanging binary data is therefore difficult if the various users use different computer platforms and software. Binary data can be stored in relatively small files if both partners are using similar systems of data exchange. The transfer rate for binary data is generally faster than that for the exchange of other file formats.

Various formats for exchanging data have been developed during recent decades. The classic example for the establishment of a data format that can be used with different computer platforms and software is the American Standard Code for Information Interchange (ASCII) that was first published in 1963 by the American Standards Association (ASA). As a 7-bit code, ASCII consists of 2^7=128 characters (codes 0 to 127). Whereas ASCII-1963 was lacking lower-case letters, in the ASCII-1967 update lower-case letters as well as various control characters such as escape and line feed, and various symbols such as brackets and mathematical operators, were also included. Since then, a number of variants appeared in order to facilitate the exchange of text written in non-English languages, such as the expanded ASCII containing 255 codes, e.g., the Latin-1 encoding.

The simplest way to exchange data between a certain piece of software and MATLAB is using the ASCII format. Although the newer versions of MATLAB provide various import routines for file types such as Microsoft Excel binaries, most data arrive in the form of ASCII files. Consider a simple data set stored in a table such as

```
SampleID   Percent C     Percent S
101        0.3657        0.0636
102        0.2208        0.1135
103        0.5353        0.5191
104        0.5009        0.5216
105        0.5415        -999
106        0.501         -999
```

The first row contains the names of the variables and the columns provide the percentages of carbon and sulfur in each sample. The absurd value `-999`

www.it-ebooks.info

indicates missing data in the data set. Two things have to be changed to convert this table into MATLAB format. First, MATLAB uses `NaN` as the representation for *Not-a-Number* that can be used to mark missing data or gaps. Second, a percent sign, `%`, should be added at the beginning of the first line. The percent sign is used to indicate nonexecutable text within the body of a program. This text is normally used to include comments in the code.

```
%SampleID  Percent C     Percent S
101        0.3657        0.0636
102        0.2208        0.1135
103        0.5353        0.5191
104        0.5009        0.5216
105        0.5415        NaN
106        0.501         NaN
```

MATLAB will ignore any text appearing after the percent sign and continue processing on the next line. After editing this table in a text editor, such as the *MATLAB Editor*, it can be saved as ASCII text file *geochem.txt* in the current working directory (Fig. 2.2). The MATLAB workspace should first be cleared by typing

```
clear
```

after the prompt in the Command Window. MATLAB can now import the data from this file with the `load` command.

```
load geochem.txt
```

MATLAB then loads the contents of the file and assigns the array to a variable `geochem` specified by the filename *geochem.txt*. Typing

```
whos
```

yields

```
Name         Size            Bytes  Class     Attributes
geochem      6x3               144  double
```

The command `save` now allows workspace variables to be stored in a binary format.

```
save geochem_new.mat
```

MAT-files are double-precision binary files using *.mat* as extension. The advantage of these binary MAT-files is that they are independent of the computer platforms running different floating-point formats. The command

```
save geochem_new.mat geochem
```

www.it-ebooks.info

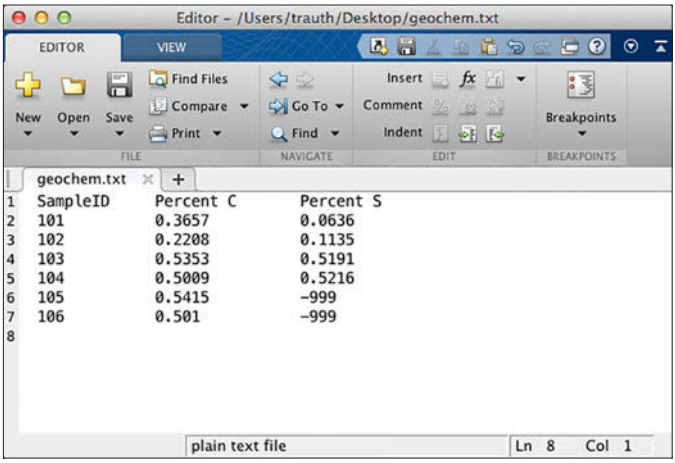


Fig. 2.2 Screenshot of MATLAB *Editor* showing the content of the file *geochem.txt*. The first line of the text needs to be commented by a percent sign at the beginning of the line, followed by the actual data array. The `-999` values need to be replaced by `NaN`s.

saves only the variable `geochem` instead of the entire workspace. The option `-ascii`, for example

```
save geochem_new.txt geochem -ascii
```

again saves the variable `geochem`, but in an ASCII file named *geochem_new.txt* in a floating-point format with 8 digits:

```
1.0100000e+02   3.6570000e-01   6.3600000e-02
1.0200000e+02   2.2080000e-01   1.1350000e-01
1.0300000e+02   5.3530000e-01   5.1910000e-01
1.0400000e+02   5.0090000e-01   5.2160000e-01
1.0500000e+02   5.4150000e-01             NaN
1.0600000e+02   5.0100000e-01             NaN
```

In contrast to the binary file *geochem_new.mat,* this ASCII file can be viewed and edited using the MATLAB Editor or any other text editor.

Such data files, especially those that are produced by electronic instruments, can look much more complicated than the example file *geochem.txt* with a single header line. In Chapters 7 and 8 we will read some of these complicated and extensive files, which are either binary or text files and usually have long headers describing the contents of the files. At this point, let us have a look at a variant of text files that contains not only one or more header lines but also unusual data types such as date and time, in a non-decimal format. We use

Movie 2.3

www.it-ebooks.info

the function `textscan` to perform this task. The MATLAB workspace should first be cleared by typing

```
clear
```

after the prompt in the Command Window. MATLAB can now import the data from the file *geochem.txt* using the `textscan` command.

```
fid = fopen('geochem.txt');
C = textscan(fid,'%u %f %f','Headerlines',1,'CollectOutput',1);
fclose(fid);
```

This script opens the file *geochem.txt* for *read only* access using `fopen` and defines the file identifier `fid`, which is then used to read the text from the file using `textscan` and to write it into the cell array `C`. The character string `%u %f %f` defines the conversion specifiers enclosed in single quotation marks, where `%u` stands for the 32-bit unsigned integer output class and `%f` stands for a 64-bit double-precision flointing-point number. The parameter `Headerlines` is set to 1, which means that a single header line is ignored while reading the file. If the parameter `CollectOutput` is 1 (i.e., is true), `textscan` concatenates output cells with the same data type into a single array. The function `fclose` closes the file defined by `fid`. The array `C` is a cell array, which is a data type with indexed containers called cells (see Section 2.5). The advantage of this data type is that it can store data of various types and sizes, such as character strings, double-precision numbers, and images in a single variable such as `C`. Typing

```
C
```

yields

```
C =
    [6x1 uint32]    [6x2 double]
```

indicating that `C` contains a 6-by-1 32-bit unsigned integer array, which is the sample ID, and a 6-by-1 double-precision array, which represents the percentages of carbon and sulfur in each sample. We can access the contents of the cells in `C` by typing

```
data1 = C{1}
data2 = C{2}
```

which yields

```
data1 =
    101
    102
```

www.it-ebooks.info

```
    103
    104
    105
    106

data2 =
    0.3657    0.0636
    0.2208    0.1135
    0.5353    0.5191
    0.5009    0.5216
    0.5415       NaN
    0.5010       NaN
```

We now concatenate the two cells into one double-precision array `data`. First, we have to change the class of `C{1}` into `double` or the class of the entire array `data` will be `uint32`. Typing

```
data(:,1) = double(C{1})
data(:,2:3) = C{2}
```

yields

```
  101.0000    0.3657    0.0636
  102.0000    0.2208    0.1135
  103.0000    0.5353    0.5191
  104.0000    0.5009    0.5216
  105.0000    0.5415       NaN
  106.0000    0.5010       NaN
```

The format of the data is as expected.

The next examples demonstrate how to read the file `geophys.txt`, which contains a single header line but also the date (in an *MM/DD/YY* format) and time (in an *HH:MM:SS.SS* format). We again use `textscan` to read the file,

```
clear

fid = fopen('geophys.txt');
data = textscan(fid,'%u %f %f %f %s %s','Headerlines',1);
fclose(fid);
```

where we skip the header, read the first column (the sample ID) as a 32-bit unsigned integer (`uint32`) with specifier `%u`, the next three columns *X*, *Y*, and *Z* as 64-bit double-precision floating-point numbers (`double`) with specifier `%f`, and then the date and time as character strings with specifier `%s`. We then convert the date and time to serial numbers, where a serial date number of 1 corresponds to *Jan-1-0000*. The year *0000* is merely a reference point and is not intended to be interpreted as a real year.

www.it-ebooks.info

```
data_date_serial = datenum(data{5});
data_time_serial = datenum(data{6});
```

Finally, we can convert the date and time serial numbers into a data and time array by typing

```
data_date = datevec(data_date_serial)
data_time = datevec(data_time_serial)
```

which yields

```
data_date =
      2013   11   18   0   0   0
      2013   11   18   0   0   0
      2013   11   18   0   0   0
      2013   11   18   0   0   0
      2013   11   18   0   0   0

data_time =
   1.0e+03 *
   2.0130   0.0010   0.0010   0.0100   0.0230   0.0091
   2.0130   0.0010   0.0010   0.0100   0.0230   0.0102
   2.0130   0.0010   0.0010   0.0100   0.0230   0.0504
   2.0130   0.0010   0.0010   0.0100   0.0240   0.0051
   2.0130   0.0010   0.0010   0.0100   0.0240   0.0233
```

The first three columns of the array `data_date` contain the year, month and day. The fourth to sixth columns of the array `data_time` contain the hour, minute and second.

We can also write data to a formatted text file using `fprintf`. As an example we again load the data from *geochem.txt* after we have commented out the first line and have replaced `-999` with `NaN`. Instead of using `load geochem.txt`, we can type

```
clear

data = load('geochem.txt');
```

to load the contents of the text file into a double-precision array `data`. We write the data to a new text file *geochem_formatted.txt* using `fprintf`. Since the function `fprintf` writes all elements of the array data to the file in column order we need to transpose the data before we save it.

```
data = data';
```

We first open the file using the permission `w` for *writing*, and discard the existing contents. We then write `data` to this file using the formatting operators `%u` for unsigned integers and `%6.4f` for fixed-point numbers with a field width of six characters and four digits after the decimal point. The

www.it-ebooks.info

control character `\n` denotes a new line after each line of three numbers.

```
fid = fopen('geochem_formatted.txt','w');
fprintf(fid,'%u %6.4f %6.4f\n',data);
fclose(fid);
```

We can view the contents of the file by typing

```
edit geochem_formatted.txt
```

which opens the file *geochem_formatted.txt*

```
101   0.3657   0.0636
102   0.2208   0.1135
103   0.5353   0.5191
104   0.5009   0.5216
105   0.5415      NaN
106   0.5010      NaN
```

in the MATLAB Editor. The format of the data is as expected.

2.7 Control Flow

Control flow in computer science helps to control the order in which computer code is evaluated. The most important kinds of control flow statements are count-controlled loops such as `for` loops and conditional statements such as `if-then` constructs. Since in this book we do not deal with the programming capabilities of MATLAB in any depth, the following introduction to the basics of control flow is rather brief and omits certain important aspects of efficient programming, such as the pre-allocation of memory prior to using `for` loops, and instructions on how the use of `for` loops can be avoided by vectorization of the MATLAB code. This introduction is instead limited to the two most important kinds of control flow statements: the aforementioned `for` loops and the `if-then` constructs. Readers interested in MATLAB as a programming environment are advised to read the more detailed chapters on control flow in the MATLAB documentation (MathWorks 2014a and c).

The `for` loops, as the first example of a MATLAB language statement, execute a series of commands between `for` and `end` a specified number of times. As an example we use such a loop to multiply the elements of an array `A` by `10`, round the result to the nearest integer, and store the result in `B`.

```
clear

rng(0)
A = rand(10,1)
for i = 1 : 10
    B(i,1) = round(10 * A(i));
```

www.it-ebooks.info

```
end
B
```

which yields

```
A =
    0.8147
    0.9058
    0.1270
    0.9134
    0.6324
    0.0975
    0.2785
    0.5469
    0.9575
    0.9649

B =
     8
     9
     1
     9
     6
     1
     3
     5
    10
    10
```

The result is as expected. We can expand the experiment by using a nested `for` loop to create a 2D array `B`.

```
rng(0)
A = rand(10,3)
for i = 1 : 10
    for j = 1 : 3
        B(i,j) = round(10 * A(i,j));
    end
end
B
```

which yields

```
A =
    0.8147    0.1576    0.6557
    0.9058    0.9706    0.0357
    0.1270    0.9572    0.8491
    0.9134    0.4854    0.9340
    0.6324    0.8003    0.6787
    0.0975    0.1419    0.7577
    0.2785    0.4218    0.7431
    0.5469    0.9157    0.3922
    0.9575    0.7922    0.6555
    0.9649    0.9595    0.1712
```

www.it-ebooks.info

```
B =
     8     2     7
     9    10     0
     1    10     8
     9     5     9
     6     8     7
     1     1     8
     3     4     7
     5     9     4
    10     8     7
    10    10     2
```

This book tries to make all of the recipes independent of the actual dimensions of the data. This is achieved by the consistent use of `size` and `length` to determine the size of the data instead of using fixed numbers such as the `30` and `3` in the above example (Section 2.4).

```
rng(0)
A = rand(10,3)
for i = 1 : size(A,1)
    for j = 1 : size(A,2)
        B(i,j) = round(10 * A(i,j));
    end
end
B
```

When working with larger data sets with many variables one might occasionally wish to automate array manipulations such as those described in Section 2.4. Let us assume, for example, that we want to replace all `NaN`s in all variables in the memory with `-999`. We first create a collection of four variables, each of which contains a single `NaN`.

```
clear

rng(0)
A = rand(3,3); A(2,1) = NaN
BC = rand(2,4); BC(2,2) = NaN
DE = rand(1,2); DE(1,1) = NaN
FG = rand(3,2); FG(2,2) = NaN
```

We list the variables in the workspace using `whos` and store this list in `variables`.

```
variables = who;
```

We then use a `for` loop to store the content of each variable in `v` using `eval` and then locate the `NaN`s in `v` using `isnan` (Section 2.4) and replace them with `-999`. The function `eval` executes a MATLAB expression stored in a text string. We assign the value of `v` to the variable in the base workspace and then clear the variables `i`, `v` and `variables`, which are no longer needed.

www.it-ebooks.info

```
for i = 1 : size(variables,1)
    v = eval(variables{i});
    v(isnan(v)==1) = -999;
    assignin('base',variables{i},v);
    eval(variables{i})
end

clear i v variables
```

Comparing the variables before and after the replacement of the `NaN`s with `-999` reveals that the script works well and that we have successfully manipulated our data.

The second important statements to control the flow of a script (apart from `for` loops) are `if-then` constructs, which evaluate an expression and then execute a group of instructions if the expression is true. As an example we compare the value of two scalars `A` and `B`.

```
clear

A = 1
B = 2
if A < B
    disp('A is less than B')
end
```

which yields

```
A is less than B
```

The script first evaluates whether `A` is less than `B` and, if it is, displays the message `A is less than B` in the Command Window. We can expand the `if-then` construct by introducing `else`, which provides an alternative statement if the expression is not true.

```
A = 1
B = 2
if A < B
    disp('A is less than B')
else
    disp('A is not less than B')
end
```

which yields

```
A is less than B
```

Alternatively, we can use `elseif` to introduce a second expression to be evaluated.

```
A = 1
```

www.it-ebooks.info

```
B = 2
if A < B
    disp('A is less than B')
elseif A >= B
    disp('A is not less than B')
end
```

The `for` loops and `if-then` constructs are extensively used in the following chapters of the book. For other aspects of programming, please refer to the MATLAB documentation (MathWorks 2014a and c).

2.8 Scripts and Functions

MATLAB is a powerful programming language. All files containing MATLAB code use *.m* as an extension and are therefore called *M-files*. These files contain ASCII text and can be edited using a standard text editor. However, the built-in Editor color-highlights various syntax elements such as comments in green, keywords such as *if*, *for* and *end* in blue, and character strings in pink. This syntax highlighting facilitates MATLAB coding.

MATLAB uses two types of M-files: scripts and functions. Whereas scripts are a series of commands that operate on data in the workspace, functions are true algorithms with input and output variables. The advantages and disadvantages of both types of M-file will now be illustrated by an example. We first start the Editor by typing

```
edit
```

This opens a new window named *untitled*. Next, we generate a simple MATLAB script by typing a series of commands to calculate the average of the elements of a data array `x`.

```
[m,n] = size(x);
if m == 1
   m = n;
end
sum(x)/m
```

The first line of the `if-then` construct yields the dimensions of the variable `x` using the command `size`. In our example `x` should be either a column vector, i.e., an array with a single column and dimensions `(m,1)`, or a row vector, i.e. an array with a single row and dimensions `(1,n)`. The `if` statement evaluates a logical expression and executes a group of commands if this expression is true. The `end` keyword terminates the last group of commands. In the example the `if-then` construct picks either `m` or `n` depending on whether `m==1` is false or true. Here, the double equal sign `'=='` makes element by element

www.it-ebooks.info

comparisons between the variables (or numbers) to the left and right of the equal signs and returns an array of the same size, made up of elements set to logical `1` where the relationship is true and to logical `0` where it is not true. In our example `m==1` returns `1` if `m` equals `1` and `0` if `m` equals any other value. The last line of the `if-then` construct computes the average by dividing the sum of elements by `m` or `n`. We do not use a semicolon here in order to allow the output of the result. We can now save our new M-file as *average.m* and type

```
clear

x = [3 6 2 -3 8];
```

in the Command Window to define an example array `x`. We then type

```
average
```

without the extension *.m* to run our script and obtain the average of the elements of the array `x` as output.

```
ans =
    3.2000
```

After typing

```
whos
```

we see that the workspace now contains

```
Name         Size            Bytes  Class     Attributes
ans          1x1                 8  double
m            1x1                 8  double
n            1x1                 8  double
x            1x5                40  double
```

The listed variables are the example array `x` and the output of the function `size`, `m` and `n`. The result of the operation is stored in the variable `ans`. Since the default variable `ans` might be overwritten during one of the succeeding operations, we need to define a different variable. Typing

```
a = average
```

however, results in the error message

```
??? Attempt to execute SCRIPT average as a function.
```

We can obviously not assign a variable to the output of a script. Moreover, all variables defined and used in the script appear in the workspace; in our example these are the variables `m` and `n`. Scripts contain sequences of

www.it-ebooks.info

commands that are applied to variables in the workspace. MATLAB functions, however, allow inputs and outputs to be defined. They do not automatically import variables from the workspace. To convert the above script into a function we need to introduce the following modifications (Fig. 2.3):

```
function y = average(x)
%AVERAGE    Average value.
%    AVERAGE(X) is the average of the elements in the array X.

% By Martin Trauth, June 27, 2014

[m,n] = size(x);
if m == 1
   m = n;
end
y = sum(x)/m;
```

The first line now contains the keyword `function`, the function name `average`, the input `x` and the output `y`. The next two lines contain comments, as indicated by the percent sign, separated by an empty line. The second comment line contains the author's name and the version of the M-file. The rest of the file contains the actual operations. The last line now defines the value of the output variable `y`, and this line is terminated by a semicolon to suppress the display of the result in the Command Window. Next we type

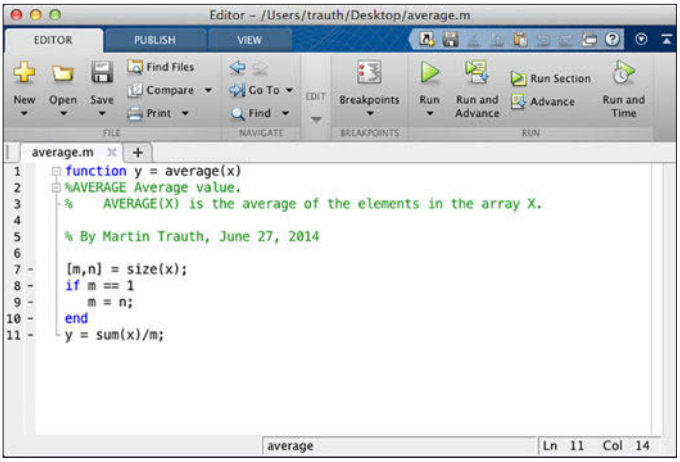


Fig. 2.3 Screenshot of the MATLAB *Editor* showing the function `average`. The function starts with a line containing the keyword `function`, the name of the function `average`, the input variable `x`, and the output variable `y`. The subsequent lines contain the output for `help average`, the copyright and version information, and also the actual MATLAB code for computing the average using this function.

www.it-ebooks.info

```
help average
```

which displays the first block of contiguous comment lines. The first executable statement (or blank line in our example) effectively ends the help section and therefore the output of *help*. Now we are independent of the variable names used in our function. The workspace can now be cleared and a new data vector defined.

```
clear

data = [3 6 2 -3 8];
```

Movie 2.4

Our function can then be run by the statement

```
result = average(data);
```

This clearly illustrates the advantages of functions compared to scripts. Typing

```
whos
```

results in

```
Name        Size            Bytes  Class     Attributes
data        1x5                40  double
result      1x1                 8  double
```

revealing that all variables used in the function do not appear in the workspace. Only the input and output as defined by the user are stored in the workspace. The M-files can therefore be applied to data as if they were real functions, whereas scripts contain sequences of commands that are applied to the variables in the workspace. If we want variables such as `m` and `n` to also appear in the memory they must be defined as *global* variables in both the function and the workspace, otherwise they are considered to be *local* variables. We therefore add one line to the function `average` with the command `global`:

```
function y = average(x)
%AVERAGE    Average value.
%    AVERAGE(X) is the average of the elements in the array X.

% By Martin Trauth, June 27, 2014

global m n
[m,n] = size(x);
if m == 1
   m = n;
end
y = sum(x)/m;
```

www.it-ebooks.info

We now type

```
global m n
```

in the Command Window. After running the function as described in the previous example we find the two variables `m` and `n` in the workspace. We have therefore transferred the variables `m` and `n` between the function `average` and the workspace.

2.9 Basic Visualization Tools

MATLAB provides numerous routines for displaying data as graphics. This section introduces the most important graphics functions. The graphics can be modified, printed and exported to be edited with graphics software other than MATLAB. The simplest function producing a graph of a variable `y` versus another variable `x` is `plot`. First, we define two one-dimensional arrays `x` and `y`, where `y` is the sine of `x`. The array `x` contains values between 0 and 2π with π/10 increments, whereas `y` is the element-by-element sine of `x`.

```
clear

x = 0 : pi/10 : 2*pi;
y = sin(x);
```

These two commands result in two one-dimensional arrays with 21 elements each, i.e., two 1-by-21 arrays. Since the two arrays `x` and `y` have the same length, we can use `plot` to produce a linear 2D graph of `y` against `x`.

```
plot(x,y)
```

This command opens a *Figure Window* named *Figure 1* with a gray background, an *x*-axis ranging from 0 to 7, a *y*-axis ranging from −1 to +1 and a blue line. We may wish to plot two different curves in a single plot, for example the sine and the cosine of `x` in different colors. The command

```
x = 0 : pi/10 : 2*pi;
y1 = sin(x);
y2 = cos(x);

plot(x,y1,'--',x,y2,'-')
```

creates a dashed blue line displaying the sine of `x` and a solid red line representing the cosine of this array (Fig. 2.4). If we create another plot, the window *Figure 1* will be cleared and a new graph displayed. The command `figure`, however, can be used to create a new figure object in a new window.

www.it-ebooks.info

```
plot(x,y1,'--')
figure
plot(x,y2,'-')
```

Instead of plotting both lines in one graph simultaneously, we can also plot the sine wave, hold the graph and then plot the second curve. The command `hold` is particularly important for displaying data while using different plot functions, for example if we wish to display the sine of `x` as a line plot and the cosine of `x` as a bar plot.

```
plot(x,y1,'r--')
hold on
bar(x,y2)
hold off
```

This command plots `y1` versus `x` as a dashed red line using `'r--'`, whereas `y2` versus `x` is shown as a group of blue vertical bars. Alternatively, we can plot both graphics in the same Figure Window but in different plots using `subplot`. The syntax `subplot(m,n,p)` divides the Figure Window into an m-by-n array of display regions and makes the `p`th display region active.

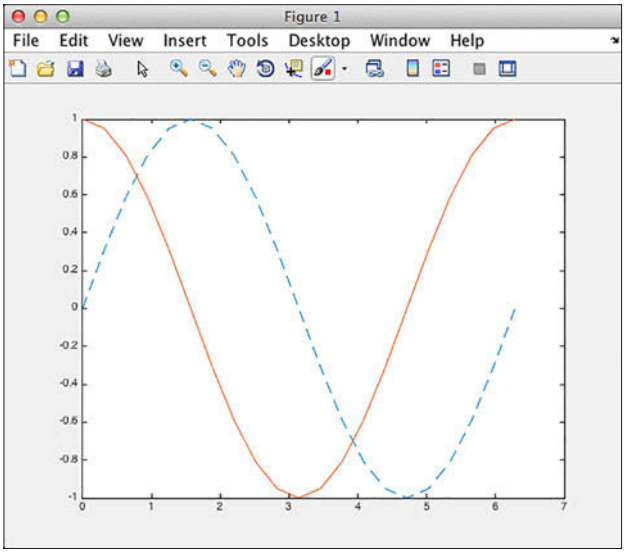


Fig. 2.4 Screenshot of the MATLAB *Figure Window* showing two curves in different colors and line types. The Figure Window allows editing of all elements of the graph after selecting *Edit Plot* from the *Tools* menu. Double clicking on the graphics elements opens an options window for modifying the appearance of the graphics. The graphics can be exported using *Save as* from the *File* menu. The command *Generate Code* from the *File* menu creates MATLAB code from an edited graph.

www.it-ebooks.info

```
subplot(2,1,1), plot(x,y1,'r--')
subplot(2,1,2), bar(x,y2)
```

For example the Figure Window is divided into two rows and one column. The 2D linear plot is displayed in the upper half of the Figure Window and the bar plot appears in the lower half. It is recommended that all Figure Windows be closed before proceeding to the next example. Subsequent plots would replace the graph in the lower display region only, or in other words, the last generated graph in a Figure Window. Alternatively, the command

```
clf
```

clears the current figure. This command can be used in larger MATLAB scripts after using the function `subplot` for multiple plots in a Figure Window.

An important modification to graphics is the scaling of the axis. By default, MATLAB uses axis limits close to the minima and maxima of the data. Using the command `axis`, however, allows the scale settings to be changed. The syntax for this command is simply `axis([xmin xmax ymin ymax])`. The command

```
plot(x,y1,'r--')
axis([0 pi -1 1])
```

sets the limits of the x-axis to 0 and π, whereas the limits of the y-axis are set to the default values -1 and $+1$. Important options of `axis` are

```
plot(x,y1,'r--')
axis square
```

which makes the x-axis and y-axis the same length, and

```
plot(x,y1,'r--')
axis equal
```

which makes the individual tick mark increments on the x-axis and y-axis the same length. The function `grid` adds a grid to the current plot, whereas the functions `title`, `xlabel` and `ylabel` allow a title to be defined and labels to be applied to the x- and y-axes.

```
plot(x,y1,'r--')
title('My first plot')
xlabel('x-axis')
ylabel('y-axis')
grid
```

These are a few examples how MATLAB functions can be used to edit the plot in the Command Window. More graphics functions will be introduced

www.it-ebooks.info

in the following chapters of this book.

2.10 Generating Code to Recreate Graphics

MATLAB supports various ways of editing all objects in a graph interactively using a computer mouse. First, the *Edit Plot* mode of the Figure Window needs to be activated by clicking on the arrow icon or by selecting *Edit Plot* from the *Tools* menu. The Figure Window also contains some other options, such as *Rotate 3D*, *Zoom* or *Insert Legend*. The various objects in a graph, however, are selected by double-clicking on the specific component, which opens the *Property Editor*. The Property Editor allows changes to be made to many features (or properties) of the graph such as axes, lines, patches and text objects.

The *Generate Code* option enables us to automatically generate the MATLAB code of a figure to recreate a similar graph with different data. We use a simple plot to illustrate the use of the Property Editor and the Generate Code option to recreate a graph.

```
clear

x = 0 : pi/10 : 2*pi;
y1 = sin(x);
plot(x,y1)
```

The default layout of the graph is that of Figure 2.4. Clicking on the arrow icon in the *Figure Toolbar* enables the Edit Plot mode. The selection handles of the graph appear, identifying the objects that are activated. Double-clicking an object in a graph opens the *Property Editor*.

As an example we can use the Property Editor to change various properties of the graph. Double-clicking the gray background of the Figure Window gives access to properties such as Figure Name, the Colormap used in the figure, and the Figure Color. We can change this color to light blue represented by the light blue square in the 4th row and 8rd column of the color chart. Moving the mouse over this square displays the RGB color code [0.68 0.92 1] (see Chapter 8 for more details on RGB colors). Activating the blue line in the graph allows us to change the line thickness to 2.0 and select a 15-point square marker. We can deactivate the Edit Plot mode of the Figure Window by clicking on the arrow icon in the Figure Toolbar.

After having made all necessary changes to the graph, the corresponding commands can even be exported by selecting Generate Code from the File menu of the Figure Window. The generated code displays in the MATLAB Editor.

www.it-ebooks.info

```
function createfigure(X1, Y1)
%CREATEFIGURE(X1, Y1)
%  X1:  vector of x data
%  Y1:  vector of y data

%  Auto-generated by MATLAB on 27-Jun-2014 13:28:13

% Create figure
figure1 = figure('Color',[0.68 0.92 1]);

% Create axes
axes1 = axes('Parent',figure1,'ColorOrderIndex',2);
box(axes1,'on');
hold(axes1,'on');

% Create plot
plot(X1,Y1,'MarkerSize',15,'Marker','square','LineWidth',2);
```

We can then rename the function `createfigure` to `mygraph` and save the file as *mygraph.m*.

```
function mygraph(X1, Y1)
%MYGRAPH(X1,Y1)
%  X1:  vector of x data
%  Y1:  vector of y data
(cont'd)
```

The automatically-generated graphics function illustrates how graphics are organized in MATLAB. The function `figure` first opens a Figure Window. Using `axes` then establishes a coordinate system, and using `plot` draws the actual line object. The Figure section in the function reminds us that the light-blue background color of the Figure Window is represented by the RGB color coding `[0.68 0.92 1]`. The Plot section reveals the square marker symbol used and the line width of 2 points.

The newly-created function `mygraph` can now be used to plot a different data set. We use the above example and

```
clear

x = 0 : pi/10 : 2*pi;
y2 = cos(x);
mygraph(x,y2)
```

The figure shows a new plot with the same layout as the previous plot. The *Generate Code* function of MATLAB can therefore be used to create templates for graphics that can be used to generate plots of multiple data sets using the same layout.

Movie 2.5

Even though MATLAB provides abundant editing facilities and the *Generate Code* function even allows the generation of complex templates

www.it-ebooks.info

for graphics, a more practical way to modify a graph for presentations or publications is to export the figure and import it into a different software such as CorelDraw or Adobe Illustrator. MATLAB graphics are exported by selecting the command *Save as* from the File menu or by using the command `print`. This function exports the graphics, either as a raster image (e.g., JPEG or GIF) or as a vector file (e.g., EPS or PDF), into the working directory (see Chapter 8 for more details on graphic file formats). In practice, the user should check the various combinations of export file formats and the graphics software used for final editing of the graphics. Readers interested in advanced visualization techniques with MATLAB are directed to the sister book *MATLAB and Design Recipes for Earth Sciences* (Trauth and Sillmann 2012).

2.11 Publishing M-Files

Another useful feature of the software is the option to publish reports on MATLAB projects in various file formats such as HTML, XML, LaTeX and many others. This feature enables you to share your results with colleagues who may or may not have the MATLAB software. The published code includes formatted commentary on the code, the actual MATLAB code, and all results of running the code including the output to the Command Window and all graphics created or modified by the code. To illustrate the use of the publishing feature we create a simple example of a commented MATLAB code to compute the sine and cosine of a time vector and display the results as two separate figures.

We start the Editor by typing `edit` in the Command Window, which opens a new window named `untitled`. An M-file to be published starts with a document title at the top of the file, followed by some comments that describe the contents and the version of the script. The subsequent contents of the file include sections of MATLAB code and comments, separated by the double percent signs `%%`. Whereas single percent signs `%` are known (from Section 2.8) to initiate comments in MATLAB, we now use double percent signs `%%` that indicate the start of new code sections in the Editor. The *code sections* feature, previously also known as *code cells* or *cell mode*, is a feature in MATLAB that enables you to evaluate blocks of commands called *sections* by using the buttons *Run*, *Run and Advance*, *Run Section*, *Advance*, and *Run and Time* on the Editor Toolstrip to evaluate either the entire script or parts of the script.

```
%% Example for Publishing M-Files
% This M-file illustrates the use of the publishing
% feature of MATLAB.
```

www.it-ebooks.info

```
% By Martin Trauth, June 27, 2014

%% Sine Wave
% We define a time vector t and compute the sine y1 of t.
% The results are displayed as linear 2D graph y1 against x.
x = 0 : pi/10 : 2*pi;
y1 = sin(x);
plot(x,y1)
title('My first plot')
xlabel('x-axis')
ylabel('y-axis')

%% Cosine Wave
% Now we compute the cosine y2 of the same time vector and
% display the results.
y2 = sin(x);
plot(x,y2)
title('My first plot')
xlabel('x-axis')
ylabel('y-axis')

%%
% The last comment is separated by the double percent sign
% without text. This creates a comment in a separate cell
% without a subheader.
```

We save the M-file as *myproject.m* and click the *Publish* button in the Publish Toolstrip. The entire script is now evaluated and the Figure Windows pop up while the script is running. Finally, a window opens up that shows the contents of the published M-file. The document title and subheaders are shown in a red font whereas the comments are in black fonts. The file includes a list of contents with jump links to proceed to the chapters of the file. The MATLAB commands are displayed on gray backgrounds but the graphics are embedded in the file without the gray default background of Figure Windows. The resulting HTML file can be easily included on a course or project webpage. Alternatively, the HTML file and included graphics can be saved as a PDF-file and shared with students or colleagues.

Movie 2.6

2.12 Creating Graphical User Interfaces

Almost all the methods of data analysis presented in this book are in the form of MATLAB scripts, i.e., series of commands that operate on data in the workspace (Section 2.8). Only in a few cases are the algorithms implemented in functions such as `canc` for adaptive filtering (Section 6.8) or `minput` for digitizing from the screen (Section 8.7). The MATLAB commands provided by The MathWorks, Inc., however, are mostly functions, i.e., algorithms with input and output variables. The most convenient variants

www.it-ebooks.info

of these functions are those with a *graphical user interface* (GUI). A GUI in MATLAB is a graphical display in one or more windows containing controls (or components) that enable the user to perform interactive tasks without typing commands in the Command Window or writing a script in the Editor. These components include pull-down menus, push buttons, sliders, text input fields and more. The GUI can read and write data files as well as performing many types of computation and displaying the results in graphics.

The manual entitled *MATLAB Creating Graphical User Interfaces* (MathWorks 2014b) provides a comprehensive guide to the creation of GUIs with MATLAB. Within this manual, however, the section on *Create a Simple GUIDE GUI* demonstrates a rather complex example with many interactive elements instead of providing the simplest possible example of a GUI. The following text therefore provides a very simple example of a GUI that computes and displays a Gaussian function for a mean and a standard deviation that can be defined by the user. Creating such a simple GUI with MATLAB requires two steps: the first step involves designing the layout of the GUI, and the second step involves adding functions to the components of the GUI. The best way to create a graphical user interface with MATLAB is using the *GUI Design Environment* (GUIDE). We start GUIDE by typing

```
guide
```

in the Command Window. Calling GUIDE opens the *GUIDE Quick Start* dialog where we can choose to open a previously created GUI or create a new one from a template. From the dialog we choose the GUIDE template *Blank GUI (Default)* and click *OK*, after which the *GUIDE Layout Editor* starts. First, we enable *Show names in component palette* in the *GUIDE Preferences* under the *File* menu and click *OK*. Second, we select *Grid and Rulers* from the *Tools* menu and enable *Show rulers*. The *GUIDE Layout Editor* displays an empty layout with dimensions of 670-by-388 pixels. We resize the layout to 500-by-300 pixels by clicking and dragging the lower right corner of the GUI.

Next, we place components such as static text, edit text, and axes onto the GUI by choosing the corresponding controls from the component palette. In our example we place two *Edit Text* areas on the left side of the GUI, along with a *Static Text* area containing the title *Mean*, with *Standard Deviation* above it. Double clicking the static text areas, the *Property Inspector* comes up in which we can modify the properties of the components. We change the *String* of the static text areas to *Mean* and *Standard Deviation*. We can also change other properties, such as the *FontName*, *FontSize*, *BackgroundColor*, and *HorizontalAlignment* of the text. Instead of the default *Edit Text* content

www.it-ebooks.info

of the edit text areas we choose 0 for the mean and 1 for the standard deviation text area. We then place an axis with dimensions of 250-by-200 pixels to the right of the GUI. Next, we save and activate the GUI by selecting *Run* from the *Tools* menu. GUIDE displays a dialog box with the question *Activating will save changes ...?*, where we click *Yes*. In the following *Save As* dialog box, we define a *FIG*-file name such as *gaussiantool.fig*.

GUIDE then saves this figure file together with the corresponding MATLAB code in a second file named *gaussiantool.m*. Furthermore, the MATLAB code is opened in the Editor and the default GUI is opened in a Figure Window with no menu or toolbar (Fig. 2.5). As we can see, GUIDE has automatically programmed the code of our GUI layout, including an initialization code at the beginning of the file that we should not edit. This code is included in the main routine named `gaussiantool`. The file also contains other functions called by `gaussiantool`, for instance the function `gaussiantool_Opening_Fcn` (executed before `gaussiantool` is made visible), `gaussiantool_OutputFnc` (sending output to the command line, not used here), `edit1_CreateFcn` and `edit2_CreateFcn` (initializing the edit text areas when they are created), and `edit1_Callback` and `edit2_Callback` (accepting text input and returning this input either as text or as a double-precision number).

We now add code to our GUI `gaussiantool`. First, we add initial values for the global variables `mmean` and `mstd` in the opening function `gaussiantool_Opening_Fcn` by adding the following lines after the last comment line marked by `%` in the first column:

```
global mmean mstd
mmean = 0;
mstd = 1;
```

The two variables must be global because they are used in the callbacks that we edit next (as in Section 2.8). The first of these callbacks `edit1_Callback` gets three more lines of code after the last comment line:

```
global mmean
mmean = str2double(get(hObject,'String'));
calculating_gaussian(hObject, eventdata, handles)
```

Movie 2.7

The first line defines the global variable `mmean`, which is then obtained by converting the text input into double precision with `str2double` in the second line. The function `edit1_Callback` then calls the function `calculating_gaussian`, which is a new function at the end of the file. This function computes and displays the Gaussian function with a mean value of `mmean` and a standard deviation of `mstd`.

www.it-ebooks.info

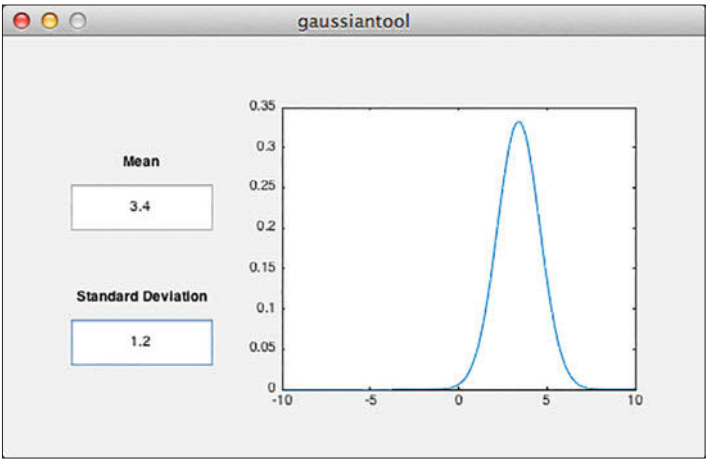


Fig. 2.5 Screenshot of the graphical user interface (GUI) `gaussiantool` for plotting a Gaussian function with a given mean and standard deviation. The GUI allows the values of the mean and standard deviation to be changed in order to update the graphics on the right. The GUI has been created using the MATLAB *GUI Design Environment* (GUIDE).

```
function calculating_gaussian(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global mmean mstd
x = -10 : 0.1 : 10;
y = normpdf(x, mmean, mstd);
plot(x,y)
```

The second callback `edit2_Callback` picks the value of the standard deviation `mstd` from the second Edit Text area, which is then also used by the function `calculating_gaussian`.

```
global mstd
mstd = str2double(get(hObject,'String'));
calculating_gaussian(hObject, eventdata, handles)
```

After saving the file *gaussiantool.m* we can run the new GUI by typing

```
gaussiantool
```

in the Command Window. The GUI starts where we can change the values of the mean and the standard deviation, then press return. The plot on the right is updated with each press of the return key. Using

```
edit gaussiantool
guide gaussiantool
```

www.it-ebooks.info

we can open the GUI code and Figure Window for further edits. Such GUIs allow a very direct and intuitive handling of functions in MATLAB that can also include animations such as the one used in `canctool` (Section 6.8), and the display of an audio-video signal. On the other hand, however, GUIs always require an interaction with the user who needs to click push buttons, move sliders and edit text input fields while the data is being analyzed. The automatic processing of large quantities of data is therefore usually carried out using scripts and functions with no graphical user interface.

Recommended Reading

Attaway S (2013) MATLAB: A Practical Introduction to Programming and Problem Solving. Elsevier, New York

Etter DM, Kuncicky DC, Moore H (2014) Introduction to MATLAB. Prentice Hall, New Jersey

Gilat A (2010) MATLAB: An Introduction with Applications. John Wiley & Sons, New York

Hanselman DC, Littlefield BL (2012) Mastering MATLAB 8. Prentice Hall, New Jersey

MathWorks (2014a) MATLAB Primer. The MathWorks, Inc., Natick, MA

MathWorks (2014b) MATLAB Creating Graphical User Interfaces. The MathWorks, Inc., Natick, MA

MathWorks (2014c) MATLAB Programming Fundamentals. The MathWorks, Inc., Natick, MA

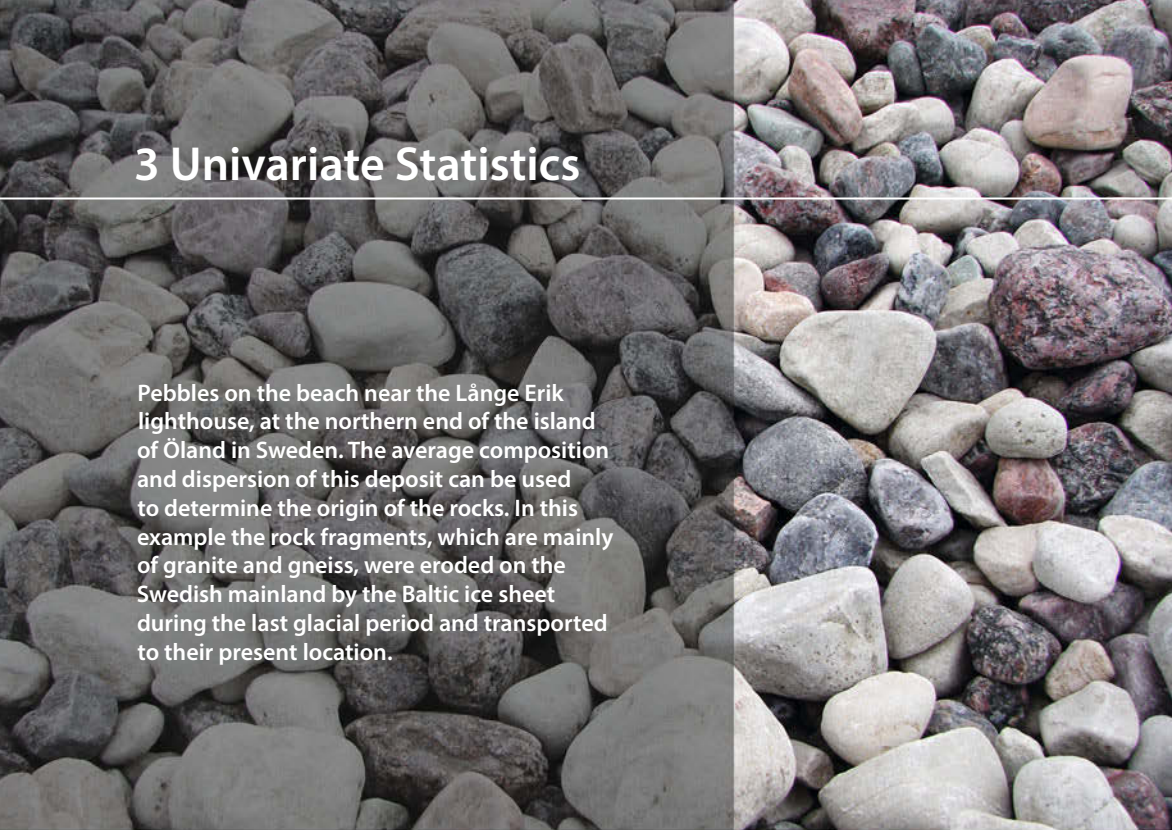
Palm WJ (2010) Introduction to MATLAB 7 for Engineers. McGraw-Hill, New York

Quarteroni A, Saleri F, Gervasio P (2014) Scientific Computing with MATLAB and Octave – 4th Edition. Springer, Berlin Heidelberg New York

Trauth MH, Sillmann E (2012) MATLAB and Design Recipes for Earth Sciences. Springer, Berlin Heidelberg New York

www.it-ebooks.info

3 Univariate Statistics



Pebbles on the beach near the Långe Erik lighthouse, at the northern end of the island of Öland in Sweden. The average composition and dispersion of this deposit can be used to determine the origin of the rocks. In this example the rock fragments, which are mainly of granite and gneiss, were eroded on the Swedish mainland by the Baltic ice sheet during the last glacial period and transported to their present location.

3.1 Introduction

The statistical properties of a single parameter are investigated by means of univariate analysis. Such a parameter could, for example, be the organic carbon content of deep-sea sediments, the sizes of grains in a sandstone layer, or the ages of sanidine crystals in a volcanic ash. Both the number and the size of *samples* that we collect from a larger *population* are often limited by financial and logistical constraints. The methods of univariate statistics assist us to draw from the sample conclusions that apply to the population as a whole. For univariate analysis we use the Statistics Toolbox (MathWorks 2014), which contains all the necessary routines.

We first need to describe the characteristics of the sample using statistical parameters, and to compute an *empirical distribution* (*descriptive statistics*) (Sections 3.2 and 3.3). A brief introduction is provided to the most important statistical parameters (such as the *measures of central tendency and dispersion*), followed by MATLAB examples. We then select a *theoretical distribution* that shows similar characteristics to the empirical distribution (Sections 3.4 and 3.5). A suite of theoretical distributions is introduced and their potential applications outlined prior to using MATLAB tools

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_3

www.it-ebooks.info

to explore these distributions. We then try to draw conclusions from the sample that can be applied to the larger population of interest (*hypothesis testing*). Sections 3.6 to 3.12 introduce the most important statistical tests for applications in earth sciences. The final section in this chapter (Section 3.13) introduces methods used to fit distributions to our own data sets.

3.2 Empirical Distributions

Let us assume that we have collected a number of measurements x_i from a specific object. The collection of data, or sample, as a subset of the population of interest, can be written as a vector x, or one-dimensional array

$$x = (x_1, x_2, \ldots, x_N)$$

containing a total of N observations. The vector x may contain a large number of data points and it may consequently be difficult to understand its properties. Descriptive statistics are therefore often used to summarize the characteristics of the data. The statistical properties of the data set may be used to define an empirical distribution, which can then be compared to a theoretical one.

The most straightforward way of investigating the sample characteristics is to display the data in a graphical form. Plotting all of the data points along a single axis does not reveal a great deal of information about the data set. However, the density of the points along the scale does provide some information about the characteristics of the data. A widely-used graphical display of univariate data is the *histogram* (Fig. 3.1). A histogram is a bar plot of a frequency distribution that is organized in intervals or *classes*. Such histogram plots provide valuable information on the characteristics of the data, such as the *central tendency*, the *dispersion* and the *general shape* of the distribution. However, quantitative measures provide a more accurate way of describing the data set than the graphical form. In purely quantitative terms, the *mean* and the *median* define the central tendency of the data set, while the data dispersion is expressed in terms of the *range* and the *standard deviation*.

Measures of Central Tendency

Parameters of central tendency or location represent the most important measures for characterizing an empirical distribution (Fig. 3.2). These values help locate the data on a linear scale. They represent a typical or best value

www.it-ebooks.info

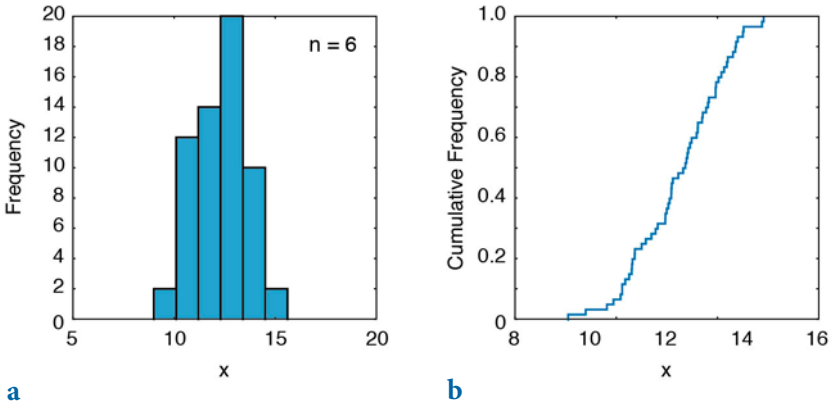


Fig. 3.1 Graphical representation of an empirical frequency distribution. **a** In a *histogram*, the frequencies are organized in *nbin* classes and plotted as a bar plot. **b** The *cumulative distribution plot* of a frequency distribution displays the totals of all observations lower than and equal to a certain value. This plot is normalized to a total number of observations of one.

that describes the data. The most popular indicator of central tendency is the *arithmetic mean*, which is the sum of all data points divided by the number of observations:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

The arithmetic mean can also be called the mean or the average of a univariate data set. The sample mean is used as an estimate of the population mean μ for the underlying theoretical distribution. The arithmetic mean is, however, sensitive to outliers, i.e., extreme values that may be very different from the majority of the data, and the *median* is therefore often used as an alternative measure of central tendency. The median is the x-value that is in the middle of the data set, i.e., 50% of the observations are smaller than the median and 50% are larger. The median of a data set sorted in ascending order is defined as

$$\tilde{x} = x_{(N+1)/2}$$

if N is odd and

$$\tilde{x} = \left(x_{(N/2)} + x_{(N/2)+1}\right)/2$$

www.it-ebooks.info

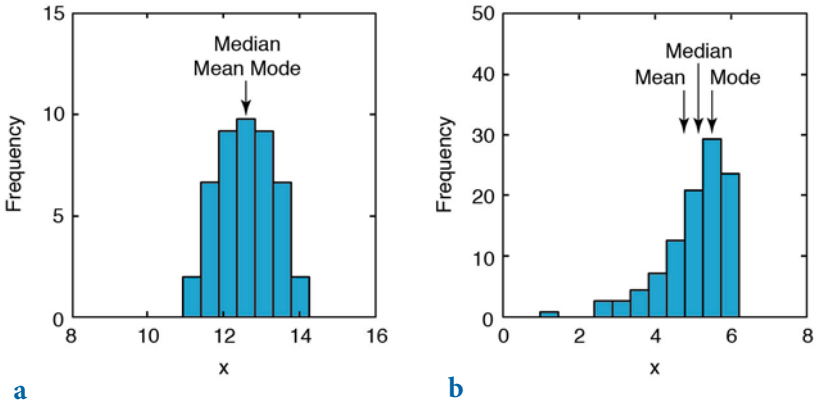


Fig. 3.2 Measures of *central tendency*. **a** In an unimodal symmetric distribution, the mean, the median and the mode are identical. **b** In a skewed distribution, the median lies between the mean and the mode. The mean is highly sensitive to outliers, whereas the median and the mode are little influenced by extremely high and low values.

if N is even. Although outliers also affect the median, their absolute values do not influence it. *Quantiles* are a more general way of dividing the data sample into groups containing equal numbers of observations. For example the three *quartiles* divide the data into four groups, the four *quintiles* divide the observations in five groups and the 99 *percentiles* define one hundred groups.

The third important measure for central tendency is the *mode*. The mode is the most frequent x-value or – if the data are grouped in classes – the center of the class with the largest number of observations. The data set has no mode if there are no values that appear more frequently than any of the other values. Frequency distributions with a single mode are called *unimodal*, but there may also be two modes (*bimodal*), three modes (*trimodal*) or four or more modes (*multimodal*) (Fig. 3.3).

The mean, median and mode are used when several quantities add together to produce a total, whereas the *geometric mean* is often used if these quantities are multiplied. Let us assume that the population of an organism increases by 10% in the first year, 25% in the second year, and then 60% in the last year. The average rate of increase is not the arithmetic mean, since the original number of individuals has increased by a factor (not a sum) of 1.1 after one year, 1.25 after the second year, and 1.6 after the third year. The average growth of the population is therefore calculated by the geometric mean:

www.it-ebooks.info

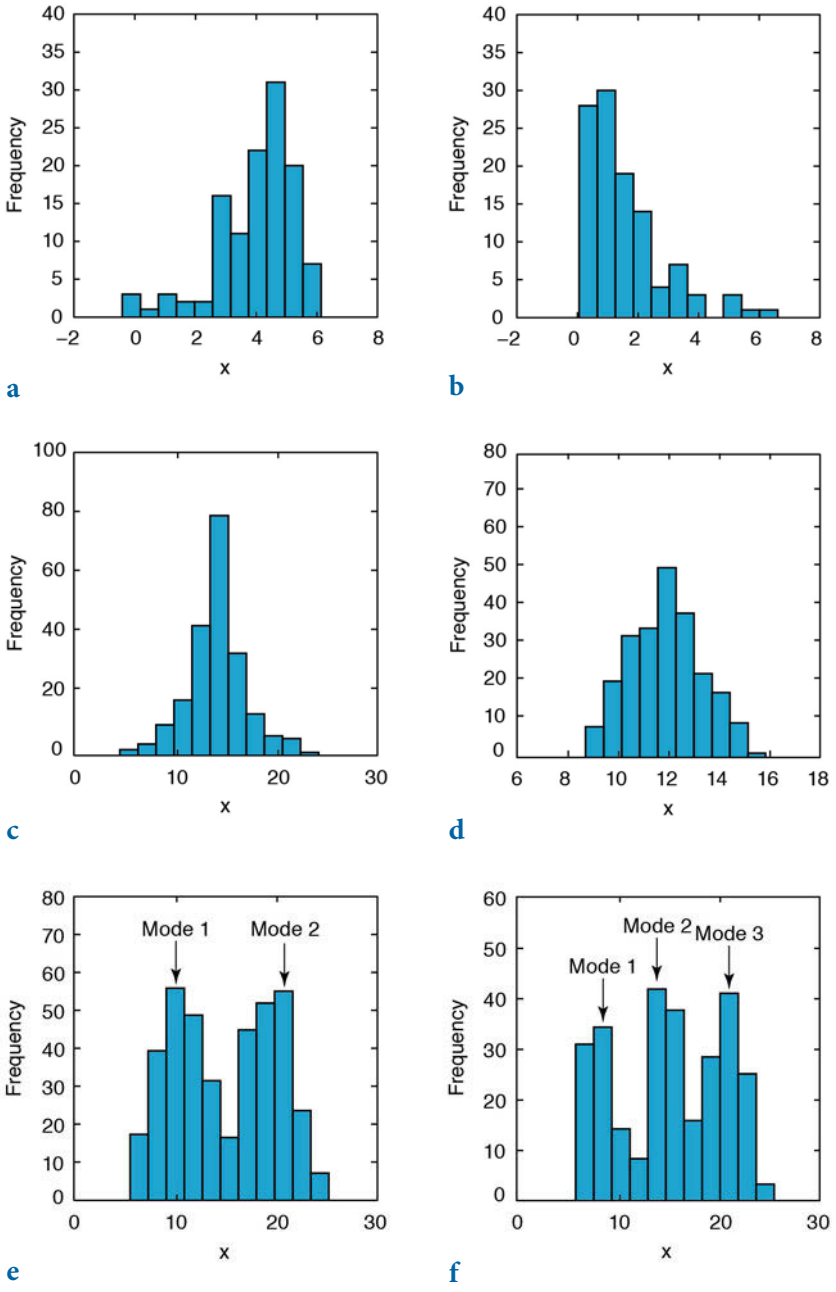


Fig. 3.3 *Dispersion* and *shape* of a distribution. **a**-**b**, Unimodal distributions showing a negative or positive skew. **c**-**d**, Distributions showing a high or low kurtosis. **e**-**f**, Bimodal and trimodal distributions showing two or three modes.

www.it-ebooks.info

$$\bar{x}_G = \left(x_1 \cdot x_2 \cdot \ldots \cdot x_N\right)^{1/N}$$

The average growth of these values is 1.3006 suggesting an approximate per annum growth in the population of 30%. The arithmetic mean would result in an erroneous value of 1.3167 or approximately 32% annual growth. The geometric mean is also a useful measure of central tendency for skewed or log-normally distributed data, in which the logarithms of the observations follow a Gaussian or normal distribution. The geometric mean, however, is not used for data sets containing negative values. Finally, the *harmonic mean*

$$\bar{x}_H = N / \left(\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_N}\right)$$

is also used to derive a mean value for asymmetric or log-normally distributed data, as is the geometric mean, but neither is robust to outliers. The harmonic mean is a better average when the numbers are defined in relation to a particular unit. The commonly quoted example is for averaging velocities. The harmonic mean is also used to calculate the mean of sample sizes.

Measures of Dispersion

Another important property of a distribution is the dispersion. Some of the parameters that can be used to quantify dispersion are illustrated in Figure 3.3. The simplest way to describe the dispersion of a data set is by the *range*, which is the difference between the highest and lowest value in the data set, given by

$$\Delta x = x_{\max} - x_{\min}$$

Since the range is defined by the two extreme data points it is very susceptible to outliers and hence it is not a reliable measure of dispersion in most cases. Using the interquartile range of the data, i.e., the middle 50% of the data, attempts to overcome this problem.

A more useful measure for dispersion is the *standard deviation*.

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

www.it-ebooks.info

The standard deviation is the average deviation of each data point from the mean. The standard deviation of an empirical distribution is often used as an estimate of the population standard deviation σ. The formula for the population standard deviation uses N instead of N–1 as the denominator. The sample standard deviation s is computed with N–1 instead of N since it uses the sample mean instead of the unknown population mean. The sample mean, however, is computed from the data x_i, which reduces the number of degrees of freedom by one. The *degrees of freedom* are the number of values in a distribution that are free to be varied. Dividing the average deviation of the data from the mean by N would therefore underestimate the population standard deviation σ.

The *variance* is the third important measure of dispersion. The variance is simply the square of the standard deviation.

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2$$

Although the variance has the disadvantage of not having the same dimensions as the original data, it is extensively used in many applications instead of the standard deviation.

In addition, both *skewness* and *kurtosis* can be used to describe the shape of a frequency distribution (Fig. 3.3). Skewness is a measure of the asymmetry of the tails of a distribution. The most popular way to compute the asymmetry of a distribution is by Pearson's mode skewness:

skewness = (*mean* – *mode*) / *standard deviation*

A negative skew indicates that the distribution is spread out more to the left of the mean value, assuming values increasing towards the right along the axis. The sample mean is in this case smaller than the mode. Distributions with positive skewness have large tails that extend towards the right. The skewness of the symmetric normal distribution is zero. Although Pearson's measure is a useful one, the following formula by Fisher for calculating the skewness is often used instead, including in the relevant MATLAB function.

$$skewness = \sum_{i=1}^{N}\frac{(x_i - \overline{x})^3}{s^3}$$

The second important measure for the shape of a distribution is the *kurtosis*. Again, numerous formulas to compute the kurtosis are available. MATLAB

www.it-ebooks.info

uses the following formula:

$$kurtosis = \sum_{i=1}^{N} \frac{(x_i - \bar{x})^4}{s^4}$$

The kurtosis is a measure of whether the data are peaked or flat relative to a normal distribution. A high kurtosis indicates that the distribution has a distinct peak near the mean, whereas a distribution characterized by a low kurtosis shows a flat top near the mean and broad tails. Higher peakedness in a distribution results from rare extreme deviations, whereas a low kurtosis is caused by frequent moderate deviations. A normal distribution has a kurtosis of three, and some definitions of kurtosis therefore subtract three from the above term in order to set the kurtosis of the normal distribution to zero.

3.3 Examples of Empirical Distributions

As an example we can analyze the data contained in the file *organicmatter_one.txt*. This file contains the organic carbon content C_{org} of lake sediments in weight percentage (wt%). In order to load the data, we type

```
clear

corg = load('organicmatter_one.txt');
```

The data file contains 60 measurements of C_{org} that can be displayed by

```
plot(corg,zeros(1,length(corg)),'o')
```

This graph shows some of the characteristics of the data. The organic carbon content of the samples ranges between 9 and 15 wt%, with most of the data clustering between 12 and 13 wt%. Values below 10 and above 14 are rare. While this kind of representation of the data undoubtedly has its advantages, histograms are a much more convenient way to display univariate data (Fig. 3.1). Histograms divide the range of the data into `nbin` equal intervals (also called bins or classes), count the number of observations `n` in each bin, and display the frequency distribution of observations as a bar plot. The bins are either defined by their edges `e` or their centers `v`. There is no fixed rule for the correct number of bins; the most suitable number will depend on the application (for example the statistical method) that is used. Ideally, the number of bins `n` should lie between 5 and 15; it should closely reflect the underlying distribution and should not result in any empty bins, i.e.,

www.it-ebooks.info

classes with no counts. In practice, the square root of the total number of observations `length(corg)`, rounded to the nearest integer using `round`, is often used as the number of bins. In our example `nbin` can be calculated using

```
nbin = round(sqrt(length(corg)))
```

which yields

```
nbin =
     8
```

The eight bin centers `v` are calculated by typing

```
vmin = min(corg) + 0.5*range(corg)/nbin;
vmax = max(corg) - 0.5*range(corg)/nbin;
vwth = range(corg)/nbin;
v = vmin : vwth : vmax;
```

which yields

```
v =
  Columns 1 through 4
    9.7383   10.3814   11.0245   11.6676
  Columns 5 through 8
   12.3107   12.9537   13.5968   14.2399
```

The smallest bin center `vmin` equals the minimum value in the data `corg` plus half of the range of the data `corg`, divided by the number of bins `nbin`. Similarly, the largest bin center `vmin` equals the maximum value of the data `corg` minus half of the range of the data `corg`, divided by `nbin`. The bin width `ewth` is calculated by dividing the range of the data `corg` by `nbin`. The corresponding nine bin edges `e` are calculated using

```
emin = min(corg);
emax = max(corg);
ewth = range(corg)/nbin;
e = emin : ewth : emax;
```

which yields

```
e =
  Columns 1 through 4
    9.4168   10.0598   10.7029   11.3460
  Columns 5 through 8
   11.9891   12.6322   13.2753   13.9184
  Column 9
   14.5615
```

The smallest and largest bin edges, `emin` and `emax`, correspond to the mininum

www.it-ebooks.info

Gallery 3.1

and maximum values in `corg`. The width of the bins `ewth`, as defined by the bin edges, is equal to `vwth`. Binning can be performed using a `for` loop in combination with an `if-then` construct. Values in `corg` that fall exactly on an edge `e` are herein assigned to the higher bin. Values falling exactly on the edge of the last bin are assigned to the last bin.

```
for i = 1 : nbin
   if i < nbin
   corgb = corg(emin+(i-1)*ewth<=corg & emin+i*ewth>corg);
   n(i) = length(corgb);
   else
   corgb = corg(emin+(i-1)*ewth<=corg & emin+i*ewth>=corg);
   n(i) = length(corgb);
   end
end
```

The result of the binning is stored in the frequency distribution `n`, which we can display with

```
n
```

which yields

```
n =
     2     2    10     7    14     9    12     4
```

Instead of using the above script we can also use the functions `histcount` and `histogram` to determine the bin centers, edges and counts. Typing

```
histogram(corg)
```

displays a histogram with six classes, using an automatic binning algorithm. The algorithm returns bins with a uniform width, chosen to cover the range of elements in `corg` and to reveal the underlying shape of the distribution. The MATLAB Help lists and explains various such methods for automatic binning. The function `histogram` also returns a structure array `h`

```
h = histogram(corg);
```

which yields

```
h =
  Histogram with properties:
            Data: [60x1 double]
          Values: [2 6 13 20 15 4]
         NumBins: 6
        BinEdges: [9 10 11 12 13 14 15]
        BinWidth: 1
       BinLimits: [9 15]
    Normalization: 'count'
```

www.it-ebooks.info

```
          FaceColor: 'auto'
          EdgeColor: [0 0 0]
```

listing the properties of the histogram. As an example we can access the bin centers `v` and the frequency distribution `n` by typing

```
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1)
n = h.Values
```

while the Figure Window is still active, which yields

```
v =
    9.5000   10.5000   11.5000   12.5000   13.5000   14.5000

n =
     2     6    13    20    15     4
```

Alternatively, we can use the bin edges `e` defined above to display the bar plot of the frequency distribution `n` by typing

```
h = histogram(corg,e);
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1)
n = h.Values
```

which yields

```
v =
  Columns 1 through 4
    9.7383   10.3814   11.0245   11.6676
  Columns 5 through 8
   12.3107   12.9537   13.5968   14.2399

n =
     2     2    10     7    14     9    12     4
```

as well as the histogram plot, as shown in Figure 3.1 a. We can also use the function `histcount` to determine `n` and `e` without plotting the histogram. Typing

```
[n,e] = histcounts(corg,e)
v = diff(e(1:2)) * 0.5 + e(1:end-1)
```

yields

```
n =
     2     2    10     7    14     9    12     4

e =
  Columns 1 through 5
    9.4168   10.0598   10.7029   11.3460   11.9891
  Columns 6 through 9
   12.6322   13.2753   13.9184   14.5615
```

www.it-ebooks.info

```
v =
  Columns 1 through 5
    9.7383   10.3814   11.0245   11.6676   12.3107
  Columns 6 through 8
   12.9537   13.5968   14.2399
```

The functions `histogram` and `histcount` provide numerous ways of binning the data, of normalizing the data, and of displaying the histogram. In the following sections, we often have to normalize the frequency distribution such that the total number of observations `sum(n)` is one. This can be achieved by typing

```
h = histogram(corg,'Normalization','probability')
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1)
n = h.Values
```

which yields

```
v =
  Columns 1 through 5
    9.5000   10.5000   11.5000   12.5000   13.5000
  Column 6
   14.5000

n =
  Columns 1 through 5
    0.0333    0.1000    0.2167    0.3333    0.2500
  Column 6
    0.0667
```

As an alternative way of plotting the data, the empirical cumulative distribution function can be displayed using `cdfplot` (Fig. 3.1 b).

```
cdfplot(corg)
```

Gallery 3.2

The most important parameters describing the distribution are the measures for central tendency and the dispersion about the average. The most popular measure for central tendency is the arithmetic mean.

```
mean(corg)

ans =
    12.3448
```

Since this measure is very susceptible to outliers, we can take the median as an alternative measure of central tendency,

```
median(corg)
```

www.it-ebooks.info

```
ans =
    12.4712
```

which does not differ by very much in this particular example. However, we will see later that this difference can be significant for distributions that are not symmetric. A more general parameter to define fractions of the data less than, or equal to, a certain value is the quantile. Some of the quantiles have special names, such as the three quartiles dividing the distribution into four equal parts, 0–25%, 25–50%, 50–75% and 75–100% of the total number of observations. We use the function `quantile` to compute the three quartiles.

```
quantile(corg,[.25 .50 .75])

ans =
    11.4054   12.4712   13.2965
```

Less than 25% of the data values are therefore lower than 11.4054, 25% are between 11.4054 and 12.4712, another 25% are between 12.4712 and 13.2965, and the remaining 25% are higher than 13.2965.

The third parameter in this context is the mode, which is the midpoint of the interval with the highest frequency. The MATLAB function `mode` to identify the most frequent value in a sample is unlikely to provide a good estimate of the peak in continuous probability distributions, such as the one in `corg`. Furthermore, the `mode` function is not suitable for finding peaks in distributions that have multiple modes. In these cases it is better to compute a histogram and calculate the peak of that histogram. We can use the function `find` to locate the class that has the largest number of observations.

```
v(find(n == max(n)))
```

or simply

```
v(n == max(n))

ans =
    12.3107
```

Both statements are identical and identify the largest element in `n`. The index of this element is then used to display the midpoint of the corresponding class `v`. If there are several elements in `n` with similar values this statement returns several solutions, suggesting that the distribution has several modes. The median, quartiles, minimum, and maximum of a data set can be summarized and displayed in a *box and whisker plot*.

```
boxplot(corg)
```

www.it-ebooks.info

The boxes have lines at the lower quartile, the median, and the upper quartile values. The whiskers are lines extending from each end of the boxes to show the extent or range of the rest of the data.

The most popular measures for dispersion are range, variance and standard deviation. We have already used the range to define the midpoints of the classes. The range is the difference between the highest and lowest value in the data set.

```
range(corg)

ans =
    5.1447
```

The variance is the average of the squared deviation of each number from the mean of a data set.

```
var(corg)

ans =
    1.3595
```

The standard deviation is the square root of the variance.

```
std(corg)

ans =
    1.1660
```

Note that, by default, the functions `var` and `std` calculate the sample variance and sample standard deviation providing an unbiased estimate of the dispersion of the population. When using `skewness` to describe the shape of the distribution, we observe a slightly negative skew.

```
skewness(corg)

ans =
   -0.2529
```

Finally, the peakedness of the distribution is described by the kurtosis. The result from the function `kurtosis`,

```
kurtosis(corg)

ans =
    2.4670
```

suggests that our distribution is slightly flatter than a Gaussian distribution since its kurtosis is less than three.

www.it-ebooks.info

Most of these functions have corresponding versions for data sets containing gaps, such as `nanmean` and `nanstd`, which treat `NaN`s as missing values. To illustrate the use of these functions we introduce a gap into our data set and compute the mean using `mean` and `nanmean` for comparison.

```
corg(25,1) = NaN;

mean(corg)

ans =
    NaN

nanmean(corg)

ans =
    12.3371
```

In this example the function `mean` follows the rule that all operations with `NaN`s result in `NaN`s, whereas the function `nanmean` simply skips the missing value and computes the mean of the remaining data.

As a second example we now explore a data set characterized by a significant skew. The data represent 120 microprobe analyses on glass shards hand-picked from a volcanic ash. The volcanic glass has been affected by chemical weathering at an initial stage and the shards therefore exhibit glass hydration and sodium depletion in some sectors. We can study the distribution of sodium (in wt%) in the 120 analyses using the same procedure as above. The data are stored in the file *sodiumcontent_one.txt*.

```
clear

sodium = load('sodiumcontent_one.txt');
```

As a first step, it is always recommended to display the data as a histogram. The square root of 120 suggests 11 classes, and we therefore display the data by typing

```
h = histogram(sodium,11)
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1)
n = h.Values
```

Since the distribution has a negative skew, the mean, the median and the mode differ significantly from each other.

```
mean(sodium)

ans =
    5.6628

median(sodium)
```

www.it-ebooks.info

```
ans =
    5.9741

v(find(n == max(n)))

ans =
    6.5800
```

The mean of the data is lower than the median, which is in turn lower than the mode. We can observe a strong negative skewness, as expected from our data.

```
skewness(sodium)

ans =
    -1.1086
```

We now introduce a significant outlier to the data and explore its effect on the statistics of the sodium content. For this we will use a different data set that is better suited to this example than the previous data set. The new data set contains higher sodium values of around 17 wt% and is stored in the file *sodiumcontent_two.txt*.

```
clear

sodium = load('sodiumcontent_two.txt');
```

This data set contains only 50 measurements, in order to better illustrate the effects of an outlier. We can use the same script used in the previous example to display the data in a histogram with seven classes, and to compute the number of observations `n` in each of the classes `v`.

```
h = histogram(sodium,7)
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1)
n = h.Values

v =
  Columns 1 through 4
   13.5500   14.2500   14.9500   15.6500

  Columns 5 through 7
   16.3500   17.0500   17.7500

n =
     1     3     3     6     6    15    16

mean(sodium)

ans =
   16.6379
```

www.it-ebooks.info

```
median(sodium)

ans =
   16.9739

v(find(n == max(n)))

ans =
   17.7500
```

The mean of the data is 16.6379, the median is 16.9739 and the mode is 17.7500. We now introduce a single, very low value of 7.0 wt% in addition to the 50 measurements contained in the original data set.

```
sodium(51,1) = 7.0;

h = histogram(sodium,11)
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1)
n = h.Values

v =
  Columns 1 through 4
    7.5500    8.6500    9.7500   10.8500
  Columns 5 through 8
   11.9500   13.0500   14.1500   15.2500
  Columns 9 through 11
   16.3500   17.4500   18.5500

n =
  Columns 1 through 8
     1     0     0     0     0     1     3     8
  Columns 9 through 11
     9    27     2
```

The histogram of this data set using a larger number of classes illustrates the distortion produced in the frequency distribution by this single outlier, showing several empty classes. The influence of this outlier on the sample statistics is also substantial.

```
mean(sodium)

ans =
   16.4490

median(sodium)

ans =
   16.9722

v(find(n == max(n)))

ans =
   17.4500
```

www.it-ebooks.info

The most significant change observed is in the mean (16.4490), which is substantially lower due to the presence of the outlier. This example clearly demonstrates the sensitivity of the mean to outliers. In contrast, the median of 16.9722 is relatively unaffected.

3.4 Theoretical Distributions

We have now described the empirical frequency distribution of our sample. A histogram is a convenient way to depict the frequency distribution of the variable x. If we sample the variable sufficiently often and the output ranges are narrow, we obtain a very smooth version of the histogram. An infinite number of measurements $N \rightarrow \infty$ and an infinitely small class width produce the random variable's *probability density function* (PDF). The probability distribution density $f(x)$ defines the probability that the variable has a value equal to x. The integral of $f(x)$ is normalized to unity, i.e., the total number of observations is one. The *cumulative distribution function* (CDF) is the sum of the frequencies of a discrete PDF or the integral of a continuous PDF. The cumulative distribution function $F(x)$ is the probability that the variable will have a value less than or equal to x.

As a next step, we need to find appropriate theoretical distributions that fit the empirical distributions described in the previous section. This section therefore introduces the most important theoretical distributions and describes their application.

Uniform Distribution

A *uniform* or *rectangular distribution* is a distribution that has a constant probability (Fig. 3.4). The corresponding probability density function is

$$f(x) = 1 / N = const.$$

where the random variable x has any of N possible values. The cumulative distribution function is

$$F(x) = x \cdot 1 / N$$

The probability density function is normalized to unity

$$\sum_{-\infty}^{+\infty} f(x)\,dx = 1$$

www.it-ebooks.info

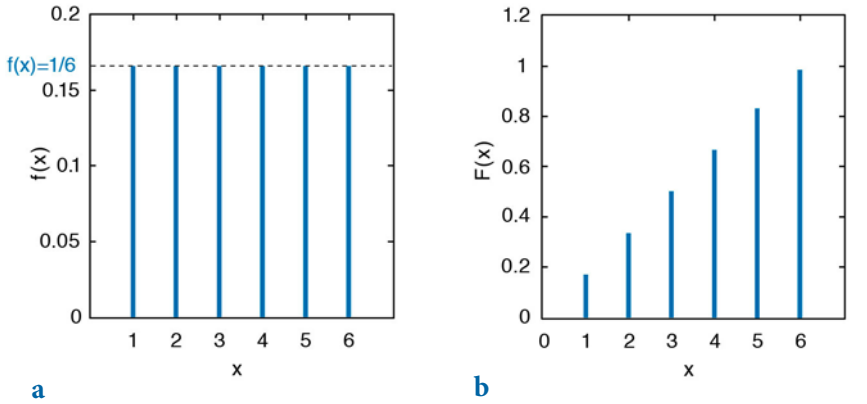


Fig. 3.4 a Probability density function $f(x)$, and **b** cumulative distribution function $F(x)$, of a uniform distribution with N=6. The 6 discrete values of the variable x have the same probability of 1/6.

i.e., the sum of all probabilities is one. The maximum value of the cumulative distribution function is therefore one.

$$F(x)_{\max} = 1$$

An example is a rolling die with N=6 faces. A discrete variable such as the faces of a die can only take a countable number of values x. The probability for each face is 1/6. The probability density function of this distribution is

$$f(x) = 1/6$$

The corresponding cumulative distribution function is

$$F(x) = x \cdot 1/6$$

where x takes only discrete values, x=1,2,…,6.

Binomial or Bernoulli Distribution

A *binomial* or *Bernoulli distribution*, named after the Swiss scientist Jakob Bernoulli (1654–1705), gives the discrete probability of x successes out of N trials, with a probability p of success in any given trial (Fig. 3.5). The probability density function of a binomial distribution is

www.it-ebooks.info

$$f(x) = \binom{N}{x} p^x (1-p)^{N-x}$$

The cumulative distribution function is

$$F(x) = \sum_{i=1}^{x} \binom{N}{i} p^i (1-p)^{N-i}$$

where

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

The binomial distribution has two parameters N and p. An example for the application of this distribution is to determine the likely outcome of drilling for oil. Let us assume that the probability of drilling success is 0.1 or 10%. The probability of x=3 successful wells out of a total number of N=10 wells is

$$f(3) = \binom{10}{3} 0.1^3 (1-0.1)^{10-3} = 0.057 \approx 6\%$$

The probability of exactly 3 successful wells out of 10 trials is therefore 6% in this example.

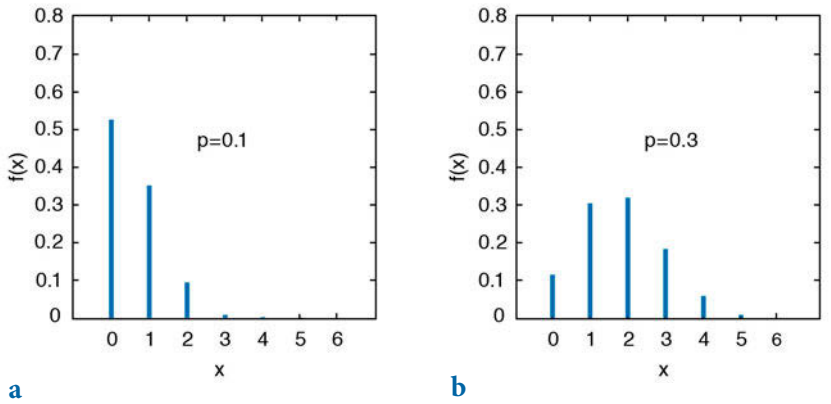


Fig. 3.5 Probability density function $f(x)$ of a binomial distribution, which gives the probability p of x successes out of N=6 trials, with probability **a** p=0.1 and **b** p=0.3 of success in any given trial.

www.it-ebooks.info

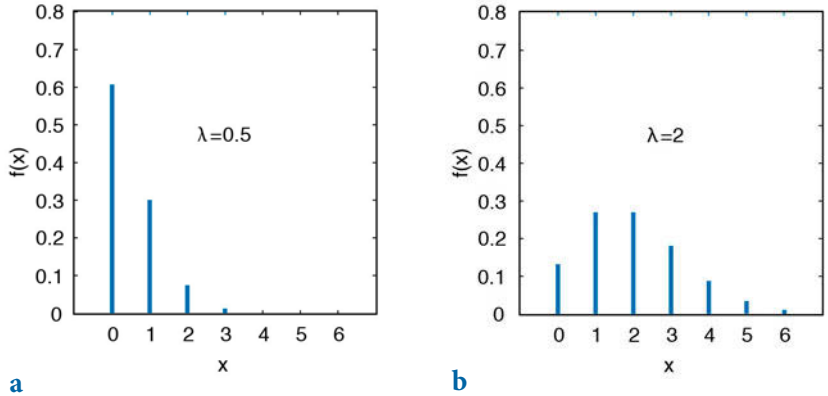


Fig. 3.6 Probability density function $f(x)$ of a Poisson distribution with different values for λ: **a** λ=0.5, and **b** λ=2.

Poisson Distribution

When the number of trials is $N \to \infty$ and the success probability is $p \to 0$, the binomial distribution approaches a *Poisson distribution* with a single parameter $\lambda = Np$ (Fig. 3.6) (Poisson 1837). This works well for $N > 100$ and $p < 0.05$ (or 5%). We therefore use the Poisson distribution for processes characterized by extremely low occurrence, e.g., earthquakes, volcanic eruptions, storms and floods. The probability density function is

$$f(x) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

and the cumulative distribution function is

$$F(x) = \sum_{i=0}^{x} \frac{e^{-\lambda}\lambda^{i}}{i!}$$

The single parameter λ describes both the mean and the variance of this distribution.

Normal or Gaussian Distribution

When $p=0.5$ (symmetric, no skew) and $N \to \infty$, the binomial distribution approaches a *normal* or *Gaussian distribution* defined by the mean μ and standard deviation σ (Fig. 3.7). The probability density function of a normal

www.it-ebooks.info

distribution is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma} \right)^2 \right)$$

and the cumulative distribution function is

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{1}{2}\left(\frac{y-\mu}{\sigma} \right)^2 \right) dy$$

The normal distribution is therefore used when the mean is both the most frequent and the most likely value. The probability of deviations is equal in either direction and decreases with increasing distance from the mean.

The *standard normal distribution* is a special member of the normal distribution family that has a mean of *zero* and a standard deviation of *one*. We can transform the equation for a normal distribution by substituting $z=(x-\mu)/\sigma$. The probability density function of this distribution is

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2} \right)$$

This definition of the normal distribution is often called the *z distribution*.

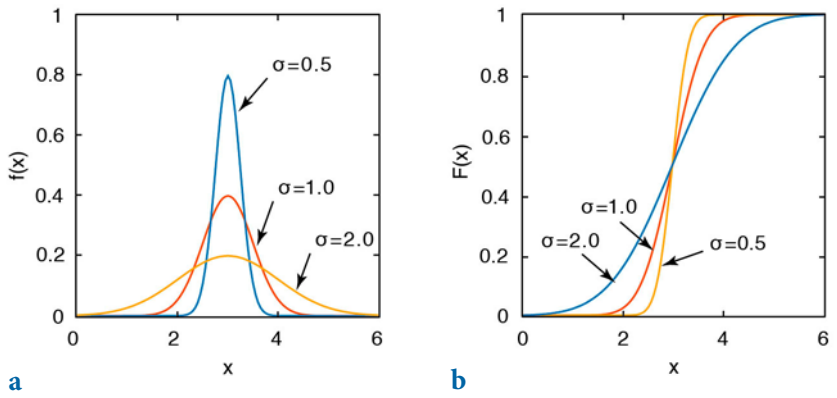


Fig. 3.7 a Probability density function $f(x)$, and **b** cumulative distribution function $F(x)$, of a Gaussian or normal distribution with a mean $\mu=3$ and various values for standard deviation σ.

www.it-ebooks.info

Logarithmic Normal or Log-Normal Distribution

The *logarithmic normal* or *log-normal distribution* is used when the data have a lower limit, e.g., mean-annual precipitation or the frequency of earthquakes (Fig. 3.8). In such cases, distributions are usually characterized by significant skewness, which is best described by a logarithmic normal distribution. The probability density function of this distribution is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}x}\exp\left(-\frac{1}{2}\left(\frac{\ln x - \mu}{\sigma}\right)^2\right)$$

and the cumulative distribution function is

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{x}\frac{1}{y}\exp\left(-\frac{1}{2}\left(\frac{\ln y - \mu}{\sigma}\right)^2\right)dy$$

where $x>0$. The distribution can be described by two parameters: the mean μ and the standard deviation σ. The formulas for the mean and the standard deviation, however, are different from the ones used for normal distributions. In practice, the values of x are logarithmized, the mean and the standard deviation are computed using the formulas for a normal distribution, and the empirical distribution is then compared with a normal distribution.

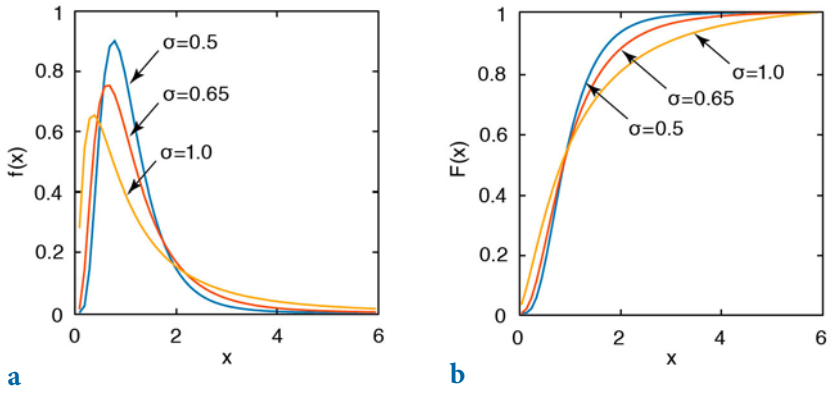


Fig. 3.8 a Probability density function $f(x)$, and **b** cumulative distribution function $F(x)$, of a logarithmic normal distribution with a mean μ=0 and with various values for σ.

www.it-ebooks.info

Student's t Distribution

The *Student's t distribution* was first introduced by William Gosset (1876–1937) who needed a distribution for small samples (Fig. 3.9). Gosset was an employee of the Irish Guinness Brewery and was not allowed to publish research results. For that reason he published his *t* distribution under the pseudonym *Student* (Student 1908). The probability density function is

$$f(x) = \frac{\Gamma\left(\frac{\Phi+1}{2}\right)}{\Gamma\left(\frac{\Phi}{2}\right)} \frac{1}{\sqrt{\Phi\pi}} \frac{1}{\left(1+\frac{x^2}{\Phi}\right)^{\frac{\Phi+1}{2}}}$$

where Γ is the Gamma function

$$\Gamma(x) = \lim_{n\to\infty} \frac{n!n^{x-1}}{x(x+1)(x+2)\ldots(x+n-1)}$$

which can be written as

$$\Gamma(x) = \int_0^\infty e^{-y} y^{x-1} dy$$

if $x>0$. The single parameter Φ of the *t* distribution is the number of degrees

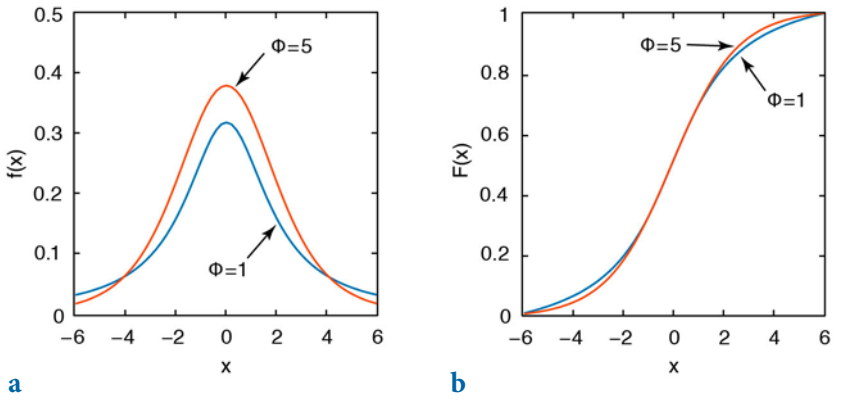
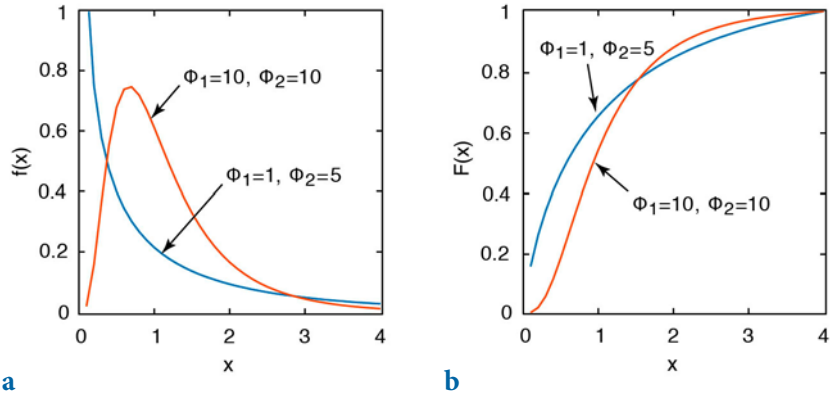


Fig. 3.9 a Probability density function $f(x)$, and **b** cumulative distribution function $F(x)$, of a Student's *t* distribution with two different values for Φ.

www.it-ebooks.info



a b

Fig. 3.10 a Probability density function $f(x)$, and **b** cumulative distribution function $F(x)$, of a Fisher's F distribution with different values for Φ_1 and Φ_2.

of freedom. In the analysis of univariate data this distribution has $n-1$ degrees of freedom, where n is the sample size. As $\Phi \rightarrow \infty$, the t distribution converges towards the standard normal distribution. Since the t distribution approaches the normal distribution for $\Phi > 30$, it is rarely used for distribution fitting. However, the t distribution is used for hypothesis testing using the t-test (Section 3.7).

Fisher's *F* Distribution

The *F distribution* was named after the statistician Sir Ronald Fisher (1890–1962). It is used for hypothesis testing using the F-test (Section 3.8). The F distribution has a relatively complex probability density function (Fig. 3.10):

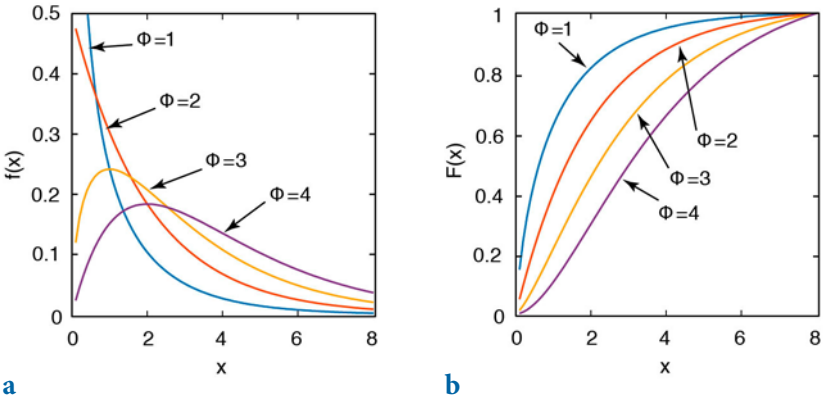
$$f(x) = \frac{\Gamma\left(\frac{\Phi_1 + \Phi_2}{2}\right)\left(\frac{\Phi_1}{\Phi_2}\right)^{\frac{\Phi_1}{\Phi_2}}}{\Gamma\left(\Phi_1/2\right)\Gamma(\Phi_1/2)} x^{\frac{\Phi_1 - 2}{2}} \left(1 + \frac{\Phi_1}{\Phi_2} x\right)^{\frac{\Phi_1 + \Phi_2}{2}}$$

where $x > 0$ and Γ is again the Gamma function. The two parameters Φ_1 and Φ_2 are the numbers of degrees of freedom.

χ^2 or Chi-Squared Distribution

The χ^2 distribution was introduced by Friedrich Helmert (1876) and Karl Pearson (1900). It is not used for fitting a distribution but has important applications in statistical hypothesis testing using the χ^2-test (Section 3.9).

www.it-ebooks.info



a b

Fig. 3.11 a Probability density function $f(x)$, and **b** cumulative distribution function $F(x)$, of a χ^2 distribution with different values for Φ.

The probability density function of the χ^2 distribution is

$$f(x) = \frac{1}{2^{\Phi/2}\,\Gamma(\Phi/2)}\, x^{\frac{\Phi-2}{2}}\, e^{-\frac{x}{2}}$$

where $x>0$, otherwise $f(x)=0$; Γ is again the Gamma function. Once again, Φ is the number of degrees of freedom (Fig. 3.11).

3.5 Examples of Theoretical Distributions

The function `randtool` is a tool for simulating discrete data sets with statistics similar to our data set. This function creates a histogram of *random numbers* from the distributions in the Statistics Toolbox (MathWorks 2014). The random numbers that have been generated by using this tool can then be exported into the workspace. We start the *graphical user interface* (GUI) of the function by typing

Movie 3.1

```
randtool
```

after the prompt. We can now create a data set similar to the one in the file *organicmatter_one.txt*. The 60 measurements have a mean of 12.3448 wt% and a standard deviation of 1.1660 wt%. The GUI uses *Mu* for μ (the mean of a population) and *Sigma* for σ (the standard deviation). After choosing *Normal* for a *Gaussian* distribution and 60 for the number of samples, we get a histogram similar to the one in the first example (Section 3.3). This synthetic distribution based on 60 measurements represents a rough estimate of the

www.it-ebooks.info

true normal distribution. If we increase the sample size the histogram looks much more like a true Gaussian distribution.

Instead of simulating discrete distributions we can use the probability density function (PDF) or the cumulative distribution function (CDF) to compute a theoretical distribution. MATLAB also provides a GUI-based function for generating PDFs and CDFs with a specific statistic, which is called `disttool`.

Movie 3.2

```
disttool
```

We choose `Normal` as the distribution and `PDF` as the function type, and then define the mean as `mu=12.3448` and the standard deviation as `sigma=1.1660`. Although the function `disttool` is GUI-based, it uses non-GUI functions such as `normpdf` and `normcdf` to calculate the probability density function and the cumulative distribution function. The MATLAB Help gives an overview of the available theoretical distributions. As an example we can use the functions `normpdf(x,mu,sigma)` and `normcdf(x,mu,sigma)` to compute the PDF and CDF of a Gaussian distribution with `mu=12.3448` and `sigma=1.1660`, evaluated for the values in `x`, to compare the results with those from our sample data set.

```
clear

mu = 12.3448;
sigma = 1.1660;

x = 5 : 0.001 : 20;
pdf = normpdf(x,mu,sigma);
cdf = normcdf(x,mu,sigma);
plot(x,pdf,x,cdf)
```

We can use these functions to familiarize ourselves with the properties of distributions. This will be important when we deal with the testing of hypotheses in the following sections. The test statistics used there follow the theoretical frequency distributions introduced in the previous sections of this chapter. In particular, the integral (or in the discrete case, the sum) of the theoretical distribution within a certain range $a \leq x \leq b$ is of great importance, as it helps in calculating the probability that a measurement will fall within this range.

As an example we can calculate the probability that a measurement will fall within the range of $\mu-\sigma \leq x \leq \mu+\sigma$, where μ is the mean and σ is the standard deviation of a Gaussian distribution. Using the PDF of the Gaussian distribution with `mu=12.3448` and `sigma=1.1660`, we find

```
pdf = pdf/sum(pdf);
```

www.it-ebooks.info

```
sum(pdf(find(x>mu-sigma,1,'first'):find(x<mu+sigma,1,'last')))
```

which yields

```
ans =
    0.6827
```

or ~68% after normalizing the PDF to unity. The expression `find(x>mu-sigma,1,'first')` returns the first index of `x` where `x` is larger than `mu-sigma`. Similarly, `find(x<mu+sigma,1,'last')` returns the last index of `x` where `x` is smaller than `mu+sigma`. Using

```
sum(pdf(find(x>mu-sigma):find(x<mu+sigma))
```

instead yields

```
ans =
   0
```

since the value of `mu-sigma` (11.1788) falls between two values of `x`, 11.178 and 11.179, similar to the value of `mu-sigma` (13.5108) falling between 13.510 and 13.511. Both, `find(x==mu-sigma)` and `find(x==mu+sigma)` return an empty matrix and hence the sum of the $\mu-\sigma \le x \le \mu+\sigma$ range is zero. Alternatively, we can calculate the probability of the $-\sigma \le x \le +\sigma$ range from the CDF by typing

```
cdf(find(x<mu+sigma,1,'last'))-cdf(find(x>mu-sigma,1,'first'))
```

which yields about the same probability of

```
ans =
    0.6825
```

or ~68%. Again,

```
cdf(find(x==mu+sigma))-cdf(find(x==mu-sigma))
```

yields no result

```
ans =
   Empty matrix: 1-by-0
```

for the same reason as before. Conversely, we can also calculate the x-values of the $\mu \pm \sigma$ range of our PDF using the inverse of the cumulative normal distribution function with `norminv(p,mu,sigma)` by typing

```
norminv((1-0.6827)/2,mu,sigma)
norminv(1-(1-0.6827)/2,mu,sigma)
```

which yields

www.it-ebooks.info

```
ans =
   11.1788
ans =
   13.5108
```

Here, the values for `p` are calculated from the complement of ~68%, which is ~32%, halved on both tails of the Gaussian distribution, e.g., `(1-0.6827)/2` and `1-(1-0.6827)/2`.

The standard deviation σ of the Gaussian distribution is important for the definition of confidence intervals. In many examples, however, the confidence of one sigma (μ±1σ) or ~68% that the true value falls within the μ±1σ range is not sufficient and higher confidence intervals such as two sigma (μ±2σ) and three sigma (μ±3σ) intervals are therefore also used. We can calculate the corresponding probabilities that the true value falls within the μ±2σ range and the μ±3σ range by typing

```
sum(pdf(find(x>mu-2*sigma,1,'first'):find(x<mu+2*sigma,1,'last')))
sum(pdf(find(x>mu-3*sigma,1,'first'):find(x<mu+3*sigma,1,'last')))
```

which yields

```
ans =
    0.9545
ans =
    0.9973
```

or ~95% and ~99%. Again, using `norminv` we can calculate the upper and lower bounds of the two sigma (μ±2σ) range

```
norminv(0.05/2,mu,sigma)
norminv(1-0.05/2,mu,sigma)
```

which yields

```
ans =
   10.0595
ans =
   14.6301
```

and the three sigma (μ±3σ) range

```
norminv(0.01/2,mu,sigma)
norminv(1-0.01/2,mu,sigma)
```

which yields

```
ans =
    9.3414
ans =
   15.3482
```

www.it-ebooks.info

Instead of using `normpdf` and `normcdf`, we can use the general function `makedist` to create a probability distribution object for any type of distribution. As an example we create a probability distribution object for a Gaussian distribution by typing

```
clear

mu = 12.3448;
sigma = 1.1660;

pd = makedist('normal',mu,sigma)
```

which yields

```
pd =
  NormalDistribution

  Normal distribution
       mu = 12.3448
    sigma =   1.166
```

When we type `whos` we can then see that we have created an object with name `pd` and class `prob.NormalDistribution`. Using this object we can determine the μ±1σ range using the probabilities `(1-0.6827)/2` and `1-(1-0.6827)/2` with the inverse cumulative distribution function `icdf` of the probability distribution object `pd`:

```
y = icdf(pd,[(1-0.6827)/2,1-(1-0.6827)/2])
```

which yields

```
y =
   11.1788   13.5108
```

We can calculate the μ±2σ and μ±3σ ranges in a similar manner. The function `normspec`

```
p = normspec(y,mu,sigma,'inside')
```

which yields

```
p =
    0.6827
```

plots the PDF and shades the region inside the μ±1σ range (Fig. 3.12).

www.it-ebooks.info

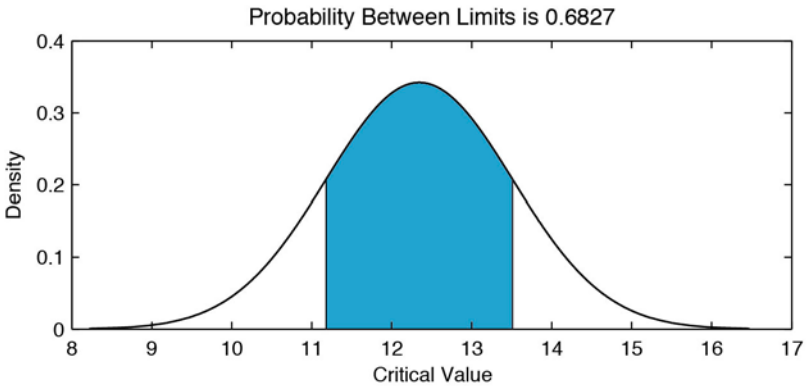


Fig. 3.12 Plot of a standard normal distribution between specified limits. As an example, the shaded area displays the $\mu \pm \sigma$ range of a Gaussian distribution with a mean $\mu=12.3448$ and a standard deviation $\sigma=1.1660$.

3.6 Hypothesis Testing

The remaining sections in this chapter are concerned with methods used to draw conclusions from the statistical sample that can then be applied to the larger population of interest (*hypothesis testing*). All hypothesis tests share the same concept and terminology. The *null hypothesis* is an assertion about the population describing the absence of a statistically significant characteristic or effect, whereas an *alternative hypothesis* is a contrasting assertion. The *p*-value of a hypothesis test is the probability, under the null hypothesis, of observing larger values for the test statistic than those calculated from the sample. The *significance level* α is the threshold of probability that controls the outcome of the tests. If the *p*-value is smaller than α, the null hypothesis can be rejected; the outcome of the test is regarded as *significant* if $p<0.05$, or *highly significant* if $p<0.01$.

A hypothesis test can be performed either as a *one-tailed* (one-sided) or *two-tailed* (two-sided) test. The term *tail* derives from the tailing off the data to the far left or far right of a probability density function as, for instance, in the standard normal distribution used in the Mann-Whitney and Ansari-Bradley tests (Sections 3.11 and 3.12). As an example, the Mann-Whitney test compares the medians of two data sets. The one-tailed Mann-Whitney test is used to test against the alternative hypothesis that the median of the first sample is either smaller or larger than the median of the second sample at a significance level of 5% (or 0.05). The two-tailed Mann-Whitney test is used when the medians are not equal at a 5% significance level, i.e., when it makes no difference which of the medians is larger. In this case, the

www.it-ebooks.info

significance level is halved, i.e., 2.5% is used instead of 5%.

We can display the standard normal distribution for the one-tailed test by typing

```
clear

pd = makedist('normal',0,1)
y = icdf(pd,[0,.95]);
[p,h] = normspec(y,0,1,'outside');
```

and obtain a plot with one blue tail to the right, with the 5% area shaded (Fig. 3.13 a). Similarly, we can display the standard normal distribution for the two-tailed test by typing

```
clear
```

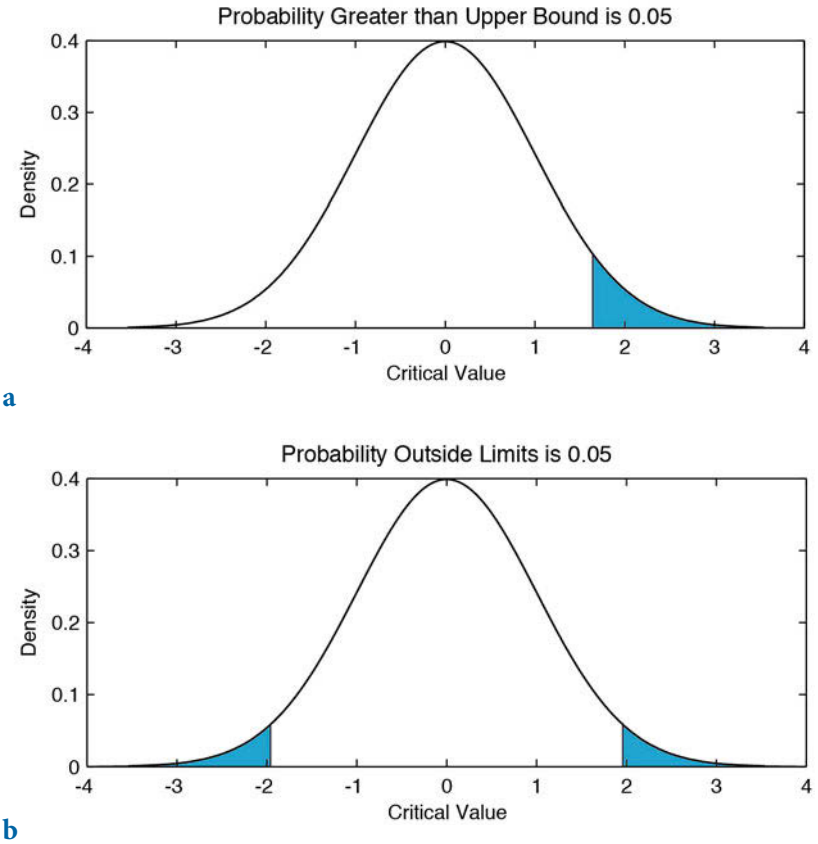


Fig. 3.13 Plot of a standard normal distribution with **a** one, or **b** two, critical regions shown as shaded areas that contain a total of 5% of the area under the curve.

www.it-ebooks.info

```
pd = makedist('normal',0,1)
y = icdf(pd,[.025,.975])
[p,h] = normspec(y,0,1,'outside');
```

and obtain a plot with two blue tails, one to the left and one to the right, with the 2.5% areas shaded (Fig. 3.13 b).

Note that we cannot prove the null hypothesis, in other words *not guilty* is not the same as *innocent*. In practice, we design hypotheses based on our data, test them, and then continue to work with those found to be true, those we could not show to be false. The inherent possibility to prove a hypothesis to be false is therefore an important requirement for our hypotheses.

The next sections introduce the most important hypothesis tests for earth science applications: the two-sample *t*-test to compare the means of two data sets, the two-sample *F*-test to compare the variances of two data sets, and the χ^2-test and Kolmogorov-Smirnov test to compare distributions (Sections 3.7 to 3.10). The Mann-Whitney and Ansari-Bradley tests are alternatives to the *t*-test and *F*-test for comparing the medians and dispersions of two data sets without requiring a normality assumption for the underlying population (Sections 3.11 and 3.12). The final section introduces methods that can be used to fit distributions to our data sets (Section 3.13).

3.7 The t-Test

The Student's *t*-test by William Sealy Gosset (Student 1908) compares the means of two distributions. The *one-sample t-test* is used to test the hypothesis that the mean of a Gaussian-distributed population has a value specified in the null hypothesis. The *two-sample t-test* is employed to test the hypothesis that the means of two Gaussian distributions are identical. In the following text the two-sample *t*-test is introduced to demonstrate hypothesis testing. Let us assume that two independent sets of n_a and n_b measurements have been carried out on the same object, for instance measurements on two sets of rock samples taken from two separate outcrops. The *t*-test can be used to determine whether both samples come from the same population, e.g., the same lithologic unit (*null hypothesis*) or from two different populations (*alternative hypothesis*). Both sample distributions must be Gaussian and the variances for the two sets of measurements should be similar. The appropriate test statistic for the difference between the two means is then

$$\hat{t} = \frac{\left|\overline{a} - \overline{b}\right|}{\sqrt{\frac{n_a + n_b}{n_a n_b} \cdot \frac{(n_a - 1) \cdot s_a^{\,2} + (n_b - 1) \cdot s_b^{\,2}}{n_a + n_b - 2}}}$$

www.it-ebooks.info

where n_a and n_b are the sample sizes, and s_a^2 and s_b^2 are the variances of the two samples a and b. The null hypothesis can be rejected if the measured t-value is higher than the critical t-value, which depends on the number of degrees of freedom $\Phi=n_a+n_b-2$ and the significance level α. The one-tailed test is used to test against the alternative hypothesis that the mean of the first sample is either smaller or larger than the mean of the second sample at a significance level of 5% (or 0.05). The one-tailed test would require the modification of the above equation by replacing the absolute value of the difference between the means with the actual difference between the means. The two-tailed t-test is used when the means are not equal at a 5% significance level, i.e., when it makes no difference which of the means is larger. In this case, the significance level is halved, i.e., 2.5% is used to compute the critical t-value.

We can now load two example data sets from two independent series of measurements. The first example shows the performance of the two-sample t-test on two distributions with means of 25.5 and 25.3 and standard deviations of 1.3 and 1.5, respectively.

```
clear

load('organicmatter_two.mat');
```

The binary file *organicmatter_two.mat* contains two data sets `corg1` and `corg2`. First, we plot both histograms in a single graph.

```
histogram(corg1,'FaceColor','b'), hold on
histogram(corg2,'FaceColor','r'), hold off
```

We then compute the sample sizes, the means and the standard deviations.

```
na = length(corg1); nb = length(corg2);
ma = mean(corg1); mb = mean(corg2);
sa = std(corg1); sb = std(corg2);
```

Next, we calculate the t-value using the translation of the equation for the t-test statistic into MATLAB code.

```
tcalc = abs((ma-mb))/sqrt(((na+nb)/(na*nb)) * ...
   (((na-1)*sa^2+(nb-1)*sb^2)/(na+nb-2)))

tcalc =
    0.7279
```

We can now compare the calculated `tcalc` value of 0.7279 with the critical `tcrit` value. This can be accomplished using the function `tinv`, which yields the inverse of the t distribution function with `na-nb-2` degrees of freedom at

www.it-ebooks.info

the 5% significance level. This is a two-sample *t*-test, i.e., the means are not equal. Computing the two-tailed critical `tcrit` value by entering 1–0.05/2 yields the upper (positive) `tcrit` value, which we compare with the absolute value of the difference between the means.

```
tcrit = tinv(1-0.05/2,na+nb-2)

tcrit =
    1.9803
```

Since the `tcalc` value calculated from the data is smaller than the critical `tcrit` value, we cannot reject the null hypothesis without another cause. We conclude therefore that the two means are identical at a 5% significance level. Alternatively, we can apply the function `ttest2(x,y,alpha)` to the two independent samples `corg1` and `corg2` at an `alpha=0.05` or a 5% significance level. The command

```
[h,p,ci,stats] = ttest2(corg1,corg2,0.05)
```

yields

```
h =
    0

p =
    0.4681

ci =
   -0.3028
    0.6547

stats =
    tstat: 0.7279
       df: 118
       sd: 1.3241
```

The result `h=0` means that we cannot reject the null hypothesis without another cause at a 5% significance level. The *p*-value of 0.4681 or ~47% (which is much greater than the significance level of 0.05 or 5%) suggests that the chances of observing more extreme *t*-values than the values in this example from similar experiments would be 4,681 in 10,000. The 95% confidence interval on the mean is [–0.3028,0.6547], which includes the theoretical (and hypothesized) difference between the means of 25.5–25.3=0.2.

The second synthetic example shows the performance of the two-sample *t*-test in an example with very different means of 24.3 and 25.5, while the standard deviations are again 1.3 and 1.5, respectively.

```
clear
```

www.it-ebooks.info

```
load('organicmatter_three.mat');
```

This file again contains two data sets `corg1` and `corg2`. As before, we plot both histograms in a single graph.

```
histogram(corg1,'FaceColor','b'), hold on
histogram(corg2,'FaceColor','r'), hold off
```

We then compute the sample sizes, the means and the standard deviations.

```
na = length(corg1); nb = length(corg2);
ma = mean(corg1); mb = mean(corg2);
sa = std(corg1); sb = std(corg2);
```

Next, we calculate the *t*-value using the translation of the equation for the *t*-test statistic into MATLAB code.

```
tcalc = abs((ma-mb))/sqrt(((na+nb)/(na*nb)) * ...
   (((na-1)*sa^2+(nb-1)*sb^2)/(na+nb-2)))

tcalc =
    4.7364
```

We can now compare the calculated `tcalc` value of 4.7364 with the critical `tcrit` value. Again, this can be accomplished using the function `tinv` at a 5% significance level. The function `tinv` yields the inverse of the *t* distribution function with `na-nb-2` degrees of freedom at the 5% significance level. This is again a two-sample *t*-test, i.e., the means are not equal. Computing the two-tailed critical `tcrit` value by entering 1–0.05/2 yields the upper (positive) `tcrit` value that we compare with the absolute value of the difference between the means.

```
tcrit = tinv(1-0.05/2,na+nb-2)

tcrit =
    1.9803
```

Since the `tcalc` value calculated from the data is now larger than the critical `tcrit` value, we can reject the null hypothesis and conclude that the means are not identical at a 5% significance level. Alternatively, we can apply the function `ttest2(x,y,alpha)` to the two independent samples `corg1` and `corg2` at an `alpha=0.05` or a 5% significance level. The command

```
[h,p,ci,stats] = ttest2(corg1,corg2,0.05)
```

yields

```
h =
     1
```

www.it-ebooks.info

```
p =
    6.1183e-06

ci =
    0.7011
    1.7086

stats =
    tstat: 4.7364
       df: 118
       sd: 1.3933
```

The result `h=1` suggests that we can reject the null hypothesis. The *p*-value is extremely low and very close to zero suggesting that the null hypothesis is very unlikely to be true. The 95% confidence interval on the mean is [0.7011,1.7086], which again includes the theoretical difference between the means of 25.5–24.3=1.2.

3.8 The F-Test

The two-sample *F*-test by Snedecor and Cochran (1989) compares the variances s_a^2 and s_b^2 of two distributions, where $s_a^2 > s_b^2$. An example is the comparison of the natural heterogeneity of two samples based on replicated measurements. The sample sizes n_a and n_b should be above 30. Both the sample and population distributions must be Gaussian. The appropriate test statistic with which to compare the variances is then

$$\hat{F} = \frac{s_a^2}{s_b^2}$$

The two variances are significantly different (i.e., we can reject the null hypothesis without another cause) if the measured *F* value is higher than the critical *F* value, which will in turn depend on the number of degrees of freedom $\Phi_a = n_a - 1$ and $\Phi_b = n_b - 1$, respectively, and the significance level α. The one-sample *F*-test, in contrast, virtually performs a χ^2-test of the hypothesis that the data come from a normal distribution with a specific variance (see Section 3.9). We first apply the two-sample *F*-test to two distributions with very similar standard deviations of 1.2550 and 1.2097.

```
clear

load('organicmatter_four.mat');
```

The quantity `F` is the quotient of the larger variance divided by the smaller variance. We can now compute the standard deviations, where

www.it-ebooks.info

```
s1 = std(corg1)
s2 = std(corg2)
```

which yields

```
s1 =
    1.2550

s2 =
    1.2097
```

The F distribution has two parameters, `df1` and `df2`, which represent the number of observations in each of the distributions reduced by one, where

```
df1 = length(corg1) - 1
df2 = length(corg2) - 1
```

which yields

```
df1 =
    59

df2 =
    59
```

Next we sort the standard deviations by their absolute values,

```
if s1 > s2
  slarger  = s1
  ssmaller = s2
else
  slarger  = s2
  ssmaller = s1
end
```

and get

```
slarger =
    1.2550

ssmaller =
    1.2097
```

We now compare the calculated `F` value with the critical `F` value. This can be accomplished using the function `finv` at a significance level of 0.05 (or 5%). The function `finv` returns the inverse of the F distribution function with `df1` and `df2` degrees of freedom, at the 5% significance level. This is a two-tailed test and we must therefore divide the p-value of 0.05 by two. Typing

```
Fcalc = slarger^2 / ssmaller^2
Fcrit = finv(1-0.05/2,df1,df2)
```

www.it-ebooks.info

yields

```
Fcalc =
    1.0762

Fcrit =
    1.6741
```

Since the `F` value calculated from the data is smaller than the critical `F` value, we cannot reject the null hypothesis without another cause. We conclude therefore that the variances are identical at a 5% significance level. Alternatively, we can apply the function `vartest2(x,y,alpha)` to the two independent samples `corg1` and `corg2` at an `alpha=0.05` or a 5% significance level. MATLAB also provides a one-sample variance test `vartest(x,variance)` analogous to the one-sample *t*-test discussed in the previous section. The one-sample variance test, however, virtually performs a χ^2-test of the hypothesis that the data in the vector `x` come from a normal distribution with a variance defined by `variance`. The χ^2-test is introduced in the next section. The command

```
[h,p,ci,stats] = vartest2(corg1,corg2,0.05)
```

yields

```
h =
     0

p =
    0.7787

ci =
    0.6429
    1.8018

stats =
    fstat: 1.0762
      df1: 59
      df2: 59
```

The result `h=0` means that we cannot reject the null hypothesis without another cause at a 5% significance level. The *p*-value of 0.7787 or ~78% (which is much greater than the significance level) means that the chances of observing more extreme values of *F* than the value in this example from similar experiments would be 7,787 in 10,000. A 95% confidence interval is [−0.6429,1.8018], which includes the theoretical (and hypothesized) ratio `var(corg1)/var(corg2)` of $1.2550^2/1.2097^2=1.0762$.

We now apply this test to two distributions with very different standard deviations, 1.8799 and 1.2939.

www.it-ebooks.info

```
clear

load('organicmatter_five.mat');
```

We again compare the calculated `Fcalc` value with the critical `Fcrit` value at a 5% significance level, using the function `finv` to compute `Fcrit`.

```
s1 = std(corg1);
s2 = std(corg2);

df1 = length(corg1) - 1;
df2 = length(corg2) - 1;

if s1 > s2
  slarger  = s1;
  ssmaller = s2;
else
  slarger  = s2;
  ssmaller = s1;
end

Fcalc = slarger^2 / ssmaller^2

Fcrit = finv(1-0.05/2,df1,df2)
```

and get

```
Fcalc =
    3.4967

Fcrit =
    1.6741
```

Since the `Fcalc` value calculated from the data is now larger than the critical `Fcrit` value, we can reject the null hypothesis. The variances are therefore different at a 5% significance level.

Alternatively, we can apply the function `vartest2(x,y,alpha)`, performing a two-sample *F*-test on the two independent samples `corg1` and `corg2` at an `alpha=0.05` or a 5% significance level.

```
[h,p,ci,stats] = vartest2(corg1,corg2,0.05)
```

yields

```
h =
     1

p =
   3.4153e-06

ci =
    2.0887
```

www.it-ebooks.info

```
    5.8539

stats =
    fstat: 3.4967
      df1: 59
      df2: 59
```

The result `h=1` suggests that we can reject the null hypothesis. The *p*-value is extremely low and very close to zero suggesting that the null hypothesis is very unlikely. The 95% confidence interval is [2.0887,5.8539], which again includes the theoretical ratio `var(corg1)/var(corg2)` of $5.0717^2/1.4504^2=3.4967$.

3.9 The χ^2-Test

The χ^2-test introduced by Karl Pearson (1900) involves the comparison of distributions, allowing two distributions to be tested for derivation from the same population. This test is independent of the distribution that is being used and can therefore be used to test the hypothesis that the observations were drawn from a specific theoretical distribution.

Let us assume that we have a data set that consists of multiple chemical measurements from a sedimentary unit. We could use the χ^2-test to test the null hypothesis that these measurements can be described by a Gaussian distribution with a typical central value and a random dispersion around it. The n data are grouped in K classes, where n should be above 30. The frequencies within the classes O_k should not be lower than four and should certainly never be zero. The appropriate test statistic is then

$$\hat{\chi}^2 = \sum_{k=1}^{K} \frac{\left(O_k - E_k\right)^2}{E_k}$$

where E_k are the frequencies expected from the theoretical distribution (Fig. 3.14). The null hypothesis can be rejected if the measured χ^2 value is higher than the critical χ^2 value, which depends on the number of degrees of freedom $\Phi=K-Z$, where K is the number of classes and Z is the number of parameters describing the theoretical distribution plus the number of variables (for instance, $Z=2+1$ for the mean and the variance from a Gaussian distribution of a data set for a single variable, and $Z=1+1$ for a Poisson distribution for a single variable).

As an example we can test the hypothesis that our organic carbon measurements contained in *organicmatter_one.txt* follow a Gaussian distribution. We must first load the data into the workspace and compute the frequency distribution `n_obs` for the data measurements using eight classes.

www.it-ebooks.info

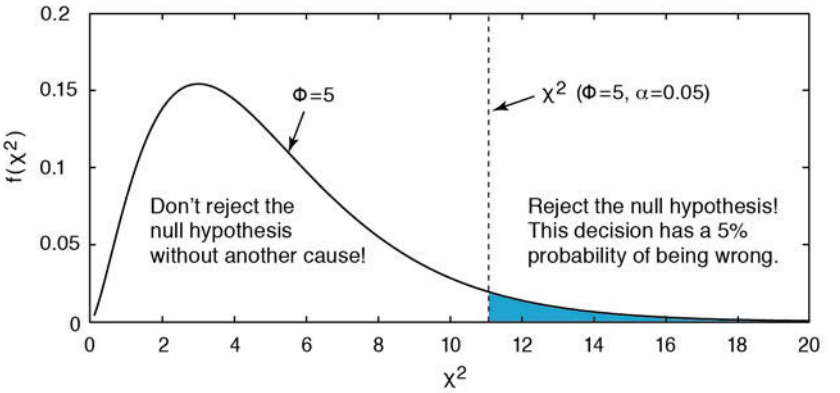


Fig. 3.14 Principles of a χ^2-test. The alternative hypothesis that the two distributions are different can be rejected if the measured χ^2 is lower than the critical χ^2. χ^2 depends on $\Phi=K-Z$, where K is the number of classes and Z is the number of parameters describing the theoretical distribution plus the number of variables. In the example the critical $\chi^2(\Phi=5, \alpha=0.05)$ is 11.0705. Since the measured $\chi^2= 5.7602$ is below the critical χ^2, we cannot reject the null hypothesis. In our example we can conclude that the sample distribution is not significantly different from a Gaussian distribution.

```
clear

corg = load('organicmatter_one.txt');

h = histogram(corg,8);
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1);
n_obs = h.Values;
```

We then use the function `normpdf` to create the expected frequency distribution `n_exp` with the mean and standard deviation of the data in `corg`.

```
n_exp = normpdf(v,mean(corg),std(corg));
```

The data need to be scaled so that they are similar to the original data set.

```
n_exp = n_exp / sum(n_exp);
n_exp = sum(n_obs) * n_exp;
```

The first command normalizes the observed frequencies `n_obs` to a total of one. The second command scales the expected frequencies `n_exp` to the sum of `n_obs`. We can now display both histograms for comparison.

```
subplot(1,2,1), bar(v,n_obs,'r')
subplot(1,2,2), bar(v,n_exp,'b')
```

An alternative way of plotting the data in `corg` is to use a normal probability plot.

www.it-ebooks.info

```
normplot(corg)
```

The function `normplot` plots the data in `corg` with the `+` symbol superimposed by a line joining the first and third quartiles of each column of `corg`, which is a robust linear fit of the sample order statistics. The line is extrapolated in each direction beyond the quartiles. If the data in `corg` are indeed normally-distributed, they will all fall on the line. In our example the data seem to agree well with the line except for the tails of the normal distribution.

Visual inspection of these plots reveals that the empirical distribution is similar to the theoretical distribution. It is, however, advisable to use a more quantitative approach to test the hypothesis of similarity. The χ^2-test explores the squared differences between the observed and expected frequencies. The quantity `chi2calc` is the sum of the squared differences divided by the expected frequencies.

```
chi2calc = sum((n_obs - n_exp).^2 ./ n_exp)

chi2calc =
    5.7602
```

The critical `chi2crit` value can be calculated using `chi2inv` value. The χ^2-test requires the number of degrees of freedom Φ. In our example we test the hypothesis that the data have a Gaussian distribution, i.e., we estimate the two parameters μ and σ. The number of degrees of freedom is Φ=8–(2+1)=5. We can now test our hypothesis at a 5% significance level. The function `chi2inv` computes the inverse of the χ^2 CDF with parameters specified by Φ for the corresponding probabilities in *p*.

```
chi2crit = chi2inv(1-0.05,5)

chi2crit =
    11.0705
```

Since the critical `chi2crit` value of 11.0705 is well above the measured `chi2calc` value of 5.7602, we cannot reject the null hypothesis without another cause. We can therefore conclude that our data follow a Gaussian distribution. Alternatively, we can apply the function `chi2gof(x)` to the sample. The command

```
[h,p] = chi2gof(corg)
```

yields

```
h =
     0
```

www.it-ebooks.info

```
p =
    0.6244

stats =
    chi2stat: 2.6136
          df: 4
       edges: [1x8 double]
           O: [8 8 5 10 10 9 10]
           E: [1x7 double]
```

The function automatically bins the data into seven classes. The result `h=0` means that we cannot reject the null hypothesis without another cause at a 5% significance level. The *p*-value of 0.6244 or ~62% (which is much greater than the significance level) means that the chances of observing either the same result or a more extreme result from similar experiments in which the null hypothesis is true would be 6,244 in 10,000. The structure array `stats` contains the calculated χ^2 value, which is 2.6136 and differs from our result of 6.2489 due to the different number of classes. The array `stats` also contains the number of degrees of freedom Φ=7–(2+1)=4, the eight edges of the seven classes automatically defined by the function `chi2gof`, and the observed and expected frequencies of the distribution.

3.10 The Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov (K-S) test introduced by Andrei N. Kolmogorow and Nikolai W. Smirnov is similar to the χ^2-test in that it also involves the comparison of distributions, allowing two distributions to be tested for derivation from the same population (Kolmogorow 1933, Smirnov 1939). This test is independent of the type of distribution that is being used and can therefore be used to test the hypothesis that the observations were drawn from a specific theoretical distribution.

Let us again assume that we have a data set that consists of multiple chemical measurements from a sedimentary unit. We can use the K-S test to test the null hypothesis that these measurements can be described by a Gaussian distribution with a typical central value (the mean) and a random dispersion around the mean (the standard deviation). The appropriate test statistic is then

$$KS = \max\left|F_n(x) - F(x)\right|$$

where $F_n(x)$ is the empirical distribution function of the n measurements and $F(x)$ is the cumulative distribution function expected from the theoretical distribution. The null hypothesis can be rejected if the measured KS value is

www.it-ebooks.info

higher than the critical *KS* value.

As an example we can again test the hypothesis that our organic carbon measurements contained in *organicmatter_one.txt* follow a Gaussian distribution. We must first load and standardize the data to have zero mean and unit standard deviation.

```
clear

corg = load('organicmatter_one.txt');
corg = (corg-mean(corg))/std(corg);
```

We then compute the empirical distribution function `cn_obs` of `corg` evaluated at the points `x` by using the function `ecdf`

```
[cn_obs,x] = ecdf(corg);
```

We then use the function `normcdf` to create the cumulative distribution function expected from the theoretical distribution `cn_exp` with a mean of zero and a standard deviation of one.

```
cn_exp = normcdf(x,0,1);
```

The test statistic is the maximum difference between the two cumulative distribution functions `cn_obs` and `cn_exp`

```
kscal = max(cn_obs-normcdf(x,0,1))
```

which yields

```
kscalc =
    0.0757
```

We can compare the two cumulative distribution functions in a plot by typing

```
plot(x,cn_obs,'b'), hold on
plot(x,normcdf(x,0,1),'r')
plot(x(find((cn_obs-normcdf(x,0,1))== ...
    max(cn_obs-normcdf(x,0,1))))*ones(2,1),[0 1],'k:')
```

The vertical black line marks the location of the maximum difference between the two cumulative distributions. The critical `kscalc` values are solutions of an *n*-th order polynomial, which can be obtained from Table 3.1 (O'Connor and Kleyner 2012). For sample sizes larger than 40 and a significance level of 0.05 (or 5%) we calculate

```
kscrit = 1.36/length(corg)^0.5
```

which yields

www.it-ebooks.info

```
kscrit =
    0.1756
```

Since the critical `kscrit` value of 0.1756 is well above the measured `kscalc` value of 0.0757, we cannot reject the null hypothesis without another cause. We can therefore conclude that our data follow a Gaussian distribution. Alternatively, we can apply the function `kstest` to the sample. Typing

Table 3.1 Critical values of KS for the Kolmogorov-Smirnov test (O'Connor and Kleyner 2012).

	Level of Significance, α			
N	0.10	0.05	0.02	0.01
1	0.95000	0.97500	0.99000	0.99500
2	0.77639	0.84189	0.90000	0.92929
3	0.63604	0.70760	0.78456	0.82900
4	0.56522	0.62394	0.68887	0.73424
5	0.50945	0.56328	0.62718	0.66853
6	0.46799	0.51926	0.57741	0.61661
7	0.43607	0.48342	0.53844	0.57581
8	0.40962	0.45427	0.50654	0.54179
9	0.38746	0.43001	0.47960	0.51332
10	0.36866	0.40925	0.45662	0.48893
11	0.35242	0.39122	0.43670	0.46770
12	0.33815	0.37543	0.41918	0.44905
13	0.32549	0.36143	0.40362	0.43247
14	0.31417	0.34890	0.38970	0.41762
15	0.30397	0.33760	0.37713	0.40420
16	0.29472	0.32733	0.36571	0.39201
17	0.28627	0.31796	0.35528	0.38086
18	0.27851	0.30936	0.34569	0.37062
19	0.27136	0.30143	0.33685	0.36117
20	0.26473	0.29408	0.32866	0.35241
21	0.25858	0.28724	0.32104	0.34427
22	0.25283	0.28087	0.31394	0.33666
23	0.24746	0.27490	0.30728	0.32954
24	0.24242	0.26931	0.30104	0.32286
25	0.23768	0.26404	0.29516	0.31657
26	0.23320	0.25907	0.28962	0.31064
27	0.22898	0.25438	0.28438	0.30502
28	0.22497	0.24993	0.27942	0.29971
29	0.22117	0.24571	0.27471	0.29466
30	0.21756	0.24170	0.27023	0.28987
31	0.21412	0.23788	0.26596	0.28530
32	0.21085	0.23424	0.26189	0.28094
33	0.20771	0.23076	0.25801	0.27677
34	0.20472	0.22743	0.25429	0.27279
35	0.20185	0.22425	0.26073	0.26897
36	0.19910	0.22119	0.24732	0.26532
37	0.19646	0.21826	0.24404	0.26180
38	0.19392	0.21544	0.24089	0.25843
39	0.19148	0.21273	0.23786	0.25518
40	0.18913	0.21012	0.23494	0.25205
>40	$1.22/N^{0.5}$	$1.36/N^{0.5}$	$1.51/N^{0.5}$	$1.63/N^{0.5}$

www.it-ebooks.info

```
[h,p,kscalc, kscrit] = kstest(corg)
```

yields

```
h =
     0

p =
    0.8562

kscalc =
    0.0757

kscrit =
    0.1723
```

The result `h=0` means that we cannot reject the null hypothesis without another cause at a 5% significance level. The *p*-value of 0.8562 or ~86% (which is much greater than the significance level) means that the chances of observing either the same result or a more extreme result from similar experiments in which the null hypothesis is true would be 8,562 in 10,000. The output variable `kscalc=0.0757` corresponds to `kscalc` in our experiment without using `kstest`. The output variable `kscrit=1.723` differs slightly from that in Table 3.1 since `kstest` uses a slightly more precise approximation for the critical value for sample sizes larger than 40 from Miller (1956).

3.11 Mann-Whitney Test

The Mann-Whitney test (also known as the Wilcoxon rank-sum test) introduced by Henry B. Mann and Donald R. Whitney (1947), can be used to determine whether two samples come from the same population, e.g., the same lithologic unit (*null hypothesis*), or from two different populations (*alternative hypothesis*). In contrast to the *t*-test, which compares the means of Gaussian distributed data, the Mann-Whitney test compares the medians without requiring a normality assumption for the underlying population, i.e., it is a non-parametric hypothesis test.

The test requires that the samples have similar dispersions. We first combine both sets of measurements (samples 1 and 2) and arrange them together in ascending order. We then sum the ranks of samples 1 and 2, where the sum of all ranks is $R=n(n+1)/2$ with n as the total number of measurements. Published literature is full of different versions of how to calculate the test statistic. Here we use the version that can be found in Hedderich and Sachs (2012, page 484):

www.it-ebooks.info

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2$$

where n_1 and n_2 are the sizes of samples 1 and 2, respectively, and R_1 and R_2 are the sums of the ranks of samples 1 and 2, respectively. The required test statistic U is the smaller of the two variables U_1 and U_2, which we compare with a critical U value that depends on the sample sizes n_1 and n_2, and on the significance level α. Alternatively, we can use the U value to calculate

$$\hat{z} = \frac{\left| U - \frac{n_1 n_2}{2} \right|}{\sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}}$$

if $n_1 \geq 8$ and $n_2 \geq 8$ (Mann and Whitney 1978, Hedderich and Sachs 2012, page 486). The null hypothesis can be rejected if the absolute measured z-value is higher than the absolute critical z-value, which depends on the significance level α (Section 3.4).

In practice, data sets often contain tied values, i.e., some of the values in the sample 1 and/or sample 2 are identical. In this case, the average ranks of the tied values are used instead of the true ranks. This means that the equation for the z-value must be corrected for tied values

$$\hat{z} = \frac{\left| U - \frac{n_1 n_2}{2} \right|}{\sqrt{\left(\frac{n_1 n_2}{S(S-1)} \right) \cdot \left(\frac{S^3 - S}{12} - \sum_{i=1}^{i=r} \frac{t_i^3 - t_i}{12} \right)}}$$

where $S = n_1 + n_2$, r is the number of tied values and t_i is the number of occurrences of the i-th tied value. Again, the null hypothesis can be rejected if the absolute measured z-value is higher than the absolute critical z-value, which depends on the significance level α.

The MATLAB code presented here has been tested with an example contained in the book by Hedderich and Sachs (2012, page 489). The example uses the Mann-Whitney test to test whether two samples (`data1` and `data2`), each consisting of eight measurements with some tied values, come from the same population (*null hypothesis*) or from two different populations (*alternative hypothesis*). We clear the workspace and define two samples,

www.it-ebooks.info

each consisting of eight measurements:

```
clear

data1 = [5 5 8 9 13 13 13 15];
data2 = [3 3 4 5 5 8 10 16];
```

We concatenate the two samples horizontally and sort the measurements in ascending order by typing

```
data = horzcat(data1,data2);
data = sort(data)
```

which yields the output

```
data =
  Columns 1 through 8
     3     3     4     5     5     5     5     8
  Columns 9 through 16
     8     9    10    13    13    13    15    16
```

autowrapping those lines that are longer than the width of the Command Window. We then determine the number of measurements `n1` and `n2` in each sample, which is eight for both of the samples in our example. Then we create a linearly-spaced vector `L` ranging from `1` to `length(data)`.

```
n1 = length(data1);
n2 = length(data2);
L = 1 : length(data)
```

which yields

```
L =
  Columns 1 through 8
     1     2     3     4     5     6     7     8
  Columns 9 through 16
     9    10    11    12    13    14    15    16
```

We next find the unique values `C` in the data. The location vectors `ia` and `ic` help to find `C = data(ia)` and to reconstruct data from `data = C(ic)`.

We then search for tied values in `data`. After we have initialized the variables by setting them all to zero, the tied values in `ic` are located using `find` and are then stored in `dties`, with the number of occurrences recorded in `nties`. Having located the tied values their ranks are averaged and stored in `icf`. We type

```
nties = 0;
dties = 0;
kties = 0;
```

www.it-ebooks.info

```
j = 1;
for i = 1 : length(ia)
    f = find(ic==i);
    if length(f) > 1
        dties(j) = max(data(f));
        nties = nties+1;
        j = j + 1;
    end
    icf(f) = mean(L(f));
end
icf
```

which yields

```
icf =
  Columns 1 through 4
    1.5000    1.5000    3.0000    5.5000
  Columns 5 through 8
    5.5000    5.5000    5.5000    8.5000
  Columns 9 through 12
    8.5000   10.0000   11.0000   13.0000
  Columns 13 through 16
   13.0000   13.0000   15.0000   16.0000
```

As we can see, instead of ranks 1 and 2, the averaged ranks 1.5 and 1.5 are used for the two duplicate values of 3 in the data set. Similarly, the averaged rank 5.5 is used for the value of 5, which occurs four times in the data set, instead of the ranks 4, 5, 6 and 7. The averaged rank 8.5 is used for the value of 8 instead of ranks 8 and 9 and the averaged rank 13 is used for the value of 13 instead of ranks 12, 13 and 14. Instead of using the above code MATLAB provides the function `tiedrank` to compute ranks adjusted for tied values:

```
icf = tiedrank(data);
```

The result obtained using `tiedrank` is `icf`, which is identical to that obtained previously. Then we count the repetitions of each tied value by typing

```
for i = 1 : nties
    kties(i) = sum(data == dties(i));
end
```

and display the values of all three variables by typing

```
nties
dties
kties
```

which yields

```
nties =
     4
```

www.it-ebooks.info

```
dties =
     3     5     8    13

kties =
     2     4     2     3
```

We see that we have four different tied values (`nties=4`), with the actual values stored in `dties` (these being the values `3`, `5`, `8` and `13`, which occur `2`, `4`, `2` and `3` times in the data set, respectively). We now use `ismember` to locate the values in `data1` and `data2` in `data`. The first output from `ismember` for the sample in `data1` is `Lia1`, which contains `1` indicating that the statement that the data in `data1` are found in `data` is true, and the `Lia1` output of `ismember` for the sample in `data1` contains `0` whenever `data1` is not a row in `data`. The second output from `ismember` is the column array `locb1`, which contains the lowest index in `data` for each row in `data1` that is also a row in `data`. We now collect the corresponding ranks in `icf` and sum them up to `R1` and `R2`.

```
[Lia1,locb1] = ismember(data1,data);
r1 = icf(locb1);
R1 = sum(r1)

[Lia2,locb2] = ismember(data2,data);
r2 = icf(locb2);
R2 = sum(r2)
```

which yields

```
R1 =
   83.5000

R2 =
   52.5000
```

We then calculate the test static `U` of the Mann-Whitney test, which is the smaller of `U1` and `U2`

```
U1 = n1*n2 + n1*(n1+1)/2 - R1
U2 = n1*n2 + n2*(n2+1)/2 - R2
U = min([U1 U2])
```

which yields

```
U1 =
   16.5000

U2 =
   47.5000

U =
   16.5000
```

www.it-ebooks.info

We use `U` to calculate the measured z-value, corrected for tied values, by typing

```
t = 0;
S = n1 + n2;
for i = 1 : nties
    t(i) = (kties(i)^3 - kties(i)) / 12;
end
T = sum(t);
zcal = abs(U - n1*n2/2) / ...
    sqrt( (n1*n2/(S*(S-1))) * ((S^3-S)/12 - T))
```

which yields

```
zcalc =
    1.6473
```

This is a two-tailed Mann-Whitney test, i.e., the alternative hypothesis is that the medians are not equal, no matter which is larger. Computing the two-tailed critical `zcrit` value using the function `norminv` for the standard normal distribution (with a mean of zero and a standard deviation of one) by entering 1–0.05/2 yields the upper (positive) `zcrit` value, which we compare with the calculated `zcalc` value

```
zcrit = norminv(1-0.05/2,0,1)
```

which yields

```
zcrit =
    1.9600
```

Since the absolute measured `zcalc` value is 1.6473, which is smaller than the critical `zcrit` value of 1.9600, we cannot reject the null hypothesis. We can therefore conclude that our samples come from the same population. Alternatively, we can use the function `ranksum` to perform a Mann-Whitney test on the same samples:

```
[p,h,stats] = ranksum(data1,data2)
```

which yields

```
P =
    0.1071

H =
     0

STATS =
    ranksum: 83.5000
```

www.it-ebooks.info

The result `h=0` means that we cannot reject the null hypothesis at a 5% significance level. The *p*-value of 0.1071 or ~11% (which is larger than the significance level) means that the chances of observing either the same result or a more extreme result, from similar experiments in which the null hypothesis is true, would be 11 in 100.

We can use the same script to test whether the two samples in *organicmatter_two.mat* come from the same lithological unit (*null hypothesis*) or from two different units (*alternative hypothesis*) without requiring a normality assumption for the underlying population. We clear the workspace and load the data.

```
clear

load('organicmatter_two.mat');
```

Alternatively, we can use the function `ranksum` to perform a Mann-Whitney test on the same samples:

```
[P,H,STATS] = ranksum(corg1,corg2)
```

which yields

```
P =
    0.5906

H =
     0

STATS =

       zval: 0.5380
    ranksum: 3733
```

The result `h=0` means that we cannot reject the null hypothesis at a 5% significance level without another cause. The *p*-value of 0.5906 or ~59% (which is much larger than the significance level) means that the chances of observing either the same result or a more extreme result from similar experiments in which the null hypothesis is true would be 5,906 in 10,000.

3.12 The Ansari-Bradley Test

The Ansari-Bradley test introduced by Abdur R. Ansari and Ralph A. Bradley (1960) can be used to determine whether two samples come from the same distribution (*null hypothesis*), or whether the samples come from distributions with the same median and shape but different dispersions (e.g., variances) (*alternative hypothesis*). In contrast to the *F*-test, which

www.it-ebooks.info

compares the dispersions of normally-distributed data, the Ansari-Bradley test compares dispersions without requiring a normality assumption for the underlying population, i.e., it is a non-parametric hypothesis test.

The test requires that the samples have similar medians, which can be achieved by subtracting the medians from the samples. The test combines both sets of measurements (samples 1 and 2) and arranges them together in ascending order. There are different ways to calculate the test statistic A_n. Here we use the one given by Hedderich and Sachs (2012, page 463)

$$A_n = \sum_{i=1}^{n} \left(\frac{n+1}{2} - \left| i - \frac{n+1}{2} \right| \right) V_i$$

where the value of the indicator function V_i is 1 for values from sample 1 and 0 for values from sample 2. The test statistic is therefore equal to the sum of the absolute values of the deviations from the mean value $(n+1)/2$ (Hedderich and Sachs 2012). For this the data are concatenated and sorted, as in the Mann-Whitney test (Section 3.11), and the smallest and largest values are then assigned rank 1, the second smallest and second largest values are assigned rank 2, and so forth. The smaller An, the larger the dispersion of the values between the two samples 1 and 2. Again, the ranking of the data may also be corrected for tied values, as was previously carried out in the Mann-Whitney test. For $n \leq 20$ we can find the critical values for the text statistic A_n in Table 1 in the open-access article by Ansari and Bradley (1960). For larger values of n we use the standard normal distribution

$$\hat{z} = \frac{A_n - \mu_{A_n}}{\sqrt{\sigma^2_{A_n}}}$$

with

$$\mu_{A_n} = \begin{cases} \frac{1}{4} n_1 (n+2) & \text{if } n \text{ is even} \\ \frac{1}{4} n_1 (n+1)^2 / n & \text{if } n \text{ is odd} \end{cases}$$

$$\sigma^2_{A_n} = \begin{cases} n_1 n_2 (n^2 - 4) / (48(n-1)) & \text{if } n \text{ is even} \\ n_1 n_2 (n+1)(n^2+3) / (48 n^2) & \text{if } n \text{ is odd} \end{cases}$$

www.it-ebooks.info

again from Hedderich and Sachs (2012, page 463). The null hypothesis can be rejected if the absolute measured z-value is higher than the absolute critical z-value, which depends on the significance level α.

Our first example uses the Ansari-Bradley test to test whether two samples `data1` and `data2`, each consisting of eight measurements, come from the same distribution (*null hypothesis*), or from distributions with the same median and shape but different dispersions (*alternative hypothesis*). We first clear the workspace and define two samples.

```
clear

data1 = [7 14 22 36 40 48 49 52];
data2 = [3 5 6 10 17 18 20 39];
```

We next subtract the medians from both data sets as the method requires the medians to be similar.

```
data1 = data1 - median(data1);
data2 = data2 - median(data2);
```

We then concatenate the two samples horizontally and sort the measurements in ascending order by typing

```
data = horzcat(data1,data2);
data = sort(data)
```

which yields

```
data =
  Columns 1 through 5
  -31.0000  -24.0000  -16.0000  -10.5000   -8.5000
  Columns 6 through 10
   -7.5000   -3.5000   -2.0000    2.0000    3.5000
  Columns 11 through 15
    4.5000    6.5000   10.0000   11.0000   14.0000
  Column 16
   25.5000
```

We determine the sample sizes for `data1` and `data2` and then calculate the size `n` of the concatenated samples.

```
n1 = length(data1);
n2 = length(data2);
n = n1 + n2;
```

We then determine the positions of the measurements in `data1` and `data2` within the concatenated and sorted vector `data` using `ismember`. The function `ismember` returns an array of the same size as `data` containing `1` (true) where the elements of `data1` are in `data`, or `0` (false) where they are not. We type

www.it-ebooks.info

```
V1 = ismember(data,data1)
V2 = ismember(data,data2)
```

which yields

```
V1 =
  Columns 1 through 8
     1     1     1     0     0     0     0     1
  Columns 9 through 16
     1     0     0     0     1     1     1     0

V2 =
  Columns 1 through 8
     0     0     0     1     1     1     1     0
  Columns 9 through 16
     0     1     1     1     0     0     0     1
```

Then we create a linearly-spaced vector `L` ranging from `1` to `length(data)`.

```
L = 1 : length(data);
```

We then calculate the test static `An` of the Ansari-Bradley test by typing

```
An = sum(((n+1)/2 - abs(L - (n+1)/2)).* V1)
```

which yields

```
An =
   31
```

According to Table 1 in Ansari and Bradley (1960), the critical value for `n1`=8 and `n2`=8 is 45 at a significance level of 5%. Alternatively, we can use the function `ansaribradley` to perform the Ansari-Bradley test by typing

```
[h,p,stats] = ansaribradley(data1,data2)
```

which yields

```
h =
     0

p =
    0.3507

stats =
        W: 31
    Wstar: -1.0564
```

The result `h=0` means that we cannot reject the null hypothesis without another cause at a 5% significance level. The *p*-value of 0.3507 or ~35% (which is greater than the significance level) means that the chances of observing

www.it-ebooks.info

either the same result or a more extreme result from similar experiments in which the null hypothesis is true would be 3,507 in 10,000.

The second example demonstrates the handling of tied values in a data set with a large sample size (>50 measurements). We create such a data set of 100 measurements by using a random number generator:

```
data1 = 3.4 + rand(1,100);
data2 = 4.3 + rand(1,100);
```

We then replace some values in `data1` and `data2` to introduce replicate (or tied) values.

```
data1(1,50:55) = 2.5;
data2(1,25:28) = 2.5;
```

We first subtract the medians from both data sets as the method requires the medians to be similar.

```
data1 = data1 - median(data1);
data2 = data2 - median(data2);
```

The two samples are then concatenated horizontally and sorted in ascending order by typing

```
data = horzcat(data1,data2);
data = sort(data);
```

For the Ansari-Bradley test we now use the function `tiedrank` with `(data,0,1)`, which computes the ranks from each end so that the smallest and largest values are assigned rank 1, the next smallest and largest are assigned rank 2, and so forth.

```
icf = tiedrank(data,0,1);
```

Then, we calculate the test statistic of the Ansari-Bradley test, which is the sum of the ranks of those measurements that are members of `data1`.

```
An = sum(icf(ismember(data,data1)))
```

which yields

```
An =
        4903
```

Alternatively, we can again use the equation introduced above by typing

```
n1 = length(data1);
n2 = length(data2);
```

www.it-ebooks.info

```
n = n1 + n2;

V1 = ismember(data,data1)
V2 = ismember(data,data2)

An = sum(((n+1)/2 - abs(icf - (n+1)/2)).* V1)
```

which also yields

```
An =
        4903
```

Finally, we calculate the *z*-value using the equations introduced above by typing

```
if mod(n,2) == 0
   display('n is even')
   muAn = 0.25 * n1 * (n+2);
   siAn = n1*n2*(n^2-4)/((48*(n-1)));
else
   display('n is odd')
   muAn = 0.25 * n1 * (n+1)^2/n;
   siAn = n1*n2*(n+1)*(n^2+3)/(48*n^2);
end

zcalc = (An - muAn)/sqrt(siAn)
```

which yields

```
zcalc =
   -0.7184
```

Since the critical *z*-value

```
zcrit = norminv(1-0.05/2,0,1)
```

which yields

```
zcrit =
    1.9600
```

is larger than the calculated *z*-value, we cannot reject the null hypothesis without another cause. Using

```
[h,p,stats] = ansaribradley(data1,data2)
```

yields a similar result

```
h =
     0

p =
```

www.it-ebooks.info

```
    0.4725

stats =
        W: 4903
    Wstar: -0.7184
```

We can use the same script to test whether the two samples in *organicmatter_four.mat* come from the same distribution (*null hypothesis*), or from distributions with the same median and shape but different dispersions (*alternative hypothesis*). We clear the workspace and load the data.

```
clear

load('organicmatter_four.mat');
```

We use the function `ansaribradley` to perform the Ansari-Bradley test by typing

```
[h,p,stats] = ansaribradley(corg1,corg2)
```

which yields

```
h =
     0

p =
    0.7448

stats =
        W: 1861
    Wstar: 0.3254
```

The result `h=0` means that we cannot reject the null hypothesis without another cause at a 5% significance level. The *p*-value of 0.7448 or ~74% (which is greater than the significance level) means that the chances of observing either the same result or a more extreme result from similar experiments in which the null hypothesis is true would be 7,448 in 10,000.

3.13 Distribution Fitting

In Section 3.9 we computed the mean and standard deviation of our sample and designed a normal distribution based on these two parameters. We then used the χ^2-test to test the hypothesis that our data indeed follow a Gaussian or normal distribution. Distribution fitting functions contained in the Statistics Toolbox provide powerful tools for estimating the distributions directly from the data. Distribution fitting functions for supported distributions all end with `fit`, as in `binofit`, or `expfit`. The function to fit normal distributions to

www.it-ebooks.info

the data is `normfit`. To demonstrate the use of this function we first generate 100 synthetic Gaussian-distributed sets of values with a mean of 6.4 and a standard deviation of 1.4.

```
clear

rng(0)
data = 6.4 + 1.4*randn(100,1);
```

We then define the midpoints `v` of nine histogram intervals with the edges `e`, display the results, and calculate the frequency distribution `n`.

```
e = 1.5 : 10.5;
h = histogram(data,e);
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1);
n = h.Values;
```

The function `normfit` yields estimates of the mean, `muhat`, and standard deviation, `sigmahat`, of the normal distribution for the observations in `data`.

```
[muhat,sigmahat] = normfit(data)

muhat =
    6.5723

sigmahat =
    1.6274
```

These values for the mean and the standard deviation are similar to the ones that we defined initially. We can now calculate the probability density function of the normal distribution with the mean `muhat` and standard deviation `sigmahat`, scale the resulting function `y` to same total number of observations in `data`, and plot the result.

```
x = 2 : 1/20 : 10;
y = normpdf(x,muhat,sigmahat);
y = trapz(v,n) * y/trapz(x,y);
bar(v,n), hold on, plot(x,y,'r'), hold off
```

Alternatively, we can use the function `mle` to fit a normal distribution (and also other distributions such as binomial or exponential distributions) to the data. The function `mle(data,'distribution',dist)` computes parameter estimates for the distribution specified by `dist`. Acceptable strings for `dist` can be obtained by typing `help mle`.

```
phat = mle(data,'distribution','normal');
```

The variable `phat` contains the values of the parameters describing the type of distribution fitted to the data. As before, we can now calculate and scale the

www.it-ebooks.info

probability density function `y`, and display the result.

```
x = 2 : 1/20 : 10;
y = normpdf(x,phat(:,1),phat(:,2));
y = trapz(v,n) * y/trapz(x,y);

bar(v,n), hold on, plot(x,y,'r'), hold off
```

In earth sciences we often encounter mixed distributions. Examples are multimodal grain size distributions (Section 8.8), multiple preferred paleocurrent directions (Section 10.6), and multimodal chemical ages in monazite reflecting multiple episodes of deformation and metamorphism in a mountain belt. Fitting Gaussian mixture distributions to the data aims to determine the means and variances of the individual distributions that combine to produce the mixed distribution. The methods described in this section help to determine the episodes of deformation in a mountain range, or to separate the different paleocurrent directions caused by tidal flow in an ocean basin.

As a synthetic example of Gaussian mixture distributions we generate two sets of 100 random numbers `ya` and `yb` with means of 6.4 and 13.3, respectively, and standard deviations of 1.4 and 1.8, respectively. We then vertically concatenate the series using `vertcat` and store the 200 data values in the variable `data`.

```
clear

rng(0)
ya = 6.4 + 1.4*randn(100,1);
yb = 13.3 + 1.8*randn(100,1);
data = vertcat(ya,yb);
```

Plotting the histogram reveals a bimodal distribution. We can also determine the frequency distribution `n` using `histogram`.

```
e = -0.5 : 30.5;
h = histogram(data,e);
v = h.BinWidth * 0.5 + h.BinEdges(1:end-1);
n = h.Values;
```

We use the function `mgdistribution.fit(data,k)` to fit a Gaussian mixture distribution with `k` components to the data. The function fits the model by maximum likelihood, using the *Expectation-Maximization (EM) algorithm*. The EM algorithm introduced by Arthur Dempster, Nan Laird and Donald Rubin (1977) is an iterative method alternating between performing an expectation step and a maximization step. The expectation step computes an expectation of the logarithmic likelihood with respect to the current estimate

www.it-ebooks.info

of the distribution. The maximization step computes the parameters that maximize the expected logarithmic likelihood computed in the expectation step. The function `mgdistribution.fit` creates an object of the *gmdistribution* class (see Section 2.5 and MATLAB Help on object-oriented programming for details on objects and classes). The function `gmdistribution.fit` treats NaN values as missing data: rows of `data` with NaN values are excluded from the fit. We can now determine the Gaussian mixture distribution with two components in a single dimension.

```
gmfit = gmdistribution.fit(data,2)

Gaussian mixture distribution with 2 components in 1 dimensions
Component 1:
Mixing proportion: 0.513162
Mean:   13.0942

Component 2:
Mixing proportion: 0.486838
Mean:    6.4730
```

Thus we obtain the means and relative mixing proportions of both distributions. In our example both normal distributions with means of 6.4730 and 13.0942 contribute ~50% (~0.49 and ~0.51, respectively) to the mixture distribution. The object `gmfit` contains several layers of information, including the mean `gmfit.mu` and the standard deviation `gmfit.Sigma` that we use to calculate the probability density function `y` of the mixed distribution.

```
x = 0 : 1/30 : 20;
y1 = normpdf(x,gmfit.mu(1,1),gmfit.Sigma(:,:,1));
y2 = normpdf(x,gmfit.mu(2,1),gmfit.Sigma(:,:,2));
```

The object `gmfit` also contains information on the relative mixing proportions of the two distributions in the layer `gmfit.PComponents`. We can use this information to scale `y1` and `y2` to the correction proportions relative to each other.

```
y1 = gmfit.PComponents(1,1) * y1/trapz(x,y1);
y2 = gmfit.PComponents(1,2) * y2/trapz(x,y2);
```

We can now superimpose the two scaled probability density functions `y1` and `y2`, and scale the result `y` to the same integral of 200 as the original data. The integral of the original data is determined using the function `trapz` to perform a trapezoidal numerical integration.

```
y = y1 + y2;
y = trapz(v,n) * y/trapz(x(1:10:end),y(1:10:end));
```

www.it-ebooks.info

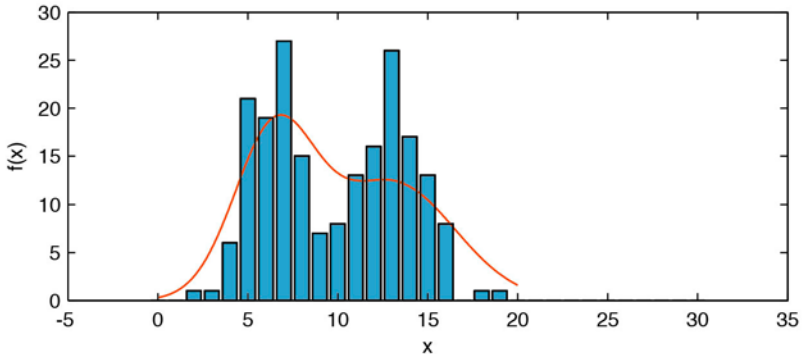


Fig. 3.15 Fitting Gaussian mixture distributions. As a synthetic example of Gaussian mixture distributions we generate two sets of 100 random numbers, with means of 6.4 and 13.3, and standard deviations of 1.4 and 1.8, respectively. The *Expectation-Maximization (EM) algorithm* is used to fit a Gaussian mixture distribution (solid line) with two components to the data (bars).

Finally, we can plot the probability density function `y` on the bar plot of the original histogram of `data`.

```
bar(v,n), hold on, plot(x,y,'r'), hold off
```

We can then see that the Gaussian mixture distribution closely matches the histogram of the data (Fig. 3.15).

Recommended Reading

Ansari AR, Bradley RA (1960) Rank-Sum Tests for Dispersion. Annals of Mathematical Statistics, 31:1174–1189. [Open access]

Bernoulli J (1713) Ars Conjectandi. Reprinted by Ostwalds Klassiker Nr. 107–108. Leipzig 1899

Dempster AP, Laird NM, Rubin DB (1977) Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society, Series B (Methodological) 39(1):1–38

Fisher RA (1935) Design of Experiments. Oliver and Boyd, Edinburgh

Helmert FR (1876) Über die Wahrscheinlichkeit der Potenzsummen der Beobachtungsfehler und über einige damit im Zusammenhang stehende Fragen. Zeitschrift für Mathematik und Physik 21:192–218

Kolmogorov AN (1933) On the Empirical Determination of a Distribution Function. Italian Giornale dell'Istituto Italiano degli Attuari 4:83–91

Mann, HB, Whitney, DR (1947) On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. Annals of Mathematical Statistics 18:50–60

Miller LH (1956) Table of Percentage Points of Kolmogorov Statistics. Journal of the American Statistical Association 51:111–121

O'Connor PDT, Kleyner A. (2012) Practical Reliability Engineering, Fifth Edition. John Wiley & Sons, New York

www.it-ebooks.info

MathWorks (2014) MATLAB Statistics Toolbox – User's Guide. The MathWorks, Inc., Natick, MA
Pearson ES (1990) Student – A Statistical Biography of William Sealy Gosset. In: Plackett RL, with the assistance of Barnard GA, Oxford University Press, Oxford
Pearson K (1900) On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. Philosophical Magazine 50:157–175
Poisson SD (1837) Recherches sur la Probabilité des Jugements en Matière Criminelle et en Matière Civile, Précédées des Regles Générales du Calcul des Probabilités. Bachelier, Imprimeur-Libraire pour les Mathematiques, Paris
Popper K (1959) The Logic of Scientific Discovery. Hutchinson & Co., London
Hedderich J, Sachs L (2012) Angewandte Statistik: Methodensammlung mit R, 14. Auflage. Springer, Berlin Heidelberg New York
Smirnov NV (1939) On the Estimation of the Discrepency between Empirical Curves of Distribution for Two Independent Samples. Bulletin of Moscow 2:3–16
Spiegel MR (2011) Schaum's Outline of Statistics, 4nd Edition. Schaum's Outlines, McGraw-Hill Professional, New York
Student (1908) On the Probable Error of the Mean. Biometrika 6:1–25
Taylor JR (1997) An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements, Second Edition. University Science Books, Sausalito, California
Wilcoxon F (1945) Individual Comparisons by Ranking Methods. Biometrics Bulletin 1:80–83

www.it-ebooks.info

4 Bivariate Statistics

Middle Pleistocene lake-sediment sequence in the Suguta Valley of northern Kenya, consisting mostly of diatomite, clays and silts with intercalated volcanic ashes. The dating of the ash layers and the typical sedimentation rates of deposits can be used to create an age-depth model for the sequence. This age model is in turn required to reconstruct environmental, and hence climatic, variations through time within the area.

4.1 Introduction

Bivariate analysis aims to understand the relationship between two variables, x and y. Examples are the length and width of a fossil, the sodium and potassium content of volcanic glass, and the organic matter content along a sediment core. When the two variables are measured on the same object, x is usually identified as the *independent variable* and y as the *dependent variable*. If both variables have been generated in an experiment, the variable manipulated by the experimenter is described as the independent variable. In some cases neither variable is manipulated and neither is independent.

The methods of bivariate statistics aim to describe the strength of the relationship between the two variables, either by a single parameter such as Pearson's correlation coefficient for linear relationships or by an equation obtained by regression analysis (Fig. 4.1). The equation describing the relationship between x and y can be used to predict the y-response from any arbitrary x within the range of the original data values used for the regression analysis. This is of particular importance if one of the two parameters is difficult to measure. In such a case, the relationship between the two variables is first determined by regression analysis on a small training set of data. The

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_4

www.it-ebooks.info

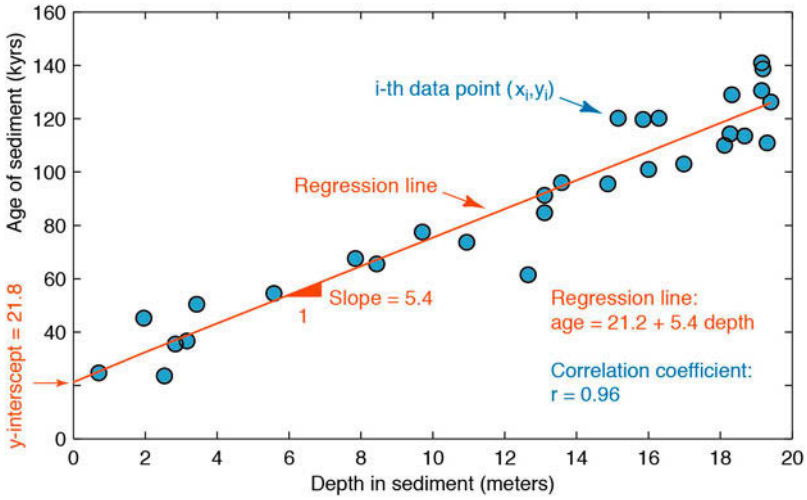


Fig. 4.1 Display of a bivariate data set. The thirty data points represent the *age* of a sediment (in kiloyears before present) at a certain *depth* (in meters) below the sediment-water interface. The combined distribution of the two variables suggests a linear relationship between *age* and *depth*, i.e., the rate of increase in the sediment age with depth is constant. A Pearson's correlation coefficient (explained in the text) of r=0.96 supports a strong linear interdependency between the two variables. Linear regression yields the equation *age*=21.2+5.4 *depth*, indicating an increase in sediment age of 5.4 kyrs per meter of sediment depth (the slope of the regression line).

regression equation can then be used to calculate the second parameter.

This chapter first introduces correlation coefficients (Section 4.2), and then explains the widely-used methods of linear and nonlinear regression analysis (Sections 4.3, 4.9 and 4.10). A selection of other methods that are also used to assess the uncertainties in regression analysis are also explained (Sections 4.4 to 4.8). All methods are illustrated by means of synthetic examples since these provide an excellent means of assessing the final outcome. We use the Statistics Toolbox (MathWorks 2014), which contains all the necessary routines for bivariate analysis.

4.2 Correlation Coefficients

Correlation coefficients are often used in the early stages of bivariate statistics. They provide only a very rough estimate of a rectilinear trend in a bivariate data set. Unfortunately, the literature is full of examples where the importance of correlation coefficients is overestimated, or where outliers in the data set lead to an extremely biased estimation of the population correlation coefficient.

www.it-ebooks.info

The most popular correlation coefficient is *Pearson's linear product-moment correlation coefficient* ρ (Pearson 1895) (Fig. 4.2). We estimate the population's correlation coefficient ρ from the sample data, i.e., we compute the sample correlation coefficient *r*, which is defined as

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{(n-1)s_x s_y}$$

where n is the number of pairs xy of data points, s_x and s_y are the univariate standard deviations. The numerator of Pearson's correlation coefficient is known as the *corrected sum of products* of the bivariate data set. Dividing the numerator by $(n-1)$ yields the *covariance*

$$cov_{xy} = \frac{1}{(n-1)}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$$

which is the summed products of deviations of the data from the sample means, divided by $(n-1)$. The covariance is a widely-used measure in bivariate statistics although it has the disadvantage of being dependent on the dimensions of the data. Dividing the covariance by the univariate standard deviations removes this dependency and leads to Pearson's correlation coefficient.

A popular way to test the significance of Pearson's correlation coefficient is to determine the probability of an *r*-value for a random sample from a population with a $\rho=0$. The significance of the correlation coefficient can be estimated using a *t*-statistic

$$t = r\sqrt{\frac{n-2}{1-r^2}}$$

The correlation coefficient is significant if the calculated *t* is greater than the critical *t* (n-2 degrees of freedom, $\alpha=0.05$). This test is, however, only valid if both variables are Gaussian distributed.

Pearson's correlation coefficient is very sensitive to disturbances in the bivariate data set. Several alternatives exist to Pearson's correlation coefficient, such as *Spearman's rank correlation coefficient* proposed by the English psychologist Charles Spearman (1863–1945). Spearman's coefficient can be used to measure statistical dependence between two variables without requiring a normality assumption for the underlying population, i.e., it is a

www.it-ebooks.info

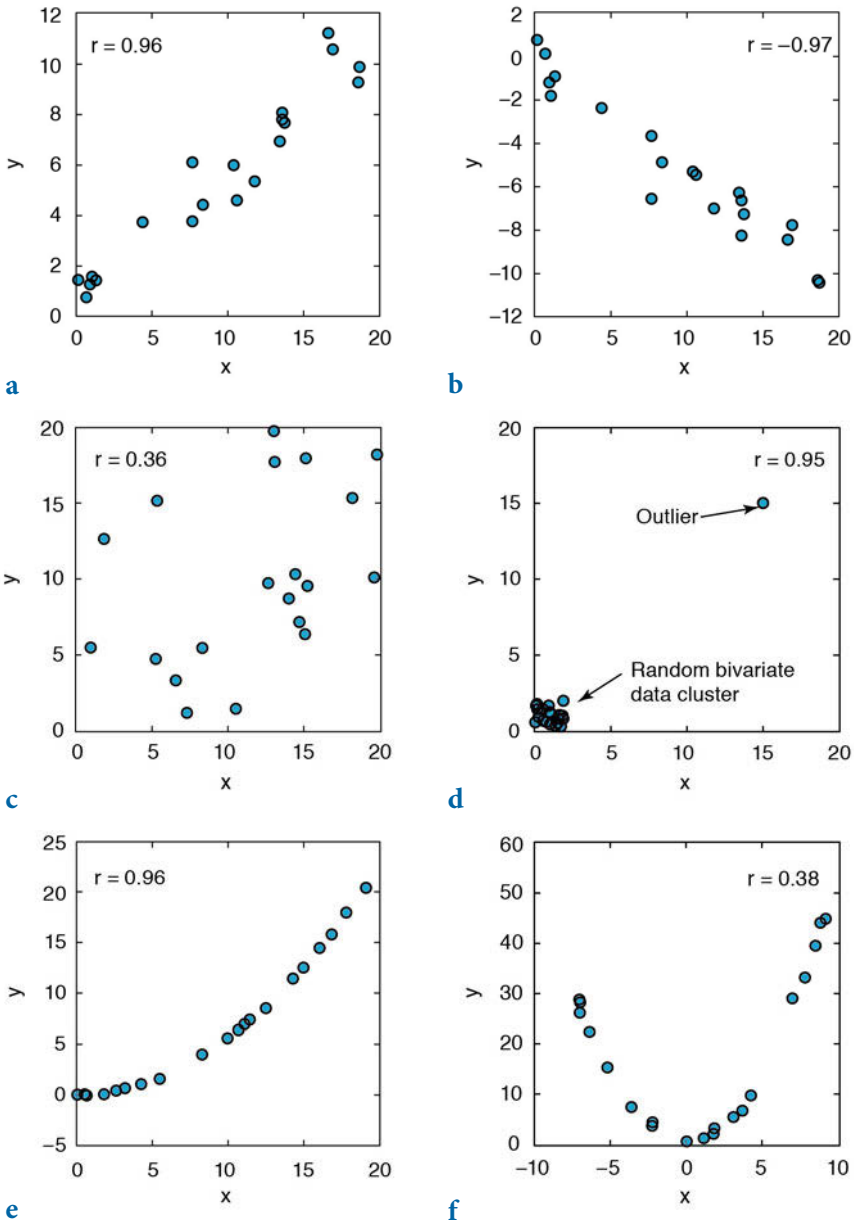


Fig. 4.2 Pearson's correlation coefficent *r* for various sample data sets. **a-b** Positive and negative linear correlation, **c** random scatter with no linear correlation, **d** an outlier causing a misleading value of *r*, **e** curvilinear relationship causing a high *r* since the curve is close to a straight line, **f** curvilinear relationship clearly not described by *r*.

www.it-ebooks.info

non-parametric measure of correlation (Spearman 1904, 1910). Furthermore, since it uses the ranks of the values in x and y rather than their numerical values, it can be used to find correlations in nonlinear data, and even in non-numerical data such as fossil names or rock types in stratigraphic sequences. Having replaced the numerical values in x and y by their ranks (whereby multiple values in x and y are replaced by their respective average ranks) the sample Spearman's rank correlation coefficient is defined as

$$r_{xy} = \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}$$

where d_i is the difference between the ranks of the two variables. Since this correlation coefficient is based on ranks rather than numerical values it is less sensitive to outliers than Pearson's correlation coefficient.

Another alternative to Pearson's correlation coefficient is the *Kendall's tau rank correlation coefficient* proposed by the British statistician Maurice Kendall (1907–1983). This is also a non-parametric measure of correlation, similar to the Spearman's rank correlation coefficient (Kendall 1938). The Kendall's tau rank correlation coefficient compares the ranks of the numerical values in x and y, which means a total of 0.5 $n(n–1)$ pairs to compare. Pairs of observations (x_i,y_i) and (x_j,y_j) are said to be concordant if the ranks for both observations are the same, and discordant if they are not. The sample Kendall's tau rank correlation coefficient is defined as

$$\tau = \frac{(P-Q)}{\frac{1}{2}n(n-1)}$$

where P is the number of concordant pairs and Q is the number of discordant pairs. Kendall's correlation coefficient typically has a lower value than Spearman's correlation coefficient.

The following example illustrates the use of the correlation coefficients and highlights the potential pitfalls when using these measures of linear trends. It also describes the resampling methods that can be used to explore the confidence level of the estimate for ρ. The synthetic data consist of two variables, the age of a sediment in kiloyears before present and the depth below the sediment-water interface in meters. The use of synthetic data sets has the advantage that we fully understand the linear model behind the data.

The data are represented as two columns contained in file *agedepth_1*.

www.it-ebooks.info

txt. These data have been generated using a series of thirty random levels (in *meters*) below the sediment surface. The linear relationship *age*=5.6 *meters*+20 was used to compute noise-free values for the variable *age*. This is the equation of a straight line with a slope of 5.6 and an intercept with the *y*-axis of 20. Some Gaussian noise with a mean of zero and a standard deviation of 10 has been added to the *age* data.

```
clear

rng(0)
meters = 20 * rand(30,1);
age =  5.6 * meters + 20;
age = age + 10.* randn(length(meters),1);

plot(meters,age,'o')
axis([0 20 0 140])

agedepth(:,1) = meters;
agedepth(:,2) = age;
agedepth = sortrows(agedepth,1);

save agedepth_1.txt agedepth -ascii
```

The synthetic bivariate data set can be loaded from the file *agedepth_1.txt*.

```
clear

agedepth = load('agedepth_1.txt');
```

We then define two new variables, `meters` and `age`, and generate a scatter plot of the data.

```
meters = agedepth(:,1);
age = agedepth(:,2);

plot(meters,age,'o')
axis([0 20 0 140])
```

In the plot, we can observe a strong linear trend suggesting some interdependency between the two variables, `meters` and `age`. This trend can be described by Pearson's correlation coefficient *r*, where *r*=1 indicates a perfect positive correlation (i.e., `age` increases with `meters`), *r*=0 suggests no correlation, and *r*=−1 indicates a perfect negative correlation. We use the function `corrcoef` to compute Pearson's correlation coefficient.

```
corrcoef(meters,age)
```

which results in the output

```
ans =
```

www.it-ebooks.info

```
    1.0000    0.9563
    0.9563    1.0000
```

The function `corrcoef` calculates a matrix of Pearson's correlation coefficients for all possible combinations of the two variables `age` and `meters`. The value of r=0.9563 suggests that the two variables `age` and `meters` are dependent on each other.

Pearson's correlation coefficient is, however, highly sensitive to outliers, as can be illustrated by the following example. Let us generate a normally-distributed cluster of thirty data with a mean of zero and a standard deviation of one. To obtain identical data values, we reset the random number generator by using the integer 10 as seed.

```
clear

rng(10)
x = randn(30,1); y = randn(30,1);

plot(x,y,'o'), axis([-1 20 -1 20]);
```

As expected, the correlation coefficient for these random data is very low.

```
corrcoef(x,y)

ans =
    1.0000    0.0302
    0.0302    1.0000
```

Now we introduce a single outlier to the data set in the form of an exceptionally high `(x,y)` value, in which `x=y`. The correlation coefficient for the bivariate data set including the outlier `(x,y)=(5,5)` is much higher than before.

```
x(31,1) = 5; y(31,1) = 5;

plot(x,y,'o'), axis([-1 20 -1 20]);

corrcoef(x,y)

ans =
    1.0000    0.5022
    0.5022    1.0000
```

Increasing the absolute `(x,y)` values for this outlier results in a dramatic increase in the correlation coefficient.

```
x(31,1) = 10; y(31,1) = 10;

plot(x,y,'o'), axis([-1 20 -1 20]);

corrcoef(x,y)
```

www.it-ebooks.info

```
ans =
    1.0000    0.7981
    0.7981    1.0000
```

and reaches a value close to r=1 if the outlier has a value of `(x,y)=(20,20)`.

```
x(31,1) = 20; y(31,1) = 20;

plot(x,y,'o'), axis([-1 20 -1 20]);

corrcoef(x,y)

ans =
    1.0000    0.9403
    0.9403    1.0000
```

We can compare the sensitivity of Pearson's correlation coefficient with that of Spearman's correlation coefficient and Kendall's correlation coefficient using the function `corr`. In contrast to `corrcoef`, this function does not calculate correlation matrices that we can later use (e.g., in Chapter 9) for calculating correlations within multivariate data sets. We type

```
r_pearson = corr(x,y,'Type','Pearson')
r_spearman = corr(x,y,'Type','Spearman')
r_kendall = corr(x,y,'Type','Kendall')
```

which yields

```
r_pearson =
    0.9403

r_spearman =
    0.1343

r_kendall =
    0.0753
```

and observe that the alternative measures of correlation result in reasonable values, in contrast to the absurd value for Pearson's correlation coefficient that mistakenly suggests a strong interdependency between the variables. Although outliers are easy to identify in a bivariate scatter, erroneous values can easily be overlooked in large multivariate data sets (Chapter 9).

Various methods exist to calculate the significance of Pearson's correlation coefficient. The function `corrcoef` also includes the possibility of evaluating the quality of the result. The p-value is the probability of obtaining a correlation as large as the observed value by random chance, when the true correlation is zero. If the p-value is small, then the correlation coefficient r is significant.

www.it-ebooks.info

```
[r,p] = corrcoef(x,y)

r =
    1.0000    0.9403
    0.9403    1.0000

p =
    1.0000    0.0000
    0.0000    1.0000
```

In our example the p-value is close to zero suggesting that the correlation coefficient is significant. We conclude from this experiment that this particular significance test fails to detect correlations attributed to an outlier. We therefore try an alternative t-test statistic to determine the significance of the correlation between x and y. According to this test, we can reject the null hypothesis that there is no correlation if the calculated t is larger than the critical t (n-2 degrees of freedom, α=0.05).

```
tcalc = r(2,1) * ((length(x)-2)/(1-r(2,1)^2))^0.5
tcrit = tinv(0.95,length(x)-2)

tcalc =
   14.8746

tcrit =
    1.6991
```

This result indeed indicates that we can reject the null hypothesis and therefore there is no correlation. As an alternative to detecting outliers, *resampling schemes* or *surrogates* such as the *bootstrap* or *jackknife* methods represent powerful tools for assessing the statistical significance of the results. These techniques are particularly useful when scanning large multivariate data sets for outliers (see Chapter 9). Resampling schemes repeatedly resample the original data set of n data points, either by choosing n–1 subsamples n times (the jackknife), or by picking an arbitrary set of subsamples with n data points *with replacement* (the bootstrap). The statistics of these subsamples provide better information on the characteristics of the population than the statistical parameters (mean, standard deviation, correlation coefficients) computed from the full data set. The function `bootstrp` allows resampling of our bivariate data set, including the outlier `(x,y)=(20,20)`.

```
rng(0)
rhos1000 = bootstrp(1000,'corrcoef',x,y);
```

This command first resamples the data a thousand times; it then calculates the correlation coefficient for each new subsample and stores the result in the variable `rhos1000`. Since `corrcoef` delivers a 2-by-2 matrix (as mentioned

www.it-ebooks.info

above), `rhos1000` has the dimensions of 1000-by-4, i.e., 1000 values for each element of the 2-by-2 matrix. Plotting the histogram of the 1000 values for the second element, i.e., the correlation coefficient of `(x,y)`, illustrates the dispersion of this parameter with respect to the presence or absence of the outlier. Since the distribution of `rhos1000` contains many empty classes, we use a large number of bins.

```
histogram(rhos1000(:,2),30)
```

The histogram shows a cluster of correlation coefficients at around r=0.1 that follow the normal distribution, and a strong peak close to r=1 (Fig. 4.3). The interpretation of this histogram is relatively straightforward. When the subsample contains the outlier the correlation coefficient is close to one, but subsamples without the outlier yield a very low (close to zero) correlation coefficient suggesting the absence of any strong interdependence between the two variables `x` and `y`.

Bootstrapping therefore provides a simple but powerful tool for either accepting or rejecting our first estimate of the correlation coefficient for the population. The application of the above procedure to the synthetic sediment data yields a clear unimodal Gaussian distribution for the correlation coefficients of the subsamples.

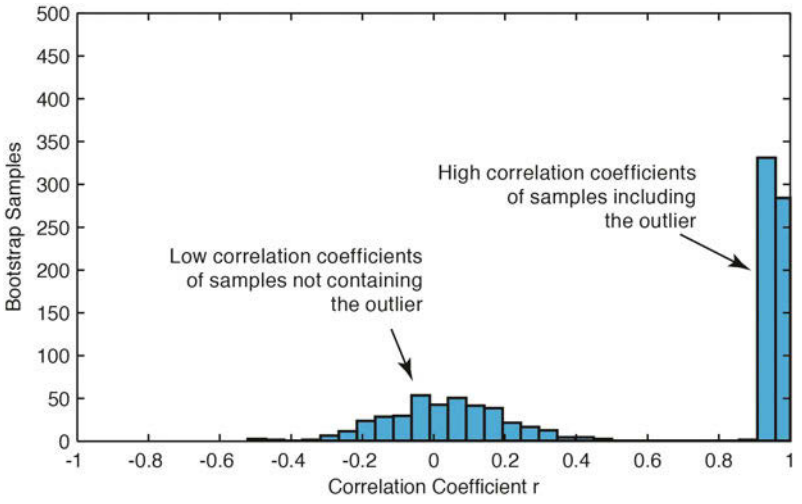


Fig. 4.3 Bootstrap result for Pearson's correlation coefficient r from 1000 subsamples. The histogram shows a roughly normally-distributed cluster of correlation coefficients at around r=0, suggesting that these subsamples do not include the outlier. The strong peak close to r=1, however, suggests that an outlier with high values for the two variables x and y is present in the corresponding subsamples.

www.it-ebooks.info

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);

corrcoef(meters,age)

ans =
    1.0000    0.9563
    0.9563    1.0000

rng(0)
rhos1000 = bootstrp(1000,'corrcoef',meters,age);

histogram(rhos1000(:,2),30)
```

Most of the values for `rhos1000` fall within the interval [0.92,0.98]. Since the correlation coefficients for the resampled data sets (in our example) have an approximately normal distribution, we can use their mean as a good estimate for the true correlation coefficient.

```
mean(rhos1000(:,2))

ans =
    0.9557
```

This value is similar to our first result of r=0.9557, but now we have confidence in the validity of this result. In our example, however, the distribution of the bootstrap estimates of the correlations from the age-depth data is quite skewed, as the upper limited is fixed at one. Nevertheless, the bootstrap method is a valuable tool for assessing the reliability of Pearson's correlation coefficient for bivariate analysis.

4.3 Classical Linear Regression Analysis

Linear regression offers another way of describing the relationship between the two variables x and y. Whereas Pearson's correlation coefficient provides only a rough measure of a linear trend, linear models obtained by regression analysis allow the prediction of arbitrary y-values for any given value of x within the data range. Statistical testing of the significance of the linear model provides some insights into the accuracy of these predictions.

Classical regression assumes that y responds to x and that the entire dispersion in the data set is contained within the y-value (Fig. 4.4). This means that x is then the independent variable (also known as the predictor variable, or the regressor). The values of x are defined by the experimenter

www.it-ebooks.info

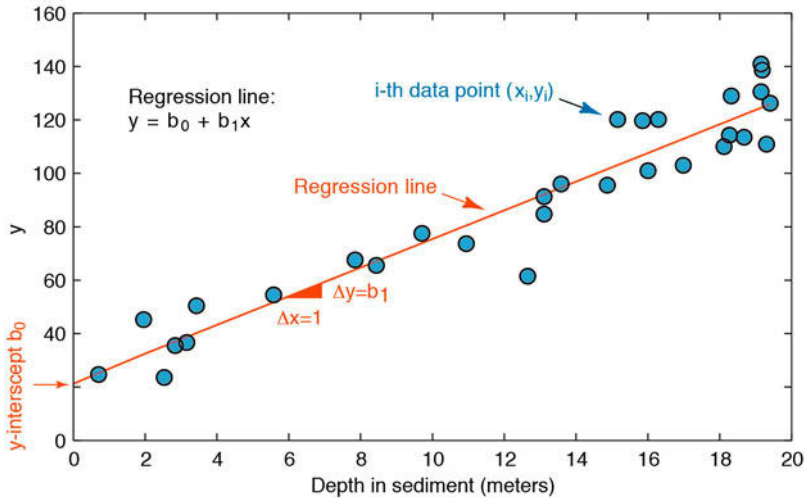


Fig. 4.4 Linear regression. Whereas classical regression minimizes the Δy deviations, reduced major axis regression minimizes the triangular area $0.5*(\Delta x \Delta y)$ between the data points and the regression line, where Δx and Δy are the distances between the predicted and the true x- and y-values. The intercept of the line with the y-axis is b_0, and the slope is b_1. These two parameters define the equation of the regression line.

and are often regarded as being free of errors. An example is the location x within a sediment core from which the variable y has been measured. The dependent variable y contains errors as its magnitude cannot be determined accurately. Linear regression minimizes the deviations Δy between the data points xy and the value y predicted by the best-fit line $y=b_0+b_1x$ using a least-squares criterion. The basic equation for a general linear model is

$$y = b_0 + b_1 x$$

where b_0 and b_1 are the regression coefficients. The value of b_0 is the intercept with the y-axis and b_1 is the slope of the line. The squared sum of the Δy deviations to be minimized is

$$\sum_{i=1}^{n} (\Delta y_i)^2 = \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_i))^2$$

Partial differentiation of the right-hand term in the equation and setting it to zero yields a simple equation for the regression coefficient b_1:

www.it-ebooks.info

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The regression line passes through the data centroid defined by the sample means, and we can therefore compute the other regression coefficient b_0,

$$b_0 = \bar{y} - b_1\bar{x}$$

using the univariate sample means and the slope b_1 computed earlier.

As an example let us again load the synthetic age-depth data from the file *agedepth_1.txt*. We can define two new variables, `meters` and `age`, and generate a scatter plot of the data.

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);
```

A significant linear trend in the bivariate scatter plot, together with a correlation coefficient greater than r=0.9, suggests a strong linear dependence between `meters` and `age`. In geological terms, this implies that the sedimentation rate was constant through time. We now try to fit a linear model to the data that will help us predict the age of the sediment at levels for which we have no age data. The function `polyfit` computes the coefficients of a polynomial $p(x)$ of a specific degree that fits the data y in a least-squared sense. In our example we fit a first degree (linear) polynomial to the data

```
p = polyfit(meters,age,1)

p =
    5.3667   21.7607
```

where `p` is a row vector containing the polynomial coefficients in descending powers. Since we are working with synthetic data we know the values for the slope and the intercept with the y-axis. The estimated slope (5.3667) and the intercept with the y-axis (21.7607) are in good agreement with the true values of 5.6 and 20, respectively. Both the data and the fitted line can be plotted on the same graph.

```
plot(meters,age,'o'), hold on
plot(meters,p(1)*meters+p(2),'r'), hold off
```

www.it-ebooks.info

Instead of using the equation for the regression line we can also use the function `polyval` to calculate the *y*-values.

```
plot(meters,age,'o'), hold on
plot(meters,polyval(p,meters),'r'), hold off
```

Movie 4.1

Both of the functions `polyfit` and `polyval` are incorporated in the GUI function `polytool`.

```
polytool(meters,age)
```

The coefficients `p(x)` and the equation obtained by linear regression can now be used to predict *y*-values for any given *x*-value. However, we can only do this within the depth interval for which the linear model was fitted, i.e., between 0 and 20 meters. As an example the age of the sediment at a depth of 17 meters is given by

```
polyval(p,17)

ans =
   112.9946
```

This result suggests that the sediment at 17 meters depth has an age of ca. 113 kyrs. The goodness-of-fit of the linear model can be determined by calculating error bounds. These are obtained by using an additional output parameter `s` from `polyfit` as an input parameter for `polyconf` to calculate the 95% (`alpha`=0.05) prediction intervals.

```
[p,s] = polyfit(meters,age,1);
[p_age,delta] = polyconf(p,meters,s,'alpha',0.05);

plot(meters,age,'o',meters,p_age,'g-',...
  meters,p_age+delta,'r--',meters,p_age-delta,'r--')
axis([0 20 0 140]), grid on
xlabel('Depth in Sediment (meters)')
ylabel('Age of Sediment (kyrs)')
```

The variable `delta` provides an estimate for the standard deviation of the error in predicting a future observation at `x` by using `p(x)`. Since the `plot` statement does not fit on one line we use an *ellipsis* (three periods, i.e., `...`) followed by *return* or *enter* to indicate that the statement continues on the next line. The plot now shows the data points, and also the regression line and the error bounds of the regression (Fig. 4.5). This graph already provides some valuable information on the quality of the result. However, in many cases a better understanding of the validity of the model is required and more sophisticated methods for testing the confidence in the results are therefore introduced in the following sections.

www.it-ebooks.info

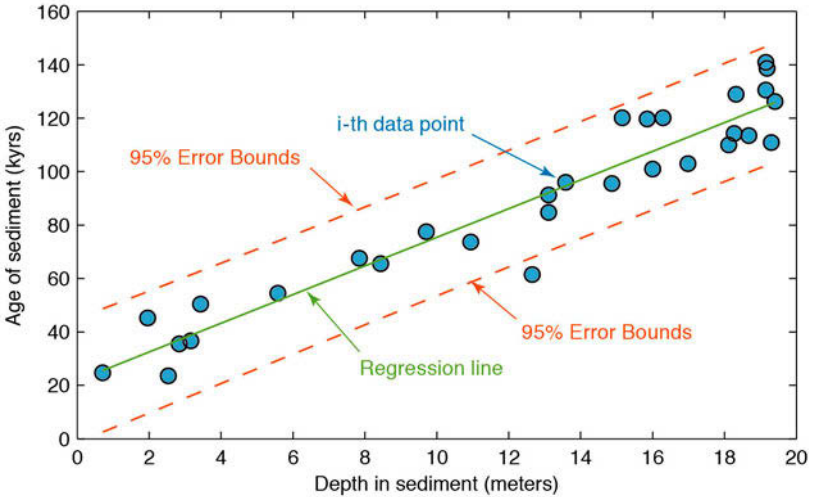


Fig. 4.5 The result of linear regression. The plot shows the original data points (circles), the regression line (solid line), and the error bounds (dashed lines) of the regression. Note that the error bounds are actually curved although they seem to be almost straight lines in the example.

4.4 Analyzing the Residuals

When we compare how much the predicted values vary from the actual or observed values, we are performing an analysis of the residuals. The statistics of the residuals provide valuable information on the quality of a model fitted to the data. For instance, a significant trend in the residuals suggests that the model does not fully describe the data. In such cases a more complex model, such as a polynomial of a higher degree, should be fitted to the data. Residuals are ideally purely random, i.e., they are Gaussian distributed with a mean of zero. We therefore test the hypothesis that our residuals are Gaussian distributed by visual inspection of the histogram and by employing a χ^2-test, as introduced in Chapter 3.

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);

p = polyfit(meters,age,1);

res = age - polyval(p,meters);
```

www.it-ebooks.info

Since plotting the residuals does not reveal any obvious pattern of behavior, no more complex model than a straight line should be fitted to the data.

```
plot(meters,res,'o')
```

An alternative way to plot the residuals is as a stem plot using `stem`.

```
subplot(2,1,1)
plot(meters,age,'o'), hold on
plot(meters,p(1)*meters+p(2),'r'), hold off

subplot(2,1,2)
stem(meters,res);
```

To explore the distribution of the residuals we can choose six classes and display the corresponding frequencies.

```
histogram(res,6)
```

The χ^2-test can be used to test the hypothesis that the residuals follow a Gaussian distribution (Section 3.9). We use `chi2gof` to perform the χ^2-test

```
[h,p,stats] = chi2gof(res)
```

which yields

```
h =
     0

p =
   NaN

stats =
    chi2stat: 0.0099
          df: 0
       edges: [-28.2530 -1.0404 3.4950 17.1012]
           O: [14 5 11]
           E: [13.7940 5.1876 11.0184]
```

The result `h=0` means that we cannot reject the null hypothesis without another cause at a 5% significance level. However, the quality of the result is not very good because the sample size of 30 measurements is very small. In such an example, pooling the data could yield very low (or even zero) expected counts. The function `chi2gof` then displays a warning that the χ^2 approximation may not be accurate. In our example we have not defined the number of bins and therefore (according to the documentation) `chi2gof` should use a default of 10 bins for pooling the data. According to Section 3.9, the corresponding number of degrees of freedom should be `10-(2+1)=7` but the output of `chi2gof` indicates `df=NaN` instead. However, if there are not

www.it-ebooks.info

enough degrees of freedom to conduct the test, `chi2gof` returns a *p*-value of `NaN`, as in our example. Alternatively, we use `chi2gof` with 6 bins

```
[h,p,stats] = chi2gof(res,'NBins',6)
```

which yields

```
h =
     0

p =
    0.1481

stats =
    chi2stat: 2.0922
          df: 1
       edges: [-28.2530 -5.5759 1.9832 9.5422 17.1012]
           O: [11 7 5 7]
           E: [8.8282 8.4603 7.3927 5.3187]
```

As indicated by `df=1` and the number of edges (`edges`), `chi2gof` now uses 4 bins instead of 6 bins defined by `NBins=6` in `chi2gof` in order to avoid bins with very low (or even zero) expected counts. The function `chi2gof` then yields a *p*-value of 0.1481 indicating that the result is significant and we cannot reject the null hypothesis that the residuals follow a Gaussian distribution.

4.5 Bootstrap Estimates of the Regression Coefficients

In this section we use the *bootstrap* method to obtain a better estimate of the regression coefficients. As an example we use the function `bootstrp` with 1000 samples (Fig. 4.6).

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);

p = polyfit(meters,age,1);

rng(0)

p_bootstrp = bootstrp(1000,'polyfit',meters,age,1);
```

The statistic of the first coefficient, i.e., the slope of the regression line is

```
histogram(p_bootstrp(:,1),15)

median(p_bootstrp(:,1))
```

www.it-ebooks.info

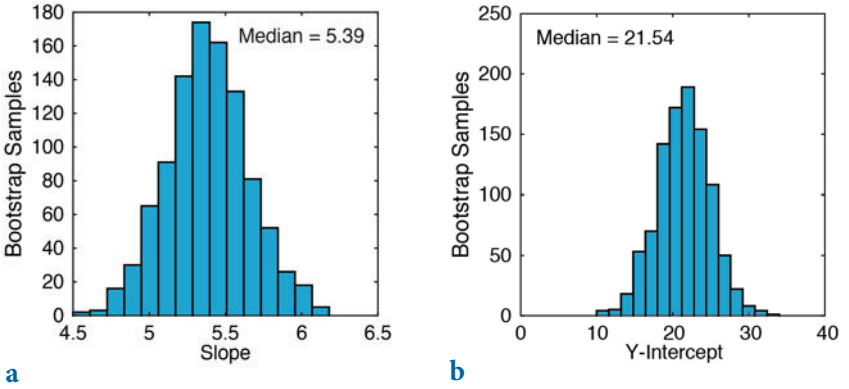


Fig. 4.6 Histogram of **a**, the first (slope of the regression line) and **b**, the second (y-axis intercept of the line) regression coefficient, as estimated from bootstrap resampling. The first coefficient is well constrained, but the second coefficient shows a broad scatter.

```
ans =
    5.3855
```

We use the median instead of the mean since we cannot expect the bootstrap results to be Gaussian distributed. In contrast, the statistic of the second coefficient shows a significant dispersion.

```
histogram(p_bootstrp(:,2),15)

median(p_bootstrp(:,2))

ans =
   21.5374
```

The true values, as used to simulate our data set, are 5.6 for the slope and 20 for the intercept with the y-axis, whereas the corresponding coefficients calculated using `polyfit` were 5.3667 and 21.7607 (Section 4.3).

4.6 Jackknife Estimates of the Regression Coefficients

The *jackknife* method is a resampling technique that is similar to the bootstrap method. From a sample with n data points, n subsamples with n–1 data points are taken. The parameters of interest, e.g., the regression coefficients, are calculated for each of the subsamples. The mean and dispersion of the coefficients are then computed. The disadvantage of this method is the limited number of n subsamples: a jackknife estimate of the regression coefficients is therefore less precise than a bootstrap estimate.
The relevant code for the jackknife is easy to generate:

www.it-ebooks.info

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);

p = polyfit(meters,age,1);

for i = 1 : 30
    j_meters = meters;
    j_age = age;
    j_meters(i) = [];
    j_age(i) = [];
    p(i,:) = polyfit(j_meters,j_age,1);
end
```

The jackknife for subsamples with $n-1=29$ data points can be obtained by a simple `for` loop. The ith data point within each iteration is deleted and regression coefficients are calculated for the remaining data points. The mean of the i subsamples gives an improved estimate of the regression coefficients. As with the bootstrap result, the slope of the regression line (first coefficient) is well defined, whereas the intercept with the y-axis (second coefficient) has a large uncertainty:

```
median(p(:,1))

ans =
    5.3663
```

compared to 5.3855 calculated by the bootstrap method and

```
median(p(:,2))

ans =
    21.7964
```

compared to 21.5374 from the bootstrap method (Section 4.5). The true values, as before, are 5.6 and 20. The histograms of the jackknife results from 30 subsamples (Fig. 4.7)

```
subplot(1,2,1), histogram(p(:,1)), axis square
subplot(1,2,2), histogram(p(:,2)), axis square
```

do not display such clear distributions for the coefficients as the histograms of the bootstrap estimates. As an alternative to the above method, MATLAB provides the function `jackknife` with which to perform a jackknife experiment.

```
p = jackknife('polyfit',meters,age,1);
```

www.it-ebooks.info

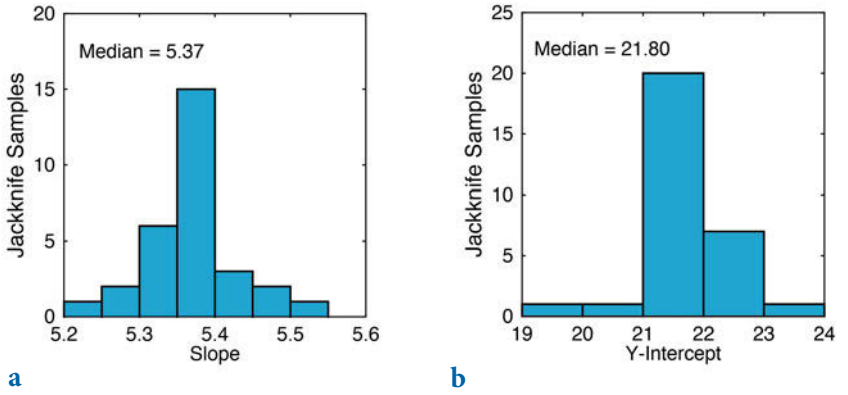


Fig. 4.7 Histogram of **a**, the first (slope of the regression line) and **b**, the second (y-axis intercept of the line) regression coefficient, as estimated from jackknife resampling. Note that the parameters are not as well defined as those from bootstrapping.

```
median(p(:,1))
median(p(:,2))

ans =
    5.3663

ans =
   21.7964

subplot(1,2,1), histogram(p(:,1)), axis square
subplot(1,2,2), histogram(p(:,2)), axis square
```

The results are identical to those obtained using the code introduced above. We have seen therefore that resampling using either the jackknife or the bootstrap method is a simple and valuable way to test the quality of regression models. The next section introduces an alternative approach for quality estimation, which is much more commonly used than the resampling methods.

4.7 Cross Validation

A third method to test the quality of the result of a regression analysis involves *cross validation*. The regression line is computed by using n–1 data points. The nth data point is predicted and the discrepancy between the prediction and the actual value is computed. The mean of the discrepancies between the actual and predicted values is subsequently determined.

In this example the cross validation is computed for n=30 data points.

www.it-ebooks.info

The resulting 30 regression lines, each computed using $n-1=29$ data points, display some dispersion in their slopes and y-axis intercepts.

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);

p = polyfit(meters,age,1);

for i = 1 : 30
    j_meters = meters;
    j_age = age;
    j_meters(i) = [];
    j_age(i) = [];
    p(i,:) = polyfit(j_meters,j_age,1);
    plot(meters,polyval(p(i,:),meters),'r'), hold on
    p_age(i) = polyval(p(i,:),meters(i));
    p_error(i) = p_age(i) - age(i);
end
hold off
```

The prediction error is – in the ideal case – Gaussian distributed with a mean of zero.

```
mean(p_error)

ans =
   -0.0487
```

The standard deviation is an unbiased mean of the deviations of the true data points from the predicted straight line.

```
std(p_error)

ans =
   10.9757
```

Cross validation gives valuable information on the goodness-of-fit of the regression result and can also be used for quality control in other fields, such as those of temporal and spatial prediction (Chapters 5 and 7).

4.8 Reduced Major Axis Regression

In some examples neither variable is manipulated and both can therefore be considered to be independent. In such cases several methods are available to compute a best-fit line that minimizes the distance from both x and y. As an example the method of *reduced major axis* (RMA) minimizes the triangular

www.it-ebooks.info

area 0.5*($\Delta x \Delta y$) between the data points and the regression line, where Δx and Δy are the distances between the predicted values of x and y and the true values of x and y (Fig. 4.4). Although this optimization appears to be complex, it can be shown that the first regression coefficient b_1 (the slope) is simply the ratio of the standard deviations of y and x.

$$b_1 = s_y / s_x$$

As with classical regression, the regression line passes through the data centroid defined by the sample mean. We can therefore compute the second regression coefficient b_0 (the y-intercept),

$$b_0 = \overline{y} - b_1 \overline{x}$$

using the univariate sample means and the slope b_1 computed earlier. Let us again load the age-depth data from the file *agedepth_1.txt* and define two variables, `meters` and `age`. It is assumed that both of the variables contain errors and that the scatter of the data can be explained by dispersions of `meters` and `age`.

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);
```

The above formula is used for computing the slope of the regression line b_1.

```
p(1,1) = std(age)/std(meters)

p =
   5.6117
```

The second coefficient b_0, i.e., the y-axis intercept, can therefore be computed by

```
p(1,2) = mean(age) - p(1,1) * mean(meters)

p =
   5.6117   18.7037
```

The regression line can be plotted by

```
plot(meters,age,'o'), hold on
plot(meters,polyval(p,meters),'r'), hold off
```

www.it-ebooks.info

This linear fit differs slightly from the line obtained from classical regression. Note that the regression line from RMA is *not* the bisector of the lines produced by the *x*-*y* and *y*-*x* classical linear regression analyses, i.e., those produced using either *x* or *y* as an independent variable while computing the regression lines.

4.9 Curvilinear Regression

It is apparent from our previous analysis that a linear regression model provides a good way of describing the scaling properties of the data. However, we may wish to check whether the data could be equally well described by a polynomial fit of a higher degree, for instance by a second degree polynomial:

$$y = b_0 + b_1 x + b_2 x^2$$

To clear the workspace and reload the original data, we type

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);
```

A second degree polynomial can then be fitted by using the function `polyfit`.

```
p = polyfit(meters,age,2)

p =
   0.0589    4.1087   26.0381
```

The first coefficient is close to zero, i.e., has little influence on predictions. The second and third coefficients are similar to those obtained by linear regression. Plotting the data yields a curve that resembles a straight line.

```
plot(meters,age,'o'), hold on
plot(meters,polyval(p,meters),'r'), hold off
```

Let us compute and plot the error bounds obtained by using an optional second output parameter from `polyfit` as an input parameter to `polyval`.

```
[p,s] = polyfit(meters,age,2);
[p_age,delta] = polyval(p,meters,s);
```

As before, this code uses an interval of $\pm 2s$, corresponding to a 95% confidence interval. Using `polyfit` not only yields the polynomial coefficients `p`, but also a structure `s` for use with `polyval` to obtain error bounds for the predictions.

www.it-ebooks.info

The variable `delta` is an estimate of the standard deviation of the prediction error of a future observation at `x` by `p(x)`. We then plot the results:

```
plot(meters,age,'o',meters,p_age,'g-',...
   meters,p_age+2*delta,'r', meters,p_age-2*delta,'r')
axis([0 20 0 140]), grid on
xlabel('Depth in Sediment (meters)')
ylabel('Age of Sediment (kyrs)')
```

We now use another synthetic data set that we generate using a quadratic relationship between meters and age.

```
clear

rng(0)
meters = 20 * rand(30,1);
age =  1.6 * meters.^2 - 1.1 * meters + 50;
age = age + 40.* randn(length(meters),1);

plot(meters,age,'o')

agedepth(:,1) = meters;
agedepth(:,2) = age;

agedepth = sortrows(agedepth,1);

save agedepth_2.txt agedepth -ascii
```

The synthetic bivariate data set can be loaded from the file *agedepth_2.txt*.

```
clear

agedepth = load('agedepth_2.txt');

meters = agedepth(:,1);
age = agedepth(:,2);

plot(meters,age,'o')
```

Fitting a second order polynomial yields a convincing regression result.

```
p = polyfit(meters,age,2)

p =
    1.8356   -7.0653   74.1526
```

As shown above, the true values for the three coefficients are +1.8, –7.1 and +74.2, which means that there are some discrepancies between the true values and the coefficients estimated using `polyfit`. The regression curve and the error bounds can be plotted by typing (Fig. 4.8)

```
plot(meters,age,'o'), hold on
```

www.it-ebooks.info

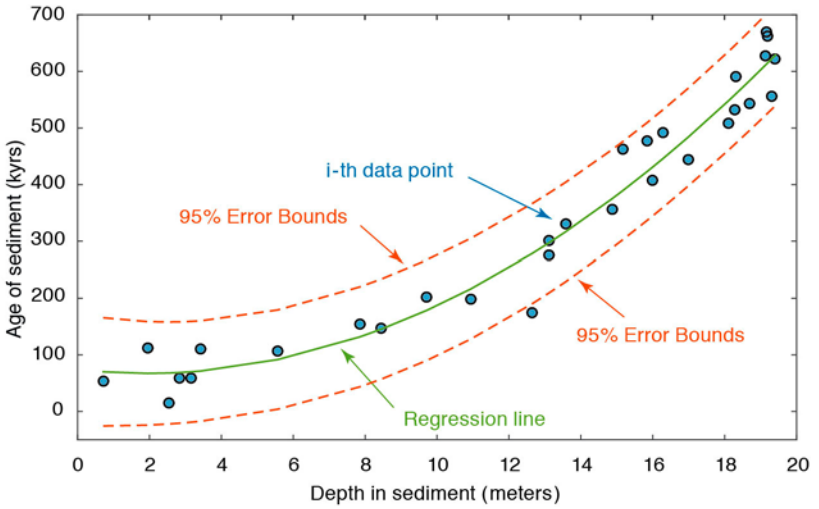


Fig. 4.8 Curvilinear regression from measurements of barium content. The plot shows the original data points (circles), the regression line for a polynomial of degree n=2 (solid line), and the error bounds (dashed lines) of the regression.

```
plot(meters,polyval(p,meters),'r'), hold off

[p,s] = polyfit(meters,age,2);
[p_age,delta] = polyval(p,meters,s);

plot(meters,age,'o',meters,p_age,'g',meters,...
   p_age+2*delta,'r--',meters,p_age-2*delta,'r--')
axis([0 20 -50 700]), grid on
xlabel('Depth in Sediment (meters)')
ylabel('Age of Sediment (kyrs)')
```

The plot shows that the quadratic model for this data is a good one. The quality of the result could again be tested by exploring the residuals, by employing resampling schemes, or by cross validation. Combining regression analysis with one of these methods provides a powerful tool in bivariate data analysis, whereas Pearson's correlation coefficient should be used only as a preliminary test for linear relationships.

4.10 Nonlinear and Weighted Regression

Many bivariate data in earth sciences follow a more complex trend than a simple linear or curvilinear trend. Classic examples for nonlinear trends are the exponential decay of radionuclides, or the exponential growth of algae populations. In such cases, MATLAB provides various tools to fit nonlinear

www.it-ebooks.info

models to the data. An easy-to-use routine to fit such models is nonlinear regression using the function `nlinfit`. To demonstrate the use of `nlinfit` we generate a bivariate data set where one variable is exponentially correlated with a second variable. We first generate evenly-spaced values between 0.3 and 3 at intervals of 0.1 and add some Gaussian noise with a standard deviation of 0.2 to make the data unevenly spaced. The resulting 26 data points are stored in the first column of the variable `data`.

```
clear

rng(0)
data(:,1) = 0.3 : 0.1 : 3;
data(:,1) = data(:,1) + 0.2*randn(size(data(:,1)));
```

We can then compute the second variable, which is the exponent of the first variable multiplied by 0.2 and increased by 3. We again add Gaussian noise, this time with a standard deviation of 0.5, to the data. Finally, we can sort the data with respect to the first column and display the result.

```
data(:,2) = 3 + 0.2 * exp(data(:,1));
data(:,2) = data(:,2) + 0.5*randn(size(data(:,2)));
data = sortrows(data,1);
plot(data(:,1),data(:,2),'o')
xlabel('x-Axis'), ylabel('y-Axis')
```

Nonlinear regression aims to estimate the two coefficients of the exponential function, i.e., the multiplier 0.2 and the summand 3. The function `p=nlinfit(data(:,1),data(:,2),fun,p0)` returns a vector `p` of coefficient estimates for a nonlinear regression of the responses in `data(:,2)` to the predictors in `data(:,1)` using the model specified by `fun`. Here, `fun` is a function handle to a function of the form `hat=modelfun(b,X)`, where `b` is a coefficient vector. A function handle is passed in an argument list to other functions, which can then execute the function using the handle. A function handle uses the *at* sign, `@`, before the function name. The variable `p0` is a vector containing initial values for the coefficients and is the same length as `p`. We can design a function handle model representing an exponential function with an input variable `t` and the coefficients `phi`. The initial values of `p` are `[0 0]`. We can then use `nlinfit` to estimate the coefficients `p` using the data `data`, the model `model`, and the initial values `p0`.

```
model = @(phi,t)(phi(1)*exp(t) + phi(2));
p0 = [0 0];
p = nlinfit(data(:,1),data(:,2),model,p0)

p =
    0.2121    2.8306
```

www.it-ebooks.info

We can now use the resulting coefficients `p(1)` and `p(2)` to calculate the function values `fittedcurve` using the model and compare the results with the original data in a graphics.

```
fittedcurve_1 = p(1)*exp(data(:,1)) + p(2);

plot(data(:,1),data(:,2),'o')
hold on
plot(data(:,1),fittedcurve_1,'r')
xlabel('x-Axis'), ylabel('y-Axis')
title('Unweighted Fit')
hold off
```

As we can see from the output of `p` and the graphics, the fitted red curve describes the data reasonably well. We can now also use `nlinfit` to perform a weighted regression. Let us assume that we know the one-sigma errors of the values in `data(:,2)`. We can generate synthetic errors and store them in the third column of `data`.

```
data(:,3) = abs(randn(size(data(:,1))));
errorbar(data(:,1),data(:,2),data(:,3),'o')
xlabel('x-Axis'), ylabel('y-Axis')
```

We can now normalize the data points so that they are weighted by the inverse of the relative errors. We therefore normalize `data(:,3)` so that the total of all errors in `data(:,3)` is one, and store the normalized errors in `data(:,4)`.

```
data(:,4) = sum(data(:,3))./data(:,3);
```

To make a weighted fit, we define the model function `model`, and then use `nlinfit` with the parameter `Weights`.

```
model = @(phi,t)(data(:,4).*(phi(1)*exp(t) + phi(2)));
p0 = [0 0];
p = nlinfit(data(:,1),data(:,5),model,p0,'Weights',data(:,4))

p =
    0.2191    2.3442
```

As before, `nlinfit` will compute weighted parameter estimates `p`. We again use the resulting coefficents `p(1)` and `p(2)` to calculate the function values `fittedcurve` using the model and compare the results with the original data.

```
fittedcurve_2 = p(1)*exp(data(:,1)) + p(2);
errorbar(data(:,1),data(:,2),data(:,3),'o')
hold on
plot(data(:,1),fittedcurve_2,'r')
xlabel('x-Axis'), ylabel('y-Axis')
title('Weighted Fit')
hold off
```

www.it-ebooks.info

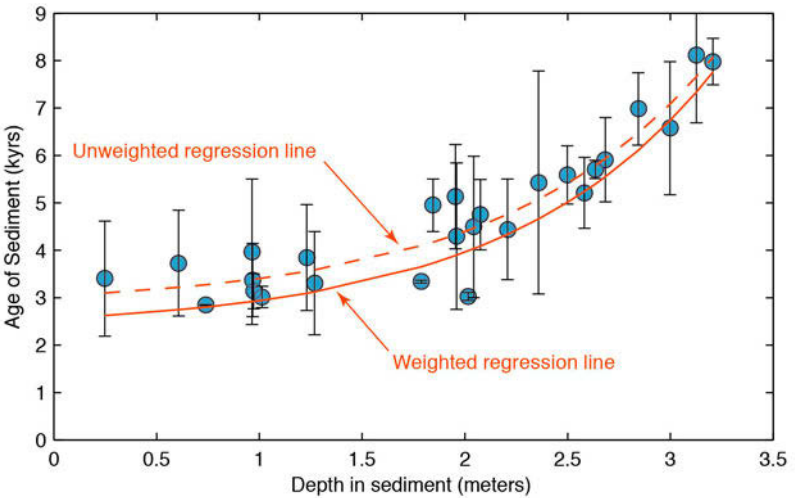


Fig. 4.9 Comparison of unweighted (dashed line) and weighted (solid line) regression result from synthetic data. The plot shows the original data points (circles), the error bars for all data points, and the regression line for an exponential model function. In the unweighted regression, the fitted curved is moved towards the first two data points (x=−0.24823 and x=0.60753) with a large error, while in the weighted regression it is moved towards the third data point (x=0.73846) with a small error.

Comparing the coefficients `p` and the red curves from the weighted regression with the previous results from the unweighted regression reveals slightly different results (Fig. 4.9):

```
errorbar(data(:,1),data(:,2),data(:,3),'o')
hold on
plot(data(:,1),fittedcurve_1,'r--')
plot(data(:,1),fittedcurve_2,'r-')
xlabel('x-Axis'), ylabel('y-Axis')
title('Comparison of Unweighted and Weighted Fit')
hold off
```

As an example, in the unweighted regression the fitted curved is moved towards the first two data points (x=0.24823 and x=0.60753) with a large error, while in the weighted experiment it is moved towards the third data point (x=0.73846) with a small error.

Recommended Reading

Albarède F (2002) Introduction to Geochemical Modeling. Cambridge University Press, Cambridge

Davis JC (2002) Statistics and Data Analysis in Geology, Third Edition. John Wiley and Sons, New York

www.it-ebooks.info

Draper NR, Smith, H (1998) Applied Regression Analysis. Wiley Series in Probability and Statistics, John Wiley and Sons, New York

Efron B (1982) The Jackknife, the Bootstrap, and Other Resampling Plans. Society of Industrial and Applied Mathematics CBMS-NSF Monographs 38

Fisher RA (1922) The Goodness of Fit of Regression Formulae, and the Distribution of Regression Coefficients. Journal of the Royal Statistical Society 85:597–612

Kendall M (1938) A New Measure of Rank Correlation. Biometrika 30:81–89

MacTavish JN, Malone PG, Wells TL (1968) RMAR; a Reduced Major Axis Regression Program Designed for Paleontologic Data. Journal of Paleontology 42/4:1076–1078

MathWorks (2014) Statistics Toolbox – User's Guide. The MathWorks, Inc., Natick, MA

Pearson K (1894–98) Mathematical Contributions to the Theory of Evolution, Part I to IV. Philosophical Transactions of the Royal Society 185–191

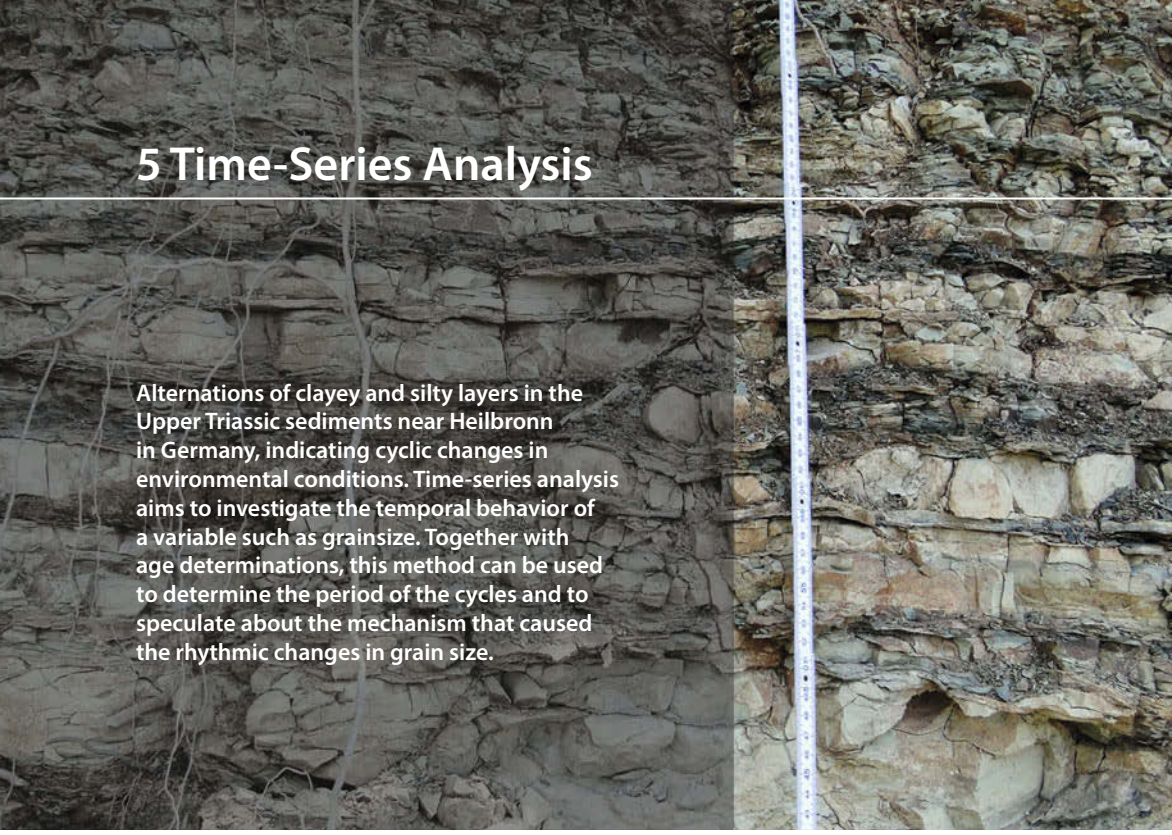
Pearson K (1895) Notes on regression and inheritance in the case of two parents. Proceedings of the Royal Society of London 58:240–242

Spearman C (1904) The proof and measurement of association between two things. American Journal of Psychology 15:72–101

Spearman C (1910) Correlation calculated from faulty data. British Journal of Psychology 3:271–295

www.it-ebooks.info

5 Time-Series Analysis



Alternations of clayey and silty layers in the Upper Triassic sediments near Heilbronn in Germany, indicating cyclic changes in environmental conditions. Time-series analysis aims to investigate the temporal behavior of a variable such as grainsize. Together with age determinations, this method can be used to determine the period of the cycles and to speculate about the mechanism that caused the rhythmic changes in grain size.

5.1 Introduction

Time-series analysis aims to investigate the temporal behavior of a variable $x(t)$. Examples include the investigation of long-term records of mountain uplift, sea-level fluctuations, orbitally-induced insolation variations and their influence on the ice-age cycles, millennium-scale variations in the atmosphere-ocean system, the effect of the El Niño/Southern Oscillation on tropical rainfall and sedimentation (Fig. 5.1), and tidal influences on noble gas emissions from bore holes. The temporal pattern of a sequence of events can be random, clustered, cyclic, or chaotic. Time-series analysis provides various tools with which to detect these temporal patterns. Understanding the underlying processes that produced the observed data allows us to predict future values of the variable. We use the Signal Processing and Wavelet Toolboxes, which contain all the necessary routines for time-series analysis (MathWorks 2014a and b).

Section 5.2 discusses signals in general and contains a technical description of how to generate synthetic signals for time-series analysis. The use of spectral analysis to detect cyclicities in a single time series (auto-spectral analysis) and to determine the relationship between two time series

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_5

www.it-ebooks.info

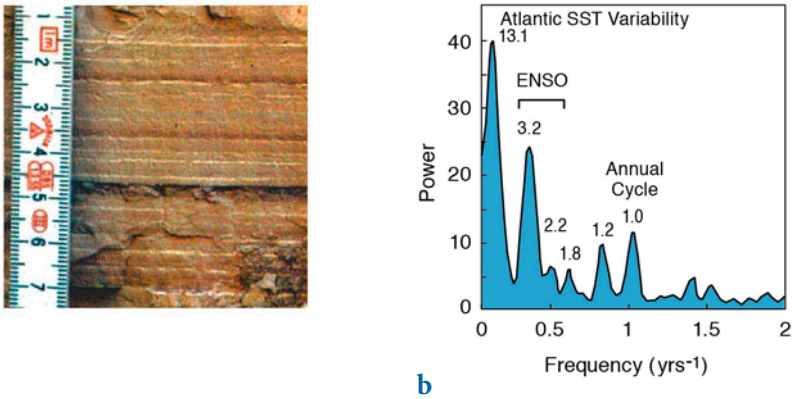


Fig. 5.1 a Photograph of ca. 30 kyr-old varved sediments from a lake in the Andes of Northwest Argentina. The distribution of the source rocks and the interannual precipitation pattern in the area suggest that the reddish-brown layers reflect cyclic recurrences of enhanced precipitation, erosion, and sediment input into the lake. **b** The power spectrum of a red-color intensity transect across 70 varves is dominated by significant peaks at frequencies of ca. 0.076, 0.313, 0.455 and 1.0 yrs^{-1}. These cyclicities suggest a strong influence of the tropical Atlantic sea-surface temperature (SST) variability, the El Niño/Southern Oscillation (ENSO), and the annual cycle that occurred 30 kyrs ago, similar to today's cyclicities (Trauth et al. 2003).

as a function of frequency (cross-spectral analysis) is then demonstrated in Sections 5.3 and 5.4. Since most time series in earth sciences have uneven time intervals, various interpolation techniques and subsequent methods of spectral analysis are required, and these are introduced in Section 5.5. Evolutionary power spectra to map changes in cyclicity through time are demonstrated in Section 5.6. An alternative technique for analyzing unevenly-spaced data is explained in Section 5.7. Section 5.8 introduces the very popular wavelet power spectrum, which is able to map temporal variations in the spectra in a similar way to the method demonstrated in Section 5.6. Section 5.9 then introduces a non-parametric method to detect abrupt transitions in central tendency and dispersion within time series. This chapter closes with an overview of nonlinear techniques, in particular the method of recurrence plots (Section 5.10).

5.2 Generating Signals

A time series is an ordered sequence of values of a variable $x(t)$ at certain times t_k.

$$x(t_k) = x(t_1), x(t_2), \ldots, x(t_N)$$

www.it-ebooks.info

If the time interval between any two successive observations $x(t_k)$ and $x(t_{k+1})$ is constant, the time series is said to be equally spaced and the sampling interval is

$$\Delta t = t_{k+1} - t_k$$

The sampling frequency f_s is the inverse of the sampling interval Δt. We generally try to sample at regular time intervals or constant sampling frequencies, but in many earth science examples this is not possible. As an example, imagine deep-sea sediments sampled at five-centimeter intervals along a sediment core. Radiometric age determinations at certain levels in the sediment core revealed significant fluctuations in the sedimentation rates. Despite the samples being evenly spaced along the sediment core they are not equally spaced on the time axis. Here, the quantity

$$\Delta t = T / N$$

where T is the full length of the time series and N is the number of data points, represents only an average sampling interval. In general, a time series $x(t_k)$ can be represented as the linear sum of a periodic component $x_p(t_k)$, a long-term component or trend $x_{tr}(t_k)$, and random noise $x_n(t_k)$.

$$x(t_k) = x_p(t_k) + x_{tr}(t_k) + x_n(t_k)$$

The long-term component is a linear or higher-degree trend that can be extracted by fitting a polynomial of a certain degree and subtracting the values of this polynomial from the data (see Chapter 4). Noise removal will be described in Chapter 6. The periodic – or cyclic in a mathematically less rigorous sense – component can be approximated by a linear combination of sine (or cosine) waves that have different amplitudes A_i, frequencies f_i, and phase angles ψ_i.

$$x_p(t_k) = \sum_i A_i \cdot \sin(2\pi f_i t_k - \psi_i)$$

The phase angle ψ helps to detect temporal shifts between signals of the same frequency. Two signals x and y with the same period are out of phase unless the difference between ψ_x and ψ_y is equal to zero (Fig. 5.2).

The frequency f of a periodic signal is the inverse of the period τ. The *Nyquist frequency* f_{nyq} is half the sampling frequency f_s and represents the maximum frequency the data can produce. As an example audio compact

www.it-ebooks.info

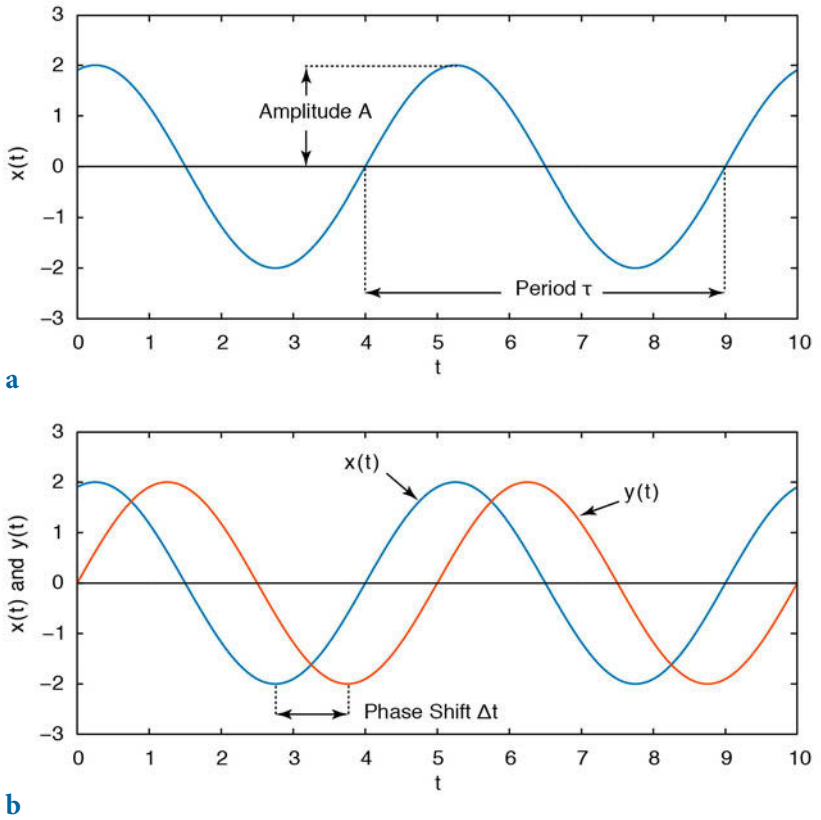


Fig. 5.2 a Periodic signal x a function of time t defined by the amplitude A, and the period τ which is the inverse of the frequency f. **b** Two signals x and y of the same period are out of phase if the difference between ψ_x and ψ_y is not equal to zero.

disks (CDs) are sampled at frequencies of 44,100 Hz (Hertz, where 1 Hz=1 cycle per second), but the corresponding Nyquist frequency is 22,050 Hz, which is the highest frequency a CD player can theoretically produce. The performance limitations of anti-alias filters used by CD players further reduce the frequency band and result in a cutoff frequency of around 20,050 Hz, which is the true upper frequency limit of a CD player.

We can now generate synthetic signals to illustrate the use of time-series analysis tools. When using synthetic data we know in advance which features the time series contains, such as periodic or random components, and we can introduce a linear trend or gaps in the time series. The user will encounter plenty of examples of the possible effects of varying the parameter settings, as well as potential artifacts and errors that can result from the application of spectral analysis tools. We will start with simple data and then apply the

www.it-ebooks.info

methods to more complex time series. The first example illustrates how to generate a basic synthetic data series that is characteristic of earth science data. First, we create a time axis `t` running from 1 to 1000 in steps of one unit, i.e., the sampling frequency is also one. We then generate a simple periodic signal `y`: a sine wave with a period of five and an amplitude of two by typing

```
clear

t = 1 : 1000;
x = 2*sin(2*pi*t/5);

plot(t,x), axis([0 200 -4 4])
```

The period of τ=5 corresponds to a frequency of f=1/5=0.2. Natural data series, however, are more complex than a simple periodic signal. The slightly more complicated signal can be generated by superimposing several periodic components with different periods. As an example we compute such a signal by adding three sine waves with the periods τ_1=50 (f_1=0.02), τ_2=15 (f_2≈0.07) and τ_3=5 (f_3=0.2). The corresponding amplitudes are A_1=2, A_2=1 and A_3=0.5.

```
t = 1 : 1000;
x = 2*sin(2*pi*t/50) + sin(2*pi*t/15) + 0.5*sin(2*pi*t/5);

plot(t,x), axis([0 200 -4 4])
```

By restricting the t-axis to the interval [0,200], only one fifth of the original data series is displayed (Fig. 5.3 a). It is, however, recommended that long data series be generated, as in the example, in order to avoid edge effects when applying spectral analysis tools for the first time.

In contrast to our synthetic time series, real data also contain various disturbances, such as random noise and first or higher-order trends. In order to reproduce the effects of noise, a random-number generator can be used to compute Gaussian noise with mean of zero and standard deviation of one. The seed of the algorithm should be set to zero using `rng(0)`. One thousand random numbers are then generated using the function `randn`.

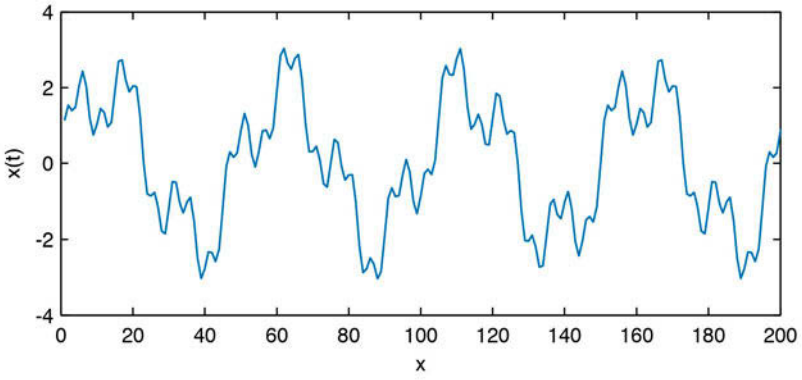
```
rng(0)
n = randn(1,1000);
```

We add this noise to the original data, i.e., we generate a signal containing additive noise (Fig. 5.3 b). Displaying the data illustrates the effect of noise on a periodic signal. Since in reality no record is totally free of noise it is important to familiarize oneself with the influence of noise on power spectra.

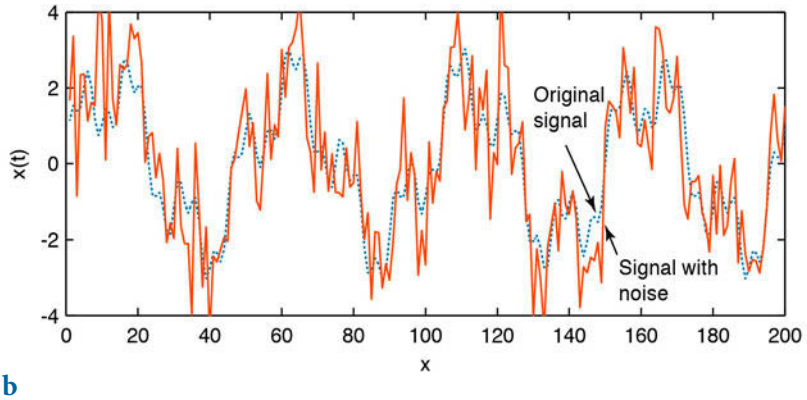
Audio 5.1

```
xn = x + n;

plot(t,x,'b-',t,xn,'r-'), axis([0 200 -4 4])
```

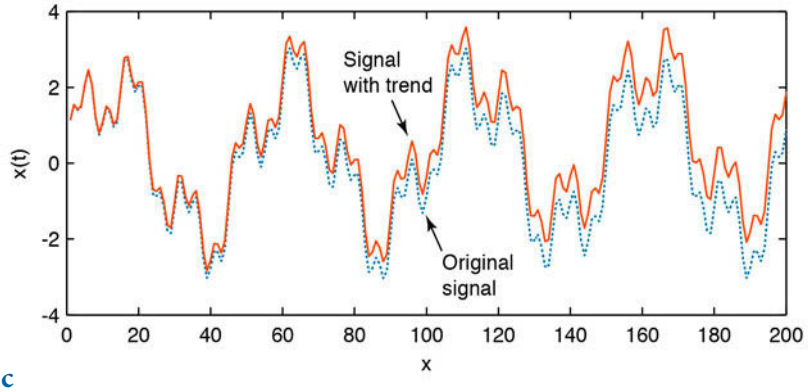
www.it-ebooks.info



a



b



c

Fig. 5.3 **a** Synthetic signal with the periodicities τ_1=50, τ_2=15 and τ_3=5, with different amplitudes, and **b** the same signal overprinted with Gaussian noise. **c** The time series shows a significant linear trend.

www.it-ebooks.info

Signal processing methods are often applied to remove a major part of the noise, although many filtering methods make arbitrary assumptions concerning the signal-to-noise ratio. Moreover, filtering introduces artifacts and statistical dependencies into the data, which may have a profound influence on the resulting power spectra.

Finally, we introduce a linear long-term trend to the data by adding a straight line with a slope of 0.005 and an intercept with the *y*-axis of zero (Fig. 5.3 c). Such trends are common in earth sciences. As an example, consider the glacial-interglacial cycles observed in marine oxygen isotope records, overprinted on a long-term cooling trend over the last six million years.

```
xt = x + 0.005*t;

plot(t,x,'b-',t,xt,'r-'), axis([0 200 -4 4])
```

In reality, more complex trends exist, such as higher-order trends or trends characterized by variations in gradient. In practice, it is recommended that such trends be eliminated by fitting polynomials to the data and subtracting the corresponding values. Our synthetic time series now contains many characteristics of a typical earth science data set. It can be used to illustrate the use of the spectral analysis tools that are introduced in the next section.

Audio 5.2

5.3 Auto-Spectral and Cross-Spectral Analysis

Auto-spectral analysis aims to describe the distribution of variance contained in a single signal $x(t)$ as a function of frequency or wavelength. A simple way to describe the variance in a signal over a time lag k is by means of the autocovariance. An unbiased estimator of the autocovariance cov_{xx} of the signal $x(t)$ with N data points sampled at constant time intervals Δt is

$$cov_{xx}(k) = \frac{1}{N-k-1} \sum_{t=1}^{N-k} (x_i - \bar{x})(x_{i+k} - \bar{x})$$

The autocovariance series clearly depends on the amplitude of $x(t)$. Normalizing the covariance by the variance σ^2 of $x(t)$ yields the autocorrelation sequence. Autocorrelation involves correlating a series of data with itself as a function of a time lag k.

$$corr_{xx}(k) = \frac{cov_{xx}(k)}{cov_{xx}(0)} = \frac{cov_{xx}(k)}{\sigma_x^{\,2}}$$

www.it-ebooks.info

A popular method used to compute power spectra in earth sciences is the method introduced by Blackman and Tukey (1958). The *Blackman-Tukey method* uses the complex Fourier transform $X_{xx}(f)$ of the autocorrelation sequence $corr_{xx}(k)$,

$$X_{xx}(f) = \sum_{k=0}^{M} corr_{xx}(k)\, e^{i 2\pi fk/f_s}$$

where M is the maximum lag and f_s the sampling frequency. The Blackman-Tukey auto-spectrum is the absolute value of the Fourier transform of the autocorrelation function. In some fields, the *power spectral density* is used as an alternative way of describing the auto-spectrum. The Blackman-Tukey power spectral density *PSD* is estimated by

$$PSD = \frac{X^*_{xx}(f) X_{xx}(f)}{f_s} = \frac{\left|X_{xx}(f)\right|^2}{f_s}$$

where $X^*_{xx}(f)$ is the conjugate complex of the Fourier transform of the autocorrelation function $X_{xx}(f)$ and f_s is the sampling frequency. The actual computation of the power spectrum can only be performed at a finite number of different frequencies by employing a *Fast Fourier Transformation* (FFT). The FFT is a method of computing a discrete Fourier transform with reduced execution time. Most FFT algorithms divide the transform into two portions of size $N/2$ at each step of the transformation. The transform is therefore limited to blocks with dimensions equal to a power of two. In practice, the *spectrum* is computed by using a number of frequencies that is close to the number of data points in the original signal $x(t)$.

The discrete Fourier transform is an approximation of the continuous Fourier transform. The continuous Fourier transform assumes an infinite signal but discrete real data are limited at both ends, i.e., the signal amplitude is zero beyond either end of the time series. In the time domain, a finite signal corresponds to an infinite signal multiplied by a rectangular window that has a value of one within the limits of the signal and a value of zero elsewhere. In the frequency domain, the multiplication of the time series by this window is equivalent to a convolution of the power spectrum of the signal with the spectrum of the rectangular window (see Section 6.4 for a definition of convolution). The spectrum of the window, however, is a $\sin(x)/x$ function, which has a main lobe and numerous side lobes on either side of the main peak, and hence all maxima in a power spectrum *leak*, i.e., they lose power on either side of the peaks (Fig. 5.4).

www.it-ebooks.info

A popular way to overcome the problem of *spectral leakage* is by windowing, in which the sequence of data is simply multiplied by a smooth bell-shaped curve with positive values. Several window shapes are available, e.g., *Bartlett* (triangular), *Hamming* (cosinusoidal) and *Hanning* (slightly different cosinusoidal) (Fig. 5.4). The use of these windows slightly modifies the equation for the Blackman-Tukey auto-spectrum to

$$X_{xx}(f) = \sum_{k=0}^{M} corr_{xx}(k) w(k)\, e^{i2\pi fk/f_s}$$

where $w(k)$ is the windowing function. The Blackman-Tukey method therefore performs auto-spectral analysis in three steps: calculation of the autocorrelation sequence $corr_{xx}(k)$, windowing and, finally, computation of the discrete Fourier transform. MATLAB allows power spectral analysis to be performed with a number of modifications to the above method. One useful modification is the Welch method (Welch 1967) (Fig. 5.5). This method involves dividing the time series into overlapping segments, computing the power spectrum for each segment, and then averaging the power spectra. The advantage of averaging the spectra is obvious: it simply improves the signal-to-noise ratio of a spectrum. The disadvantage is a loss of resolution in the spectra.

Cross-spectral analysis correlates two time series in the frequency domain.

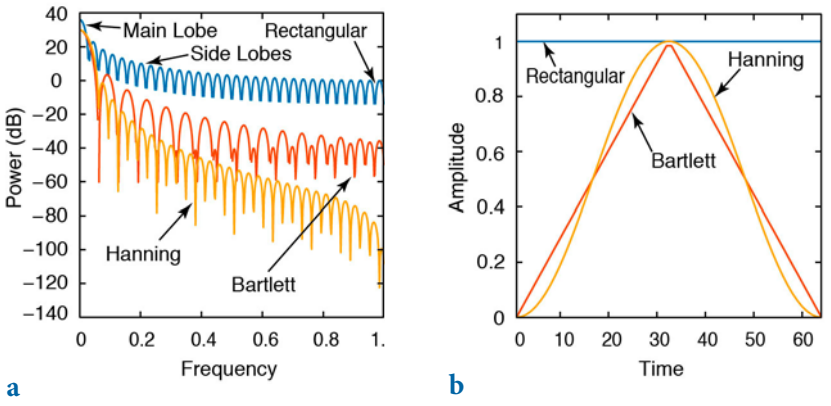


Fig. 5.4 Spectral leakage. **a** The amplitudes of the side lobes relative to that of the main lobe are reduced by multiplying the corresponding time series by **b** a smooth bell-shaped window function. A number of different windows with advantages and disadvantages are available for use instead of the default rectangular window, including *Bartlett* (triangular) and *Hanning* (cosinusoidal) windows. Graph generated using the function `wvtool`.

www.it-ebooks.info

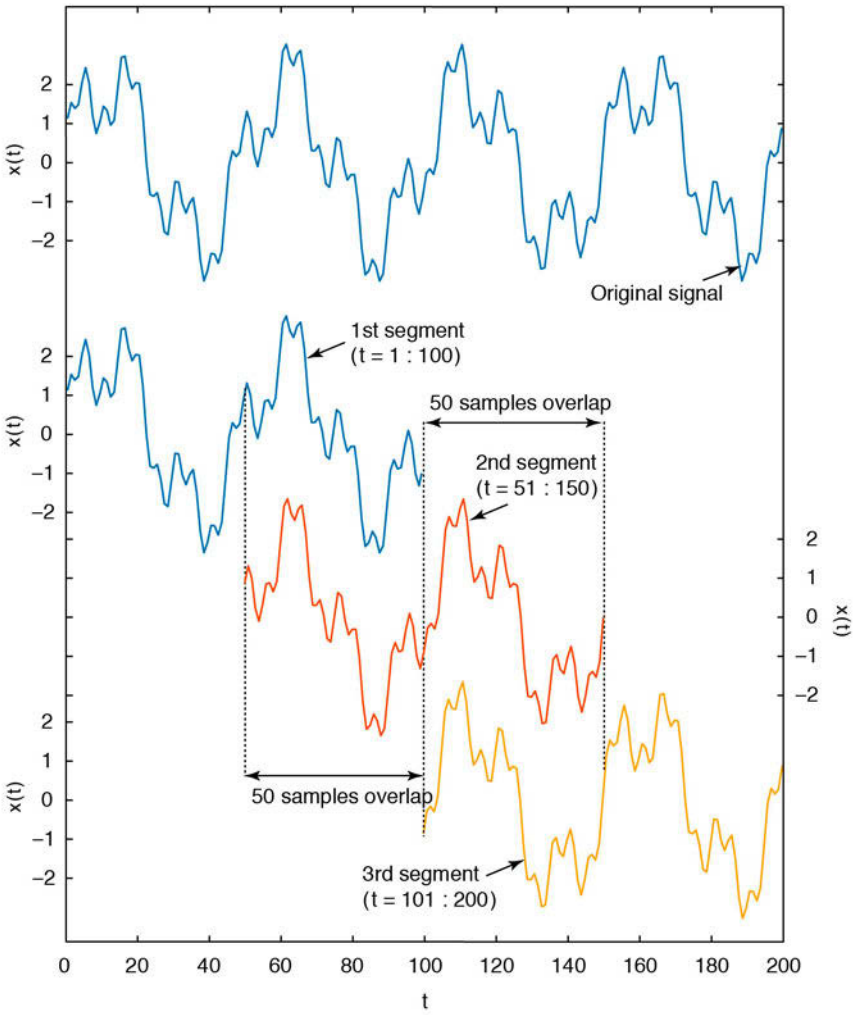


Fig. 5.5 Principle of Welch's power spectral analysis. The time series is first divided into overlapping segments; the power spectrum for each segment is then computed and all spectra are averaged to improve the signal-to-noise ratio of the power spectrum.

The cross-covariance is a measure of the variance between two signals over a time lag k. An unbiased estimator of the cross-covariance cov_{xy} of two signals, $x(t)$ and $y(t)$, with N data points sampled at constant time intervals Δt, is

$$cov_{xy}(k) = \frac{1}{N-k-1} \sum_{t=1}^{N-k} (x_i - \bar{x})(y_{i+k} - \bar{y})$$

www.it-ebooks.info

The cross-covariance series again depends on the amplitudes of $x(t)$ and $y(t)$. Normalizing the covariance by the standard deviations of $x(t)$ and $y(t)$ yields the cross-correlation sequence.

$$corr_{xy}(k) = \frac{cov_{xy}(k)}{cov_{xy}(0)} = \frac{cov_{xy}(k)}{\sigma_x \sigma_y}$$

The *Blackman-Tukey method* uses the complex Fourier transform $X_{xy}(f)$ of the cross-correlation sequence $corr_{xy}(k)$

$$X_{xy}(f) = \sum_{k=0}^{M} corr_{xy}(k)\, e^{i2\pi fk/f_s}$$

where M is the maximum lag and f_s the sampling frequency. The absolute value of the complex Fourier transform $X_{xy}(f)$ is the cross-spectrum while the angle of $X_{xy}(f)$ represents the phase spectrum. The phase difference is important in calculating leads and lags between two signals, a parameter often used to propose causalities between two processes documented by the signals. The correlation between two spectra can be calculated by means of the coherence:

$$C_{xy} = \frac{\left|X_{xy}(f)\right|^2}{X_{xx}(f)X_{yy}(f)}$$

The coherence is a real number between 0 and 1, where 0 indicates no correlation and 1 indicates maximum correlation between $x(t)$ and $y(t)$ at the frequency f. A significant degree of coherence is an important precondition for computing phase shifts between two signals.

5.4 Examples of Auto-Spectral and Cross-Spectral Analysis

The Signal Processing Toolbox provides numerous methods for computing spectral estimators for time series. The introduction of object-oriented programming with MATLAB has led to the launch of a new set of functions performing spectral analyses. Type `help spectrum` for more information about object-oriented spectral analysis. The non-object-oriented functions to perform spectral analyses, however, are still available. One of the oldest functions in this toolbox is `periodogram(x,window,nfft,fs)` which computes the power spectral density `Pxx` of a time series `x(t)` using the periodogram

www.it-ebooks.info

method. This method was invented by Arthur Schuster in 1898 for studying the climate and calculates the power spectrum by performing a Fourier transform directly on a sequence without requiring prior calculation of the autocorrelation sequence. The periodogram method can therefore be considered a special case of the Blackman and Tukey (1958) method, applied with the time lag k set to unity (Muller and Macdonald 2000). At the time of its introduction in 1958, the indirect computation of the power spectrum via an autocorrelation sequence was faster than calculating the Fourier transformation for the full data series `x(t)` directly. After the introduction of the Fast Fourier Transform (FFT) by Cooley and Tukey (1965), and subsequent faster computer hardware, the higher computing speed of the Blackman-Tukey approach compared to the periodogram method became relatively unimportant.

For this next example we again use the synthetic time series `x`, `xn` and `xt` generated in Section 5.2 as the input:

```
clear

t = 1 : 1000; t = t';
x = 2*sin(2*pi*t/50) + sin(2*pi*t/15) + 0.5*sin(2*pi*t/5);

randn('seed',0)
n = randn(1000,1);
xn = x + n;

xt = x + 0.005*t;
```

We then compute the periodogram by calculating the Fourier transform of the sequence `x`. The fastest possible Fourier transform using `fft` computes the Fourier transform for `nfft` frequencies, where `nfft` is the next power of two closest to the number of data points `n` in the original signal `x`. Since the length of the data series is `n=1000`, the Fourier transform is computed for `nfft=1024` frequencies, while the signal is padded with `nfft-n=24` zeros.

```
Xxx = fft(x,1024);
```

If `nfft` is even, as in our example, then `Xxx` is symmetric. For example, as the first `(1+nfft/2)` points in `Xxx` are unique, the remaining points are symmetrically redundant. The power spectral density is defined as `Pxx2=(abs(Xxx).^2)/Fs`, where `Fs` is the sampling frequency. The function `periodogram` also scales the power spectral density by the length of the data series, i.e., it divides by `Fs=1` and `length(x)=1000`.

```
Pxx2 = abs(Xxx).^2/1000;
```

www.it-ebooks.info

We now drop the redundant part in the power spectrum and use only the first `(1+nfft/2)` points. We also multiply the power spectral density by two to keep the same energy as in the symmetric spectrum, except for the first data point.

```
Pxx = [Pxx2(1); 2*Pxx2(2:512)];
```

The corresponding frequency axis runs from `0` to `Fs/2` in `Fs/(nfft-1)` steps, where `Fs/2` is the Nyquist frequency. Since `Fs=1` in our example, the frequency axis is

```
f = 0 : 1/(1024-1) : 1/2;
```

We then plot the power spectral density `Pxx` in the Nyquist frequency range from `0` to `Fs/2`, which in our example is from `0` to `1/2`. The Nyquist frequency range corresponds to the first 512 or `nfft/2` data points. We can plot the power spectral density over the frequency by typing

```
plot(f,Pxx), grid
```

The graphical output shows that there are three significant peaks at the positions of the original frequencies of the three sine waves (1/50, 1/15, and 1/5). Alternatively, we can also plot the power spectral density over the period by typing

```
plot(1./f,Pxx), axis([0 100 0 1000]), grid
```

where we observe the three periods 50, 15, and 5, as expected. Since the values on the *x*-axis of this plot are not evenly spaced (in constrast to those on the frequency axis), we find the long periods poorly resolved and a broad peak at a period of 50 in this graphics. The code for the power spectral density can be rewritten to make it independent of the sampling frequency,

```
Fs = 1;

t = 1/Fs :1/Fs : 1000/Fs; t = t';
x = 2*sin(2*pi*t/50) + sin(2*pi*t/15) + 0.5*sin(2*pi*t/5);

nfft = 2^nextpow2(length(t));
Xxx = fft(x,nfft);

Pxx2 = abs(Xxx).^2 /Fs /length(x);
Pxx = [Pxx2(1); 2*Pxx2(2:512)];
f = 0 : Fs/(nfft-1) : Fs/2;

plot(f,Pxx), grid
axis([0 0.5 0 max(Pxx)])
```

www.it-ebooks.info

where the function `nextpow2` computes the next power of two closest to the length of the time series `x(t)`. This code allows the sampling frequency to be modified and the differences in the results to be explored. We can now compare the results with those obtained using the function `periodogram(x,window,nfft,fs)`.

```
[Pxx,f] = periodogram(x,[],1024,1);
```

This function allows the windowing of the signals with various window shapes to overcome spectral leakage. However, we use the default rectangular window by choosing an empty vector `[]` for `window` to compare the results with the above experiment. The power spectrum `Pxx` is computed using an FFT of length `nfft=1024`, which is the next power of two closest to the length of the series `x(t)` and which is padded with zeros to make up the number of data points to the value of `nfft`. A sampling frequency `fs` of one is used within the function in order to obtain the correct frequency scaling for the *f*-axis. We display the results by typing

```
plot(f,Pxx), grid
xlabel('Frequency')
ylabel('Power')
title('Auto-Spectrum')
```

or alternatively

```
plot(1./f,Pxx), axis([0 100 0 1000]), grid
xlabel('Period')
ylabel('Power')
title('Auto-Spectrum')
```

The graphical output is almost identical to our Blackman-Tukey plot and again shows that there are three significant peaks at the positions of the original frequencies (or periods) of the three sine waves. The same procedure can also be applied to the noisy data:

```
[Pxx,f] = periodogram(xn,[],1024,1);

plot(f,Pxx), grid
xlabel('Frequency')
ylabel('Power')
title('Auto-Spectrum')
```

Let us now increase the noise level by introducing Gaussian noise with a mean of zero and a standard deviation of five.

```
rng(0)
n = 5 * randn(size(x));
xn = x + n;
```

www.it-ebooks.info

```
[Pxx,f] = periodogram(xn,[],1024,1);

plot(f,Pxx), grid
xlabel('Frequency')
ylabel('Power')
title('Auto-Spectrum')
```

This spectrum now resembles a real data spectrum in the earth sciences and the spectral peaks are set against a significant background noise level. The peak of the highest frequency even disappears into the noise and cannot be distinguished from maxima that are attributed to noise. Both spectra can be compared on the same plot (Fig. 5.6):

```
[Pxx,f] = periodogram(x,[],1024,1);
[Pxxn,f] = periodogram(xn,[],1024,1);

subplot(1,2,1)
plot(f,Pxx), grid
xlabel('Frequency')
ylabel('Power')

subplot(1,2,2)
plot(f,Pxxn), grid
xlabel('Frequency')
ylabel('Power')
```

Next, we explore the influence of a linear trend on a spectrum. Long-term trends are common features in earth science data. We will see that this trend is misinterpreted as a very long period by the FFT, producing a large peak

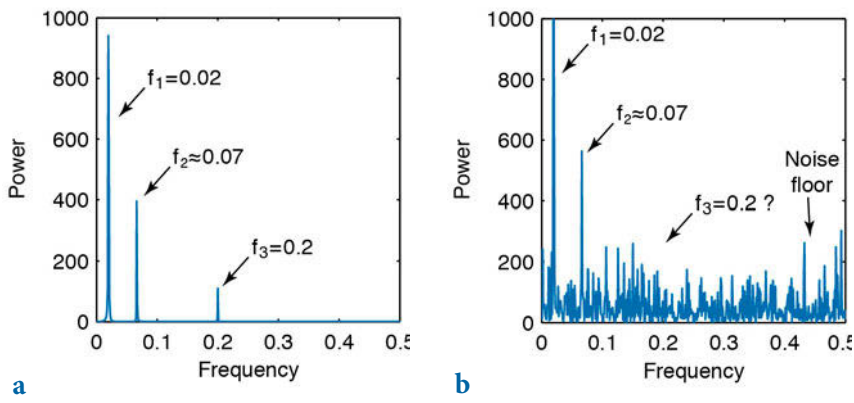


Fig. 5.6 Comparison of the auto-spectra for **a** the noise-free, and **b** the noisy synthetic signals with the periods τ_1=50 (f_1=0.02), τ_2=15 (f_2≈0.07) and τ_3=5 (f_3=0.2). The highest frequency peak disappears completely into the background noise and cannot be distinguished from peaks attributed to the Gaussian noise.

www.it-ebooks.info

with a frequency close to zero (Fig. 5.7).

```
[Pxx,f] = periodogram(x,[],1024,1);
[Pxxt,f] = periodogram(xt,[],1024,1);

subplot(1,2,1)
plot(f,Pxx), grid
xlabel('Frequency')
ylabel('Power')

subplot(1,2,2)
plot(f,Pxxt), grid
xlabel('Frequency')
ylabel('Power')
```

To eliminate the long-term trend, we use the function `detrend`. This function removes linear trends, defined as either a single straight-line fit from the vector `x`, or a continuous, piecewise linear trend from `x` with one or more breakpoints defined by the user.

```
xdt = detrend(xt);

subplot(2,1,1)
plot(t,x,'b-',t,xt,'r-'), grid
axis([0 200 -4 4])

subplot(2,1,2)
plot(t,x,'b-',t,xdt,'r-'), grid
axis([0 200 -4 4])
```

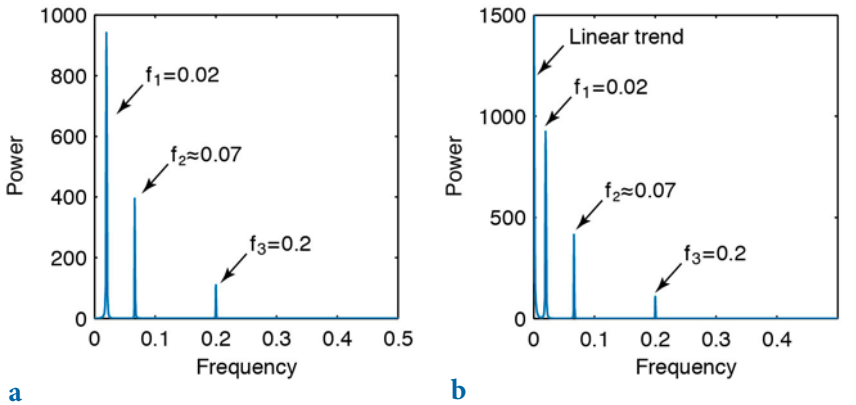


Fig. 5.7 Comparison of the auto-spectra for **a** the original noise-free signal with the periods τ_1=50 (f_1=0.02), τ_2=15 (f_2≈0.07) and τ_3=5 (f_3=0.2), and **b** the same signal overprinted on a linear trend. The linear trend is misinterpreted by the FFT as a very long period with a high amplitude.

www.it-ebooks.info

The resulting spectrum no longer shows the low-frequency peak.

```
[Pxxt,f] = periodogram(xt,[],1024,1);
[Pxxdt,f] = periodogram(xdt,[],1024,1);

subplot(1,2,1)
plot(f,Pxx), grid
xlabel('Frequency')
ylabel('Power')

subplot(1,2,2)
plot(f,Pxxdt), grid
xlabel('Frequency')
ylabel('Power')
```

Some data contain a high-order trend that can be removed by fitting a higher-order polynomial to the data and subtracting the corresponding $x(t)$ values.

We now use two sine waves with identical periodicities τ=5 (equivalent to f=0.2) and amplitudes equal to two to compute the cross-spectrum of two time series. The sine waves show a relative phase shift of t=1. In the argument of the second sine wave this corresponds to $2\pi/5$, which is one fifth of the full wavelength of τ=5.

```
clear

t = 1 : 1000;
x = 2*sin(2*pi*t/5);
y = 2*sin(2*pi*t/5 + 2*pi/5);

plot(t,x,'b-',t,y,'r-')
axis([0 50 -2 2]), grid
```

The cross-spectrum is computed by using the function `cpsd`, which uses Welch's method for computing power spectra (Fig. 5.8). `Pxy` is complex and contains both amplitude and phase information.

```
[Pxy,f] = cpsd(x,y,[],0,1024,1);

plot(f,abs(Pxy)), grid
xlabel('Frequency')
ylabel('Power')
title('Cross-Spectrum')
```

The function `cpsd(x,y,window,noverlap,nfft,fs)` specifies the number of FFT points `nfft` used to calculate the cross power spectral density, which is 1024 in our example. The parameter `window` is empty in our example and the default rectangular window is therefore used to obtain eight sections of `x` and `y`. The parameter `noverlap` defines the number of overlapping samples, which is zero in our example. The sampling frequency `fs` is 1 in this example. The

www.it-ebooks.info

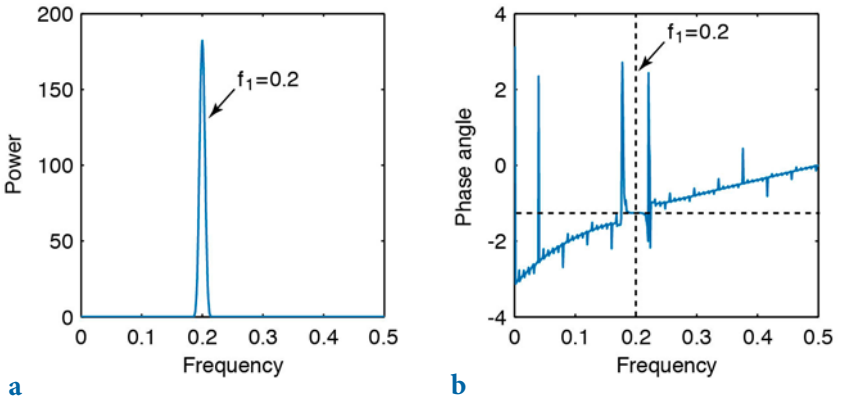


Fig. 5.8 Cross-spectrum of two sine waves with identical periodicities of τ=5 (equivalent to f=0.2) and amplitudes of 2. The sine waves show a relative phase shift of t=1. In the argument of the second sine wave this corresponds to $2\pi/5$, which is one fifth of the full wavelength of τ=5. **a** The magnitude shows the expected peak at f=0.2. **b** The corresponding phase difference in radians at this frequency is 1.2566, which equals $(1.2566 \cdot 5)/(2 \cdot \pi) = 1.0000$, which is the phase shift of 1 that we introduced initially.

coherence of the two signals is one for all frequencies, since we are working with noise-free data.

```
[Cxy,f] = mscohere(x,y,[],0,1024,1);

plot(f,Cxy), grid
xlabel('Frequency')
ylabel('Coherence')
title('Coherence')
```

We use the function `mscohere(x,y,window,noverlap,nfft,fs)` which specifies the number of FFT points `nfft`=1024, the default rectangular window (`window`=`[]`), and no overlapping data points (`noverlap`=0). The complex part of `Pxy` is required for computing the phase shift between the two signals using the function `angle`.

```
phase = angle(Pxy);

plot(f,phase), grid
xlabel('Frequency')
ylabel('Phase Angle')
title('Phase Spectrum')
```

The phase shift at a frequency of f=0.2 (period τ=5) can be interpolated from the phase spectrum

```
interp1(f,phase,0.2)
```

www.it-ebooks.info

which produces the output

```
ans =
  -1.2566
```

The phase spectrum is normalized to one full period $\tau=2\pi$ and the phase shift of −1.2566 therefore equals $(-1.2566 \cdot 5)/(2 \cdot \pi) = -1.0000$, which is the phase shift of one that we introduced initially.

We now use two sine waves with different periodicities to illustrate cross-spectral analysis. Both signals, `x` and `y`, have a periodicity of 5 but a phase shift of 1.

```
clear

t = 1 : 1000;
x = sin(2*pi*t/15) + 0.5*sin(2*pi*t/5);
y = 2*sin(2*pi*t/50) + 0.5*sin(2*pi*t/5+2*pi/5);

plot(t,x,'b-',t,y,'r-')
axis([0 100 -3 3]), grid
```

We can now compute the cross-spectrum `Pxy`, which clearly shows the common period of $\tau=5$ (or frequency of $f=0.2$).

```
[Pxy,f] = cpsd(x,y,[],0,1024,1);

plot(f, abs(Pxy)), grid
xlabel('Frequency')
ylabel('Power')
title('Cross-Spectrum')
```

The coherence shows a high value that is close to one at $f=0.2$.

```
[Cxy,f] = mscohere(x,y,[],0,1024,1);

plot(f,Cxy), grid
xlabel('Frequency')
ylabel('Coherence')
title('Coherence')
```

The complex part of the cross-spectrum `Pxy` is required for calculating the phase shift between the two sine waves.

```
[Pxy,f] = cpsd(x,y,[],0,1024,1);
phase = angle(Pxy);

plot(f,phase), grid
```

The phase shift at a frequency of $f=0.2$ (period $\tau=5$) is

```
interp1(f,phase,0.2)
```

www.it-ebooks.info

which produces the output of

```
ans =
  -1.2572
```

The phase spectrum is normalized to one full period $\tau=2\pi$ and the phase shift of −1.2572 therefore equals $(-1.2572\cdot5)/(2\cdot\pi)=-1.0004$, which is again the phase shift of one that we introduced initially.

5.5 Interpolating and Analyzing Unevenly-Spaced Data

We can now use our experience in analyzing evenly-spaced data to run a spectral analysis on unevenly-spaced data. Such data are very common in earth sciences, for example in the field of paleoceanography, where deep-sea cores are typically sampled at constant depth intervals. The transformation of evenly-spaced length-parameter data to time-parameter data in an environment with changing length-time ratios results in unevenly-spaced time series. Numerous methods exist for interpolating unevenly-spaced sequences of data or time series. The aim of these *interpolation techniques* for $x(t)$ data is to estimate the x-values for an equally-spaced t vector from the irregularly-spaced $x(t)$ actual measurements. *Linear interpolation* predicts the x-values by effectively drawing a straight line between two neighboring measurements and by calculating the x-value at the appropriate point along that line. However, this method has its limitations. It assumes linear transitions in the data, which introduces a number of artifacts including the loss of high-frequency components of the signal and the limiting of the data range to that of the original measurements.

Cubic-spline interpolation is another method for interpolating data that are unevenly spaced. Cubic splines are piecewise continuous curves requiring at least four data points for each step. The method has the advantage that it preserves the high-frequency information contained in the data. However, steep gradients in the data sequence, which typically occur adjacent to extreme minima and maxima, could cause spurious amplitudes in the interpolated time series. Since all these (and other) interpolation techniques might introduce artifacts into the data, it is always advisable to (1) keep the total number of data points constant before and after interpolation, (2) report the method employed for estimating the evenly-spaced data sequence, and (3) explore the effect of interpolation on the variance of the data.

Following this brief introduction to interpolation techniques we can apply the most popular linear and cubic spline interpolation techniques to unevenly-spaced data. Having interpolated the data we can then use the spectral tools that have previously been applied to evenly-spaced data

www.it-ebooks.info

(Sections 5.3 and 5.4). We must first load the two time series:

```
clear

series1 = load('series1.txt');
series2 = load('series2.txt');
```

Both synthetic data sets contain a two-column matrix with 339 rows. The first column contains ages in kiloyears, which are unevenly spaced. The second column contains oxygen-isotope values measured on calcareous microfossils (foraminifera). The data sets contain 100, 40 and 20 kyr cyclicities and they are overlain by Gaussian noise. In the 100 kyr frequency band, the second data series has shifted by 5 kyrs with respect to the first data series. To plot the data we type

```
plot(series1(:,1),series1(:,2))
figure
plot(series2(:,1),series2(:,2))
```

The statistics for the spacing of the first data series can be computed by

```
intv1 = diff(series1(:,1));

plot(intv1)
```

The plot shows that the spacing varies around a mean interval of 3 kyrs, with a standard deviation of ca. 1 kyr. The minimum and maximum values for the time axis

```
min(series1(:,1))
max(series1(:,1))
```

of t_{min}=0 and t_{max}=997 kyrs provide some information about the temporal range of the data. The second data series

```
intv2 = diff(series2(:,1));

plot(intv2)

min(series2(:,1))
max(series2(:,1))
```

has a similar range, from 0 to 997 kyrs. We see that both series have a mean spacing of 3 kyrs and range from 0 to ca. 1000 kyrs. We now interpolate the data to an evenly-spaced time axis. While doing this, we follow the rule that the number of data points should not be increased. The new time axis runs from 0 to 996 kyrs, with 3 kyr intervals.

```
t = 0 : 3 : 996;
```

www.it-ebooks.info

We can now interpolate the two time series to this axis with *linear* and *spline* interpolation methods, using the function `interp1`.

```
series1L = interp1(series1(:,1),series1(:,2),t,'linear');
series1S = interp1(series1(:,1),series1(:,2),t,'spline');

series2L = interp1(series2(:,1),series2(:,2),t,'linear');
series2S = interp1(series2(:,1),series2(:,2),t,'spline');
```

In the `linear` interpolation method the linear interpolant is the straight line between neighboring data points. In the `spline` interpolation the interpolant is a piecewise polynomial (the *spline*) between these data points. The method `spline` with `interp1` uses a piecewise cubic spline interpolation, i.e., the interpolant is a third-degree polynomial. The results are compared by plotting the first series before and after interpolation.

```
plot(series1(:,1),series1(:,2),'ko'), hold on
plot(t,series1L,'b-',t,series1S,'r-'), hold off
```

We can already observe some significant artifacts at ca. 370 kyrs. Whereas the linearly-interpolated points are always within the range of the original data, the spline interpolation method produces values that are unrealistically high or low (Fig. 5.9). The results can be compared by plotting the second data series.

```
plot(series2(:,1),series2(:,2),'ko'), hold on
plot(t,series2L,'b-',t,series2S,'r-'), hold off
```

In this series, only a few artifacts can be observed. The function `interp1` also provides an alternative to `spline`, which is `pchip`. The name `pchip` stands for

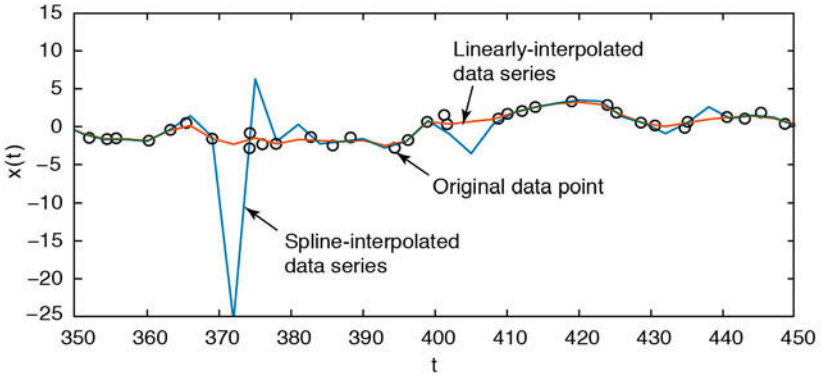


Fig. 5.9 Interpolation artifacts. Whereas the linearly interpolated points are always within the range of the original data, the spline interpolation method results in unrealistic high and low values.

www.it-ebooks.info

Piecewise Cubic Hermite Interpolating Polynomial and this method performs a shape-preserving piecewise cubic interpolation. The function avoids the typical artifacts of the splines as it preserves the original shape of the data series. We can apply the function used above to calculate the power spectrum, computing the FFT for 256 data points with a sampling frequency of 1/3 kyr^{-1}.

```
[Pxx,f] = periodogram(series1L,[],256,1/3);

plot(f,Pxx)
xlabel('Frequency')
ylabel('Power')
title('Auto-Spectrum')
```

Significant peaks occur at frequencies of approximately 0.01, 0.025 and 0.05, corresponding approximately to the 100, 40 and 20 kyr cycles. Analysis of the second time series

```
[Pxx,f] = periodogram(series2L,[],256,1/3);

plot(f,Pxx)
xlabel('Frequency')
ylabel('Power')
title('Auto-Spectrum')
```

also yields significant peaks at frequencies of 0.01, 0.025 and 0.05 (Fig. 5.10). We now compute the cross-spectrum for both data series.

```
[Pxy,f] = cpsd(series1L,series2L,[],128,256,1/3);

plot(f,abs(Pxy))
xlabel('Frequency')
ylabel('Power')
title('Cross-Spectrum')
```

The correlation, as indicated by the high value for the coherence, is quite convincing.

```
[Cxy,f] = mscohere(series1L,series2L,[],128,256,1/3);

plot(f,Cxy)
xlabel('Frequency')
ylabel('Magnitude Squared Coherence')
title('Coherence')
```

We can observe a fairly high coherence at frequencies of 0.01, 0.025 and 0.05. The complex part of `Pxy` is required for calculating the phase difference for each frequency.

```
phase = angle(Pxy);
```

www.it-ebooks.info

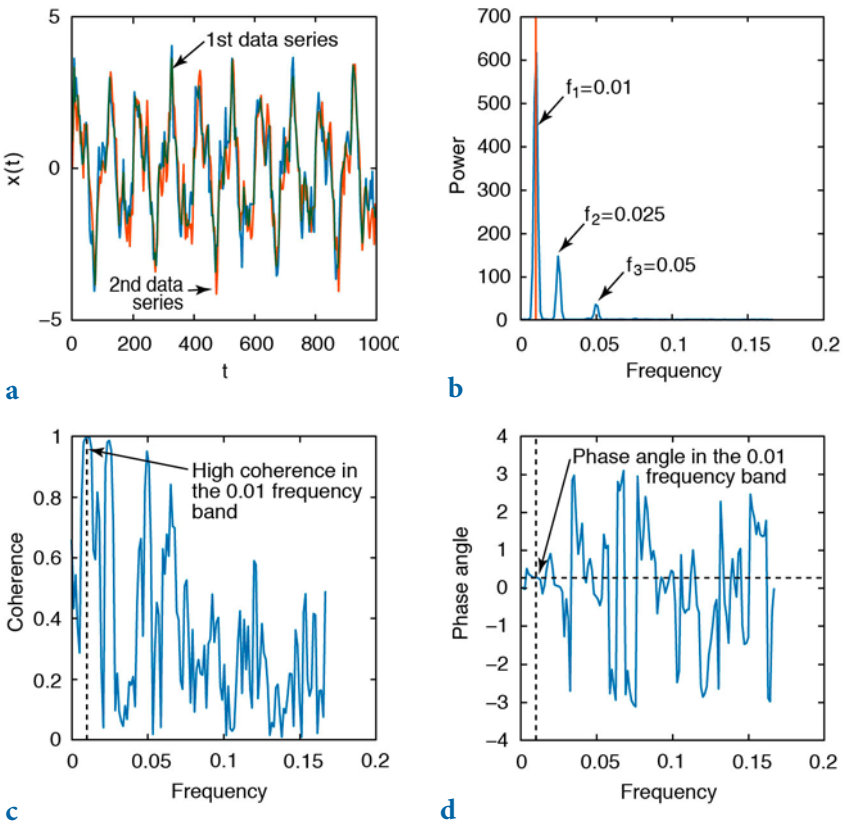


Fig. 5.10 Result from cross-spectral analysis of the two linearly-interpolated signals: **a** signals in the time domain, **b** cross-spectrum of both signals, **c** coherence of the signals in the frequency domain, and **d** phase spectrum in radians.

```
plot(f,phase)
xlabel('Frequency')
ylabel('Phase Angle')
title('Phase spectrum')
```

The phase shift at a frequency of f=0.01 is calculated using

```
interp1(f,phase,0.01)
```

which produces the output of

```
ans =
  -0.2796
```

The phase spectrum is normalized to a full period $\tau=2\pi$ and the phase

www.it-ebooks.info

shift of –0.2796 therefore equals (–0.2796·100 kyrs)/(2·π)=–4.45 kyrs. This corresponds roughly to the phase shift of 5 kyrs introduced to the second data series with respect to the first series.

The Signal Processing Toolbox also contains a GUI function named `sptool` (for *Signal Processing Tool*), which is a more convenient tool for spectral analysis but is not described in any detail herein.

Movie 5.1

5.6 Evolutionary Power Spectrum

The amplitude of spectral peaks usually varies with time. This is particularly true for paleoclimate time series. Paleoclimate records usually show trends, not only in the mean and variance but also in the relative contributions of rhythmic components such as the Milankovitch cycles in marine oxygen-isotope records. Evolutionary power spectra have the ability to map such changes in the frequency domain. The evolutionary or windowed power spectrum is a modification of the method introduced in Section 5.3, which computes the spectrum of overlapping segments of the time series. These overlapping segments are relatively short compared to the windowed segments used by the Welch method (Section 5.3), which is used to increase the signal-to-noise ratio of power spectra. The evolutionary power spectrum method therefore uses the Short-Time Fourier Transform (STFT) instead of the Fast Fourier Transformation (FFT). The output from the evolutionary power spectrum is the short-term, time-localized frequency content of the signal. There are various methods to display the results. For instance, time and frequency can be plotted on the *x*- and *y*-axes, respectively, or vice versa, with the color of the plot being dependent on the height of the spectral peaks.

As an example we use a data set that is similar to those used in Section 5.5. The data series contains three main periodicities of 100, 40 and 20 kyrs and additive Gaussian noise. The amplitudes, however, change through time and this example can therefore be used to illustrate the advantage of the evolutionary power spectrum method. In our example the 40 kyr cycle appears only after ca. 450 kyrs, whereas the 100 and 20 kyr cycles are present throughout the time series. We first load from the file *series3.txt* and display the data (Fig. 5.11).

```
clear

series3 = load('series3.txt');
plot(series3(:,1),series3(:,2))
xlabel('Time (kyr)')
ylabel('d18O (permille)')
title('Signal with Varying Cyclicities')
```

www.it-ebooks.info

Since both the standard and the evolutionary power spectrum methods require evenly-spaced data, we interpolate the data to an evenly-spaced time vector `t`, as demonstrated in Section 5.5.

```
t = 0 : 3 : 1000;
series3L = interp1(series3(:,1),series3(:,2),t,'linear');
```

We then compute a non-evolutionary power spectrum for the full length of the time series (Fig. 5.12). This exercise helps us to compare the differences between the results of the standard and the evolutionary power spectrum methods.

```
[Pxx,f] = periodogram(series3L,[],1024,1/3);
plot(f,Pxx)
xlabel('Frequency')
ylabel('Power')
title('Power Spectrum')
```

Audio 5.3

The auto-spectrum shows significant peaks at 100, 40 and 20 kyr cyclicities, as well as some noise. The power spectrum, however, does not provide any

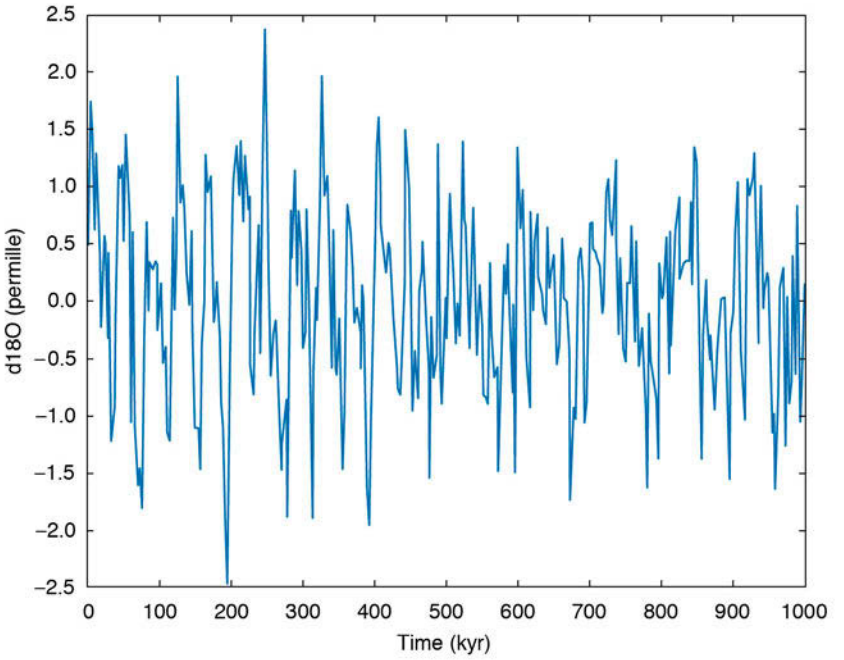


Fig. 5.11 Synthetic data set containing three main periodicities of 100, 40, and 20 kyrs and additive Gaussian noise. Whereas the 100 and 20 kyr cycles are present throughout the time series, the 40 kyr cycle only appears at around 450 kyrs before present.

www.it-ebooks.info

information about fluctuations in the amplitudes of these peaks. The non-evolutionary power spectrum simply represents an average of the spectral information contained in the data.

We now use the function `spectrogram` to map the changes in the power spectrum with time. By default, the time series is divided into eight segments with a 50% overlap. Each segment is windowed with a Hamming window to suppress spectral leakage (Section 5.3). The function `spectrogram` uses similar input parameters to those used in `periodogram` in Section 5.3. We then compute the evolutionary power spectrum for a window of 64 data points with a 50 data point overlap. The STFT is computed for `nfft`=256. Since the spacing of the interpolated time vector is 3 kyrs, the sampling frequency is 1/3 kyr^{-1}.

```
spectrogram(series3L,64,50,256,1/3)
title('Evolutionary Power Spectrum')
xlabel('Frequency (1/kyr)')
ylabel('Time (kyr)')
colormap(jet)
```

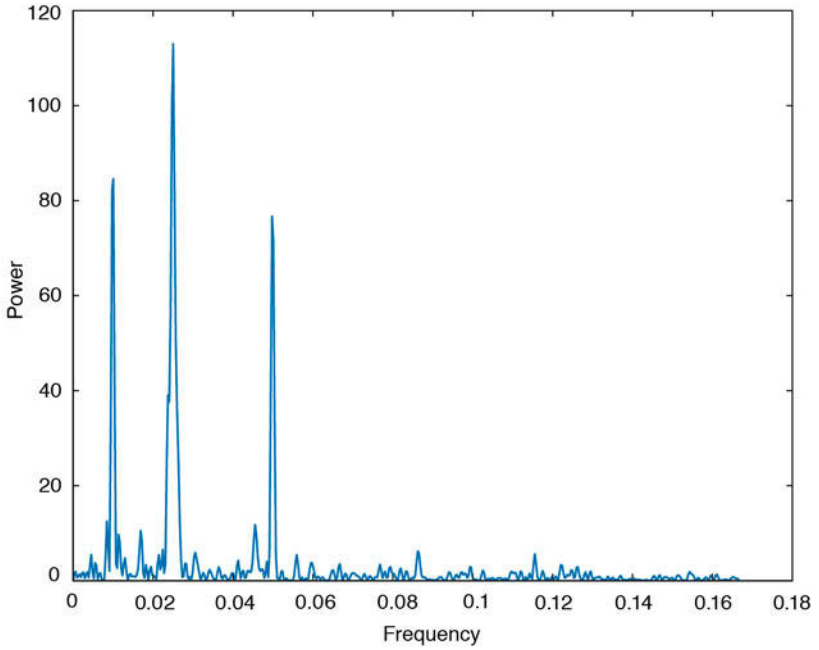


Fig. 5.12 Power spectrum for the complete time series. showing significant peaks at 100, 40 and 20 kyrs. The plot, however, does not provide any information on the temporal behavior of the cyclicities.

www.it-ebooks.info

The output from `spectrogram` is a color plot (Fig. 5.13) that displays red vertical stripes representing significant maxima at frequencies of 0.01 and 0.05 kyr^{-1} (i.e., every 100 and 20 kyrs). There is also a 40 kyr cycle (corresponding to a frequency of 0.025 kyr^{-1}), but this only occurs after ca. 450 kyrs, as documented by the vertical red stripe in the lower half of the graph.

To improve the visibility of the significant cycles, the colors used in the graph can be modified using the colormap editor.

Movie 5.2 `colormapeditor`

The colormap editor displays the colormap of the figure as a strip of rectangular cells. The nodes that separate regions of uniform slope in the RGB colormap can be shifted by using the mouse, which introduces distortions in the colormap and results in modification of the spectrogram colors. For example shifting the yellow node towards the right increases the contrast between the vertical peak areas at 100, 40 and 20 kyrs, and the background.

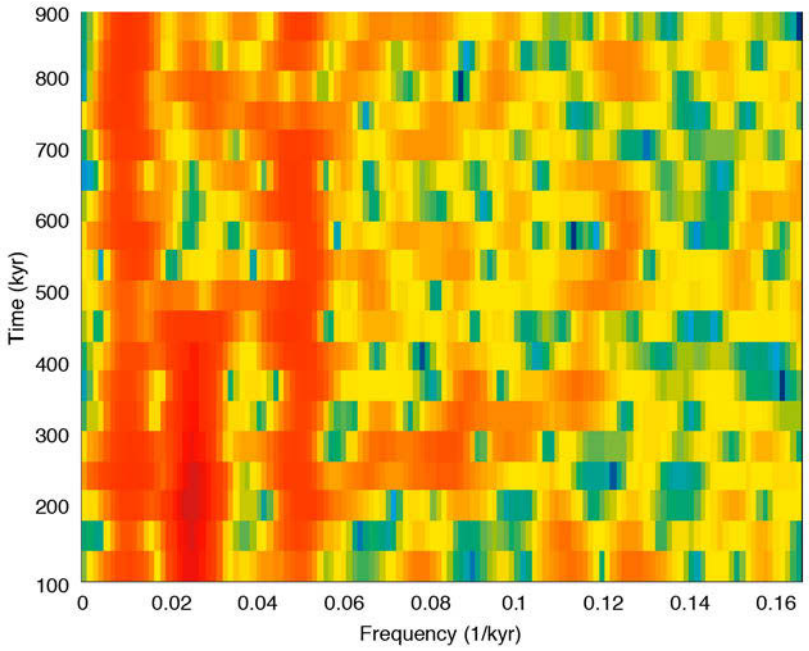


Fig. 5.13 Evolutionary power spectrum using `spectrogram`, which computes the short-time Fourier transform STFT of overlapping segments of the time series. We use a Hamming window of 64 data points and 50 data points overlap. The STFT is computed for `nfft`=256. Since the spacing of the interpolated time vector is 3 kyrs the sampling frequency is 1/3 kyr^{-1}. The plot shows the onset of the 40 kyr cycle at around 450 kyrs before present.

www.it-ebooks.info

5.7 Lomb-Scargle Power Spectrum

The power spectrum methods introduced in the previous sections require evenly-spaced data. In earth sciences, however, time series are often unevenly spaced. Although interpolating the unevenly-spaced data to a grid of evenly-spaced times is one way to overcome this problem (Section 5.5), interpolation introduces numerous artifacts into the data, in both the time and frequency domains. For this reason an alternative method of time-series analysis has become increasingly popular in earth sciences, the Lomb-Scargle algorithm (e.g., Scargle 1981, 1982, 1989, 1990, Press et al. 1992, Schulz et al. 1998).

The Lomb-Scargle algorithm only evaluates the data of the time series at the times t_i that are actually measured. Assuming a series $y(t)$ of N data points, the Lomb-Scargle normalized periodogram P_x, as a function of angular frequency $\omega=2\pi f > 0$, is given by

$$P_x(\omega)=\frac{1}{2s^2}\left\{\frac{\left[\sum_j (y_i-\bar{y})\cos\omega(t_j-\tau)\right]^2}{\sum_j \cos^2\omega(t_j-\tau)}+\frac{\left[\sum_j (y_i-\bar{y})\sin\omega(t_j-\tau)\right]^2}{\sum_j \sin^2\omega(t_j-\tau)}\right\}$$

where

$$\bar{y}=\frac{1}{N}\sum_{i=1}^{N} y_i$$

and

$$s^2=\frac{1}{N-1}\sum_{i=1}^{N}(y_i-\bar{y})^2$$

are the arithmetic mean and the variance of the data (Section 3.2). The constant τ, which is defined by the relationship

$$\tan(2\omega\tau)=\frac{\sum_j \sin 2\omega t_j}{\sum_j \cos 2\omega t_j}$$

is an offset that makes $P_x(\omega)$ independent of shifting the t_i values by any

www.it-ebooks.info

constant amount. Scargle (1982) showed that this particular choice of the offset τ has the consequence that the solution for $P_x(\omega)$ is identical to a least-squares fit of sine and cosine functions to the data series $y(t)$:

$$y(t) = A\cos\omega t + B\sin\omega t$$

The least-squares fit of harmonic functions to data series in conjunction with spectral analysis had previously been investigated by Lomb (1976), and hence the method is called the normalized Lomb-Scargle Fourier transform. The term *normalized* refers to the factor s^2 in the dominator of the equation for the periodogram.

Scargle (1982) has shown that the Lomb-Scargle periodogram has an exponential probability distribution with a mean equal to one, assuming that the noise is Gaussian distributed. The probability that $P_x(\omega)$ will be between some positive quantity z and $z+dz$ is $\exp(-z)dz$. If we scan M independent frequencies, the probability of none of them having a value larger than z is $(1-\exp(-z))M$. We can therefore compute the false-alarm probability of the null hypothesis (i.e., the probability that a given peak in the periodogram is not significant) using

$$P(>z) \equiv 1-(1-e^{-z})^M$$

Press et al. (1992) suggested using the Nyquist criterion (Section 5.2) to determine the number of independent frequencies M, assuming that the data were evenly spaced. In this case, the appropriate value for the number of independent frequencies is $M=2N$, where N is the length of the time series.

More detailed discussions of the Lomb-Scargle method are given in Scargle (1989) and Press et al. (1992). An excellent summary of the method and a TURBO PASCAL program to compute the normalized Lomb-Scargle power spectrum of paleoclimatic data have been published by Schulz and Stattegger (1998). A convenient MATLAB algorithm `lombscargle` for computing the Lomb-Scargle periodogram has been published by Brett Shoelson (The MathWorks, Inc.) and can be downloaded from *File Exchange* at

`http://www.mathworks.de/matlabcentral/fileexchange/993-lombscargle-m`

The following MATLAB code is based on the original FORTRAN code published by Scargle (1989). Significance testing uses the methods proposed by Press et al. (1992) explained above.

We first load the synthetic data that were generated to illustrate the use of the evolutionary or windowed power spectrum method in Section 5.6.

www.it-ebooks.info

The data contain periodicities of 100, 40 and 20 kyrs, as well as additive Gaussian noise, and are unevenly spaced about the time axis. We define two new vectors `t` and `x` that contain the original time vector and the synthetic oxygen-isotope data sampled at times `t`.

```
clear

series3 = load('series3.txt');
t = series3(:,1);
x = series3(:,2);
```

We then generate a frequency axis `f`. Since the Lomb-Scargle method is not able to deal with the frequency of zero (i.e., with an infinite period) we start at a frequency value that is equivalent to the spacing of the frequency vector. The variable `ofac` is the oversampling parameter that influences the resolution of the frequency axis about the `N(frequencies)=N(datapoints)` case. We also need the highest frequency `fhi` that can be analyzed by the Lomb-Scargle algorithm: the Nyquist frequency `fnyq` that would be obtained if the `N` data points were evenly spaced over the same time interval is commonly used for `fhi`. The following code uses the input parameter `hifac`, which is defined by Press et al. (1992) as `hifac=fhi/fnyq`.

```
int = mean(diff(t));
ofac = 4; hifac = 1;
f = ((2*int)^(-1))/(length(x)*ofac): ...
    ((2*int)^(-1))/(length(x)*ofac): ...
    hifac*(2*int)^(-1);
```

where `int` is the mean sampling interval. We normalize the data by subtracting the mean.

```
x = x - mean(x);
```

We can now compute the normalized Lomb-Scargle periodogram `px` as a function of the angular frequency `wrun` using the translation of Scargle's FORTRAN code into MATLAB code.

```
for k = 1:length(f)
    wrun = 2*pi*f(k);
    px(k) = 1/(2*var(x)) * ...
       ((sum(x.*cos(wrun*t - ...
       atan2(sum(sin(2*wrun*t)),sum(cos(2*wrun*t)))/2))).^2) ...
       /(sum((cos(wrun*t - ...
       atan2(sum(sin(2*wrun*t)),sum(cos(2*wrun*t)))/2)).^2)) + ...
       ((sum(x.*sin(wrun*t - ...
       atan2(sum(sin(2*wrun*t)),sum(cos(2*wrun*t)))/2))).^2) ...
       /(sum((sin(wrun*t - ...
       atan2(sum(sin(2*wrun*t)),sum(cos(2*wrun*t)))/2)).^2));
end
```

www.it-ebooks.info

The significance level for any peak in the power spectrum `px` can now be computed. The variable `prob` indicates the false-alarm probability for the null hypothesis: a low `prob` therefore indicates a highly significant peak in the power spectrum.

```
prob = 1-(1-exp(-px)).^(2*length(x));
```

We now plot the power spectrum and the probabilities (Fig. 5.14):

```
plot(f,px)
xlabel('Frequency')
ylabel('Power')
title('Lomb-Scargle Power Spectrum')

figure
plot(f,prob)
xlabel('Frequency')
ylabel('Probability')
title('Probabilities')
```

The two plots suggest that all three peaks are highly significant since the errors are extremely low at the cyclicities of 100, 40 and 20 kyrs.

An alternative way of displaying the significance levels was suggested by Press et al. (1992). In this method the equation for the false-alarm probability of the null hypothesis is inverted to compute the corresponding power of the significance levels. As an example we choose a significance level of 95%. However, this number can also be replaced by a vector of several significance levels such as `signif=[0.90 0.95 0.99]`. We can now type

```
m = floor(0.5*ofac*hifac*length(x));
effm = 2*m/ofac;
signif = 0.95;
levels = log((1-signif.^(1/effm)).^(-1));
```

where `m` is the true number of independent frequencies and `effm` is the effective number of frequencies using the oversampling factor `ofac`. The second plot displays the spectral peaks and the corresponding probabilities.

```
plot(f,px)
hold on
for k = 1:length(signif)
    line(f,levels(:,k)*ones(size(f)),'LineStyle','--')
end
xlabel('Frequency')
ylabel('Power')
title('Lomb-Scargle Power Spectrum')
hold off
```

All three spectral peaks at frequencies of 0.01, 0.025 and 0.05 kyr^{-1} exceed the

www.it-ebooks.info

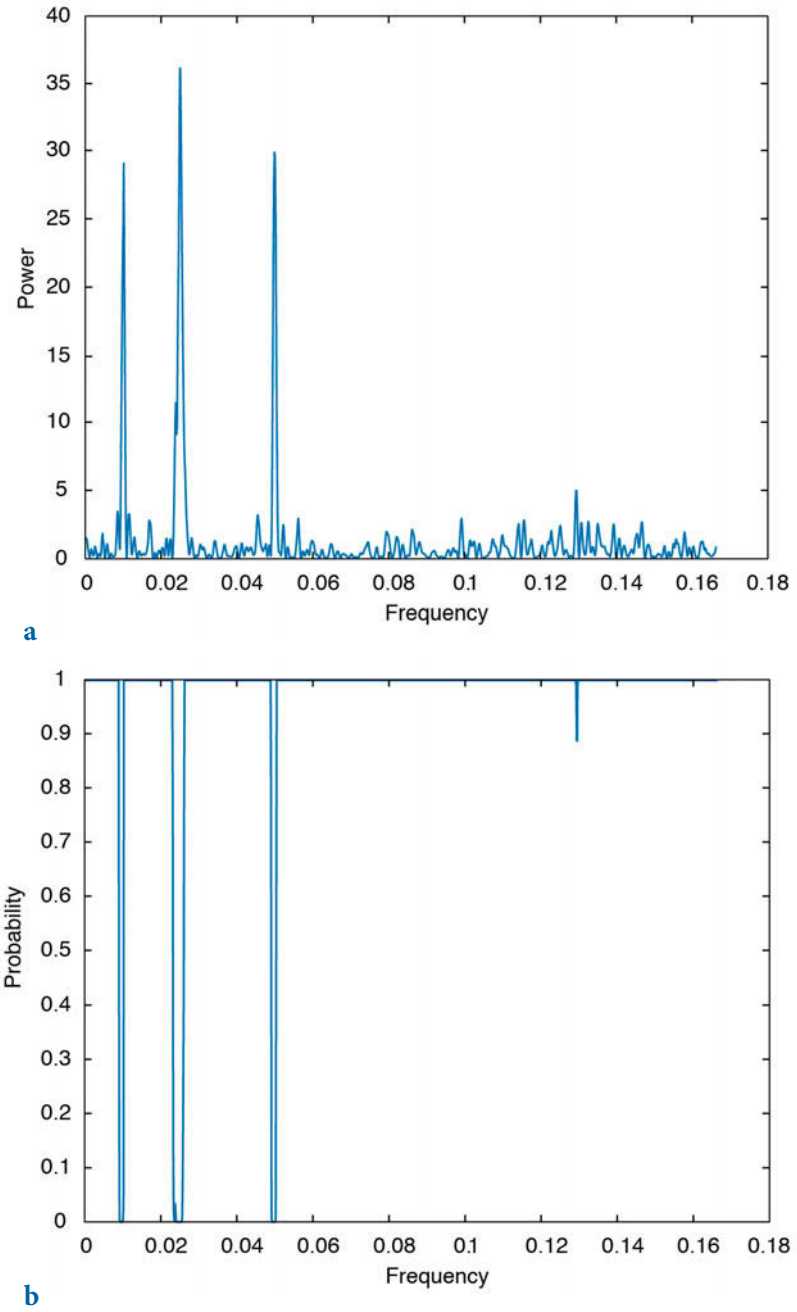


Fig. 5.14 **a** Lomb-Scargle power spectrum and **b** the false-alarm probability of the null hypothesis. The plot suggests that the 100, 40 and 20 kyr cycles are highly significant.

www.it-ebooks.info

95% significant level, suggesting that they represent significant cyclicities. We have therefore obtained similar results to those obtained using the periodogram method. However, the Lomb-Scargle method has the advantage that is does not require any interpolation of unevenly-spaced data, as well as permitting quantitative significance testing.

5.8 Wavelet Power Spectrum

Section 5.6 demonstrated the use of a modification to the power spectrum method for mapping changes in cyclicity through time. A similar modification could, in theory, be applied to the Lomb-Scargle method, which would have the advantage that it could then be applied to unevenly-spaced data. Both methods, however, assume that the data are composites of sine and cosine waves that are globally uniform in time and have infinite time spans. The evolutionary power spectrum method divides the time series into overlapping segments and computes the Fourier transform of these segments. To avoid spectral leakage, the data are multiplied by windows that are smooth bell-shaped curves with positive values (Section 5.3). The higher the temporal resolution of the evolutionary power spectrum the lower the accuracy of the result. Moreover, short time windows contain a large number of high-frequency cycles whereas the low-frequency cycles are underrepresented.

In contrast to the Fourier transform, the *wavelet transform* uses base functions (*wavelets*) that have smooth ends *per se* (Lau and Weng 1995, Mackenzie et al. 2001). Wavelets are small packets of waves; they are defined by a specific frequency and decay towards either end. Since wavelets can be stretched and translated in both frequency and time, with a flexible resolution, they can easily map changes in the time-frequency domain. We use the functions for wavelet analysis that are included in the Wavelet Toolbox (MathWorks 2014b). There is also, however, a very popular wavelet toolbox produced by Christopher Torrence and Gilbert P. Compo (1998), which is freely available online from

`http://paos.colorado.edu/research/wavelets/`

A wavelet transformation mathematically decomposes a signal $y(t)$ into elementary functions $\psi_{a,b}(t)$ derived from a *mother wavelet* $\psi(t)$, by dilation and translation,

$$\psi_{a,b} = \frac{1}{(a)^{1/2}} \psi\left(\frac{t-b}{a}\right)$$

www.it-ebooks.info

where b denotes the position (translation) and a (>0) the scale (dilation) of the wavelet (Lau and Weng 1995). The wavelet transform of the signal $y(t)$ about the mother wavelet $\psi(t)$ is defined as the convolution integral

$$W(b,a) = \frac{1}{(a)^{1/2}} \int \psi^* \left(\frac{t-b}{a} \right) y(t)\, dt$$

where ψ^* is the complex conjugate of ψ. There are many mother wavelets available in the literature, such as the classic *Haar* wavelet, the *Morlet* wavelet, or the *Daubechies* wavelet. The most popular wavelet in geosciences is the Morlet wavelet introduced by French geophysicist Jean Morlet (1931–2007), which is defined by

$$\psi_0(\eta) = \pi^{-1/4} \exp(i\omega_0 \eta) \exp(-\eta^2/2)$$

where η is the time and ω_0 is the wave number (Torrence and Compo 1998). The wave number is the number of oscillations within the wavelet itself. We can easily compute a discrete version of the Morlet wavelet `wave` by translating the above equation into MATLAB code, where `eta` is the non-dimensional time and `w0` is the wave number. Changing `w0` produces wavelets with different wave numbers. Note that it is important not to use `i` for index in `for` loops, since it is used here for imaginary unit (Fig. 5.15).

```
clear

eta = -10 : 0.1 : 10;
w0 = 6;
wave = pi.^(-1/4) .* exp(i*w0*eta) .* exp(-eta.^2/2);

plot(eta,wave)
xlabel('Position')
ylabel('Scale')
title('Morlet Mother Wavelet')
```

In order to familiarize ourselves with wavelet power spectra, we use a pure sine wave with a period five and additive Gaussian noise.

```
clear

rng(0)
t = 0 : 0.5 : 50;
x = sin(2*pi*t/5) + randn(size(t));
```

As a first step, we need to define the mother wavelet and its wave number `w0`.

```
mother = 'morl';
w0 = 6;
```

www.it-ebooks.info

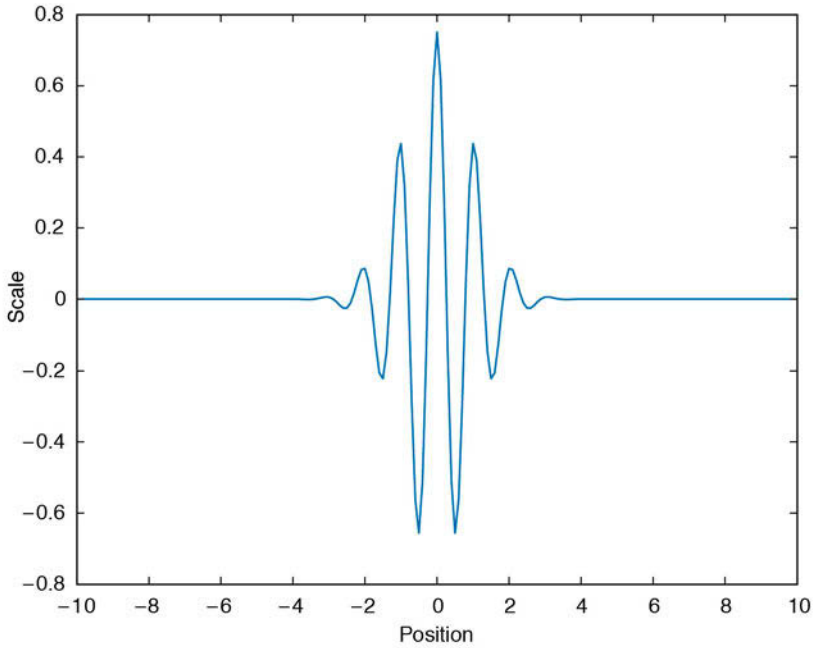


Fig. 5.15 Morlet mother wavelet with wave number 6.

We then need to define the values of the scales for which the wavelet transform will be computed. These values define how much a wavelet is stretched or compressed to map the variability of the time series at different wavelengths. Scales with smaller values correspond to higher frequencies and can therefore map rapidly-changing details, whereas those with higher values can map the long-term variations. The definition of the scales first requires the sampling interval `dt` of our time series `x`. We then use the default spacing `ds` of 0.4875 for a Morlet wavelet, following the instructions contained in the Wavelet Toolbox manual (MathWorks 2014b). The smallest value for the scales `s0` is usually chosen to be twice the sampling interval, i.e., `2*dt`. We next calculate the number of scales `nb`, which depends on the length of the time series and the spacing of the scales. Finally, we calculate the scales `scales` themselves depending on the smallest scale, the number of scales, and the spacing of the scales, using equations provided in the Wavelet Toolbox manual (MathWorks 2014b).

```
dt = 0.5;
ds = 0.4875;
s0 = 2*dt;
nb = fix(log2(length(x))/ds)+1;
scales = s0*2.^((0:nb-1)*ds);
```

www.it-ebooks.info

In the next step we compute the real or complex continuous wavelet coefficients using the function `cwt` contained in the Wavelet Toolbox.

```
coefs = cwt(x,scales,mother);
```

The function `scal2frq` converts scales `scales` to pseudo-frequencies, using the mother wavelet `mother` and the sampling period `dt`.

```
f = scal2frq(scales,mother,dt);
```

We use a filled contour plot to portray the power spectrum, i.e., the absolute value of the wavelet coefficients (Fig. 5.16 a).

```
contour(t,f,abs(coefs),...
   'LineStyle','none',...
   'LineColor',[0 0 0],...
   'Fill','on')
xlabel('Time')
ylabel('Frequency')
title('Wavelet Power Spectrum')
set(gcf,'Colormap',jet)
set(gca,'YLim',[0 0.9],...
   'XGrid','On',...
   'YGrid','On')
```

Alternatively, we can compute the wavelet transform using the fast Fourier transform (FFT) algorithm implemented in the function `cwtft`. This approach is used in the freely available wavelet toolbox produced by Torrence and Compo (1998). We first define the scales using the values for the smallest scales `s0`, the sampling interval `ds` of the scales, and the number of scales `nb` from above, merged into a structure array `sc`.

```
sc.s0 = s0;
sc.ds = ds;
sc.nb = nb;
```

Then, we create a structure array `sig` that contains the signal `x`, the sampling interval (or period) `dt`, the mother wavelet `mother`, and the scales `sc`.

```
sig = struct('val',x,...
    'period',dt,...
    'wavelet',mother,...
    'scales',sc);
```

The output from `cwtft` is a structure array `cwtstruct` that includes the wavelet coefficients `cfs` and the scales `scales`. The default mother wavelet is the Morlet wavelet.

```
cwtstruct = cwtft(sig);
```

www.it-ebooks.info

We convert the scales to pseudo-frequencies using the equation for the Morlet wavelet, which we find in the wavelet definitions in the `cwtft` help section.

```
f = 1./(4*pi*cwtstruct.scales/(w0+sqrt(2+w0^2)));
```

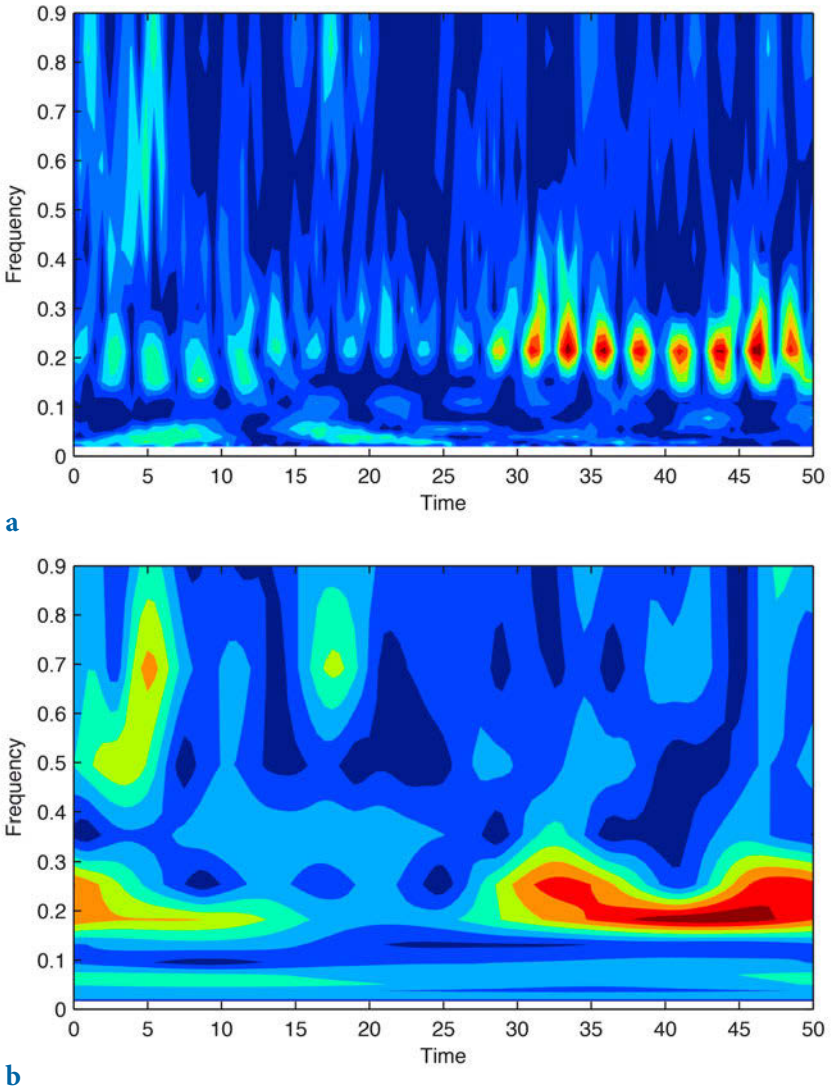


Fig. 5.16 Wavelet power spectrum showing a significant period at 5 cycles that persists throughout the full length of the time vector. The wavelet power spectrum has been calculated using **a** the continuous 1D wavelet transform `cwt` and **b** the continuous wavelet transform using the FFT algorithm `cwtft`.

www.it-ebooks.info

We again use a filled contour plot to portray the power spectrum, i.e., the absolute value of the wavelet coefficients (Fig. 5.16 b).

```
contour(t,f,abs(cwtstruct.cfs),...
    'LineStyle','none',...
    'LineColor',[0 0 0],...
    'Fill','on')
xlabel('Time')
ylabel('Frequency')
title('Wavelet Power Spectrum Using FFT Algorithm')
set(gcf,'Colormap',jet)
set(gca,'YLim',[0 0.9],...
    'XGrid','On',...
    'YGrid','On')
```

As we can see, the wavelet power spectrum derived using `cwtft` is much smoother than that computed with `cwt`, since `cwtft` uses sinusoids to smooth the coefficients. However, the smoothing causes a significant loss of detail in the contour plot.

We now apply this concept to the synthetic data from the example to demonstrate the windowed power spectrum method and load the synthetic data contained in file `series3.txt`, remembering that the data contain periodicities of 100, 40, and 20 kyrs as well as additive Gaussian noise, and that they are unevenly spaced about the time axis.

```
clear

series3 = load('series3.txt');
```

As for the Fourier transform and in contrast to the Lomb-Scargle algorithm, the wavelet transform requires evenly-spaced data, and we therefore interpolate the data using `interp1`.

```
t = 0 : 3 : 1000;
series3L = interp1(series3(:,1),series3(:,2),t,'linear');
```

Again, we first need to define the mother wavelet and its wave number `w0`.

```
mother = 'morl';
w0 = 6;
```

We then define the scales, as demonstrated in the first example. Unlike the previous example the sampling interval `dt` of our time series is now 3.

```
dt = 3;
ds = 0.4875;
s0 = 2*dt;
nb = fix(log2(length(series3L))/ds)+1;
scales = s0*2.^((0:nb-1)*ds);
```

www.it-ebooks.info

We compute the wavelet coefficients using `cwt`.

```
coefs = cwt(series3L,scales,mother);
```

We convert the scales `scales` to pseudo-frequencies using the mother wavelet `mother` and the sampling period `dt`.

```
f = scal2frq(scales,mother,dt);
```

We use a filled contour plot to portray the power spectrum (Fig. 5.17 a).

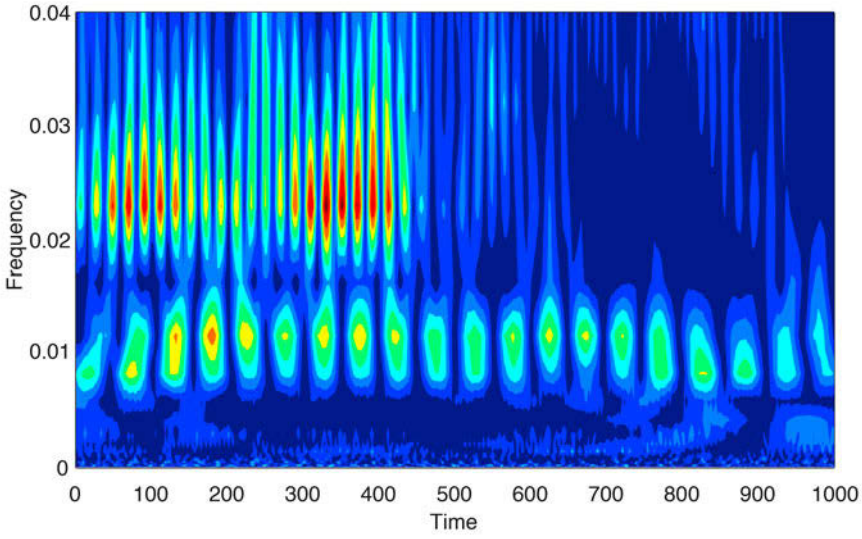
```
contour(t,f,abs(coefs),...
   'LineStyle','none',...
   'LineColor',[0 0 0],...
   'Fill','on')
xlabel('Time')
ylabel('Frequency')
title('Wavelet Power Spectrum')
set(gcf,'Colormap',jet)
set(gca,'YLim',[0 0.04],...
   'XGrid','On',...
   'YGrid','On')
```

The graph shows horizontal clusters of peaks at around 0.01 and 0.025 kyr^{-1}, corresponding to 100 and 40 kyr cycles. The 40 kyr cycle (a frequency of 0.025 kyr^{-1}) only appears at ca. 450 kyrs before present. Using `cwtft` instead of `cwt` again creates a much smoother result (Fig. 5.17 b).

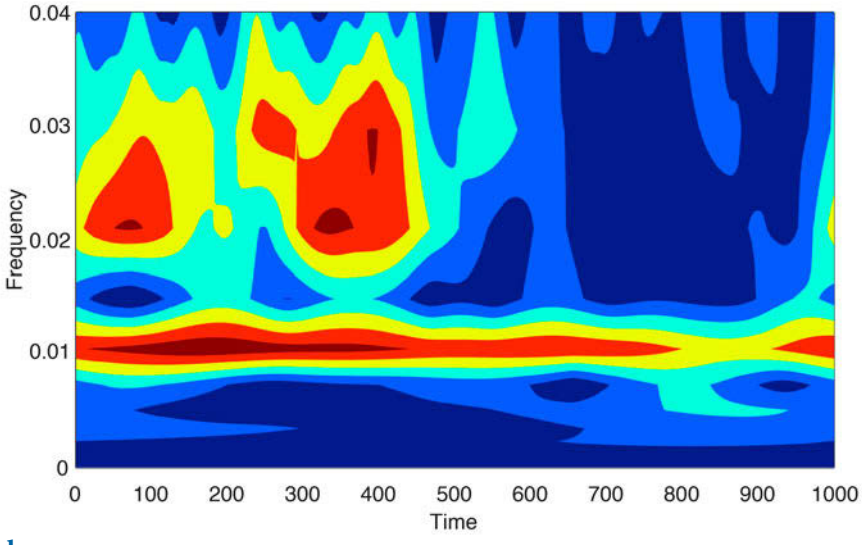
```
sc.s0 = s0;
sc.ds = ds;
sc.nb = nb;
sig = struct('val',series3L,...
    'period',dt,...
    'wavelet',mother,...
    'scales',sc);
cwtstruct = cwtft(sig);
scales = cwtstruct.scales

f = 1./(4*pi*cwtstruct.scales/(w0+sqrt(2+w0^2)));

contour(t,f,abs(cwtstruct.cfs),...
    'LineStyle','none',...
    'LineColor',[0 0 0],...
    'Fill','on')
xlabel('Time')
ylabel('Frequency')
title('Wavelet Power Spectrum Using FFT Algorithm')
set(gcf,'Colormap',jet)
set(gca,'YLim',[0 0.04],...
    'XGrid','On',...
    'YGrid','On')
```

Compared to the windowed power spectrum method, the wavelet power

www.it-ebooks.info



a



b

Fig. 5.17 Wavelet power spectrum for the synthetic data series contained in *series_3.txt*. The plot clearly shows significant periodicities at frequencies of 0.01, 0.025, and 0.05 kyr^{-1} corresponding to the 100, 40, and 20 kyr cycles. The 100 kyr cycle is present throughout the entire time series, whereas the 40 kyr cycle only appears at around 450 kyrs before present. The 20 kyr cycle is relatively weak but is probably present throughout the entire time series. The wavelet power spectrum has been calculated using **a** the continuous 1D wavelet transform `cwt` and **b** the continuous wavelet transform using FFT algorithm `cwtft`.

www.it-ebooks.info

spectrum clearly shows a much higher resolution on both the time and the frequency axes. Instead of dividing the time series into overlapping segments and computing the power spectrum for each segment, the wavelet transform uses short packets of waves that better map temporal changes in the cyclicities. The disadvantage of both the windowed power spectrum and the wavelet power spectrum is, however, the requirement for evenly-spaced data. The Lomb-Scargle method overcomes this problem but (as with the power spectrum method) has limitations in its ability to map temporal changes in the frequency domain.

5.9 Detecting Abrupt Transitions in Time Series

A number of methods are available to detect abrupt changes in time series in the time domain. An example of such such methods for use in climate time series is the *rampfit method* (Mudelsee and Stattegger 1997, Mudelsee 2000), and examples suitable for use in the frequency domain are the evolutionary *power spectrum* and the *wavelet power spectrum* (e.g., Lau and Weng 1995, Mackenzie et al. 2001). In most cases, trends and events in both time and frequency domains are detected by computing the statistical parameters of the data (e.g., measures of central tendency and dispersion) contained in a sliding window of length L. The precision of these parameters depends on the length of the window, i.e., an accurate value for the mean and the variance is obtained if L is large. However, a larger window reduces the accuracy of the estimated changes in these parameters. This problem is often referred to as *Grenander's uncertainty principle of statistics* (Grenander 1958). Performing a statistical test to assess differences in central tendency and dispersion between two different sliding windows, however, partly overcomes this problem, provided only the location of a sharp transition in statistical parameters is required.

The classic t-test and F-test statistics are often used to compare the means and variances of two sets of measurements and could therefore be used to detect changes in the location and dispersion between two sliding windows. These two tests, however, make the basic assumption that the samples came from a population with a Gaussian distribution (Sections 3.7 and 3.8). The non-parametric Mann-Whitney and Ansari-Bradley tests provide a solution to this problem that is independent of the distribution (Sections 3.11 and 3.12). The Mann-Whitney test (Mann and Whitney 1947, Lepage 1971) performs a two-sided rank sum test of the null hypothesis that two samples come from identical continuous distributions with identical medians, against the alternative that they do not have identical medians. The Ansari-Bradley test performs a two-sided test that two independent samples come from the

www.it-ebooks.info

same distribution, against the alternative that they come from distributions that have the same median and shapes but different dispersions (Ansari and Bradley 1960, Lepage 1971).

The example below demonstrates the Mann-Whitney and Ansari-Bradley tests on two synthetic records that contain significant changes in the central tendency (mean, median, mode) and dispersion (range, variance, quantiles) in the middle of the time series (Fig. 5.18). The time axis runs from 0.1 to 500 kyr at sampling intervals of 0.1 kyr. At 250 kyr the mean of the log-normal distributed data changes abruptly from 1.0 to 1.5 and the standard deviation changes from 0.5 to 1.3 (Fig. 5.18 a).

```
clear
rng(0)
t = 0.1 : 0.1 : 500;
y1 = 0.1 * random('logn',1,   0.5, 1, length(t),1);
y2 = 0.1 * random('logn',1.5, 1.3, 1, length(t),1);
y = y1(1:length(t)/2);
y(length(t)/2+1:length(t)) = y2(length(t)/2+1:length(t));
```

We first use a Mann-Whitney test with paired sliding windows of three different lengths, in order to detect any abrupt change in the mean. We choose sliding window lengths of 300, 500, and 1,000 data points, i.e., in each step we apply the Mann-Whitney test to two samples of 150 data points, two samples of 250 data points, and two samples of 500 data points. Note that when running a Mann-Whitney test on different sets of data the length of the window needs to be adjusted to the length of the time series, and to the required accuracy with which the transition in the mean is to be identified.

```
w = [300 500 1000];
```

We use the function `ranksum` introduced in Section 3.11 to perform the Mann-Whitney test.

```
for j = 1:length(w)
na = w(j);
nb = w(j);
for i = w(j)/2+1:length(y)-w(j)/2
    [p,h] = ranksum(y(i-w(j)/2:i-1),y(i+1:i+w(j)/2));
    mwreal(j,i) = p;
end
mwreal(j,1:w(j)/2) = mwreal(j,w(j)/2+1) * ones(1,w(j)/2);
mwreal(j,length(y)-w(j)/2+1:length(y)) = ...
        mwreal(j,length(y)-w(j)/2) * ones(1,w(j)/2);
end
```

We then display the results.

```
subplot(2,1,1)
```

www.it-ebooks.info

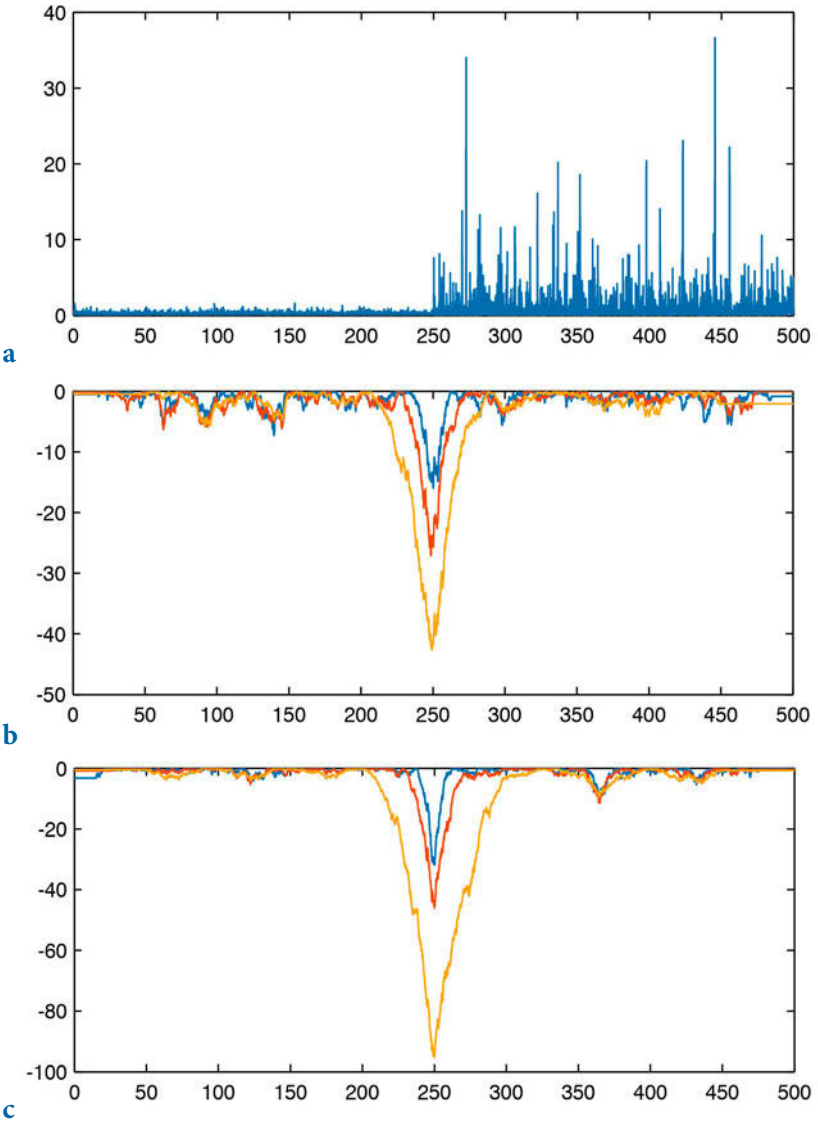


Fig. 5.18 Mann-Whitney and Ansari-Bradley tests on synthetic dust flux record. **a** Lognormal distributed noise. After 250 kyrs the mean and variance of the data shifts towards a lower value. **b** Result of a Mann-Whitney test for three different lengths of the paired sliding windows (150, 250 and 500 data points, equivalent to 15, 25 and 50 kyrs). The length of the window clearly influences the amplitudes and widths of the parameter maxima, whereas the location of the transition in the mean is well defined. **c** Result of a Ansari-Bradley test for three different lengths of the paired sliding windows (150, 250 and 500 data points, equivalent to 15, 25 and 50 kyrs). The length of the window clearly influences the amplitudes and widths of the parameter maxima, and the location of the transition in the dispersion is well defined.

www.it-ebooks.info

```
plot(t,y)
title('Synthetic signal of lognormal distributed noise')
subplot(2,1,2)
plot(t,log(mwreal))
title('Results from Mann-Whitney U-test')
```

The result from the Mann-Whitney test reveals that the length of the window influences the amplitudes and widths of the maxima of the test parameter, whereas the location of the transition in the means is well defined (Fig. 5.18 b). We next use an Ansari-Bradley test for the same three different lengths of paired sliding windows (150, 250 and 500 data points) to detect any abrupt change in the standard deviation. We use the function `ansaribradley` introduced in Section 3.12 to perform the Ansari-Bradley test.

```
for j = 1:length(w)
df1 = w(j) - 1;
df2 = w(j) - 1;
for i = w(j)/2+1:length(y)-w(j)/2
    [h,p] = ansaribradley(y(i-w(j)/2:i-1),y(i+1:i+w(j)/2));
    abreal(j,i) = p;
end
abreal(j,1:w(j)/2) = abreal(j,w(j)/2+1) * ones(1,w(j)/2);
abreal(j,length(y)-w(j)/2+1:length(y)) = ...
    abreal(j,length(y)-w(j)/2) * ones(1,w(j)/2);
end
```

We then display the results.

```
subplot(2,1,1)
plot(t,y)
title('Synthetic signal of lognormal distributed noise')
subplot(2,1,2)
plot(t,log(abreal))
title('Results from Ansari-Bradley test')
```

The length of the window again clearly influences the amplitudes and widths of the maxima of the test parameters, and the location of the transition in the dispersion is again well defined (Fig. 5.18 c). This method has been successfully applied to records of terrigenous dust flux preserved in marine sediments offshore subtropical West Africa, the eastern Mediterranean Sea, and the Arabian Sea, in order to detect trends, rhythms and events in the African Plio-Pleistocene climate (Trauth et al. 2009).

5.10 Nonlinear Time-Series Analysis (by N. Marwan)

The methods described in the previous sections detect linear relationships in the data. However, natural processes on the earth often show a more complex and chaotic behavior, and methods based on linear techniques may

www.it-ebooks.info

therefore yield unsatisfactory results. In recent decades, new techniques for nonlinear data analysis derived from chaos theory have become increasingly popular. Such methods have been employed to describe nonlinear behavior by, for example, defining the scaling laws and fractal dimensions of natural processes (Turcotte 1997, Kantz and Schreiber 1997). However, most methods of nonlinear data analysis require either long or stationary data series and these requirements are rarely satisfied in the earth sciences. While most nonlinear techniques work well on synthetic data, these methods are unable to describe nonlinear behavior in real data.

During the last decades, *recurrence plots* have become very popular in science and engineering as a new method of nonlinear data analysis (Eckmann 1987, Marwan 2007). Recurrence is a fundamental property of dissipative dynamical systems. Although small disturbances in such systems can cause exponential divergence in their states, after some time the systems will return to a state that is close to a former state and then pass again through a similar evolution. Recurrence plots allow such recurrent behavior of dynamical systems to be visually portrayed. The method is now widely accepted as a useful tool for the nonlinear analysis of short and nonstationary data sets.

Phase Space Portrait

The starting point for most nonlinear data analyses is the construction of a phase space portrait for a system. The state of a system can be described by its state variables $x_1(t)$, $x_2(t)$, …, $x_d(t)$. As an example, suppose the two variables *temperature* and *pressure* are used to describe the thermodynamic state of the earth's mantle as a complex system. The d state variables at time t form a vector in a d-dimensional space, which is known as the phase space. The state of a system typically changes with time and the vector in the phase space therefore describes a trajectory representing the temporal evolution (i.e., the dynamics) of the system. The trajectory provides essential information on the dynamics of the system, such as whether systems are periodic or chaotic.

In many applications the observation of a natural process does not yield all possible state variables, either because they are not known or because they cannot be measured. However, due to coupling between the system's components, we can reconstruct a phase space trajectory from a single observation u_i:

$$x_i = (u_i, u_{i+\tau}, \ldots, u_{i+(m-1)\tau})^T$$

where m is the embedding dimension and τ is the time delay (index based;

www.it-ebooks.info

the real time delay is $\tau=\Delta t$). This reconstruction of the phase space is called *time delay embedding*. The reconstruction of the phase space is not exactly the same as the original phase space, but its topological properties are preserved provided that the embedding dimension is sufficiently large. In practice, the embedding dimension must be more than twice the dimension of the attractor (i.e., $m>2d+1$). The reconstructed trajectory is then sufficiently accurate for subsequent data analysis.

As an example we now explore the phase space portrait of a harmonic oscillator such as an undamped pendulum. We first create the position vector `x1` and the velocity vector `x2`

```
clear

t = 0 : pi/10 : 3*pi;
x1 = sin(t);
x2 = cos(t);
```

The phase space portrait

```
plot(x1,x2)
xlabel('x_1')
ylabel('x_2')
```

is a circle, suggesting an exact recurrence of each state after one complete cycle (Fig. 5.19). Using the time delay embedding we can reconstruct this phase space portrait using only a single observation, e.g., the velocity vector, and a time delay of five, which corresponds to a quarter of the period of our pendulum.

```
tau = 5;
plot(x2(1:end-tau),x2(1+tau:end))
xlabel('x_1')
ylabel('x_2')
```

As we can see, the reconstructed phase space is almost the same as the original phase space. Next, we compare this phase space portrait with one for a typical nonlinear system, the Lorenz system (Lorenz 1963). Weather patterns often to not change in a predictable manner. In 1963, Edward Lorenz introduced a simple three-dimensional model to describe the chaotic behavior exhibited by turbulence in the atmosphere. The variables defining the Lorenz system are the intensity of atmospheric convection, the temperature difference between ascending and descending currents, and the distortion of the vertical temperature profiles from linearity. Small variations in the initial conditions can cause dramatically divergent weather patterns, a behavior often referred to as the butterfly effect. The dynamics of the Lorenz

www.it-ebooks.info

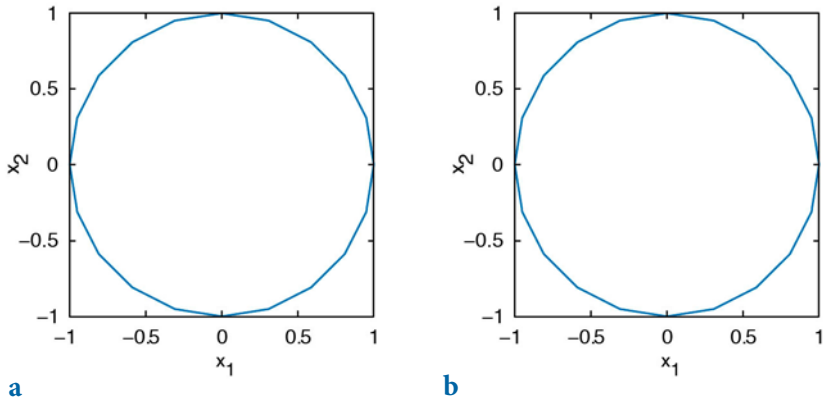


Fig. 5.19 a Original, and **b** reconstructed phase space portrait for a periodic system. The reconstructed phase space is almost the same as the original phase space.

system are described by three coupled nonlinear differential equations:

$$\frac{dx}{dt} = s(y(t) - x(t)),$$
$$\frac{dy}{dt} = -x(t)z(t) + rx(t) - y(t),$$
$$\frac{dz}{dt} = x(t)y(t) - bz(t).$$

Integrating the differential equation yields a simple MATLAB code for computing the *xyz* triplets of the Lorenz system. As system parameters controlling the chaotic behavior we use `s=10`, `r=28` and `b=8/3`; the time delay is `dt=0.01`. The initial values for the position vectors are `x1=8`, `x2=9` and `x3=25`. These values, however, can be changed to any other values, which of course will then change the behavior of the system.

```
clear

dt = .01;
s = 10;
r = 28;
b = 8/3;
x1 = 8; x2 = 9; x3 = 25;
for i = 1 : 5000
   x1 = x1 + (-s*x1*dt) + (s*x2*dt);
   x2 = x2 + (r*x1*dt) - (x2*dt) - (x3*x1*dt);
   x3 = x3 + (-b*x3*dt) + (x1*x2*dt);
   x(i,:) = [x1 x2 x3];
end
```

www.it-ebooks.info

Typical traces of a variable (such as the first variable) can be viewed by plotting `x(:,1)` over time (Fig. 5.20).

```
t = 0.01 : 0.01 : 50;
plot(t,x(:,1))
xlabel('Time')
ylabel('Temperature')
```

We next plot the phase space portrait for the Lorenz system (Fig. 5.21).

```
plot3(x(:,1),x(:,2),x(:,3))
grid, view(70,30)
xlabel('x_1')
ylabel('x_2')
zlabel('x_3')
```

In contrast to the simple periodic system described above, the trajectories of the Lorenz system obviously do not precisely follow the previous course, but recur very close to it. Moreover, if we follow two very close segments of the trajectory, we see that they run into different regions of the phase space with time. The trajectory is obviously circling around a fixed point in the phase space and then, after a random time period, circling around another. The curious orbit of the phase states around fixed points is known as the Lorenz attractor.

These observed properties are typical of chaotic systems. While small disturbances in such a system cause exponential divergences in its state, the system returns approximately to a previous state through a similar course. The reconstruction of the phase space portrait using only the first state and a time delay of six

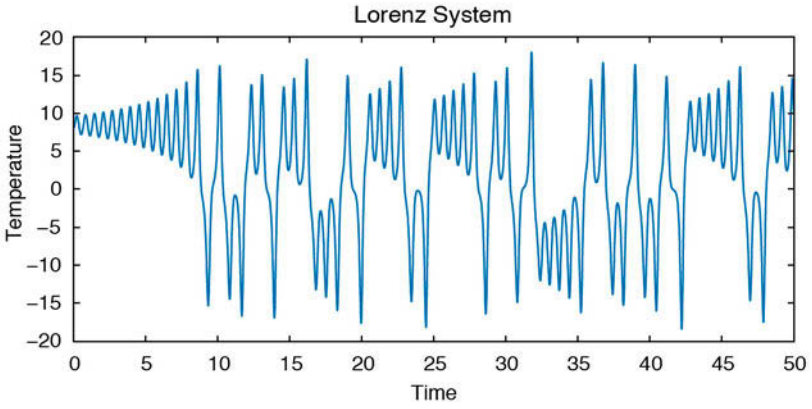


Fig. 5.20 The Lorenz system. As system parameters we use `s=10`, `r=28` and `b=8/3`; the time delay is `dt=0.01`.

www.it-ebooks.info

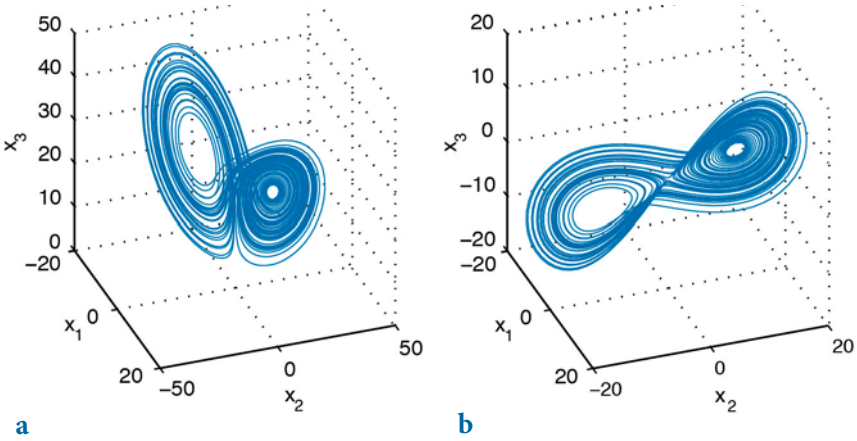


Fig. 5.21 a The phase space portrait for the Lorenz system. In contrast to the simple periodic system, the trajectories of the Lorenz system obviously do not follow precisely the previous course, but recur very close to it. **b** The reconstruction of the phase space portrait using only the first state and a time delay of 6 reveals a topologically similar phase portrait to a, with the two typical ears.

```
tau = 6;
plot3(x(1:end-2*tau,1),x(1+tau:end-tau,1),x(1+2*tau:end,1))
grid, view([100 60])
xlabel('x_1'), ylabel('x_2'), zlabel('x_3')
```

Movie 5.3

reveals a similar phase portrait with the two typical ears (Fig. 5.21). The characteristic properties of chaotic systems can also be observed in this reconstruction.

The time delay and embedding dimension need to be chosen from a previous analysis of the data. The delay can be estimated with the help of the autocovariance or autocorrelation function. For our example of a periodic oscillation,

```
clear

t = 0 : pi/10 : 3*pi;
x = sin(t);
```

we compute and plot the autocorrelation function

```
for i = 1 : length(x) - 2
    r = corrcoef(x(1:end-i),x(1+i:end));
    C(i) = r(1,2);
end

plot(C)
```

www.it-ebooks.info

```
xlabel('Delay'), ylabel('Autocorrelation')
grid on
```

We now choose a delay such that the autocorrelation function for the first time period equals zero. In our case this is five, which is the value that we have already used in our example of phase space reconstruction. The appropriate embedding dimension can be estimated using the false nearest neighbors method, or more simple, using recurrence plots, which are introduced in the next subsection. The embedding dimension is gradually increased until the majority of the diagonal lines are parallel to the line of identity.

The phase space trajectory or its reconstruction is the basis of several measures defined in nonlinear data analysis, such as *Lyapunov exponents*, *Rényi entropies*, or *dimensions*. The book on nonlinear data analysis by Kantz and Schreiber (1997) is recommended for more detailed information on these methods. Phase space trajectories or their reconstructions are also necessary for constructing recurrence plots.

Recurrence Plots

The phase space trajectories of dynamic systems that have more than three dimensions are difficult to portray visually. *Recurrence plots* provide a way of analyzing systems with higher dimensions. They can be used, e.g., to detect transitions between different regimes, or to detect interrelationships or synchronisations between different systems (Marwan 2007). The method was first introduced by Eckmann and others (1987). The recurrence plot is a tool that displays the recurrences of states in the phase space through a two-dimensional plot.

$$R_{i,j} = \begin{cases} 0 & \left\| x_i - x_j \right\| > \varepsilon \\ 1 & \left\| x_i - x_j \right\| \leq \varepsilon \end{cases}$$

If the distance between two states, i and j , on the trajectory is smaller than a given threshold ε, the value of the recurrence matrix R is one; otherwise it is zero. This analysis is therefore a pairwise test of all states. For N states we compute N^2 tests. The recurrence plot is then the two-dimensional display of the N-by-N matrix, where black pixels represent $R_{i,j}$=1 and white pixels indicate $R_{i,j}$=0, with a coordinate system representing two time axes. Such recurrence plots can help to find a preliminary characterization of the dynamics of a system or to find transitions and interrelationships within a system (cf. Fig. 5.22).

www.it-ebooks.info

As a first example we load the synthetic time series containing 100 kyr, 40 kyr and 20 kyr cycles already used in the previous sections. Since the data are unevenly spaced, we need to linearly interpolate the data to an evenly-spaced time axis.

```
clear

series1 = load('series1.txt');
t = 0 : 3 : 996;
series1L = interp1(series1(:,1),series1(:,2),t,'linear');
```

We start with the assumption that the phase space is only one-dimensional. Calculating the distances between all points of the phase space trajectory produces the distance matrix `s`.

```
N = length(series1L);
```

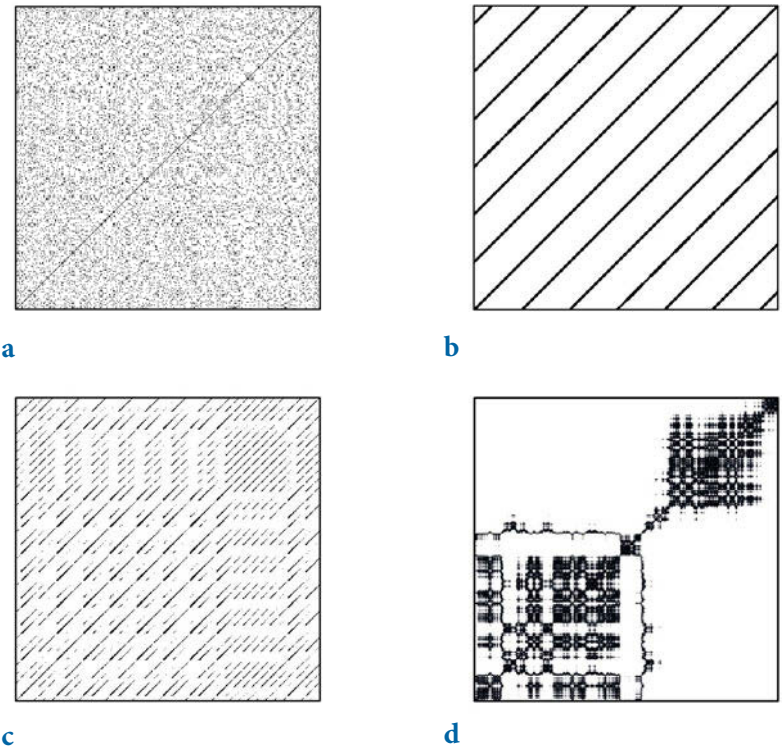


Fig. 5.22 Recurrence plots representing typical dynamical behaviors: **a** stationary uncorrelated data (white noise), **b** periodic oscillation, **c** chaotic data (Roessler system), and **d** non-stationary data with abrupt changes.

www.it-ebooks.info

```
S = zeros(N, N);

for i = 1 : N,
    S(:,i) = abs(repmat(series1L(i), N, 1 ) - series1L(:));
end
```

We can now plot the distance matrix

```
imagesc(t,t,S)
colormap jet
colorbar
xlabel('Time'), ylabel('Time')
axis xy
```

for the data set, where a colorbar provides a quantitative measure of the distances between states (Fig. 5.23). We now apply a threshold ε to the distance matrix to generate the black/white recurrence plot (Fig. 5.24).

```
imagesc(t,t,S<1)
colormap([1 1 1;0 0 0])
xlabel('Time'), ylabel('Time')
axis xy
```

Both plots reveal periodically recurring patterns. The distances between these periodically recurring patterns represent the cycles contained in the time series. The most significant periodic patterns have periods of 200 and 100 kyrs. The 200 kyr period is the most significant because of the superposition of the 100 and 40 kyr cycles, which are common divisors of 200 kyrs. Moreover, there are smaller substructures within the recurrence plot that have periods of 40 and 20 kyrs.

As a second example we now apply the method of recurrence plots to the Lorenz system. We again generate *xyz* triplets from the coupled differential equations.

```
clear

dt = .01;
s = 10;
r = 28;
b = 8/3;
x1 = 8; x2 = 9; x3 = 25;
for i = 1 : 5000
   x1 = x1 + (-s*x1*dt) + (s*x2*dt);
   x2 = x2 + (r*x1*dt) - (x2*dt) - (x3*x1*dt);
   x3 = x3 + (-b*x3*dt) + (x1*x2*dt);
   x(i,:) = [x1 x2 x3];
end
```

We then choose the resampled first component of this system and reconstruct a phase space trajectory by using an embedding of m=3 and τ=2.

www.it-ebooks.info

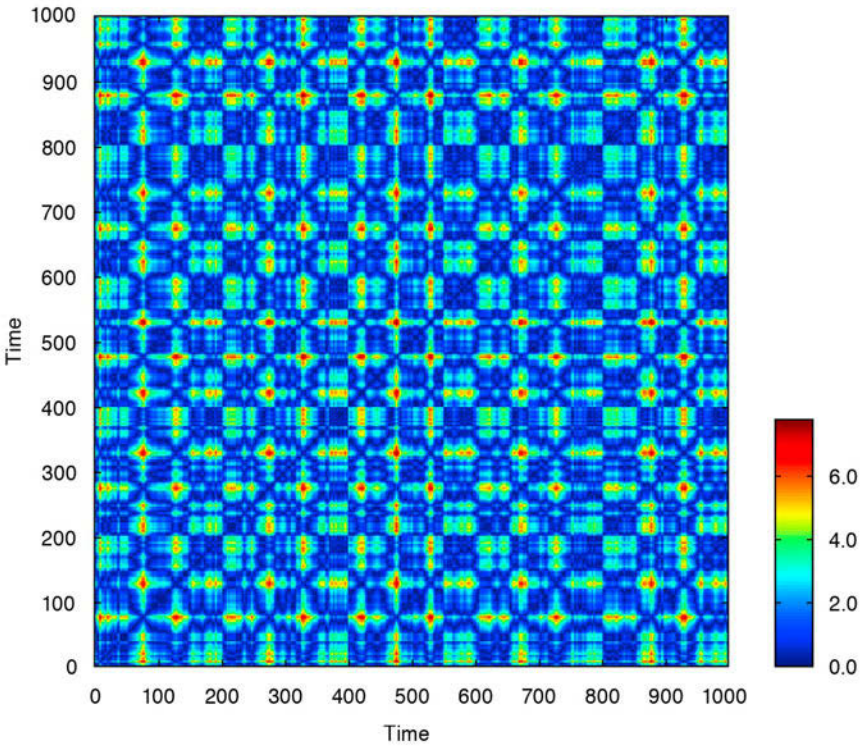


Fig. 5.23 Display of the distance matrix from the synthetic data, providing a quantitative measure for the distances between states at particular times; blue colors indicate small distances and red colors represent large distances.

```
t = 0.01 : 0.05 : 50;
y = x(1:5:5000,1);
m = 3; tau = 2;

N = length(y);
N2 = N - tau*(m - 1);
```

The original data series had a length of 5,000 data points, reduced to 1,000 data points (equivalent to 50 seconds), but because of the time delay method the reconstructed phase space trajectory has a length of 996 data points. We can create the phase space trajectory with

```
for mi = 1:m
   xe(:,mi) = y([1:N2] + tau*(mi-1));
end
```

We can accelerate the pair-wise test between each pairs of points on the

www.it-ebooks.info

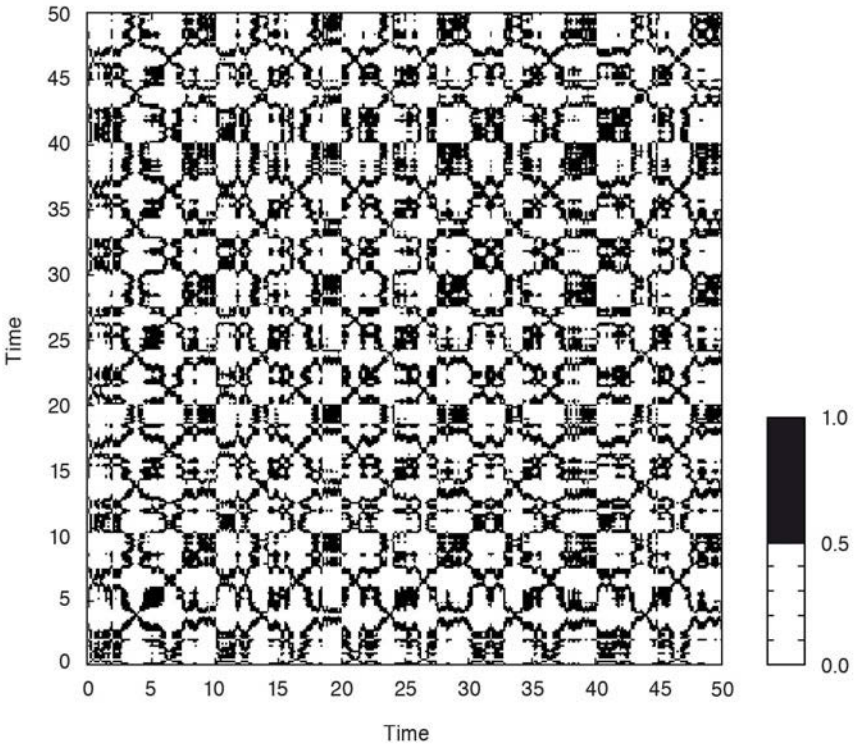


Fig. 5.24 The recurrence plot for the synthetic data derived from the distance matrix, as shown in Fig. 5.23, after applying a threshold of ε=1.

trajectory with a fully vectorized algorithm supported by MATLAB. For this we need to transfer the trajectory vector into two test vectors whose element-wise test will provide the pair-wise test of the trajectory vector:

```
x1 = repmat(xe,N2,1);
x2 = reshape(repmat(xe(:),1,N2)',N2*N2,m);
```

From these vectors we calculate the recurrence plot using the Euclidean norm without any FOR loop (see Section 9.4 for details on Euclidean distances).

```
S = sqrt(sum((x1 - x2).^ 2,2 ));
S = reshape(S,N2,N2);

imagesc(t(1:N2),t(1:N2),S<10)
colormap([1 1 1;0 0 0])
xlabel('Time'), ylabel('Time')
axis xy
```

This recurrence plot reveals many short diagonal lines (Fig. 5.25). These

www.it-ebooks.info

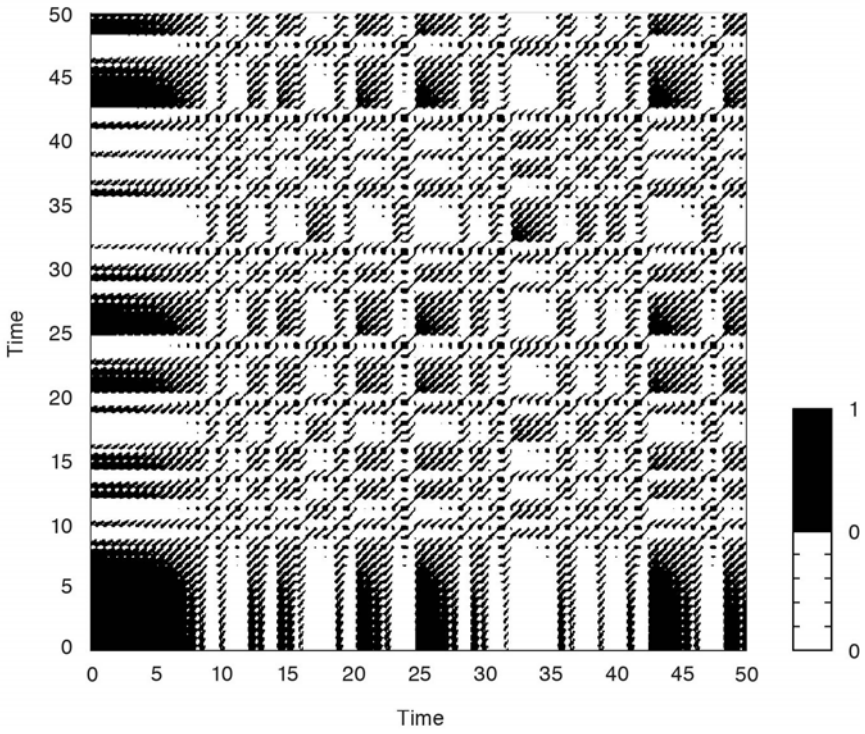


Fig. 5.25 The recurrence plot for the Lorenz system using a threshold of ε=2. The regions with regular diagonal lines reveal unstable periodic orbits, typical of chaotic systems.

Movie 5.4

lines represent periods of time during which the phase space trajectory runs parallel to earlier or later sequences in this trajectory, i.e., periods of times during which the states and dynamics were similar. The distances between these diagonal lines represent the periods of the cycles, which vary and are not constant, in contrast to those for a harmonic oscillation (Fig. 5.22).

Recurrence Quantification

The structure of recurrence plots can also be described by a suite of quantitative measures. Several measures are based on the distribution of the lengths of diagonal or vertical lines, as well as on the local proximity configuration. These measures can be used to trace hidden transitions within a process. As an example we will consider two measures: the recurrence rate and the transitivity coefficient. The recurrence rate is the density of points in the recurrence plot and corresponds to the recurrence probability

www.it-ebooks.info

of the system. The transitivity coefficient has its roots in graph theory and characterizes the regularity or complexity of the system.

We load the synthetic time from the file *series3.txt*, interpolate the data to an annual time axis, and reconstruct its phase space trajectory using an embedding dimension of 5 and a time delay of 3 (Fig. 5.26).

```
clear

series3 = load('series3.txt');

t = 0 : 1 : 996;
series3L = interp1(series3(:,1),series3(:,2),t,'linear');
plot(t,series3L)
xlabel('Time')

N = length(series3L);
tau = 3; m=5;
N2 = N - tau*(m - 1);

xe = zeros(N2,m);
for mi = 1:m
   xe(:,mi) = series3L([1:N2] + tau*(mi-1));
end
```

Using the vectorized approach we calculate the recurrence plot by applying a threshold of 1.2 to the distance matrix (Fig. 5.27).

```
x1 = repmat(xe,N2,1);
x2 = reshape(repmat(xe(:),1,N2)',N2*N2,m);

S = sqrt(sum((x1 - x2).^ 2,2));
S = reshape(S,N2,N2);
```

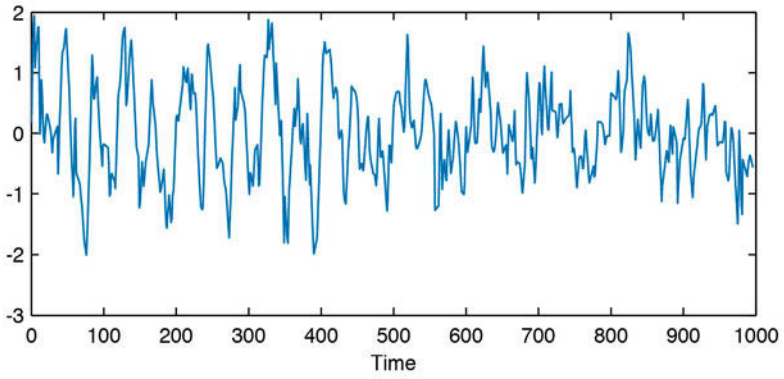


Fig. 5.26 Time series of the synthetic data used in the example of quantitative measures of recurrence plots.

www.it-ebooks.info

```
R = S<1.2;

imagesc(t(1:N2),t(1:N2),R)
colormap([1 1 1;0 0 0])
xlabel('Time'), ylabel('Time')
axis square xy
```

To calculate the recurrence rate we can simply compute the mean of the matrix `R`

```
RR = mean(R(:))
```

which yields

```
RR =
  0.1399
```

The probability that the system returns to a randomly selected previous state is therefore about 14%.

The transitivity coefficient is a graph-theoretical measure of the probability that three connected network nodes (triples) are completely interconnected,

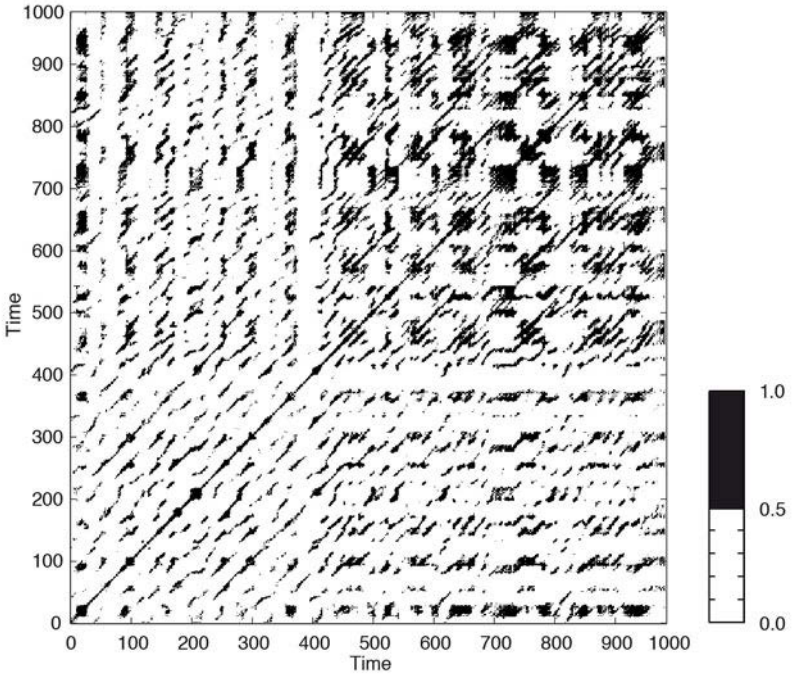


Fig. 5.27 Recurrence plot for the synthetic data in Fig. 5.26, using an embedding of m=5 and τ = 3 and applying a threshold of ε=1.2.

www.it-ebooks.info

i.e., that they form a closed triangle:

$$\tau = \frac{\sum_{i,j,k}^{N} A_{ij} A_{ij} A_{ik}}{\sum_{i,j,k=1}^{N} A_{ij} A_{ik} \left(1 - \delta_{ik}\right)}$$

This measure can be intuitively understood with respect to recurrences in the phase space. We identify a recurrence of states by a network link: close points on the phase space trajectory are connected by a link. Three connected points form a triple, but only if all three points recur closely to each other, thus forming a triangle. Such a triangular configuration will remain along the phase space trajectory if the dynamic is very regular (recurring states remain recurring over a long period of time). However, if the dynamic is chaotic, then parts of the phase space trajectory that were initially close will subsequently diverge and the triangular configuration will break down, although the corresponding triple nodes might remain interconnected for some time. The probability of finding triangles is therefore higher for regular dynamics but lower for chaotic dynamics. This explanation is, of course, rather simplified but a theoretically substantiated explanation can be found in Donner et al. (2011).

In order to calculate the probability that triples also form triangles we need to compute the number of connected triples and the number of triangles, which can be achieved directly from the recurrence plot but excluding the main diagonal.

```
A = R - eye(size(R));
```

The number of triangles and triples is then

```
numTripl = sum(sum(A * A));
numTria = trace(A * A * A);
```

and finally, the transitivity coefficient is the fraction

```
Trans = numTria/numTripl
```

which yields

```
Trans =
   0.5930
```

This number means that the system does not have regular dynamics (which would yield a transitivity coefficient close to one).

Changes in the dynamics, such as transition points and regime changes,

www.it-ebooks.info

are often of interest. Recurrence analysis can be used to detect different types of such transitions. Applying moving windows along the main diagonal of the recurrence plot, we divide it into sub-recurrence plots and calculate the recurrence measures of these sub-plots. In our example we choose a moving window length of 150 and an overlap of 20%

```
w = 150;
```

Movie 5.5

We then calculate the recurrence rate and transitivity coefficient within these moving windows (Fig. 5.28).

```
w = 150;
Trans = zeros(length(R)-w,1);
RR = zeros(length(R)-w,1);
for i = 1:w/5:length(R)-w
   subR = R(i:i+w,i:i+w);
   RR(i) = mean(subR(:));
   subA = A(i:i+w,i:i+w);
   numTripl = sum(sum(subA * subA));
   numClosTria = trace(subA * subA * subA);
   Trans(i) = numClosTria/numTripl;
end

plot(t(round(w/2) + (1:w/5:length(RR))), RR(1:w/5:end),...
  t(round(w/2) + (1:w/5:length(RR))), Trans(1:w/5:end))
xlabel('Time')
legend('recurrence rate','transitivity coeff',4)
```

The results suggest slight changes in the dynamics with respect to recurrence probability (due to the visible amplitude variations in the time series) and regularity. For a reliable interpretation of the variations in the recurrence

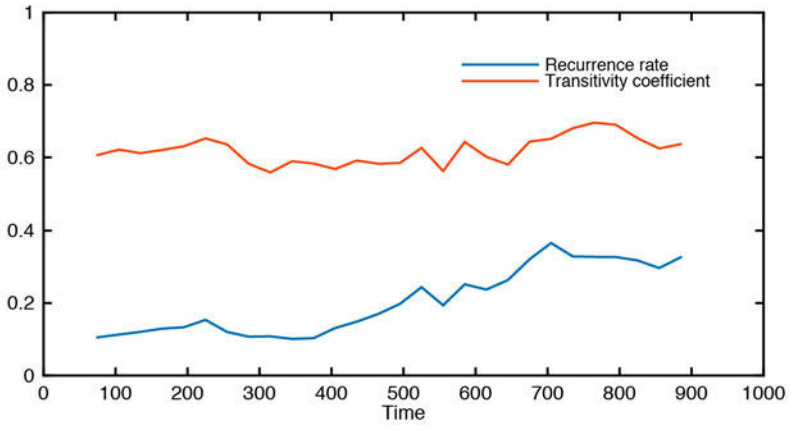


Fig. 5.28 Recurrence rate and transitivity coefficient for the synthetic data in Fig. 5.26, using a moving window of 150 data points and an overlap of 20%.

www.it-ebooks.info

measures, a statistical test should be applied (Marwan 2011). Other more complex measures that quantify other aspects of the dynamics (e.g., predictability, or laminar phases) are included in the Cross Recurrence Plot Toolbox for MATLAB, available from

http://tocsy.pik-potsdam.de/CRPtoolbox/

Bivariate and multivariate extensions of recurrence plots allow nonlinear correlation tests and synchronization analyses to be carried out. A detailed introduction to methods based on recurrence plots can be found on the following web site:

http://www.recurrence-plot.tk

The analysis of recurrence plots has already been applied to many problems in earth sciences. The comparison of the dynamics of modern precipitation data with paleo-rainfall data inferred from annual-layered lake sediments in the northwestern Argentine Andes provides a good example of such analyses (Marwan et al. 2003). In this instance the recurrence plot method was applied to red-color intensity transects across varved lake sediments that were approximately 30 kyrs old (Section 8.7). Comparing the recurrence plots from the sediments with those from modern precipitation data revealed that the reddish layers document the more intense rainy seasons that occurred during La Niña years. The application of linear techniques was, however, not able to link the increased flux of reddish clays with either the El Niño or La Niña phase of the El Niño/Southern Oscillation. Moreover, recurrence plots helped to prove the hypothesis that longer rainy seasons, enhanced precipitation, and the stronger influence of the El Niño/Southern Oscillation caused an increase in the number of landslides 30 kyrs ago (Marwan et al. 2003, Trauth et al. 2003).

Recommended Reading

Ansari AR, Bradley RA (1960) Rank-Sum Tests for Dispersion. Annals of Mathematical Statistics, 31:1174–1189. [Open access]

Blackman, RB, Tukey, JW (1958) The Measurement of Power Spectra. Dover NY

Cooley JW, Tukey JW (1965) An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation 19(90):297–301.

Donner RV, Heitzig J, Donges JF, Zou Y, Marwan N, Kurths J (2011) The Geometry of Chaotic Dynamics – A Complex Network Perspective. European Physical Journal B, 84:653–672

Eckmann JP, Kamphorst SO, Ruelle D (1987) Recurrence Plots of Dynamical Systems. Europhysics Letters 5:973–977

Grenander U (1958) Bandwidth and variance in estimation of the spectrum. Journal of the Royal Statistical Society Series B 20:152–157

www.it-ebooks.info

Holschneider M (1995) Wavelets, an Analysis Tool. Oxford University Press, Oxford
Kantz H, Schreiber T (1997) Nonlinear Time Series Analysis. Cambridge University Press, Cambridge
Lau KM, Weng H (1995) Climate Signal Detection Using Wavelet Transform: How to make a Time Series Sing. Bulletin of the American Meteorological Society 76:2391–2402
Lepage Y (1971) A combination of Wilcoxon's and Ansari-Bradley's statistics. Biometrika 58:213–271
Lomb NR (1972) Least-Squared Frequency Analysis of Unequally Spaced Data. Astro-physics and Space Sciences 39:447–462
Lorenz EN (1963) Deterministic Nonperiodic Flow. Journal of Atmospheric Sciences 20:130–141
Mackenzie D, Daubechies I, Kleppner D, Mallat S, Meyer Y, Ruskai MB, Weiss G (2001) Wavelets: Seeing the Forest and the Trees. Beyond Discovery, National Academy of Sciences, December 2001, available online at http://www.beyonddiscovery.org
Mann, HB, Whitney, DR (1947) On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. Annals of Mathematical Statistics 18:50–60
Marwan N, Thiel M, Nowaczyk NR (2002) Cross Recurrence Plot Based Synchronization of Time Series. Nonlinear Processes in Geophysics 9(3/4):325–331
Marwan N, Trauth MH, Vuille M, Kurths J (2003) Nonlinear Time-Series Analysis on Present-Day and Pleistocene Precipitation Data from the NW Argentine Andes. Climate Dynamics 21:317–332
Marwan N, Romano MC, Thiel M, Kurths J (2007) Recurrence Plots for the Analysis of Complex Systems. Physics Reports, 438: 237–329
Marwan N (2011) How to avoid potential pitfalls in recurrence plot based data analysis. International Journal of Bifurcation and Chaos 21:1003–1017
MathWorks (2014a) Signal Processing Toolbox – User's Guide. The MathWorks, Inc., Natick, MA
MathWorks (2014b) Wavelet Toolbox – User's Guide. The MathWorks, Inc., Natick, MA
Mudelsee M, Stattegger M (1997) Exploring the structure of the mid-Pleistocene revolution with advanced methods of time-series analysis. International Journal of Earth Sciences 86:499–511
Mudelsee M (2000) Ramp function regression: A tool for quantifying climate transitions. Computers and Geosciences 26:293–307
Muller RA, MacDonald GJ (2000) Ice Ages and Astronomical Causes – Data, Spectral Analysis and Mechanisms. Springer Verlag, Berlin Heidelberg New York
Press WH, Teukolsky SA, Vetterling WT (2007) Numerical Recipes: The Art of Scientific Computing – Third Edition. Cambridge University Press, Cambridge
Romano M, Thiel M, Kurths J, von Bloh W (2004) Multivariate Recurrence Plots. Physics Letters A 330(3–4):214–223
Scargle JD (1981) Studies in Astronomical Time Series Analysis. I. Modeling Random Processes in the Time Domain. The Astrophysical Journal Supplement Series 45:1–71
Scargle JD (1982) Studies in Astronomical Time Series Analysis. II. Statistical Aspects of Spectral Analysis of Unevenly Spaced Data. The Astrophysical Journal 263:835–853
Scargle JD (1989) Studies in Astronomical Time Series Analysis. III. Fourier Transforms, Autocorrelation Functions, and Cross-Correlation Functions of Unevenly Spaced Data. The Astrophysical Journal 343:874–887
Schulz M, Stattegger K (1998) SPECTRUM: Spectral Analysis of Unevenly Spaced Paleoclimatic Time Series. Computers & Geosciences 23:929–945
Schuster A (1898) On the investigation of hidden periodicities with application to a supposed

www.it-ebooks.info

26 day period of meteorological phenomena. Terrestrial Magmetism and Atmospheric Electricity 3:13–41

Takens F (1981) Detecting Strange Attractors in Turbulence. Lecture Notes in Mathematics, 898:366–381

Torrence C, Compo GP (1998) A Practical Guide to Wavelet Analysis. Bulletin of the American Meteorological Society 79:61–78

Trulla LL, Giuliani A, Zbilut JP, Webber Jr CL (1996) Recurrence Quantification Analysis of the Logistic Equation with Transients. Physics Letters A 223(4):255–260

Turcotte DL (1992) Fractals and Chaos in Geology and Geophysics. Cambridge University Press, Cambridge

Trauth MH, Bookhagen B, Marwan N, Strecker MR (2003) Multiple Landslide Clusters Record Quaternary Climate Changes in the NW Argentine Andes. Palaeogeography Palaeoclimatology Palaeoecology 194:109–121

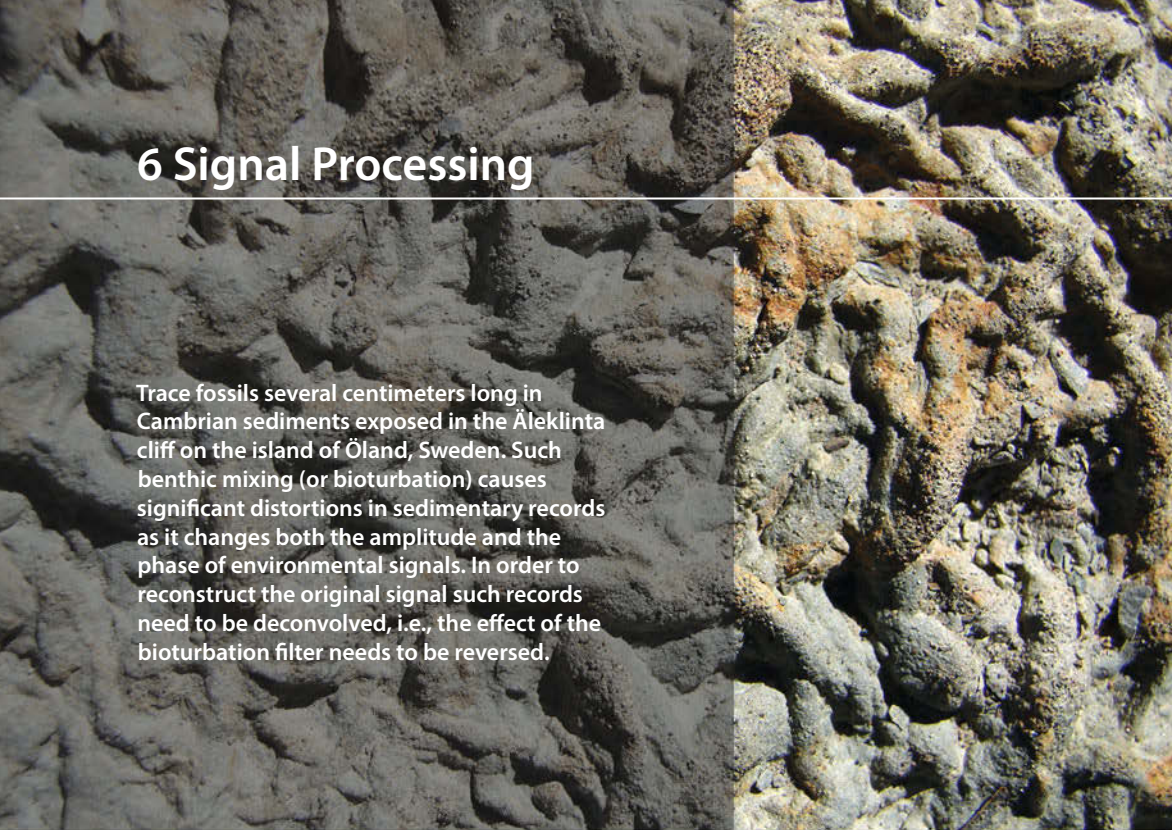
Trauth MH, Larrasoaña JC, Mudelsee M (2009) Trends, rhythms and events in Plio-Pleistocene African climate. Quaternary Science Reviews 28:399–411

Weedon G (2003) Time-Series Analysis and Cyclostratigraphy – Examining Stratigraphic Records of Environmental Change. Cambridge University Press, Cambridge

Welch PD (1967) The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging over Short, Modified Periodograms. IEEE Trans. Audio Electroacoustics AU-15:70–73

www.it-ebooks.info

6 Signal Processing



Trace fossils several centimeters long in Cambrian sediments exposed in the Äleklinta cliff on the island of Öland, Sweden. Such benthic mixing (or bioturbation) causes significant distortions in sedimentary records as it changes both the amplitude and the phase of environmental signals. In order to reconstruct the original signal such records need to be deconvolved, i.e., the effect of the bioturbation filter needs to be reversed.

6.1 Introduction

Signal processing involves techniques for manipulating a signal in order to minimize the effects of noise, to correct all kinds of unwanted distortions, and to separate out various components of interest. Most signal processing algorithms include the design and realization of filters. A *filter* can be described as a system that transforms signals. *System theory* provides the mathematical background for filter design and realization. A filter has an input and an output, with the *output signal* $y(t)$ being modified with respect to the *input signal* $x(t)$ (Fig. 6.1). The *signal transformation* can be carried out through a mathematical process known as *convolution* or, if filters are involved, as *filtering*.

This chapter deals with the design and realization of *digital filters* with the help of a computer. Many natural processes, however, resemble *analog filters* that act over a range of spatial dimensions. A single rainfall event is not recorded in lake sediments because short and low-amplitude events are smeared over a longer time span. Bioturbation also introduces serious distortions, for instance in deep-sea sediment records. In addition to such natural filters, the field collection and sampling of geological data

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_6

www.it-ebooks.info

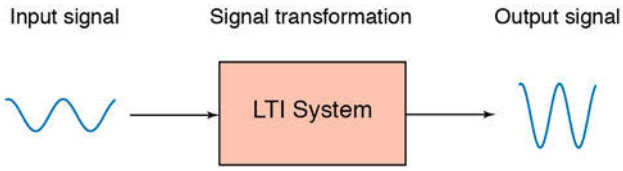


Fig. 6.1 Schematic of a linear time-invariant (LTI) system. The input signal is transformed into an output signal.

alters and smoothes the data with respect to its original form. For example a finite sized sediment sample is integrated over a certain period of time and therefore smoothes the natural signal. Similarly, the measurement of magnetic susceptibility in a sediment core with the help of a loop sensor introduces significant smoothing since the loop response is integrated over a section of the core.

The characteristics of these natural filters are often difficult to determine, whereas numerical filters are designed with well-defined characteristics. In addition, artificial filters are time invariant in most cases, whereas natural filters, such as mixing within the water body of a lake or bioturbation at the water-sediment interface, may vary with time. An easy way to describe or predict the effect of a filter is to explore the filter output from a simple input signal as a sine wave, a square wave, a sawtooth function, a ramp function, or a step function. Although there is an endless variety of such input signals, most systems or filters are described by their impulse response, i.e., the output resulting from the input of a unit impulse.

This chapter starts with a technical section (Section 6.2) on generating periodic signals, trends, and noise, following on from Section 5.2 of the previous chapter. Section 6.3 then considers linear time-invariant systems, providing the mathematical background for filters. The succeeding sections (6.4 to 6.9) deal with the design, the realization, and the application of linear time-invariant filters. Section 6.10 then considers the use of adaptive filters originally developed for use in the telecommunication industry. Adaptive filters automatically extract noise-free signals from duplicate measurements on the same object. Such filters can be used in a large number of applications, for example to remove noise from duplicate paleoceanographic time series, or to improve the signal-to-noise ratio of parallel color-intensity transects across varved lake sediments (see Chapter 5, Fig. 5.1). Adaptive filters are also widely used in geophysics for noise canceling. We use the Signal

www.it-ebooks.info

Processing Toolbox (MathWorks 2014), which contains all the necessary routines for time-series analysis.

6.2 Generating Signals

MATLAB provides numerous tools for generating basic signals that can be used to illustrate the effects of filters. In Chapter 5 we generated a signal by adding together three sine waves with different amplitudes and periods. In the following example the time vector is transposed in order to generate column vectors.

```
clear

t = (1:100)';
x = 2*sin(2*pi*t/50) + sin(2*pi*t/10) + 0.5*sin(2*pi*t/5);

plot(t,x), axis([0 100 -4 4])
```

Frequency-selective filters are very common in earth sciences. They are used to remove specific frequency bands from the data. As an example we can design a filter to suppress that portion of the signal that has a periodicity of τ=10, leaving the other two cycles unaffected. The effects of such filters on simple periodic signals can also be used to predict signal distortions of natural filters.

A *step function* is another basic input signal that can be used to explore filter characteristics. It describes the transition from a value of one towards a value of zero at a specific time. The function `stairs` draws a stairstep graph of the elements of `x`.

```
t = (1:100)';
x = [ones(50,1);zeros(50,1)];

stairs(t,x), axis([0 100 -2 2])
```

This signal can be used to study the effects of a filter on a sudden transition. An abrupt climate change could be regarded as an example. Most natural filters tend to smooth such a transition and smear it over a longer time period.

A *unit impulse* is a third important signal type that we will use in the following examples. This signal equals zero at all times except at a single data point, where it equals one. The function `stem` plots the data sequence `x` as stems from the *t*-axis, using circles for the data values.

```
t = (1:100)';
x = [zeros(49,1);1;zeros(50,1)];

stem(t,x), axis([0 100 -4 4])
```

www.it-ebooks.info

The unit impulse is the most popular synthetic signal used to study the performance of a filter. The output of the filter, i.e., the impulse response, describes the characteristics of a filter very well. Moreover, the output of a linear time-invariant filter can be described by the superposition of impulse responses that have been scaled by multiplying the output of the filter by the amplitude of the input signal.

6.3 Linear Time-Invariant Systems

Filters can be described as systems with an input $x(t)$ and output $y(t)$. We will therefore first describe the characteristics of systems in general before then considering filters. Important characteristics of a system are

- *Continuity* – A system with continuous inputs $x(t)$ and outputs $y(t)$ is a continuous system. Most natural systems are continuous. However, after sampling natural signals we obtain discrete data series and model these natural systems as discrete systems, with discrete inputs and outputs.

- *Linearity* – For linear systems, the output $y(t)$ of the linear combination of several input signals $x_i(t)$, where

  $$x(t) = k_1 x_1(t) + k_2 x_2(t)$$

  is the same as the linear combination of the outputs $y_i(t)$:

  $$y(t) = k_1 y_1(t) + k_2 y_2(t)$$

  Important properties of linearity are scaling and additivity (*superposition*), which allow the input and output to be multiplied by a constant k_i, either before or after transformation. Superposition allows additive components of the input to be extracted and transformed separately. Fortunately, many natural systems follow a linear pattern of behavior. Complex linear signals such as additive harmonic components can be separated out and transformed independently. Milankovitch cycles provide an example of linear superposition in paleoclimate records, although there is an ongoing debate about the validity of this theory. Numerous nonlinear systems also exist in nature, which do not possess the properties of scaling and additivity. An example of a linear system is

```
x = (1:100)';
y = 2*x;
```

www.it-ebooks.info

```
plot(x,y)
```

where `x` is the input signal and `y` is the output signal. An example of a nonlinear system is

```
x = (-100:100)';
y = x.^2;

plot(x,y)
```

- *Time invariance* – The system output $y(t)$ does not change as a result of a delay in the input $x(t+i)$: the system characteristics are constant with time. Unfortunately, natural systems often change their characteristics with time. For instance, benthic mixing or bioturbation depends on various environmental parameters such as nutrient supply, and the system's properties consequently vary significantly with time. In such a case it is difficult to determine the actual input of the system from the output, e.g., to extract the actual climate signal from a bioturbated sedimentary record.

- *Invertibility* – An invertible system is a system in which the original input signal $x(t)$ can be reproduced from the system's output $y(t)$. This is an important property if unwanted signal distortions are to be corrected, in which case the known system is inverted and the output then used to reconstruct the undisturbed input. For example, a core logger measuring magnetic susceptibility with a loop sensor integrates the signal over a specific core interval, with the sensitivity highest at the position of the loop and decreasing down-core and up-core. This system is invertible, i.e., we can compute the input signal $x(t)$ from the output signal $y(t)$ by inverting the system. The inverse of the above linear system is

```
x = (1:100)';
y = 0.5*x;

plot(x,y)
```

where `x` is the input signal and `y` is the output signal. A nonlinear system

```
x = (-100:100)';
y = x.^2;

plot(x,y)
```

is not invertible. Since this system yields equal responses for different inputs, such as `y=4` for inputs `x=-2` and `x=+2`, the input `x` cannot be reconstructed from the output `y`. A similar situation can also occur in linear systems, such as

www.it-ebooks.info

```
x = (1:100)';
y = zeros(size(x));

plot(x,y)
```

The output `y` is zero for all inputs `x`, and the output therefore does not contain any information about the input.

- *Causality* – The system response only depends on present and past inputs $x(0)$, $x(-1)$, …, whereas future inputs $x(+1)$, $x(+2)$, … have no effect on the output $y(0)$. All real-time systems, such as telecommunication systems, must be causal since they cannot have future inputs available to them. All systems and filters in MATLAB are indexed as causal. In earth sciences, however, numerous non-causal filters are used. The filtering of images and signals extracted from sediment cores are examples where the future inputs are available at the time of filtering. Output signals have to be delayed after filtering in order to compensate for the differences between causal and non-causal indexing.

- *Stability* – A system is stable if the output $y(t)$ of a finite input $x(t)$ is also finite. Stability is critical in filter design, where filters often have the disadvantage of provoking divergent outputs. In such cases, the filter design has to be revised and improved.

Linear time-invariant (LTI) systems are very popular as a special type of filter. Such systems have all the advantages that have been described above, as well as being easy to design and use in many applications. The following Sections 6.4 to 6.9 describe the design, realization and application of LTI-type filters to extract specific frequency components from signals. These filters are mainly used to reduce the noise level in signals. Unfortunately, however, many natural systems do not behave as LTI systems in that the signal-to-noise ratio often varies with time. Section 6.10 describes the application of adaptive filters that automatically adjust their characteristics in a time-variable environment.

6.4 Convolution, Deconvolution and Filtering

Convolution is a mathematical description of a system transformation. Filtering is an application of the convolution process. A running mean of length five provides an example of such a simple filter. The output of an arbitrary input signal is

www.it-ebooks.info

$$y(t) = \frac{1}{5}\sum_{k=-2}^{2} x(t-k)$$

The output $y(t)$ is simply the average of the five input values $x(t–2)$, $x(t–1)$, $x(t)$, $x(t+1)$ and $x(t+2)$. In other words, all the five consecutive input values are multiplied by a factor of 1/5 and summed to form $y(t)$. In this example all input values are multiplied by the same factor, i.e., they are equally weighted. The five factors used in the above operation are therefore called filter weights b_k. The filter can be represented by the vector

```
b = [0.2 0.2 0.2 0.2 0.2]
```

consisting of the five identical filter weights. Since this filter is symmetric, it does not shift the signal on the time axis: the only function of this filter is to smooth the signal. Running means of a given length are often used to smooth signals, mainly for cosmetic reasons. Modern spreadsheet software usually contains running means as a function for smoothing data series. The effectiveness of a smoothing filter increases with the filter length.

The weights that a filter of arbitrary length uses can be varied. As an example let us consider an asymmetric filter of five weights.

```
b = [0.05 0.08 0.14 0.26 0.47]
```

The sum of all of the filter weights is one and therefore it does not introduce any additional variance into the signal. However, since it is highly asymmetric, it shifts the signal along the time axis, i.e., it introduces a phase shift.

The general mathematical representation of the filtering process is the *convolution*:

$$y(t) = \sum_{k=-N_1}^{N_2} b_k \cdot x(t-k)$$

where b_k is the vector of *filter weights*, and N_1+N_2 is the *order of the filter*, which is the length of the filter reduced by one. Filters with five weights, as in our example, have an order of four. In contrast to this format, MATLAB uses the engineering standard for indexing filters, i.e., filters are always defined as causal. The convolution used by MATLAB is therefore

$$y(t) = \sum_{k=0}^{N} b_k \cdot x(t-k)$$

www.it-ebooks.info

where N is the order of the filter. A number of the frequency-domain tools provided by MATLAB cannot simply be applied to non-causal filters that have been designed for applications in earth sciences. Hence, it is common to carry out phase corrections in order to simulate non-causality. For example frequency-selective filters, as will be introduced in Section 6.9, can be applied using the function `filtfilt`, which provides zero-phase forward and reverse filtering.

Movies 6.1-6.4

The functions `conv` and `filter` that provide digital filtering in MATLAB are best illustrated in terms of a simple running mean. The n elements of the vector $x(t_1)$, $x(t_2)$, $x(t_3)$, …, $x(t_n)$ are replaced by the arithmetic means of subsets of the input vector. For instance, a running mean over three elements computes the mean of inputs $x(t_{n-1})$, $x(t_n)$, $x(t_{n+1})$ to obtain the output $y(t_n)$. We can illustrate this simply by generating a random signal

```
clear

t = (1:100)';
rng(0)
x1 = randn(100,1);
```

designing a filter that averages three data points of the input signal

```
b1 = [1 1 1]/3;
```

and convolving the input vector with the filter

```
y1 = conv(x1,b1);
```

The elements of `b1` are the weights of the filter. In our example all filter weights are the same and equal to 1/3. Note that the `conv` function yields a vector that has a length of $n+m-1$, where m is the length of the filter.

```
m1 = length(b1);
```

We can explore the contents of our workspace to check the length of the input and output of `conv`. Typing

```
whos
```

yields

```
Name        Size            Bytes  Class     Attributes
b1          1x3                24  double
m1          1x1                 8  double
t         100x1               800  double
x1        100x1               800  double
y1        102x1               816  double
```

www.it-ebooks.info

Here, we see that the actual input series `x1` has a length of 100 data points, whereas the output `y1` has two additional elements. Convolution generally introduces $(m-1)/2$ data points at each end of the data series. To compare input and output signals, we therefore clip the output signal at either end.

```
y1 = y1(2:101,1);
```

A more general way to correct the phase shifts of `conv` is

```
y1 = y1(1+(m1-1)/2:end-(m1-1)/2,1);
```

which of course only works for an odd number of filter weights. An alternative is to use `same` for the `shape` parameter in `conv(a,b,shape)` in order to return the most central 100 data points of the convolution that have the same size as the input signal `x1`.

```
y1 = conv(x1,b1,'same');
```

We can then plot both input and output signals for comparison, using `legend` to display a legend for the plot.

```
plot(t,x1,'b-',t,y1,'r-')
legend('x1(t)','y1(t)')
```

This plot illustrates the effect that the running mean has on the original input series. The output `y1` is significantly smoother than the input signal `x1`. If we increase the length of the filter we obtain an even smoother signal output `y2`.

```
b2 = [1 1 1 1 1]/5;
m2 = length(b2);

y2 = conv(x1,b2,'same');

plot(t,x1,'b-',t,y1,'r-',t,y2,'g-')
legend('x1(t)','y1(t)','y2(t)')
```

The reverse of convolution is deconvolution, which is often used in signal processing to reverse the effects of a filter. We use the first example of a random signal `x1`, design a filter that averages three data points of the input signal, and convolve the input vector with the filter, which yields the output `y1`.

```
clear

t = (1:100)';
rng(0)
x1 = randn(100,1);
```

www.it-ebooks.info

```
b1 = [1 1 1]/3;
y1 = conv(x1,b1);
```

We use `deconv` to reverse the convolution and compare the deconvolution result `x1d` with the original signal `x1`.

```
x1d = deconv(y1,b1);

plot(t,x1,'b:',t,x1d,'r')
```

As we can see, there is no difference. There is a difference, however, if we add noise to the signal and deconvolve the result by typing

```
y1n = y1 + 0.05*randn(size(y1));
x1nd = deconv(y1n,b1);

plot(t,x1,'b:',t,x1nd,'r')
```

The next section provides a broader definition of filters.

6.5 Comparing Functions for Filtering Data Series

The filters described in the previous section were very simple examples of *nonrecursive filters*, in which the filter output $y(t)$ depends only on the filter input $x(t)$ and the filter weights b_k. Prior to introducing a broader description of linear time-invariant filters, we replace the function `conv` by `filter`, which can also be used for *recursive filters*. In this case, the output $y(t_n)$ depends not only on the filter input $x(t)$, but also on previous elements of the output $y(t_{n-1})$, $y(t_{n-2})$, $y(t_{n-3})$ and so on (Section 6.6). We will first use `conv` for nonrecursive filters in order to compare the results of `conv` and `filter`.

```
clear

t = (1:100)';
rng(0)
x3 = randn(100,1);
```

We design a filter that averages five data points of the input signal.

```
b3 = [1 1 1 1 1]/5;
m3 = length(b3);
```

The input signal can be convolved using the function `conv`.

```
y3 = conv(x3,b3,'same');
```

We next follow a similar procedure with the function `filter` and compare the result with that obtained using the function `conv`. In contrast to the function

www.it-ebooks.info

`conv` without using `same`, the function `filter` yields an output vector with the same length as the input vector. Unfortunately, the function `filter` assumes that the filter is causal. The filter weights are indexed n, n–1, n–2 and so on, and therefore no future elements of the input vector, such as $x(n+1)$, x(n+2) etc. are needed to compute the output $y(n)$. This is of great importance in electrical engineering, the classic field of MATLAB application, where filters are often applied in real time. In earth sciences, however, the entire signal is, in most applications, available at the time of processing the data. The data series is filtered by

```
y4 = filter(b3,1,x3);
```

and the phase correction is then carried out using

```
y4 = y4(1+(m3-1)/2:end-(m3-1)/2,1);
y4(end+1:end+m3-1,1) = zeros(m3-1,1);
```

which works only for an odd number of filter weights. This command simply shifts the output by `(m-1)/3` towards the lower end of the t-axis, and then fills the data to the end with zeros. Comparing the ends of both outputs illustrates the effect of this correction, where

```
y3(1:5,1)
y4(1:5,1)
```

yields

```
ans =
    0.0225
    0.1950
    0.2587
   -0.1103
   -0.5638

ans =
    0.0225
    0.1950
    0.2587
   -0.1103
   -0.5638
```

This was the lower end of the output. We can see that both vectors `y3` and `y4` contain the same elements. We now explore the upper end of the data vector, where

```
y3(end-5:end,1)
y4(end-5:end,1)
```

www.it-ebooks.info

yields the output

```
ans =
    0.5491
    0.0181
   -0.5868
   -0.8124
   -0.8498
   -0.8333

ans =
    0.5491
    0.0181
         0
         0
         0
         0
```

The vectors are identical up to element `y(end-m3+1)`, but then the second vector `y4` contains zeros instead of true data values. Plotting the results with

```
subplot(2,1,1), plot(t,x3,'b-',t,y3,'r-')
subplot(2,1,2), plot(t,x3,'b-',t,y4,'r-')
```

or in one single plot,

```
plot(t,x3,'b-',t,y3,'g-',t,y4,'r-')
```

shows that the results from using `conv` are identical with those from using `filter`, except at the upper end of the data vector. These observations are important for our next steps in signal processing, particularly if we are interested in leads and lags between various components of signals.

6.6 Recursive and Nonrecursive Filters

We now expand the nonrecursive filters by a recursive component, such that the output $y(t_n)$ depends not only on the filter input $x(t)$, but also on previous output values $y(t_{n-1})$, $y(t_{n-2})$, $y(t_{n-3})$, and so on. This filter requires not only the nonrecursive filter weights b_i, but also the recursive filters weights a_i (Fig. 6.2), and can be described by the *difference equation*:

$$y(t) = \sum_{k=-N_1}^{N_2} b_k \cdot x(t-k) - \sum_{k=1}^{M} a_k \cdot y(t-k)$$

Although this is a non-causal version of the difference equation, MATLAB again uses the causal indexing,

www.it-ebooks.info

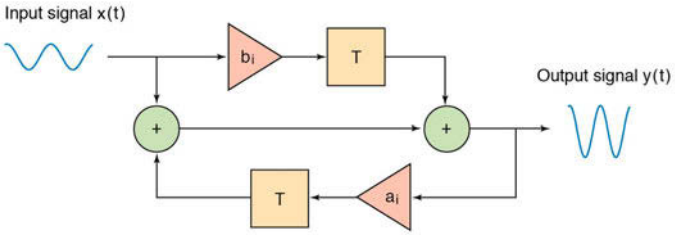


Fig. 6.2 Schematic of a linear time-invariant filter with an input $x(t)$ and an output $y(t)$. The filter is characterized by its weights a_i and b_i. and the delay elements T. Nonrecursive filters require only nonrecursive weights b_i whereas recursive filters also require the recursive filter weights a_i.

$$y(t) = \sum_{k=0}^{N} b_k \cdot x(t-k) - \sum_{k=1}^{M} a_k \cdot y(t-k)$$

with the known problems in the design of zero-phase filters. The larger of the two quantities M, and N_1+N_2 or N, is the order of the filter.

We use the same synthetic input signal `x5` as in the previous example to illustrate the performance of a recursive filter.

```
clear

t = (1:100)';
rng(0)
x5 = randn(100,1);
```

This input is then filtered using a recursive filter with a set of weights `a5` and `b5`,

```
b5 = [0.0048    0.0193    0.0289    0.0193    0.0048];
a5 = [1.0000   -2.3695    2.3140   -1.0547    0.1874];

m5 = length(b5);

y5 = filter(b5,a5,x5);
```

and the output `y5` corrected for the phase

```
y5 = y5(1+(m5-1)/2:end-(m5-1)/2,1);
y5(end+1:end+m5-1,1) = zeros(m5-1,1);
```

We can now plot the results.

```
plot(t,x5,'b-',t,y5,'r-')
```

www.it-ebooks.info

This filter clearly changes the signal dramatically. The output contains only low-frequency components and all higher frequencies have been eliminated. A comparison of the periodograms for the input and the output reveals that all frequencies above f=0.1 (corresponding to a period of τ=10) have been suppressed.

```
[Pxx,f] = periodogram(x5,[],128,1);
[Pyy,f] = periodogram(y5,[],128,1);

plot(f,Pxx,f,Pyy)
```

We have now designed a frequency-selective filter, i.e., a filter that eliminates certain frequencies while leaving other frequencies relatively unaffected. The next section introduces tools that are used to characterize a filter in the time and frequency domains and to predict the effect of a frequency-selective filter on arbitrary signals.

6.7 Impulse Response

The impulse response is a very convenient way of describing the characteristics of a filter (Fig. 6.3). The impulse response h is useful in LTI systems where the convolution of the input signal $x(t)$ with h is used to obtain the output signal $y(t)$.

$$y(t) = \sum_{k=-N_1}^{N_2} h_k \cdot x(t-k)$$

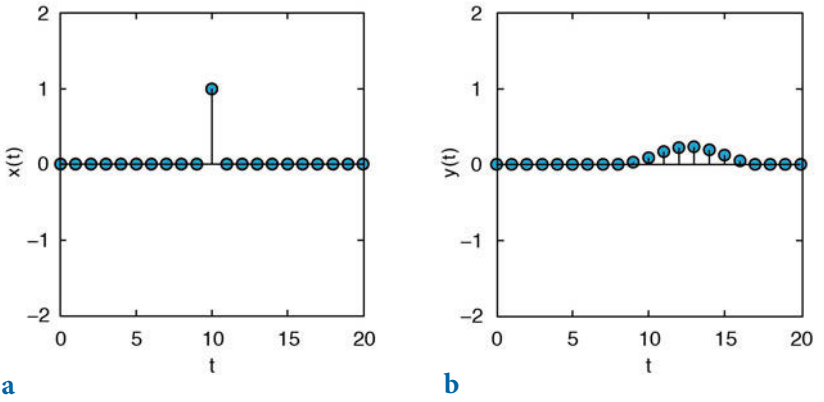


Fig. 6.3 Transformation of **a** a unit impulse to compute **b** the impulse response of a system. The impulse response is often used to describe and predict the performance of a filter.

www.it-ebooks.info

It can be shown that the values of the impulse response h are identical to the filter weights in nonrecursive filters, but not in recursive filters. The above convolution equation is often written in a short form:

$$y(t) = h(t) * x(t)$$

In many examples convolution in the time domain is replaced by a simple multiplication of the Fourier transforms $H(f)$ and $X(f)$ in the frequency domain.

$$Y(f) = H(f) \cdot X(f)$$

The output signal $y(t)$ in the time domain is then obtained by a reverse Fourier transform of $Y(f)$. Signals are often convolved in the frequency domain rather than the time domain because of the relative simplicity of the multiplication. However, the Fourier transformation itself introduces a number of artifacts and distortions, and convolution in the frequency domain is therefore not without problems. In the following examples we apply the convolution only in the time domain.

First, we generate a unit impulse:

```
clear

t = (0:20)';
x6 = [zeros(10,1);1;zeros(10,1)];

stem(t,x6), axis([0 20 -4 4])
```

The function `stem` plots the data sequence `x6` as stems from the x-axis, terminated with circles for the data value. This can be a better way to plot digital data than using the continuous lines generated by `plot`. We now feed this into the filter and explore the output `y6`. The impulse response is identical to the weights of nonrecursive filters.

```
b6 = [1 1 1 1 1]/5;
m6 = length(b6);

y6 = filter(b6,1,x6);
```

We again correct this for the phase shift of the function `filter`, although this might not be important in this example.

```
y6 = y6(1+(m6-1)/2:end-(m6-1)/2,1);
y6(end+1:end+m6-1,1) = zeros(m6-1,1);
```

www.it-ebooks.info

We obtain an output vector `y6` of the same length and phase as the input vector `x6`. We now plot the results for comparison.

```
stem(t,x6)
hold on
stem(t,y6,'filled','r')
axis([0 20 -2 2])
hold off
```

In contrast to `plot`, the function `stem` accepts only one data series and the second series `y6` is therefore overlaid on the same plot using the function `hold`. The effect of the filter is clearly seen on the plot: it averages the unit impulse over a length of five elements. Furthermore, the values of the output `y6` equal the filter weights of `a6`; in our example these values are 0.2 for all elements of `a6` and `y6`.

For a recursive filter, however, the output `y6` does not match the filter weights. Once again, we first generate an impulse:

```
clear

t = (0:20)';
x7 = [zeros(10,1);1;zeros(10,1)];
```

An arbitrary recursive filter with weights of `a7` and `b7` is then designed.

```
b7 = [0.0048    0.0193    0.0289    0.0193    0.0048];
a7 = [1.0000   -2.3695    2.3140   -1.0547    0.1874];

m7 = length(b7);

y7 = filter(b7,a7,x7);

y7 = y7(1+(m7-1)/2:end-(m7-1)/2,1);
y7(end+1:end+m7-1,1) = zeros(m7-1,1);
```

The stem plot of the input `x2` and the output `y2` shows an interesting impulse response:

```
stem(t,x7)
hold on
stem(t,y7,'filled','r')
axis([0 20 -2 2])
hold off
```

The signal is smeared over a broader area, and is also shifted towards the right. This filter therefore not only affects the amplitude of the signal, but also shifts the signal towards lower or higher values. Such phase shifts are usually unwanted characteristics of filters, although in some applications shifts along the time axis might be of particular interest.

www.it-ebooks.info

6.8 Frequency Response

We next investigate the frequency response of a filter, i.e., the effect of a filter on the amplitude and phase of a signal (Fig. 6.4). The frequency response $H(f)$ of a filter is the Fourier transform of the impulse response $h(t)$. The absolute value of the complex frequency response $H(f)$ is the magnitude response of the filter $A(f)$.

$$A(f) = |H(f)|$$

The argument of the complex frequency response $H(f)$ is the phase response of the filter.

$$\Phi(f) = \arg\left(H(f)\right)$$

Since MATLAB filters are all causal it is difficult to explore the phase of signals using the corresponding functions included in the Signal Processing Toolbox. The user's guide for this toolbox simply recommends that the filter output be delayed in the time domain by a fixed number of samples, as we have done in the previous examples. As another example a sine wave with a period of 20 and an amplitude of 2 is used as an input signal.

```
clear

t = (1:100)';
x8 = 2*sin(2*pi*t/20);
```

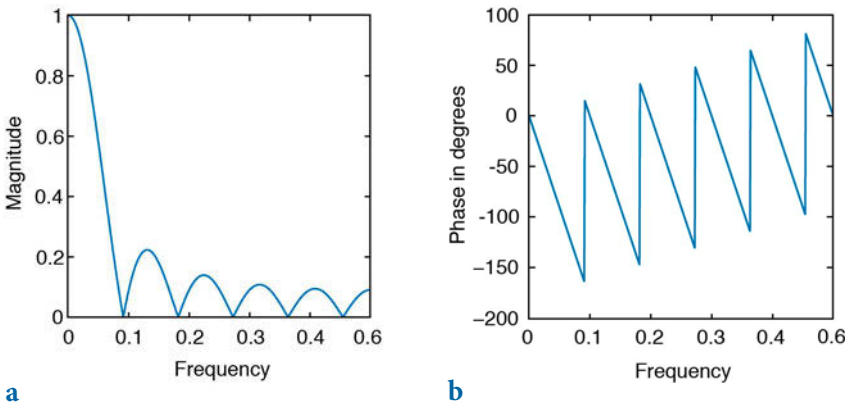


Fig. 6.4 a Magnitude and **b** phase response of a running mean over eleven elements.

www.it-ebooks.info

A running mean over eleven elements is designed and this filter applied to the input signal `x8`.

```
b8 = ones(1,11)/11;
m8 = length(b8);

y8 = filter(b8,1,x8);
```

The phase of the output `y8` is corrected for causal indexing.

```
y8 = y8(1+(m8-1)/2:end-(m8-1)/2,1);
y8(end+1:end+m8-1,1) = zeros(m8-1,1);
```

Both input and output of the filter are plotted.

```
plot(t,x8,t,y8)
```

The filter clearly reduces the amplitude of the sine wave. Whereas the input signal `x8` has an amplitude of 2, the output `y8` has an amplitude of

```
max(y8)

ans =
    1.1480
```

The filter reduces the amplitude of a sine with a period of 20 by

```
1-max(y8(40:60))/2

ans =
    0.4260
```

i.e., by approximately 43%. The elements 40 to 60 are used for computing the maximum value of `y8`, in order to avoid edge effects. Nevertheless, the filter does not affect the phase of the sine wave, i.e., both input and output are in phase.

The same filter, however, has a different impact on a different input signal. Let us design another sine wave `x9`, with a similar amplitude but with a different period of 15.

```
clear

t = (1:100)';
x9 = 2*sin(2*pi*t/15);
```

Applying a similar filter and correcting the output `y9` for the phase shift of the function `filter` yields

```
b9 = ones(1,11)/11;
m9 = length(b9);
```

www.it-ebooks.info

```
y9 = filter(b9,1,x9);

y9 = y9(1+(m9-1)/2:end-(m9-1)/2,1);
y9(end+1:end+m9-1,1) = zeros(m9-1,1);
```

The output `y9` is again in phase with the input `x9`, but the amplitude is dramatically reduced compared to that of the input.

```
plot(t,x9,t,y9)

1-max(y9(40:60))/2

ans =
    0.6768
```

Movie 6.5

The running mean over eleven elements reduces the amplitude of this signal by ~68%. More generally, the filter response clearly depends on the frequency of the input. The frequency components of a more complex signal containing multiple periodicities are affected in a different way. The frequency response of a filter

```
clear

b10 = ones(1,11)/11;
```

can be computed using the function `freqz`.

```
[h,w] = freqz(b10,1,512);
```

The function `freqz` returns the complex frequency response `h` of the digital filter `b10`. The frequency axis is normalized to π. We transform the frequency axis `w` to the true frequency values `f`. The true frequency values `f` are `w` times the sampling frequency (which is one in our example) divided by `2*pi`.

```
f = 1*w/(2*pi);
```

Next, we calculate and display the magnitude of the frequency response.

```
magnitude = abs(h);

plot(f,magnitude)
xlabel('Frequency'), ylabel('Magnitude')
title('Magnitude')
```

This plot can be used to predict the effect of the filter for any frequency of an input signal. We can interpolate the magnitude of the frequency response to calculate the increase or reduction in a signal's amplitude for a specific frequency. As an example the interpolation of `magnitude` for a frequency of 1/20

www.it-ebooks.info

```
1-interp1(f,magnitude,1/20)

ans =
    0.4260
```

results in the expected ~43% reduction in the amplitude of a sine wave with a period of 20. The sine wave with a period of 15 experiences an amplitude reduction of

```
1-interp1(f,magnitude,1/15)

ans =
    0.6751
```

i.e., approximately 68%, which is similar to the value observed previously. It is very important that such a running mean wipes out certain frequencies, e.g., those for which `magnitude=0`. As an example, applying the filter to a signal with a period of approximately `1/0.09082` completely eliminates that signal. Furthermore, since the magnitude of the frequency response is the absolute value of the complex frequency response `h`, the magnitude response is actually negative between `~0.09082` and `~0.1816`, between `~0.2725` and `~0.3633`, and between `~0.4546` and the Nyquist frequency. All signal components having frequencies within these intervals are mirrored on the *t*-axis. As an example we try a sine wave with a period of 7 (e.g., a frequency of approximately `0.1429`), which is within the first interval with a negative magnitude response:

```
t = (1:100)';
x10 = 2*sin(2*pi*t/7);

b10 = ones(1,11)/11;
m10 = length(b10);

y10 = filter(b10,1,x10);

y10 = y10(1+(m10-1)/2:end-(m10-1)/2,1);
y10(end+1:end+m10-1,1) = zeros(m10-1,1);

plot(t,x10,t,y10)
```

The sine wave with a period of 7 experiences an amplitude reduction of

```
1-interp1(f,magnitude,1/7)

ans =
    0.7957
```

i.e., approximately 80%, but also changes the sign as we can see from the plot. The elimination of certain frequencies and flipping of the signal have

www.it-ebooks.info

important consequences when interpreting causality in earth sciences. These filters should therefore be avoided completely, even though they are offered as standards in spreadsheet programs. As an alternative, filters with a specific frequency response should be used, such as a Butterworth lowpass filter (Section 6.9).

The frequency response can be calculated for all kinds of filters. It is a valuable tool for predicting the effects of a filter on signals in general. The phase response can also be calculated from the complex frequency response `h` of the filter (Fig. 6.4):

```
phase = 180*angle(h)/pi;

plot(f,phase)
xlabel('Frequency'), ylabel('Phase in degrees')
title('Phase')
```

The phase angle `phase` is plotted in degrees. We observe frequent jumps in this plot that are an artifact of the function `arctangent` within the function `angle`. We can unwrap the phase response to eliminate those jumps that are equal to or larger than 180°, with the help of the function `unwrap`.

```
plot(f,180*unwrap(angle(h))/pi)
xlabel('Frequency'), ylabel('Phase in degrees')
title('Phase')
```

In our example this has no effect since no jumps occur that are equal to or larger than 180°. Since the filter has a linear phase response `phase`, no shifts occur in the frequency components of the signals relative to each other. We would therefore not expect any distortions of the signal in the frequency domain. The phase shift of the filter on a specific period can be computed using

```
interp1(f,180*unwrap(angle(h))/pi,1/20) * 20/360

ans =
   -5.0000
```

and

```
interp1(f,180*unwrap(angle(h))/pi,1/15) * 15/360

ans =
   -5.0000
```

for the sine waves with periods of 20 and 15, respectively. Since MATLAB uses causal indexing for filters, the phase needs to be corrected in a similar way to the delayed output of the filter. In our example we used a filter with

www.it-ebooks.info

a length of eleven. We therefore have to correct the phase by (11–1)/2=5, which suggests a zero phase shift for the filter for both frequencies.

This also works for recursive filters. Consider a simple sine wave with a period of 8 and the previously employed recursive filter.

```
clear

t = (1:100)';
x11 = 2*sin(2*pi*t/8);

b11 = [0.0048    0.0193    0.0289    0.0193    0.0048];
a11 = [1.0000   -2.3695    2.3140   -1.0547    0.1874];

m11 = length(b11);

y11 = filter(b11,a11,x11);
```

We correct the output for the phase shift introduced by causal indexing and plot both input and output signals.

```
y11= y11(1+(m11-1)/2:end-(m11-1)/2,1);
y11(end+1:end+m11-1,1) = zeros(m11-1,1);

plot(t,x11,t,y11)
```

The magnitude is reduced by

```
1-max(y11(40:60))/2

ans =
    0.6465
```

which is also supported by the magnitude response

```
[h,w] = freqz(b11,a11,512);

f = 1*w/(2*pi);

magnitude = abs(h);

plot(f,magnitude)
xlabel('Frequency'), ylabel('Magnitude')
title('Magnitude Response')

1-interp1(f,magnitude,1/8)

ans =
    0.6462
```

The phase response

```
phase = 180*angle(h)/pi;
```

www.it-ebooks.info

```
f = 1*w/(2*pi);

plot(f,180*unwrap(angle(h))/pi)
xlabel('Frequency'), ylabel('Phase in degrees')
title('Magnitude Response')

interp1(f,180*unwrap(angle(h))/pi,1/8) * 8/360

ans =
    -5.0557
```

must again be corrected for causal indexing. Since the sampling interval was one and the filter length is five, we have to add (5–1)/2=2 to the phase shift of –5.0557. This suggests a corrected phase shift of –3.0557, which is exactly the delay seen on the plot.

```
plot(t,x11,t,y11), axis([30 40 -2 2])
```

The next section gives an introduction to the design of filters with a desired frequency response. These filters can be used to amplify or suppress different components of arbitrary signals.

6.9 Filter Design

We now aim to design filters with a specific frequency response. We first generate a synthetic signal `x12` with two periods, 50 and 5. The power spectrum of the signal shows the expected peaks at frequencies of 0.02 and 0.20.

```
clear

t = 0 : 1000;
x12 = 2*sin(2*pi*t/50) + sin(2*pi*t/5);

plot(t,x12), axis([0 200 -4 4])

[Pxx,f] = periodogram(x12,[],1024,1);

plot(f,Pxx)
xlabel('Frequency')
ylabel('Power')
```

The Butterworth filter design technique is widely used in order to create filters of any order with a lowpass, highpass, bandpass and bandstop configuration (Fig. 6.5). In our example we would like to design a five-order lowpass filter with a cutoff frequency of 0.10. The inputs of the function `butter` are the order of the filter and the cutoff frequency normalized to the Nyquist

www.it-ebooks.info

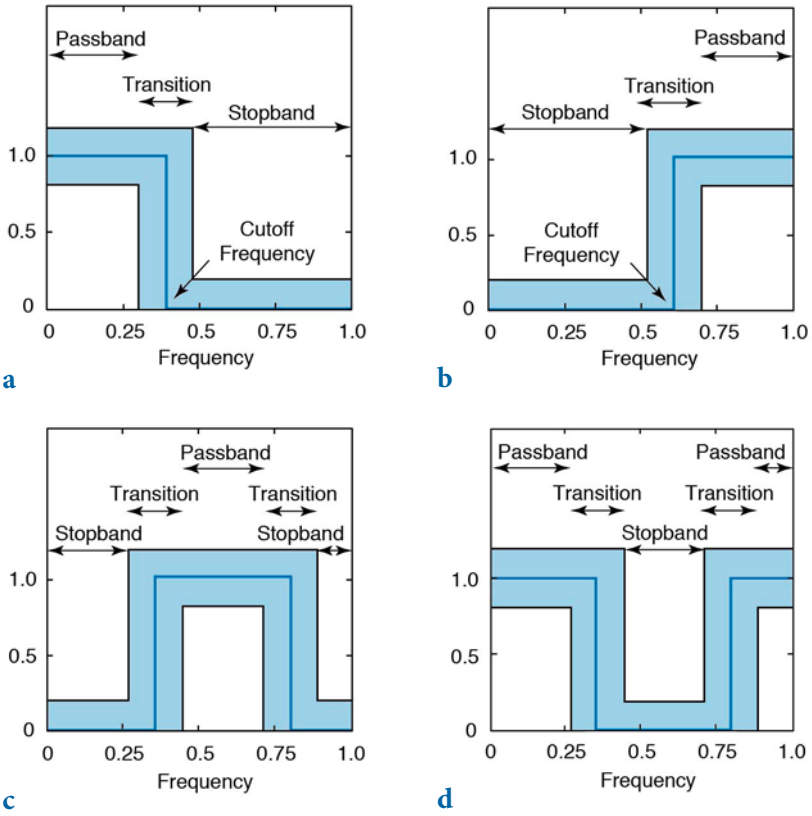


Fig. 6.5 Frequency responses for the fundamental types of frequency-selective filters. **a** Lowpass filter to suppress the high-frequency component of a signal. In earth sciences, such filters are often used to suppress high-frequency noise in a low-frequency signal. **b** Highpass filters to remove all low frequencies and trends in natural data. **c-d** Bandpass and bandstop filters to extract or suppress a certain frequency band. The solid line in all graphics depicts the ideal frequency response of a frequency-selective filter, while the gray band shows the tolerance for a low-order design of such a filter. In practice, the frequency response lies within the gray band.

frequency, which in our example is 0.5 (i.e., half of the sampling frequency). The outputs from `butter` are the filter weights `a12` and `b12`.

```
[b12,a12] = butter(5,0.1/0.5);
```

The frequency characteristics of the filter show a relatively smooth transition from the passband to the stopband, but the advantage of the filter is its low order.

```
[h,w] = freqz(b12,a12,1024);
```

www.it-ebooks.info

```
f = 1*w/(2*pi);

plot(f,abs(h)), grid
xlabel('Frequency')
ylabel('Magnitude')
```

We can again apply the filter to the signal by using the function `filter`. Frequency-selective filters such as lowpass, highpass, bandpass and bandstop filters are designed to suppress certain frequency bands, but phase shifts should be avoided. The function `filtfilt` provides zero phase-shift forward and reverse digital filtering. After filtering in the forward direction, the filtered sequence is reversed and runs back through the filter. The magnitude of the signal is not affected by this operation, since it is either 0 or 100% of the initial amplitude, depending on the frequency. Any phase shifts introduced by the filter are canceled out by the forward and reverse application of the same filter. This function also helps to overcome the problems with causal indexing of filters in MATLAB by eliminating the phase differences between the causal and non-causal versions of the same filter. Filtering, and then plotting the results clearly illustrates the effects of the filter.

```
xf12 = filtfilt(b12,a12,x12);

plot(t,x12,'b-',t,xf12,'r-')
axis([0 200 -4 4])
```

Movie 6.6

We might now wish to design a new filter with a more rapid transition from passband to stopband. Such a filter requires a higher order, i.e., it needs a larger number of filter weights `a12` and `b12`. We now create a 15-order Butterworth filter as an alternative to the above filter.

```
[b13,a13] = butter(15,0.1/0.5);

[h,w] = freqz(b13,a13,1024);

f = 1*w/(2*pi);

plot(f,abs(h)), grid
xlabel('Frequency')
ylabel('Magnitude')
```

The frequency response is clearly improved. The entire passband is relatively flat at a value of 1.0, whereas the stopband is approximately zero everywhere. We next modify our input signal by introducing a third period of 5. This signal is then used to illustrate the operation of a Butterworth bandstop filter.

```
clear

t = 0 : 1000;
```

www.it-ebooks.info

```
x14 = 2*sin(2*pi*t/50) + sin(2*pi*t/10) + 0.5*sin(2*pi*t/5);
plot(t,x14), axis([0 200 -4 4])

[Pxx,f] = periodogram(x14,[],1024,1);

plot(f,Pxx)
```

The new Butterworth filter is a bandstop filter. The stopband of the filter is between the frequencies 0.05 and 0.15. It can therefore be used to suppress the period of 10, corresponding to a frequency of 0.1.

```
rng(0)
xn14 = x14 + randn(1,length(t));

[b14,a14] = butter(5,[0.05 0.15]/0.5,'stop');
xf14 = filtfilt(b14,a14,x14);

[Pxx,f] = periodogram(xf14,[],1024,1);

plot(f,Pxx)

figure
plot(t,xn14,'b-',t,xf14,'r-'), axis([0 200 -4 4])
```

Audio 6.1

The plots show the effect of this filter. The frequency band between 0.05 and 0.15, and therefore also the frequency of 0.1, have been successfully removed from the signal.

6.10 Adaptive Filtering

The fixed filters used in the previous sections make the basic assumption that the signal degradation is known and does not change with time. However, *a priori* knowledge of the signal and noise statistical characteristics is not usually available in most applications. In addition, both the noise level and the variance of the genuine signal can be highly nonstationary with respect to time, e.g., stable isotope records during a glacial-interglacial transition. Fixed filters cannot thus be used in a nonstationary environment without any knowledge of the signal-to-noise ratio.

Adaptive filters, widely used in the telecommunication industry, could help to overcome these problems. An adaptive filter is an inverse modeling process that iteratively adjusts its own coefficients automatically without requiring any *a priori* knowledge of the signal and the noise. The operation of an adaptive filter includes (1) a filtering process, the purpose of which is to produce an output in response to a sequence of data, and (2) an adaptive process, providing a mechanism for the adaptive control of the filter weights (Haykin 1991).

www.it-ebooks.info

In most practical applications the adaptive process is oriented towards minimizing an estimation error e. The estimation error e at an instant i is defined by the difference between the desired response d_i and the actual filter output y_i, which is the filtered version of a signal x_i, as shown by

$$e_i = d_i - y_i$$

where i=1, 2, …, N and N is the length of the input data vector. In the case of a nonrecursive filter characterized by a vector of filter weights W with f elements, the filter output y_i is given by the inner product of the transposed vector W and the input vector X_i.

$$y_i = W^T \cdot X_i$$

The choice of desired response d that is used in the adaptive process depends on the application. Traditionally, d is a combination signal that is comprised of a signal s and random noise n_0. The signal x contains noise n_1 that is uncorrelated with the signal s but correlated in some unknown way with the noise n_0. In noise canceling systems the practical objective is to produce a system output y that is a best fit in the least-squares sense to the desired response d.

Different approaches have been developed to solve this multivariate minimum error optimization problem (e.g., Widrow and Hoff 1960, Widrow et al. 1975, Haykin 1991). The selection of one algorithm over another is influenced by various factors including the rate of convergence (the number of adaptive steps required for the algorithm to converge closely enough to an optimum solution), the misadjustment (the measure of the amount by which the final value of the mean-squared error deviates from the minimum squared error of an optimal filter, e.g., Wiener 1945, Kalman and Bucy 1961), and the tracking (the capability of the filter to work in a nonstationary environment, i.e., to track changing statistical characteristics of the input signal) (Haykin 1991).

The simplicity of the least-mean-squares (LMS) algorithm, originally developed by Widrow and Hoff (1960), has made it the benchmark against which other adaptive filtering algorithms are tested. For applications in earth sciences we use this filter to extract the noise from two signals S and X, both containing the same signal s, but with uncorrelated noise n_1 and n_2 (Hattingh 1988). As an example, consider a simple duplicate set of measurements on the same material, e.g., two parallel stable isotope records from the same foraminifera species. You would expect two time-series, each

www.it-ebooks.info

with N elements, containing the same desired signal overlain by different, uncorrelated noise. The first record is used as the primary input S

$$S = (s_1, s_2, \ldots, s_N)$$

and the second record as the reference input X.

$$X = (x_1, x_2, \ldots, x_N)$$

As demonstrated by Hattingh (1988), the desired noise-free signal can be extracted by filtering the reference input X using the primary input S as the desired response d. The minimum error optimization problem is solved by the least-mean-square norm. The mean-squared error e_i^2 is a second-order function of the weights in the nonrecursive filter. The dependence of e_i^2 on the unknown weights W may be seen as a multidimensional paraboloid with a uniquely defined minimum point. The weights corresponding to the minimum point on this error performance surface define the optimal Wiener solution (Wiener 1945). The value computed for the weight vector W using the LMS algorithm represents an estimator whose expected value approaches the Wiener solution as the number of iterations approaches infinity (Haykin 1991). Gradient methods are used to reach the minimum point on the error performance surface. To simplify the optimization problem, Widrow and Hoff (1960) developed an approximation for the required gradient function that can be computed directly from the data. This leads to a simple relationship for updating the filter-weight vector W.

$$W_{i+1} = W_i + 2 \cdot u \cdot e_i \cdot X_i$$

The new parameter estimate W_{i+1} is based on the previous set of filter weights W_i plus a term that is the product of a bounded step size u, a function of the input state X_i and a function of the error e_i. In other words, error e_i calculated from the previous step is fed back into the system to update filter coefficients for the next step (Fig. 6.6). The fixed convergence factor u regulates the speed and stability of adaption. A low value of u ensures a higher level of accuracy, but more data are needed to enable the filter to reach the optimum solution. In the modified version of the LMS algorithm by Hattingh (1988), this problem is overcome by feeding the data back so that the filter can have another chance to improve its own coefficients and adapt to the changes in the data.

www.it-ebooks.info

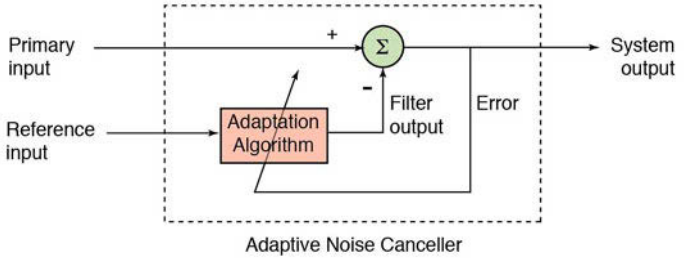


Fig. 6.6 Schematic of an adaptive filter. Each iteration involves a new estimate of the filter weights W_{i+1} based on the previous set of filter weights W_i plus a term that is the product of a bounded step size u, a function of the filter input X_i, and a function of the error e_i. In other words, error e_i calculated from the previous step is fed back into the system to update filter coefficients for the next step (modified from Trauth 1998).

In the following example introducing the use of the function `canc`, each of these loops is called an iteration, and many of these loops are required if optimal results are to be achieved. This algorithm extracts the noise-free signal from two vectors, `x` and `s`, containing the correlated signals and uncorrelated noise. As an example we generate two signals `yn1` and `yn2` containing the same sine wave but different Gaussian noise.

```
clear

x = 0 : 0.1 : 100; x = x';
y = sin(x);
rng(0)
yn1 = y + 0.5*randn(size(y));
yn2 = y + 0.5*randn(size(y));

plot(x,yn1,x,yn2)
```

The algorithm `canc` formats both signals, feeds them into the filter loop, corrects the signals for phase shifts, and formats the signals for the output. The required inputs are the signals `x` and `s`, the step size `u`, the filter length `l` and the number of iterations `iter`. In our example the two noisy signals are `yn1` and `yn2`. We make an arbitrary choice of a filter with `l=5` filter weights. A value of `u` in the range of $0 < u < l/\lambda_{max}$, where λ_{max} is the largest eigenvalue of the autocorrelation matrix for the reference input, leads to reasonable results (Haykin 1991) (Fig. 6.7). The value of `u` is computed using

```
k = kron(yn1,yn1');
u = 1/max(eig(k))
```

where `kron` returns the Kronecker tensor product of `yn1` and `yn1'` (which is a matrix formed by taking all possible products between the elements of `yn1`

www.it-ebooks.info

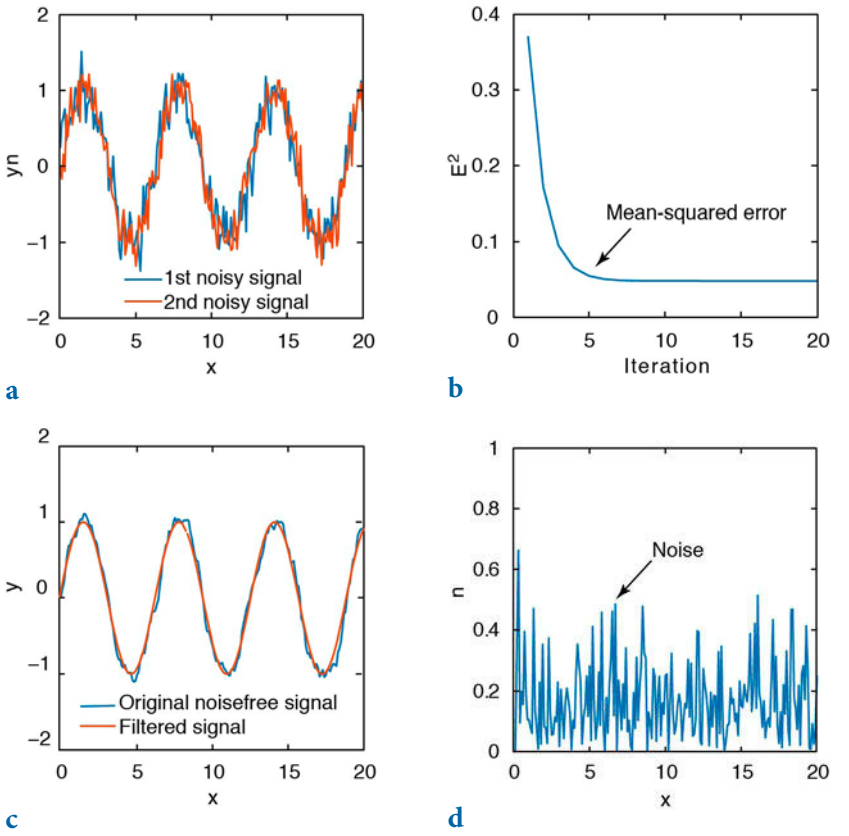


Fig. 6.7 Output of the adaptive filter. **a** The duplicate records corrupted by uncorrelated noise are fed into the adaptive filter with 5 weights with a convergence factor of 0.0019. After 20 iterations, the filter yields **b** the learning curve, **c** the noise-free record, and **d** the noise extracted from the duplicate records.

and `yn1'`), and `eig` returns the eigenvector of `k`. This yields

```
u =
    0.0019
```

We now run the adaptive filter `canc` for 20 iterations and use the above value of `u`.

```
[z,e,mer,w] = canc(yn1,yn2,0.0019,5,20);
```

The output variables from `canc` are the filtered primary signal `z`, the extracted noise `e`, the mean-squared error `mer` for the number of iterations `it` performed with stepsize `u`, and the filter weights `w` for each data point in `yn1` and `yn2`. The

www.it-ebooks.info

plot of the mean-squared error `mer`

```
plot(mer)
```

illustrates the performance of the adaptive filter, although the chosen step size `u=0.0019` clearly results in a relatively rapid convergence. In most examples a smaller step size decreases the rate of convergence but improves the quality of the final result. We therefore reduce `u` and run the filter again with further iterations.

```
[z,e,mer,w] = canc(yn1,yn2,0.0001,5,20);
```

The plot of the mean-squared error `mer` against the iterations

```
plot(mer)
```

now converges after about six iterations. In practice, the user should vary the parameters `u` and `l` to obtain the optimum result. We can now compare the filter output with the original noise-free signal.

```
plot(x,y,'b',x,z,'r')
```

This plot shows that the noise level of the signal has been reduced dramatically by the filter. Next, the plot

```
plot(x,e,'r')
```

shows the noise extracted from the signal. Using the last output from `canc` we can calculate and display the mean filter weights of the final iteration from `w`

```
wmean = mean(w)
plot(wmean)
```

The frequency characteristic of the filter provides a more illustrative representation of the effect of the filter.

```
[h,w] = freqz(www,1,1024);
f = 1*w/(2*pi);
plot(f,abs(h))
```

which clearly shows that the filter is a lowpass filter with a relatively smooth transition band. This means that it does not have the quality of a recursive filter designed, for example, using the Butterworth approach. However, the filter weights are calculated in an optimization process rather than chosen arbitrarily.

The strength of an adaptive filter is in filtering a time series with a variable signal-to-noise ratio along the time axis. Since the filter-weight vector W is

www.it-ebooks.info

updated for each individual data point, these filters are even used in real-time applications such as telecommunication systems. We examine this behavior through an example in which the signal-to-noise ratio in the middle of the time series (x=500) is reduced from about 10% to zero.

```
clear

x = 0 : 0.1 : 100; x = x';
y = sin(x);
rng(0)
yn1 = y + 0.5*randn(size(y));
yn2 = y + 0.5*randn(size(y));

yn1(501:1001) =  y(501:1001);
yn2(501:1001) =  y(501:1001);

plot(x,yn1,x,yn2)
```

The value of u is again computed by

```
k = kron(yn1,yn1');
u = 1/max(eig(k))
```

which yields

```
u =
    0.0016
```

Audio 6.2

We now run the adaptive filter canc for 20 iterations and use the above value of u.

```
[z,e,mer,w] = canc(yn1,yn2,0.0016,5,20);
```

The plot of the mean-squared error mer versus the number of performed iterations it with stepsize u

```
plot(mer)
```

illustrates the performance of the adaptive filter, although the chosen step size u=0.0016 clearly results in a relatively rapid convergence. Again, we can now compare the filter output with the original noise-free signal.

```
plot(x,y,'b',x,z,'r')
```

This plot shows that the filter output y is almost the same as the noise-free signal x. The plot

```
plot(x,e,'r')
```

shows the noise extracted from the signal. Here we can observe some signal

www.it-ebooks.info

components that have been removed by the filter in error, from within the noise-free segment of the time series beyond x=500. Using the last output of canc, we can calculate and display the filter weights w of the final iteration

```
surf(w(3:999,:)), shading interp
```

which shows nicely the adaptation of the filter weights before and after the change in the signal-to-noise ratio. We plot only the middle part of w, as the edges 1:(l-1)/2 and end-(l-1)/2:end are all zero due to the length of the filter l. We can also use this example to demonstrate the effect that the values of u and l have on the performance of the adaptive filter. In theory, a smaller u leads to more accurate results, but the rapid adaptation to a changing signal-to-noise ratio does not work well. Larger values of l also give better results but the number of data points lost through the filtering process increases by (l-1)/2.

The graphical user interface (GUI) version canctool can be used as an alternative to canc. We use the same example data set as before; this function was created using the *GUI Design Environment* (GUIDE) (see Section 2.10). We again generate two signals, yn1 and yn2, containing the same sine wave but different Gaussian noise.

```
clear

x = 0 : 0.1 : 100; x = x';
y = sin(x);

rng(0)
yn1 = y + 0.5*randn(size(y));
yn2 = y + 0.5*randn(size(y));
```

We next run canctool

```
canctool(yn1,yn2)
```

and watch the filter iteratively adjusting its filter weights. The function canctool accepts the primary and reference signals as inputs. The GUI provides default values for the filter length, the convergence rate, and the number of iterations. The user must adjust these values to the specific data, for example, using the formula for a suitable convergence rate that has been described above. As an example, we choose a filter length of 11, a convergence rate of 0.00001, and 30 iterations. Once the calculation is complete, canctool exports the filtered primary signal yy, the filtered reference signal zz (using the same set of filter weights), the noise ee extracted from the primary signal, and the mean-squared error mer for each iteration, to the workspace using the function assignin. The application of this algorithm has been demonstrated

Movie 6.7

www.it-ebooks.info

on duplicate oxygen-isotope records from ocean sediments (Trauth 1998). This work by M.H. Trauth illustrates the use not only of the modified LMS algorithm, but also of another type of adaptive filter, the recursive least-squares (RLS) algorithm, in various different environments (Haykin 1991, Trauth 1998).

Recommended Reading

Alexander ST (1986) Adaptive Signal Processing: Theory and Applications. Springer, Berlin Heidelberg New York

Buttkus B (2000) Spectral Analysis and Filter Theory in Applied Geophysics. Springer, Berlin Heidelberg New York

Cowan CFN, Grant PM (1985) Adaptive Filters. Prentice Hall, Englewood Cliffs, New Jersey

Grünigen DH (2004) Digitale Signalverarbeitung, mit einer Einführung in die kontinuierlichen Signale und Systeme, Dritte bearbeitete und erweiterte Auflage. Fachbuchverlag Leipzig, Leipzig

Hattingh M (1988) A new Data Adaptive Filtering Program to Remove Noise from Geophysical Time- or Space Series Data. Computers & Geosciences 14(4):467–480

Haykin S (2003) Adaptive Filter Theory. Prentice Hall, Englewood Cliffs, New Jersey

Kalman R, Bucy R (1961) New Results in Linear Filtering and Prediction Theory. ASME Tans. Ser. D Jour. Basic Eng. 83:95–107

MathWorks (2014) Signal Processing Toolbox – User's Guide. The MathWorks, Inc., Natick, MA

Sibul LH (1987) Adaptive Signal Processing. IEEE Press

Trauth MH (1998) Noise Removal from Duplicate Paleoceanographic Time-Series: The Use of adaptive Filtering Techniques. Mathematical Geology 30(5):557–574

Weeks, M (2007) Digital Signal Processing Using MATLAB and Wavelts. Infinity Science Press, Jones and Bartlett Publishers, Boston Toronto London Singapor

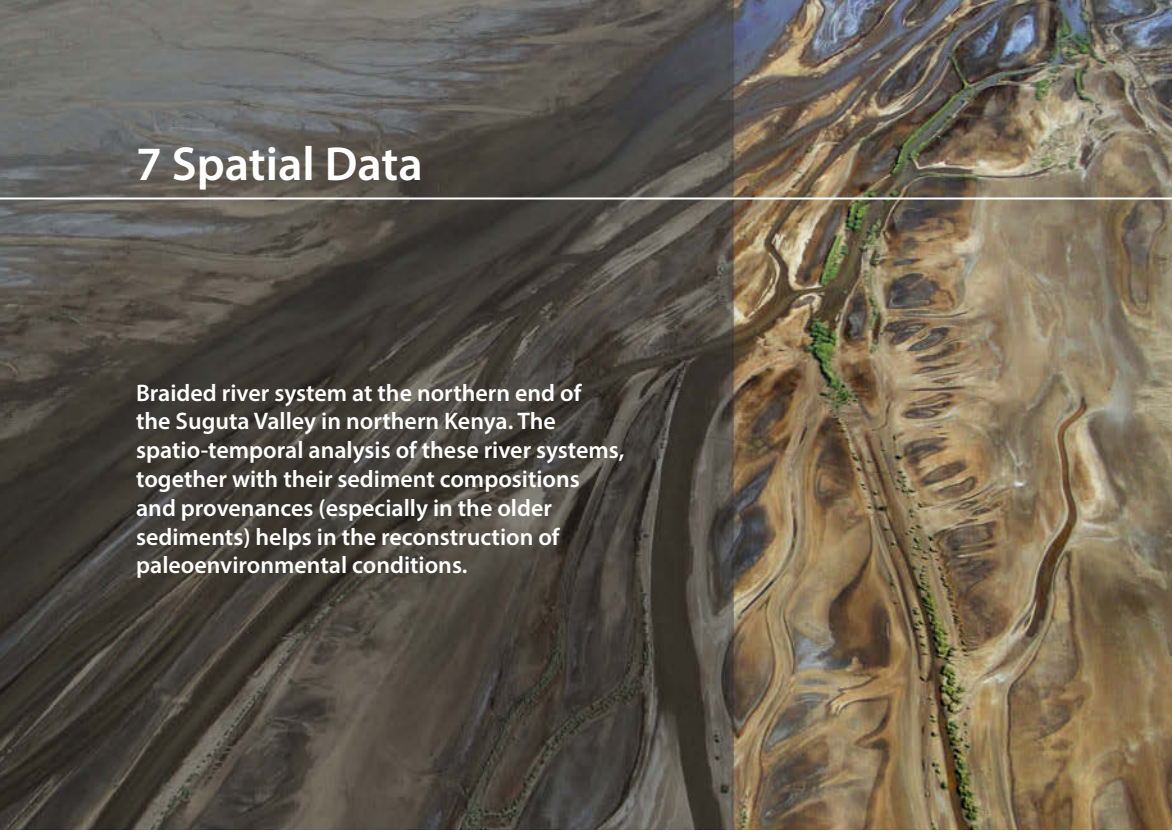
Widrow B, Hoff Jr. M (1960) Adaptive Switching Circuits. IRE WESCON Conv. Rev. 4:96–104

Widrow B, Glover JR, McCool JM, Kaunitz J, Williams CS, Hearn RH, Zeidler JR, Dong E, Goodlin RC (1975) Adaptive Noise Cancelling: Principles and Applications. Proc. IEEE 63(12):1692–1716

Wiener N (1949) Extrapolation, Interpolation and Smoothing of Stationary Time Series, with Engineering Applications. MIT Press, Cambridge, Mass (reprint of an article originally issued as a classified National Defense Research Report, February, 1942)

www.it-ebooks.info

7 Spatial Data



Braided river system at the northern end of the Suguta Valley in northern Kenya. The spatio-temporal analysis of these river systems, together with their sediment compositions and provenances (especially in the older sediments) helps in the reconstruction of paleoenvironmental conditions.

7.1 Types of Spatial Data

Most data in earth sciences are spatially distributed, either as vector data, (points, lines, polygons) or as raster data (gridded topography). Vector data are generated by digitizing map objects such as drainage networks or outlines of lithologic units. Raster data can be obtained directly from a satellite sensor output, but gridded data can also, in most cases, be interpolated from irregularly-distributed field samples (gridding).

The following section introduces the use of vector data by using coastline data as an example (Section 7.2). The acquisition and handling of raster data are then illustrated using digital topographic data (Sections 7.3 to 7.5). The availability and use of digital elevation data has increased considerably since the early 90s. With a resolution of 5 arc minutes (about 9 km), ETOPO5 was one of the first data sets for topography and bathymetry. In October 2001 it was replaced by ETOPO2, which has a resolution of 2 arc minutes (about 4 km), and in March 2009 the ETOPO1 became available, which has a resolution of 1 arc minutes (about 2 km). There is also a data set for topography called GTOPO30 completed in 1996 that has a horizontal grid spacing of 30 arc seconds (about 1 km). More recently, the 30 and 90 m resolution data

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_7

www.it-ebooks.info

from the Shuttle Radar Topography Mission (SRTM) have replaced the older data sets in most scientific studies. Section 7.6 demonstrates how to export 3D graphics objects such as digital terrain models to create interactive documents such as Apple iBooks Author documents or interactive Adobe PDF documents, in particular for use with tablet computers.

The second part of this chapter deals with the estimation of continuous surfaces from unevenly-spaced data and the statistics of spatial data (Sections 7.7 to 7.8). In earth sciences, most data are collected in an irregular pattern. Access to rock samples is often restricted to natural outcrops such as shoreline cliffs and the walls of a gorge, or anthropogenic outcrops such as road cuttings and quarries. The sections on interpolating such unevenly-spaced data illustrate the use of the most important gridding routines and outline the potential pitfalls when using these methods. Sections 7.9 to 7.11 introduce various methods for statistically analyzing spatial data, including the application of statistical tests to point distributions (Section 7.9), the spatial analysis of digital elevation models (Section 7.10), and an overview of geostatistics and kriging (Section 7.11).

This chapter requires the Mapping Toolbox (MathWorks 2014a), although most graphics routines used in our examples can be easily replaced by standard MATLAB functions. Section 7.6 requires Simulink 3D Animation to create animated 3D graphics objects (MathWorks 2014b). An alternative and useful mapping toolbox by Rich Pawlowicz (Earth and Ocean Sciences, at the University of British Columbia) is available from

```
http://www.eos.ubc.ca/~rich/
```

The handling and processing of large spatial data sets requires a computing system with at least 4 GB physical memory.

7.2 The Global Geography Database GSHHG

The Global Self-consistent, Hierarchical, High-resolution Geography (GSHHG) database is an amalgamation of two public domain databases by Paul Wessel (SOEST, University of Hawaii, Honolulu, HI) and Walter Smith (NOAA Laboratory for Satellite Altimetry, Silver Spring, MD) (Wessel and Smith 1996). The GSHHG database combines the older GSHHS database (Soluri and Woodson 1990, Wessel and Smith 1996), which is a shoreline database, with the poor quality Antarctica data replaced by the more accurate data from Bohlander and Scambos (2007), together with rivers and borders from the CIA World Data Bank II (WDBII) (Gorny 1977). The GSHHG data can be downloaded from the web page of the US National Geophysical Data Center (NGDC):

www.it-ebooks.info

```
http://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html
```

As an example we use the data from the file *coastline.txt*, which contains an excerpt from the older GSHHG shoreline database. The two columns in this file represent the *longitude/latitude* coordinates of `NaN`-separated polygons or coastline segments.

```
NaN          NaN
42.892067    0.000000
42.893692    0.001760
NaN          NaN
42.891052    0.001467
42.898093    0.007921
42.904546    0.013201
42.907480    0.016721
42.910414    0.020828
42.913054    0.024642
42.915987    0.028749
42.918921    0.032562
42.922441    0.035789
(cont'd)
```

The `NaN`s perform two functions: they provide a means of identifying break points in the data and they serve as pen-up commands when the Mapping Toolbox plots vector maps. The shorelines can be displayed by using

```
clear

data = load('coastline.txt');

plot(data(:,1),data(:,2)), axis equal
xlabel('Longitude'), ylabel('Latitude')
```

More advanced plotting functions that are contained in the Mapping Toolbox (MathWorks 2014a) allow an alternative version of this plot to be generated (Fig. 7.1).

```
axesm('MapProjection','lambert', ...
      'MapLatLimit',[0 15], ...
      'MapLonLimit',[35 55], ...
      'Frame','on', ...
      'MeridianLabel','on', ...
      'ParallelLabel','on');
plotm(data(:,2),data(:,1));
```

Note that the input for `plotm` must be in the correct order, with the *latitude* entered first, followed by the *longitude* (i.e., the second column of the data matrix must be entered first). In contrast, the function `plot` requires an *xy* input, i.e., input must be in the opposite order, with the first column of the matrix entered first. The function `axesm` defines the map axis and sets various

www.it-ebooks.info

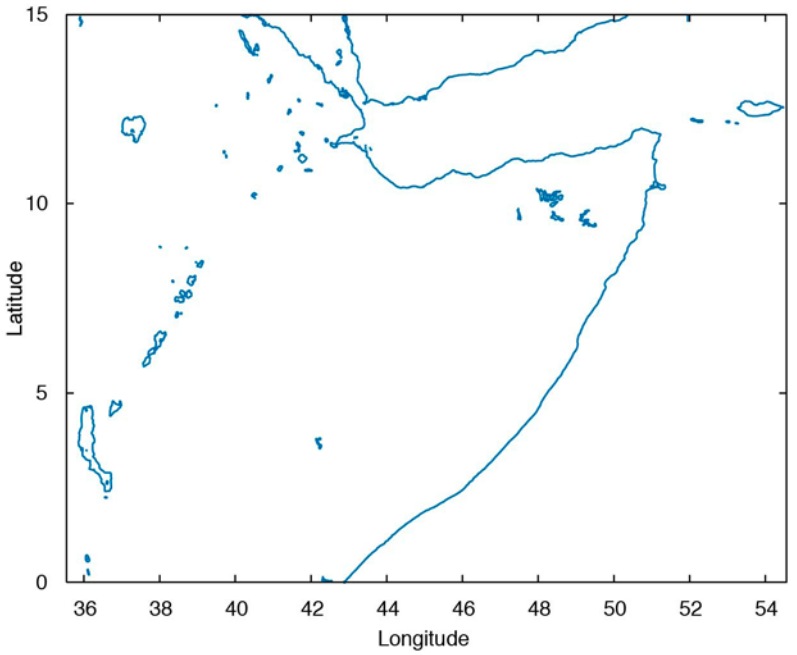


Fig. 7.1 Shoreline display from the GSHHS data set. The map shows an area between latitudes 0° and 15° north, and longitudes 40° and 50° east. This simple map is made using the function `plot`, with equal axis aspect ratios (data from Wessel and Smith 1996).

map properties such as the map projection, the map limits, and the axis labels.

7.3 The 1-Minute Gridded Global Relief Data ETOPO1

ETOPO1 is a global data base of topographic and bathymetric data on a regular 1 arc-minute grid (about 2 km) (Amante and Eakins 2009). Older ETOPO2v2 and ETOPO5 global relief grids have been superseded but are still available. ETOPO1 is a compilation of data from a variety of sources. It can be downloaded from the US National Geophysical Data Center (NGDC) web page

```
http://www.ngdc.noaa.gov/mgg/global/
```

As an example we use the data from the file *grid01-258.asc* that has the following content:

www.it-ebooks.info

```
NCOLS  1801
NROWS  2401
XLLCENTER   30.0000000000
YLLCENTER -20.0000000000
CELLSIZE 0.016666666667
NODATA_VALUE   -32768
    294    299    293    288    285    282 ...
    294    294    289    291    281    276 ...
    288    285    285    283    280    272 ...
    280    276    277    277    283    279 ...
    271    272    271    274    280    283 ...
    267    270    268    272    272    276 ...
(cont'd)
```

The headers document the size of the data matrix (e.g., 1801 columns and 2401 rows in our example), the coordinates of the lower-left corner (e.g., x=30 and y=–20), the cell size (e.g., ~0.0167=1/60 degree latitude and longitude), and the *–32768* flag for data voids. We comment the header by typing `%` at the beginning of the first six lines

```
%NCOLS  1801
%NROWS  2401
%XLLCENTER   30.0000000000
%YLLCENTER -20.0000000000
%CELLSIZE 0.016666666667
%NODATA_VALUE   -32768
    294    299    293    288    285    282 ...
    294    294    289    291    281    276 ...
    288    285    285    283    280    272 ...
    280    276    277    277    283    279 ...
    271    272    271    274    280    283 ...
    267    270    268    272    272    276 ...
(cont'd)
```

and load the data into the workspace.

```
clear

ETOPO1 = load('grid01-258.asc');
```

We flip the matrix up and down. The *–32768* flag for data voids must then be replaced by the MATLAB representation for Not-a-Number `NaN`.

```
ETOPO1 = flipud(ETOPO1);
ETOPO1(find(ETOPO1 == -32768)) = NaN;
```

Finally, we check whether the data are now correctly stored in the workspace by printing the minimum and maximum elevations for the area.

```
max(ETOPO1(:))
min(ETOPO1(:))
```

www.it-ebooks.info

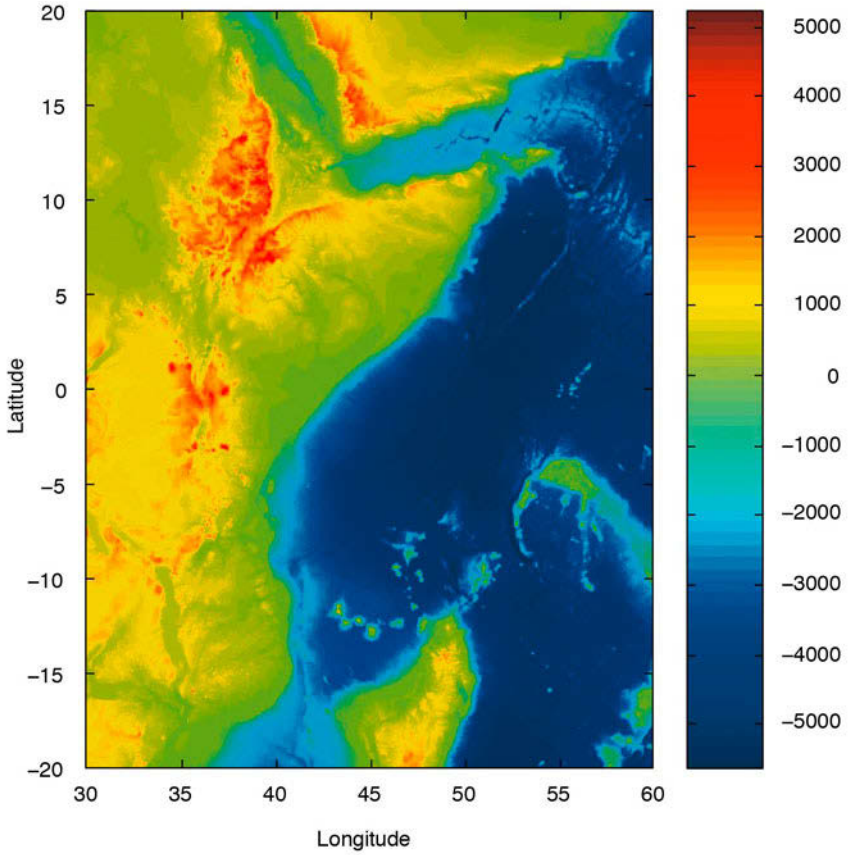


Fig. 7.2 Display from the ETOPO1 elevation data set. The map uses the function `surf` to generate a colored surface. The colorbar provides information on the colormap used to portray the topographic and bathymetric variations (data from Amante and Eakins 2009).

In this example the maximum elevation for the area is 5,677 m and the minimum elevation is −5,859 m. The reference level is the sea level at 0 m. We now define a coordinate system using the information that the lower-left corner is latitude 20° south and longitude 30° east. The resolution is 1 arc minute, corresponding to 1/60 of a degree.

```
[LON,LAT] = meshgrid(30:1/60:60,-20:1/60:20);
```

We now generate a colored surface from the elevation data using the function `surf`.

```
surf(LON,LAT,ETOPO1)
colormap jet
```

www.it-ebooks.info

```
shading interp
axis equal, view(0,90)
colorbar
```

This script opens a new figure window and generates a colored surface. The surface is highlighted by a set of color shades, and is displayed in an overhead view (Fig. 7.2). Additional display methods will be described in the section on SRTM elevation data (Section 7.5).

Interactive 7.1

7.4 The 30-Arc Seconds Elevation Model GTOPO30

The 30 arc second (about 1 km) global digital elevation data set GTOPO30 contains only elevation data, not bathymetry. The data set has been developed by the Earth Resources Observation System Data Center and is available from the U.S. Geological Survey web page

```
https://lta.cr.usgs.gov/GTOPO30
```

The model uses a variety of international data sources but is mainly based on raster data from the Digital Terrain Elevation Model (DTEM) and vector data from the Digital Chart of the World (DCW). The GTOPO30 data set has been divided into 33 tiles. The tile names refer to the longitude and latitude of the upper-left (northwest) corner of the tile. The tile name *e020n40* refers to the coordinates of the upper-left corner of the tile, i.e., longitude 20 degrees east and latitude 40 degrees north. As an example we select and download the tile *e020n40*, which is provided as a 24.9 MB compressed *tar* file. After decompressing the file we obtain eight files containing the raw data and header files in various formats. The *tar* file also provides a GIF image of a shaded relief display of the data.

Importing the GTOPO30 data into the workspace is simple. The Mapping Toolbox provides an import routine `gtopo30` that reads the data and stores it onto a regular data grid. We import only a subset of the original matrix:

```
clear

latlim = [-5 5]; lonlim = [30 40];
GTOPO30 = gtopo30('E020N40',1,latlim,lonlim);
```

This script reads the data from the tile *e020n40* (without file extension) at full resolution (scale factor =1) into the matrix `GTOPO30`, which has the dimensions of 1,200 by 1,200 cells. The coordinate system is defined by using the *lon/lat* limits listed above. The resolution is 30 arc seconds, corresponding to 1/120 of a degree.

```
[LON,LAT] = meshgrid(30:1/120:40-1/120,-5:1/120:5-1/120);
```

www.it-ebooks.info

We need to reduce the limits by a factor of 120, in order to obtain a matrix of similar dimensions to the GTOPO30 matrix. A grayscale image can be generated from the elevation data using the function `surf`. The fourth power of the colormap `gray` is flipped vertically in order to obtain dark colors for high elevations and light colors for low elevations.

```
surf(LON,LAT,GTOPO30)
shading interp
colormap(flipud(gray.^4))
axis equal, view(0,90)
colorbar
```

Interactive 7.2

This script opens a new figure window and generates the gray surface using interpolated shading, displayed in an overhead view (Fig. 7.3).

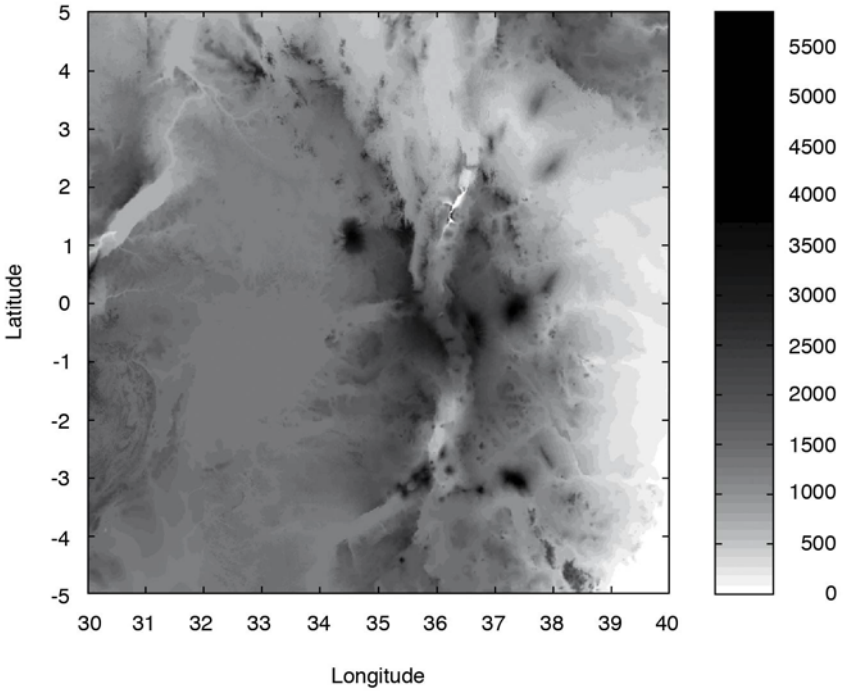


Fig. 7.3 Display from the GTOPO30 elevation data set. The map uses the function `surf` to generate a gray surface. The fourth power of the colormap `gray` is used to intensify the gray shades of the map at higher elevations and the colormap is then flipped vertically in order to obtain dark colors for high elevations and light colors for low elevations (data from the U.S. Geological Survey).

www.it-ebooks.info

7.5 The Shuttle Radar Topography Mission SRTM

The Shuttle Radar Topography Mission (SRTM) was a radar system on board the Space Shuttle *Endeavour* during an 11-day mission in February 2000 (Farr et al. 2000, 2007). SRTM was an international project spearheaded by the National Geospatial-Intelligence Agency (NGA) and the National Aeronautics and Space Administration (NASA). Detailed information on the SRTM project, including a gallery of images and a user's forum, can be accessed through the NASA web page:

```
http://www2.jpl.nasa.gov/srtm/
```

The data were processed at the Jet Propulsion Laboratory. They are distributed through the United States Geological Survey's (USGS) National Map Viewer and Download Platform:

```
http://viewer.nationalmap.gov/viewer/
```

Alternatively, the raw data files can be downloaded from

```
http://dds.cr.usgs.gov/srtm/
```

This directory contains zipped files of SRTM DEMs from various areas of the world, processed by the SRTM global processor and sampled at resolutions of 1 arc second (SRTM-1, 30 meter grid) and 3 arc seconds (SRTM-3, 90 meter grid). As an example we download the 1.7 MB file *s01e036.hgt.zip* from

```
http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/Africa/
```

containing SRTM-3 data for the Kenya Rift Valley in East Africa. All elevations are in meters referenced to the WGS84 EGM96 geoid, as documented at

```
http://earth-info.nga.mil/GandG/wgs84/index.html
```

The name of this file refers to the longitude and latitude of the lower-left (southwest) pixel of the tile, i.e., latitude one degree south and longitude 36 degrees east. SRTM-3 data contain 1,201 lines and 1,201 samples, with similar numbers of overlapping rows and columns. After having downloaded and unzipped the file we save *s01e036.hgt* in our working directory. The digital elevation model is provided as 16-bit signed integer data in a simple binary raster. The bit order is *big-endian* (Motorola's standard) with the most significant bit first. The data are imported into the workspace using

```
clear
```

www.it-ebooks.info

```
fid = fopen('S01E036.hgt','r');
SRTM = fread(fid,[1201,inf],'int16','b');
fclose(fid);
```

This script opens the file *s01e036.hgt* for read only access using `fopen` and defines the file identifier `fid`, which is then used to read the binaries from the file (using `fread`) and to write them into the matrix `SRTM`. Function `fclose` closes the file defined by `fid`. The matrix first needs to be transposed and flipped vertically.

```
SRTM = SRTM'; SRTM = flipud(SRTM);
```

The *–32768* flag for data voids can be replaced by `NaN`, which is the MATLAB representation for Not-a-Number.

```
SRTM(find(SRTM == -32768)) = NaN;
```

The SRTM data contain numerous gaps that might cause spurious effects during statistical analysis or when displaying the digital elevation model in a graph. A popular way to eliminate gaps in digital elevation models is by filling gaps with the arithmetic means of adjacent elements. We use the function `nanmean` since it treats `NaN`s as missing values and returns the mean of the remaining elements that are not `NaN`s. The following double `for` loop averages `SRTM(i-1:i+1,j-1:j+1)` arrays, i.e., averages over three-by-three element wide areas of the digital elevation model.

```
for i = 2 : 1200
    for j = 2 : 1200
        if isnan(SRTM(i,j)) == 1
            SRTM(i,j) = nanmean(nanmean(SRTM(i-1:i+1,j-1:j+1)));
        end
    end
end
clear i j
```

If there are still `NaN`s in the data set (as in our example) causing errors when importing the data set into a *Virtual Reality Modeling Language* (VRML) client, the double `for` loop can be run a second time. Finally, we check whether the data are now correctly stored in the workspace by printing the minimum and maximum elevations of the area.

```
max(SRTM(:))

ans =
   3992

min(SRTM(:))
```

www.it-ebooks.info

```
ans =
   1504
```

In our example the maximum elevation of the area is 3,992 m above sea level and the minimum is 1,504 m. A coordinate system can be defined using the information that the lower-left corner is *s01e036*. The resolution is 3 arc seconds, corresponding to 1/1,200 of a degree.

```
[LON,LAT] = meshgrid(36:1/1200:37,-1:1/1200:0);
```

A shaded grayscale map can be generated from the elevation data using the function `surfl`. This function displays a shaded surface with simulated lighting.

```
surfl(LON,LAT,SRTM)
shading interp
colormap gray
view(0,90)
```

This script opens a new figure window and generates the shaded-relief map using interpolated shading, as well as a gray colormap, displayed in an overhead view. Since SRTM data contain a large amount of noise, we first smooth the data using an arbitrary 9-by-9 pixel moving average filter. The new matrix is then stored in the matrix `SRTM_FILTERED`.

```
B = 1/81 * ones(9,9);
SRTM_FILTERED = filter2(B,SRTM);
```

The corresponding shaded-relief map is generated by

```
surfl(LON,LAT,SRTM_FILTERED)
shading interp
colormap gray
view(0,90)
```

After having generated the shaded-relief map (Fig. 7.4), the plot must be exported to a graphics file. For instance, the figure may be written into a JPEG format with a 70% quality level and 300 dpi resolution.

Interactive 7.3

```
print -djpeg70 -r300 srtmimage
```

The new file *srtmimage.jpg* has a size of 320 KB; the decompressed image has a size of 16.5 MB. This file can now be imported into another software package such as Adobe Photoshop.

www.it-ebooks.info

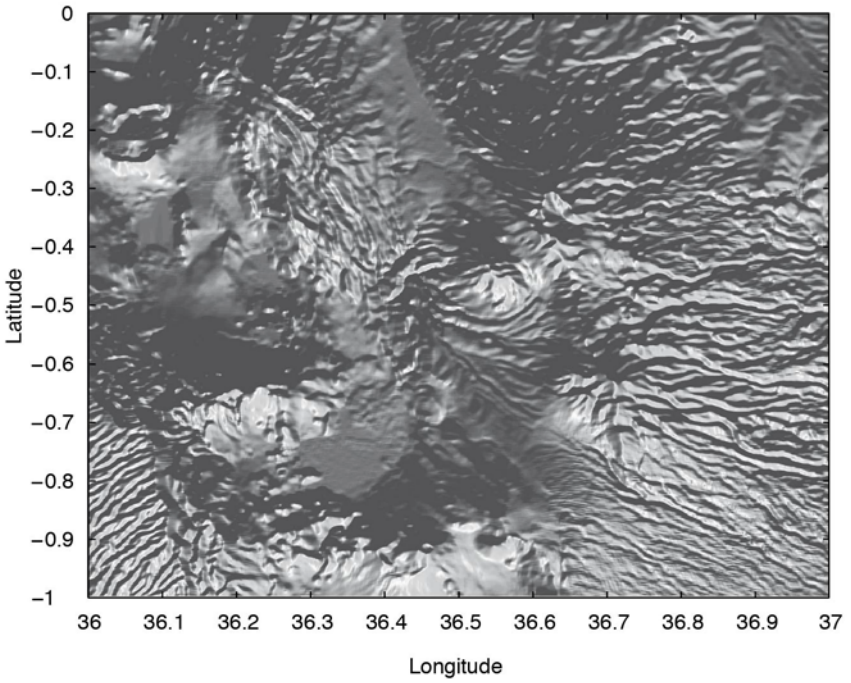


Fig. 7.4 Display from the filtered SRTM elevation data set. The map uses the function `surfl` to generate a shaded-relief map with simulated lighting, using interpolated shading and a gray colormap, displayed in an overhead view (data from Farr et al. 2000, 2008).

7.6 Exporting 3D Graphics to Create Interactive Documents

Since the introduction of electronic devices with touch controls, interactive 3D graphics objects have become increasingly popular in multimedia electronic books (*ebooks*). The Simulink 3D Animation available from The MathWorks, Inc. provides the necessary tools to create and export 3D graphics objects for inclusion in documents such as multimedia ebooks, interactive webpages, and presentations (MathWorks 2014b). The VRML files are text files with the file extension *.wrl* for *world*, short for *Virtual Reality World*. Opening such a file in, for example, the MATLAB editor reveals that it contains vertices and edges for 3D polygons, together with parameters such as surface colors, textures, transparency and so on. The VRML files can be viewed using a browser plugin (of which there are many different types available online), or using 3D software such as the open-source *MeshLab* software

`http://meshlab.sourceforge.net`

www.it-ebooks.info

or the free *Blender* software available at

```
http://www.blender.org
```

These tools (as well as others) can be used to convert the VRML format into other 3D graphics object file formats such as the *Universal 3D .u3d* and *COLLADA .dae* formats. As an example the U3D format is the format required to place a 3D graphics object onto a PDF page with Adobe Acrobat. The DAE format is used to include 3D graphics objects in Apple iBooks Author files, for use with Apple iPads and other iOS devices with touch controls.

The digital terrain models created in previous sections are great examples for such interactive 3D objects. We use a modified script to import and display an SRTM data set from the previous section as an example. The data are imported into the workspace using

```
clear

fid = fopen('S01E036.hgt','r');
SRTM = fread(fid,[1201,inf],'int16','b');
fclose(fid);
```

The matrix first needs to be transposed and flipped vertically.

```
SRTM = SRTM'; SRTM = flipud(SRTM);
```

The SRTM data contain numerous gaps that might cause spurious effects during statistical analysis or when displaying the digital elevation model in a graph. We can use the method described in the previous section to eliminate the gaps. In this case, however, we have to search the data set for additional possible `NaN`s, even after having searched for `NaN`s and replaced them with the means of the surrounding pixels, because the VRML clients cannot handle the character string *NaN* and therefore produces an error message when importing the file into software such as MeshLab. Since it is for display only we can also use a simplified method of replacing the gaps marked by *-32768* with the mean of the entire DEM.

```
SRTM(find(SRTM == -32768)) = mean(SRTM(:));
```

A coordinate system can then be defined using the information that the lower-left corner is *s01e036*. The resolution is 3 arc seconds, corresponding to 1/1,200 of a degree.

```
[LON,LAT] = meshgrid(36:1/1200:37,-1:1/1200:0);
```

www.it-ebooks.info

We also need to reduce the size of the array by a factor of 10 because most VRML clients limit the number of vertices in a VRML file. You can run an experiment with a larger number of vertices and see whether it works with your VRML software.

```
LON = LON(1:10:end,1:10:end);
LAT = LAT(1:10:end,1:10:end);
SRTM = SRTM(1:10:end,1:10:end);
```

We also eliminate the edges of the data set, as these may cause problems when writing the VRML files.

```
LON = LON(2:end-1,2:end-1);
LAT = LAT(2:end-1,2:end-1);
SRTM = SRTM(2:end-1,2:end-1);
```

3D graphics can be generated from the elevation data using the function `trimesh` following Delaunay triangulation of the data set using `delaunay`.

```
tri = delaunay(LON,LAT);
trimesh(tri,LON,LAT,SRTM)
axis([35.5 37.5 -1.5 0.5 -500 4500]), axis off
```

Next, we determine the size of the array `SRTM` and reshape the 119-by-119 array `SRTM` to a `n`-by-1 array, where `n=xdim*ydim=14161`.

```
[xdim ydim] = size(SRTM);
SRTM = SRTM(:);
```

We then determine the range of the z-values and define the spacing in x and y directions.

```
zrange = range(SRTM);
xspace = 10;
yspace = 10;
```

We again use the colormap `demcmap` to display the SRTM data set.

```
cmap = demcmap(SRTM,256);
```

Alternatively, we can use any other colormap, even one that we have created ourselves, to display the digital terrain model. We then normalize the elevation data to the `[1,length(cmap)]` range.

```
cmap = cmap(round((SRTM-min(SRTM)) ...
    .*(size(cmap,1)-1)./zrange)+1,:);
```

The Simulink 3D Animation User's Guide (MathWorks 2014b) contains a great introduction in its chapter *Interact with Virtual Reality Worlds*, which

www.it-ebooks.info

explains the following steps in greater detail than provided here. We first need to make sure that all virtual reality worlds are closed and deleted.

```
out = vrwho;
for i=1:length(out)
    while (get(out(i),'opencount')~=0)
        close(out(i));
    end
    delete(out(i));
end
```

We then create a new virtual reality world `myworld` using `vrworld`.

```
myworld = vrworld('');
open(myworld)
```

Our world then needs a shape and a grid. We create the shape `newShape` with the name `Landscape` and the grid `newGrid` using `vrnode`. The function `vrnode` creates a handle to either an existing or a new node.

```
shapeName = ['Landscape'];
newShape = vrnode(myworld,shapeName,'Shape');
newGrid = vrnode(newShape,'geometry','DEM','ElevationGrid');
```

We can access the contents of the variable `newShape` from the `vrworld` class in a similar manner to structure arrays, e.g., by typing

```
getfield(newShape.geometry)
```

which lists a number of the defining parameters of `newShape` such as, for example, the color. The nodes of the world can be accessed using one of the two commands

```
nodes(myworld)
mynodes = get(myworld,'Nodes')
```

We can access the fields of the node `Landscape` using one of the two commands

```
fields(myworld.Landscape)
fields(mynodes.DEM)
```

and the `DEM` using

```
fields(myworld(1))
fields(mynodes(2))
```

both with a detailed output of all the parameter settings of the nodes. We can also list the fields in `newShape` and `newGrid` using

```
fields(newShape)
```

www.it-ebooks.info

```
fields(newGrid)
```

and modify the values of the various parameters of `newShape` and `newGrid`. As an example we can reduce the height values to 20% of the original SRTM values in order to change the proportions of the 3D graphics.

```
setfield(newGrid, ...
    'xDimension',xdim,...
    'zDimension',ydim,...
    'xSpacing',xspace,...
    'zSpacing',yspace,...
    'height',0.2*SRTM);
```

We then use the same colormap `cmap` that was created previously (before using `demcmap`) based on elevation

```
GridColor = vrnode(newGrid,...
        'color','TerrainColor',...
        'Color');
GridColor.color = cmap;
getfield(newGrid,'color')
```

and save the world `myworld` created with Simulink 3D Animation to a new VRML file *srtm_2.wrl*.

```
save(myworld,'srtm.wrl')
close(myworld)
delete(myworld)
```

Interactive 7.4

We then close and delete the world `myworld`. Importing the resulting VRML file into a VRML client such as MeshLab or Blender reveals the terrain model does indeed have colors, as defined by `demcmap`. We can use the file in the VRML format to create an interactive document. However, the software to create such interactive documents, such as Adobe Acrobat or Apple iBooks Author, may require different file formats; these can easily be converted with the MeshLab or Blender software.

7.7 Gridding and Contouring

The previous data sets were all stored in evenly-spaced two-dimensional arrays. Most data in earth sciences, however, are obtained from irregular sampling patterns. The data are therefore unevenly-spaced and need to be interpolated in order to allow a smooth and continuous surface to be computed from our measurements in the field. *Surface estimation* is typically carried out in two major steps (Fig. 7.5). Firstly, the number of *control points* needs to be selected, and secondly, the value of the variable of interest

www.it-ebooks.info

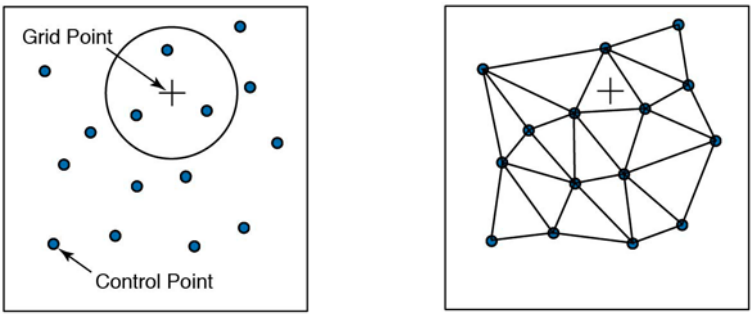


Fig. 7.5 Methods for selecting the control points to use for estimating the values at grid points. **a** Construction of a circle around the grid point (plus sign) with a radius defined by spatial autocorrelation of the *z*-values at the control points (small circles). **b** Triangulation: the control points are selected from the vertices of the triangle surrounding the grid point, with the option of also including the vertices of the adjoining triangles.

needs to be estimated for the *grid points*. Control points are the unevenly-spaced field measurements, such as the thicknesses of sandstone units at different outcrops or the concentrations of a chemical tracer in water wells. The data are generally represented as *xyz* triplets, where *x* and *y* are spatial coordinates and *z* is the variable of interest. In such cases most gridding methods require continuous and unique data. However, spatial variables in earth sciences are often discontinuous and not spatially unique: for example, the sandstone unit may be faulted or folded. Furthermore, gridding requires spatial autocorrelation, i.e., the neighboring data points should be correlated with each other through a specific relationship. There is no point in making a surface estimation if the *z* variables are random and have no autocorrelation. Having selected the control points, a number of different methods are available for calculating the *z*-values at the evenly-spaced grid points.

Various techniques exist for selecting the control points. Most methods make arbitrary assumptions on the autocorrelation of the *z*-variable. The *nearest-neighbor criterion* includes all control points within a circular neighborhood of the grid point, where the radius of the circle is specified by the user (Fig. 7.5 a). Since the degree of spatial autocorrelation is likely to decrease with increasing distance from the grid point, considering too many distant control points is likely to lead to erroneous results when computing values for the grid points. On the other hand, using radii that are too small may limit the number of control points used in calculating the grid point values to a very small number, resulting in a noisy estimate of the modeled surface.

www.it-ebooks.info

It is perhaps due to these difficulties that *triangulation* is often used as an alternative method for selecting the control points (Fig. 7.5 b). In this technique, all control points are connected in a triangular network. Every grid point is located within the triangular area formed by three control points. The z-value of the grid point is computed from the z-values of the three grid points. A modification of this form of gridding also uses the three points at the apices of the three adjoining triangles. The *Delauney triangulation* method uses a triangular net in which the acuteness of the triangles is minimized, i.e., the triangles are as close as possible to equilateral.

Kriging, introduced in Section 7.11, is an alternative approach to select control points. It is often regarded as the ultimate method of gridding. Some people even use the term *geostatistics* synonymously with kriging. Kriging is a method for quantifying the spatial autocorrelation and hence the circle's dimension. More sophisticated versions of kriging use an elliptical area instead of a circle.

As mentioned above, the second step in surface estimation is the actual computation of the z-values for the grid points. The *arithmetic mean* of the measured z-values at the control points

$$\overline{z} = \frac{1}{N} \sum_{i=1}^{N} z_i$$

provides the easiest way of computing the values at the grid points. This is a particularly useful method if there are only a limited number of control points. If the study area is well covered by control points and the distance between these points is highly variable, the z-values of the grid points should be computed using a *weighted mean*. This involves weighting the z-values at the control points by the inverse of the distance d_i from the grid points.

$$\overline{z} = \frac{\sum_{i=1}^{N} (z_i / d_i)}{\sum_{i=1}^{N} (1 / d_i)}$$

Depending on the spatial scaling relationship of the variable z, the inverse of the square root of the distance may be used to weight the z-values, rather than simply the inverse of distance. The fitting of 3D *splines* to the control points offers another method for computing the grid point values that is commonly used in the earth sciences. Most routines used in surface estimation involve *cubic polynomial splines*, i.e., a third-degree 3D polynomial is fitted to at

www.it-ebooks.info

least six adjacent control points. The final surface is a composite comprising different portions of these splines. MATLAB has, from the start, provided a biharmonic spline interpolation method, which was developed by Sandwell (1987). This gridding method is particularly well suited for producing smooth surfaces from noisy data sets with unevenly-distributed control points. As an example we use synthetic *xyz* data representing the vertical distance between the surface of an imaginary stratigraphic horizon that has been displaced by a normal fault, and a reference surface. The foot wall of the fault shows roughly horizontal strata, whereas the hanging wall is characterized by the development of two large sedimentary basins. The *xyz* data are irregularly distributed and so need to be interpolated onto a regular grid. The *xyz* data are stored as a three-column table in a file named *normalfault.txt*.

```
4.3229698e+02   7.4641694e+01   9.7283620e-01
4.4610209e+02   7.2198697e+01   6.0655065e-01
4.5190255e+02   7.8713355e+01   1.4741054e+00
4.6617169e+02   8.7182410e+01   2.2842172e+00
4.6524362e+02   9.7361564e+01   1.1295175e-01
4.5526682e+02   1.1454397e+02   1.9007110e+00
4.2930233e+02   7.3175896e+01   3.3647807e+00
(cont'd)
```

The first and second column contain the coordinates *x* (between 420 and 470 of an arbitrary spatial coordinate system) and *y* (between 70 and 120), while the third column contains the vertical *z*-values. The data are loaded using

```
clear

data = load('normalfault.txt');
```

Initially, we wish to create an overview plot of the spatial distribution of the control points. In order to label the points in the plot, numerical *z*-values of the third column are converted into character string representations with a maximum of two digits.

```
labels = num2str(data(:,3),2);
```

The 2D plot of our data is generated in two steps. Firstly, the data are displayed as empty circles using the `plot` command. Secondly, the data are labeled using the function `text(x,y,'string')`, which adds text contained in `string` to the *xy* locations. The value 1 is added to all *x* coordinates in order to produce a small offset between the circles and the text.

```
plot(data(:,1),data(:,2),'o'), hold on
text(data(:,1)+1,data(:,2),labels), hold off
```

www.it-ebooks.info

This plot helps us to define the axis limits for gridding and contouring: `xlim`=[420 470] and `ylim`=[70 120]. The function `meshgrid` transforms the domain specified by vectors `x` and `y` into arrays `XI` and `YI`. The rows of the output array `XI` are copies of the vector `x` and the columns of the output array `YI` are copies of the vector `y`. We choose 1.0 as the grid interval.

```
x = 420:1:470; y = 70:1:120;
[XI,YI] = meshgrid(x,y);
```

The biharmonic spline interpolation is used to interpolate the irregular-spaced data at the grid points specified by `XI` and `YI`.

```
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');
```

The option `v4` selects the biharmonic spline interpolation, which was the sole gridding algorithm available until MATLAB4 was replaced by MATLAB5. MATLAB provides various tools with which to display the results. The simplest way to display the gridding results is as a contour plot using `contour`. By default, the number of contour levels and the values of the contour levels are chosen automatically. The choice of the contour levels depends on the minimum and maximum values of *z*.

```
contour(XI,YI,ZI)
```

Alternatively, the number of contours can be chosen manually, e.g., ten contour levels.

```
contour(XI,YI,ZI,10)
```

Contouring can also be performed at values specified in a vector `v`. Since the maximum and minimum values of `z` are

```
min(data(:,3))

ans =
   -27.4357

max(data(:,3))

ans =
   21.3018
```

we choose

```
v = -40 : 10 : 20;
```

The command

```
[c,h] = contour(XI,YI,ZI,v);
```

www.it-ebooks.info

yields contour matrix `c` and a handle `h` that can be used as input to the function `clabel`, which labels contours automatically.

```
clabel(c,h)
```

Alternatively, the plot can be labeled manually by selecting the `manual` option in the function `clabel`. This function places labels onto locations that have been selected with the mouse. Labeling is terminated by pressing the `return` key.

```
[c,h] = contour(XI,YI,ZI,v);
clabel(c,h,'manual')
```

Filled contours are an alternative to the empty contours used above. This function is used together with `colorbar`, which displays a legend for the plot. In addition, we can plot the locations (small circles) and z-values (contour labels) of the true data points (Fig. 7.6).

```
contourf(XI,YI,ZI,v), colorbar, hold on
plot(data(:,1),data(:,2),'ko')
text(data(:,1)+1,data(:,2),labels), hold off
```

A pseudocolor plot is generated using the function `pcolor`. Black contours are also added at the same levels as in the above example.

```
pcolor(XI,YI,ZI), shading flat, hold on
contour(XI,YI,ZI,v,'k'), hold off
```

The third dimension is added to the plot using the `mesh` command. We can also use this example to introduce the function `view(az,el)` to specify the direction of viewing, where `az` is the azimuth or horizontal rotation and `el` is the elevation (both in degrees). The values `az`=–37.5 and `el`=30 define the default view for all 3D plots,

```
mesh(XI,YI,ZI), view(-37.5,30)
```

whereas `az`=0 and `el`=90 is directly overhead and the default 2D view:

```
mesh(XI,YI,ZI), view(0,90)
```

The function `mesh` provides one of many methods available in MATLAB for 3D presentation, another commonly used function being `surf`. The figure may be rotated by selecting the *Rotate 3D* option on the *Edit Tools* menu. We also introduce the function `colormap`, which uses predefined color look-up tables for 3D graphics. Typing `help graph3d` lists a number of built-in colormaps, although colormaps can also be arbitrarily modified and generated by the user. As an example we use the colormap *hot*, which is a *black-red-yellow-*

www.it-ebooks.info

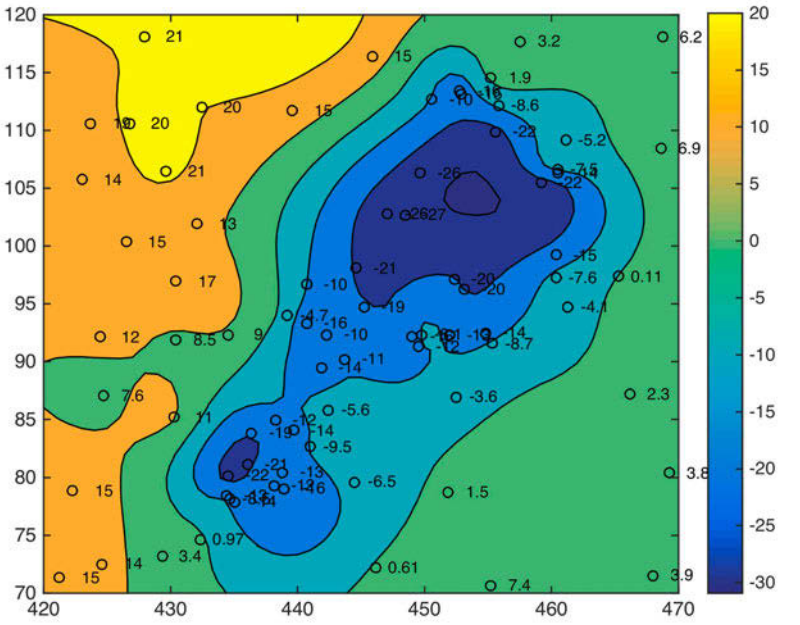


Fig. 7.6 Contour plot with the locations (small circles) and *z*-values (contour labels) of the true data points.

white colormap.

```
surf(XI,YI,ZI), colormap('hot'), colorbar
```

Using *Rotate 3D* only rotates the 3D plot, not the colorbar. The function `surfc` combines both a surface and a 2D contour plot in one graph.

```
surfc(XI,YI,ZI)
```

The function `surfl` can be used to illustrate an advanced application for 3D visualization, generating a 3D colored surface with interpolated shading and lighting. The axis labeling, ticks, and background can be turned off by typing `axis off`. In addition, black 3D contours can be added to the surface, as above. The grid resolution is increased prior to data plotting in order to obtain smooth surfaces (Fig. 7.7).

Interactive 7.7

```
[XI,YI] = meshgrid(420:0.25:470,70:0.25:120);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');

surf(XI,YI,ZI), shading interp, light, axis off, hold on
contour3(XI,YI,ZI,v,'k'), hold off
```

www.it-ebooks.info

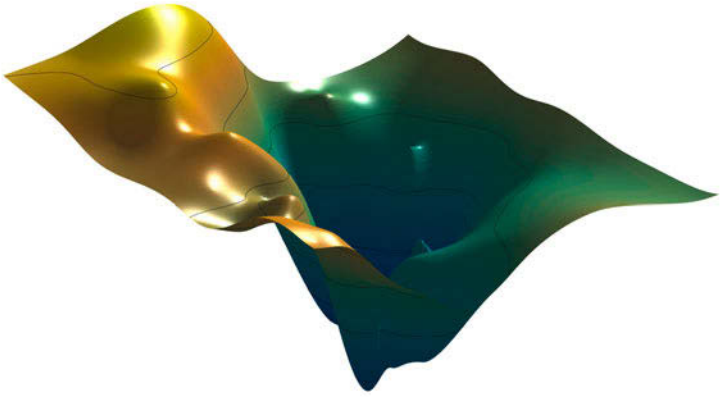


Fig. 7.7 Three-dimensional colored surface with interpolated shading and simulated lighting. The axis labeling, ticks and background are turned off. The plot also contains 3D contours, in black.

The biharmonic spline interpolation described in this section provides a solution to most gridding problems. It was therefore, for some time, the only gridding method that came with MATLAB. However, different applications in earth sciences require different methods of interpolation, although they all have their problems. The next section compares biharmonic spline interpolation with other gridding methods and summarizes their strengths and weaknesses.

7.8 Comparison of Methods and Potential Artifacts

The first example in this section illustrates the use of the *bilinear interpolation* technique for gridding irregular-spaced data. Bilinear interpolation is an extension of the one-dimensional technique of linear interpolation introduced in Section 5.5. In the two-dimensional case, linear interpolation is first performed in one direction, and then in the other direction. The bilinear method would appear to be one of the simplest interpolation techniques, which might intuitively not be expected to produce serious artifacts or distortions in the data. The opposite is true, however, as this method has a number of disadvantages and other methods are therefore preferred in many applications.

The sample data used in the previous section can again be loaded to study the effects of a bilinear interpolation.

www.it-ebooks.info

```
clear

data = load('normalfault.txt');
labels = num2str(data(:,3),2);
```

We now choose the option `linear` while using the function `griddata` to interpolate the data.

```
[XI,YI] = meshgrid(420:0.25:470,70:0.25:120);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'linear');
```

The results are plotted as contours. The plot also includes the locations of the control points.

```
v = -40 : 10 : 20;
contourf(XI,YI,ZI,v), colorbar, hold on
plot(data(:,1),data(:,2),'o'), hold off
```

The new surface is restricted to the area that contains control points: by default, bilinear interpolation does not extrapolate beyond this region. Furthermore, the contours are rather angular compared to the smooth shape of the contours from the biharmonic spline interpolation. The most important character of the bilinear gridding technique, however, is illustrated by a projection of the data in a vertical plane.

```
plot(XI,ZI,'k'), hold on
plot(data(:,1),data(:,3),'ro')
text(data(:,1)+1,data(:,3),labels)
title('Linear Interpolation'), hold off
```

This plot shows the projection of the estimated surface (vertical lines) and the labeled control points. The z-values at the grid points never exceed the z-values of the control points. As with the linear interpolation of time series (Section 5.5), bilinear interpolation causes significant smoothing of the data and a reduction in high-frequency variations.

Biharmonic spline interpolations are, in many ways, the other extreme. They are often used for extremely irregular-spaced and noisy data.

```
[XI,YI] = meshgrid(420:0.25:470,70:0.25:120);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');

v = -40 : 10 : 20;
contourf(XI,YI,ZI,v), colorbar, hold on
plot(data(:,1),data(:,2),'o'), hold off
```

The contours suggest an extremely smooth surface. This solution is very useful in many applications but the method also produces a number of artifacts. As we can see from the next plot, the estimated values at the grid

www.it-ebooks.info

points are often beyond the range of the measured z-values.

```
plot(XI,ZI,'k'), hold on
plot(data(:,1),data(:,3),'o')
text(data(:,1)+1,data(:,3),labels)
title('Biharmonic Spline Interpolation'), hold off
```

This can sometimes be appropriate and does not smooth the data in the way that bilinear gridding does. However, introducing very close control points with different z-values can cause serious artifacts. As an example we introduce one reference point with a z-value of +5 close to a reference point with a negative z-value of around –26.

```
data(79,:) = [450 105 5];
labels = num2str(data(:,3),2);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');

v = -40 : 10 : 20;
contourf(XI,YI,ZI,v), colorbar, hold on
plot(data(:,1),data(:,2),'ko')
text(data(:,1)+1,data(:,2),labels), hold off
```

The extreme gradient at the location (450,105) results in a paired *low* and *high* (Fig. 7.8). In such cases, it is recommended that one of the two control points be deleted and the z-value of the remaining control point be replaced by the arithmetic mean of both z-values.

Extrapolation beyond the area supported by control points is a common feature of spline interpolation (see also Section 5.5). Extreme local trends combined with large areas with no data often result in unrealistic estimates. To illustrate these edge effects we eliminate all control points in the upper-left corner.

```
[i,j] = find(data(:,1)<435 & data(:,2)>105);
data(i,:) = [];

labels = num2str(data(:,3),2);

plot(data(:,1),data(:,2),'ko'), hold on
text(data(:,1)+1,data(:,2),labels), hold off
```

We again use the biharmonic spline interpolation technique.

```
[XI,YI] = meshgrid(420:0.25:470,70:0.25:120);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');

v = -40 : 10 : 40;
contourf(XI,YI,ZI,v)
caxis([-40 40])
colorbar
hold on
```

www.it-ebooks.info

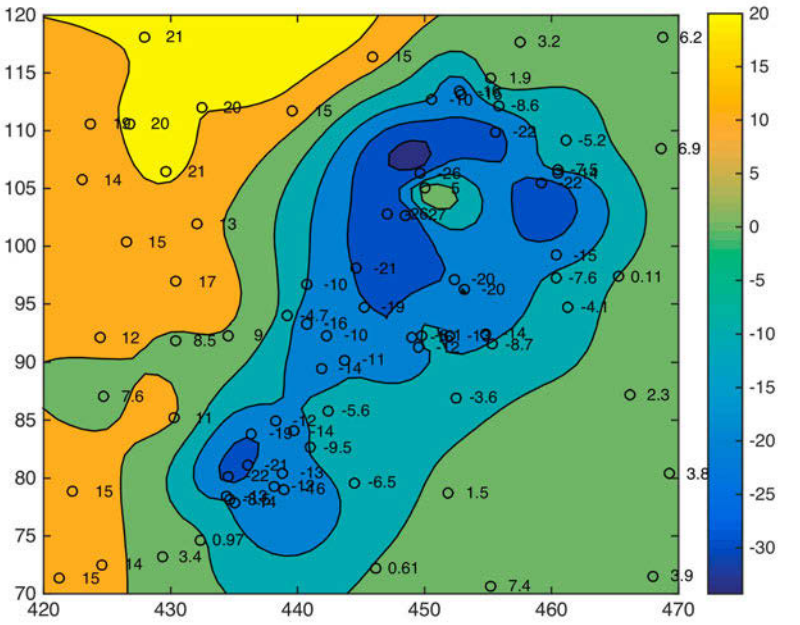


Fig. 7.8 Contour plot of a data set gridded using a biharmonic spline interpolation. At the location (450,105), very close control points with different z-values have been introduced. Interpolation causes a paired low and high, which is a common artefact in spline interpolation of noisy data.

```
plot(data(:,1),data(:,2),'ko')
text(data(:,1)+1,data(:,2),labels)
hold off
```

As can be seen from the plot, this method extrapolates the gradients beyond the area with control points, up to the edge of the map (Fig. 7.9). Such an effect is particular undesirable when gridding closed data, such as percentages, or data that have only positive values. In such cases, it is recommended that the estimated z-values be replaced by `NaN`. For instance, we delete the areas with z-values larger than 20, which are regarded as unrealistic values. The resulting plot now contains a sector with no data.

```
ZID = ZI;
ZID(find(ZID > 20)) = NaN;

contourf(XI,YI,ZID,v)
caxis([-40 40])
colorbar
hold on
plot(data(:,1),data(:,2),'ko')
```

www.it-ebooks.info

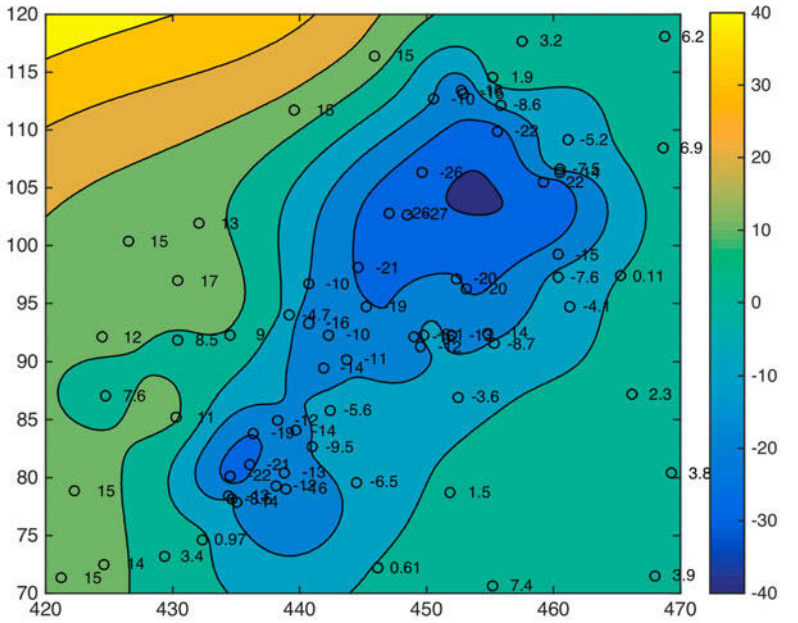


Fig. 7.9 Contour plot of a data set gridded using a biharmonic spline interpolation. No control points are available in the upper left corner. The spline interpolation then extrapolates beyond the area with control points using the gradients at the map edges, resulting in unrealistic z estimates at the grid points.

```
text(data(:,1)+1,data(:,2),labels)
hold off
```

Alternatively, we can eliminate a rectangular area with no data.

```
ZID = ZI;
ZID(131:201,1:71) = NaN;

contourf(XI,YI,ZID,v)
caxis([-40 40])
colorbar
hold on
plot(data(:,1),data(:,2),'ko')
text(data(:,1)+1,data(:,2),labels)
hold off
```

In some examples the area with no control points is simply concealed by placing a legend over this part of the map.

Other interpolation methods available with `griddata` are the cubic (`cubic`), natural neighbor (`natural`), and nearest neighbor (`nearest`) interpolation methods. The `cubic` option interpolates the data in two dimensions using

www.it-ebooks.info

a spline, i.e., a third-degree 3D polynomial is fitted to at least six adjacent control points, resulting in a surface (and its first derivative) that is continuous. The resulting surface is much smoother than those calculated using linear (`linear`), natural (`natural`), or nearest neighbor (`nearest`) techniques but not as smooth as that resulting from a biharmonic spline interpolation (`v4`). For this reason the typical artifacts of splines do not occur to the same extent. We can compare all of these methods in the next example. We first clear the workspace and reload the data from *normalfault.txt*.

```
clear

data = load('normalfault.txt');
data(79,:) = [450 105 5];
labels = num2str(data(:,3),2);
```

We then create titles for the results from the different interpolation methods.

```
titles = ['linear ';'nearest';'natural';'cubic  ';'biharmo'];
```

Since we store the titles in a single character array, we use spaces to expand the names of the methods so that they are all the same length. We again define the axis limits for gridding and contouring: `xlim`=[420 470] and `ylim`=[70 120]. The function `meshgrid` transforms the domain specified by vectors `x` and `y` into arrays `XI` and `YI`. The rows of the output array `XI` are copies of the vector `x` and the columns of the output array `YI` are copies of the vector `y`. We choose 1.0 as the grid interval.

```
x = 420:1:470; y = 70:1:120;
[XI,YI] = meshgrid(x,y);
```

We then use `griddata` with all available options and store the results in a three-dimensional array `ZI`.

```
ZI(:,:,1) = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'linear');
ZI(:,:,2) = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'nearest');
ZI(:,:,3) = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'natural');
ZI(:,:,4) = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'cubic');
ZI(:,:,5) = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');
```

We compare the results in five different graphics in separate figure windows, slightly offset on the computer display, using `figure`. The data are displayed as filled contours at values specified in a vector `v`.

```
v = -40 : 10 : 20;

for i = 1 : 5
    figure('Position',[50 i*100-50 500 300])
    contourf(XI,YI,ZI(:,:,i),v), colorbar, hold on
```

www.it-ebooks.info

```
    plot(data(:,1),data(:,2),'ko')
    text(data(:,1)+1,data(:,2),labels), hold off
    title(titles(i,:))
end
```

According to the MATLAB documentation, The MathWorks, Inc. has now decided to move the linear (`linear`), the natural (`natural`) and the nearest neighbor (`nearest`) techniques to the new function `scatteredInterpolant`, while the biharmonic (`v4`) and cubic spline (`cubic`) options remain in the `griddata` function. In fact the code of `griddata` invokes the new `scatteredInterpolant` function instead of the original codes of `linear`, `natural`, or `nearest`. The new function works in a very similar manner to `griddata`, as we can easily explore by typing

```
FLIN = scatteredInterpolant(data(:,1),data(:,2),data(:,3),...
            'linear','linear');
FNEA = scatteredInterpolant(data(:,1),data(:,2),data(:,3),...
            'nearest','nearest');
FNAT = scatteredInterpolant(data(:,1),data(:,2),data(:,3),...
            'natural','none');

ZI(:,:,6) = FLIN(XI,YI);
ZI(:,:,7) = FNEA(XI,YI);
ZI(:,:,8) = FNAT(XI,YI);

titles(6:8,:) = ['scatlin';'scatnea';'scatnat'];

for i = 6:8
    figure('Position',[350 (i-5)*100-50 500 300])
    contourf(XI,YI,ZI(:,:,i),v), colorbar, hold on
    plot(data(:,1),data(:,2),'ko')
    text(data(:,1)+1,data(:,2),labels), hold off
    title(titles(i,:))
end
```

Another very useful MATLAB gridding method is *splines with tension* by Wessel and Bercovici (1998), available for download from the author's webpage:

`http://www.soest.hawaii.edu/wessel/tspline/`

The *tsplines* use biharmonic splines in tension t, where the parameter t can vary between 0 and 1. A value of t=0 corresponds to a standard cubic spline interpolation. Increasing t reduces undesirable oscillations between data points, e.g., the paired *lows* and *highs* observed in one of the previous examples. The limiting situation $t \rightarrow 1$ corresponds to linear interpolation.

www.it-ebooks.info

7.9 Statistics of Point Distributions

This section is about the statistical distribution of objects within an area, which may help explain the relationship between these objects and properties of the area. For instance, the spatial concentration of hand-axes in an archaeological site may suggest that a larger population of hominins lived in that part of the area, the clustered occurrence of fossils may document environmental conditions that were favorable to those particular organisms, the alignment of volcanoes may often help in mapping tectonic structures concealed beneath the surface.

Various methods for the statistical analysis of point distributions are introduced below. We first consider a test for a uniform spatial distribution of objects, followed by a test for a random spatial distribution and finally, a simple test for a clustered distributions of objects.

Test for Uniform Distribution

In order to illustrate the test for a uniform distribution we first need to compute some synthetic data. The function `rand` computes uniformly-distributed pseudo-random numbers drawn from a uniform distribution within the interval [0,1]. We compute *xy* data using `rand` and multiply the data by ten to obtain data within the interval [0,10].

```
clear

rng(0)
data = 10 * rand(100,2);
```

We can use the χ^2–test introduced in Section 3.8 to test the hypothesis that the data have a uniform distribution. The *xy* data are now organized into 25 classes that are square subareas with dimensions of 2-by-2. This definition of the classes ignores the rule of thumb that the number of classes should be close to the square root of the number of observations (see Section 3.3). Our choice of classes, however, does not result in any empty classes, which should be avoided when applying the χ^2–test. Furthermore, 25 classes produce integer values for the expected number of observations that are easier to work with. We display the data as blue circles in a plot of *y* versus *x*. The rectangular areas are outlined with red lines (Fig. 7.10).

```
plot(data(:,1),data(:,2),'o')
hold on
x = 0:10; y = ones(size(x));
for i = 1:4, plot(x,2*i*y,'r-'), end
for i = 1:4, plot(2*i*y,x,'r-'), end
```

www.it-ebooks.info

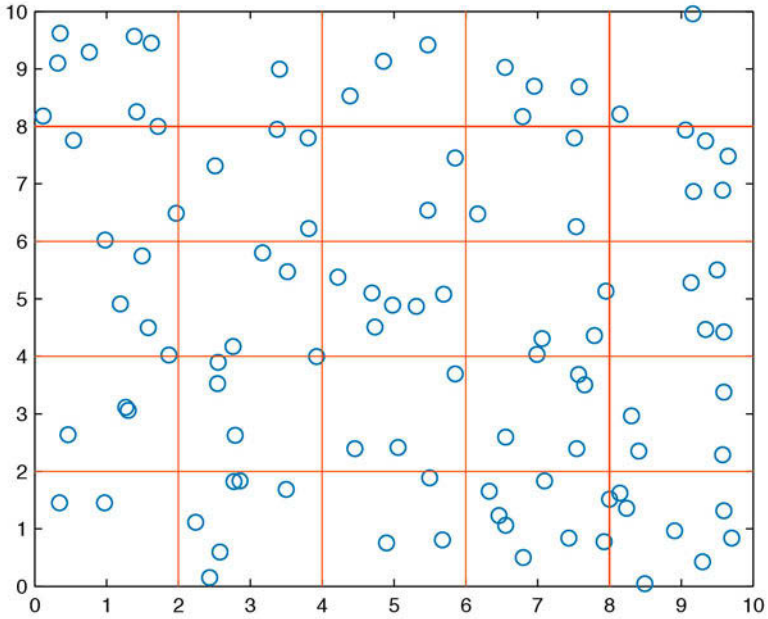


Fig. 7.10 Two-dimensional plot of a point distribution. The distribution of objects in the field is tested for uniform distribution using the χ^2–test. The *xy* data are organized into 25 classes that are subareas with dimensions of 2-by-2.

```
hold off
```

A three-dimensional version of a histogram `hist3` is used to display the spatial data, organized in classes (Fig. 7.11).

```
hist3(data,[5 5]), view(30,70)
```

As with the equivalent two-dimensional function, the function `hist3` can be used to compute the frequency distribution `n_obs` of the data.

```
n_obs = hist3(data,[5 5]);
n_obs = n_obs(:);
```

For a uniform distribution, the theoretical frequencies for the different classes are identical. The expected number of objects in each square area is the size of the total area 10·10=100 divided by the 25 subareas or classes, which comes to four. To compare the theoretical frequency distribution with the actual spatial distribution of objects, we generate a 5-by-5 array with an identical number of objects.

```
n_exp = 4 * ones(25,1);
```

www.it-ebooks.info

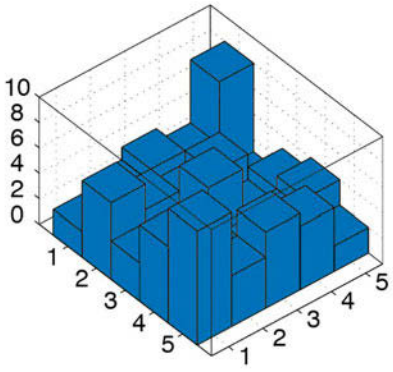


Fig. 7.11 Three-dimensional histogram displaying the numbers of objects for each subarea. The histogram was created using `hist3`.

The χ^2-test explores the squared differences between the observed and expected frequencies (Section 3.9). The quantity χ^2 is defined as the sum of the squared differences divided by the expected frequencies.

```
chi2_data = sum((n_obs - n_exp).^2 ./n_exp)

chi2 =
    22.5000
```

The critical χ^2 can be calculated using `chi2inv`. The χ^2-test requires the degrees of freedom Φ. In our example we test the hypothesis that the data are uniformly distributed, i.e., we estimate only one parameter (Section 3.5). The number of degrees of freedom is therefore Φ=25–(1+1)=23. We test the hypothesis at a p=95% significance level. The function `chi2inv` computes the inverse of the χ^2 CDF with parameters specified by Φ for the corresponding probabilities in p.

```
chi2_theo = chi2inv(0.95,25-1-1)

ans =
    35.1725
```

Since the critical χ^2 of 35.1725 is well above the measured χ^2 of 22.5000, we cannot reject the null hypothesis and therefore conclude that our data follow a uniform distribution.

www.it-ebooks.info

Test for Random Distribution

The following example illustrates the test for random distribution of objects within an area. We use the uniformly-distributed data generated in the previous example and display the point distribution.

```
clear

rng(5)
data = 10 * rand(100,2);
plot(data(:,1),data(:,2),'o')
hold on
x = 0:10; y = ones(size(x));
for i = 1:9, plot(x,i*y,'r-'), end
for i = 1:9, plot(i*y,x,'r-'), end
hold off
```

We then generate the three-dimensional histogram and use the function `hist3` to count the objects per class. In contrast to the previous test, we now count the subareas containing a certain number of observations. The number of subareas is larger than would normally be used for the previous test. In our example we use 49 subareas or classes.

```
hist3(data,[7 7])
view(30,70)

counts = hist3(data,[7 7]);
counts = counts(:);
```

The frequency distribution of those subareas that contain a specific number of objects follows a Poisson distribution (Section 3.5) if the objects are randomly distributed. First, we compute a frequency distribution of the subareas containing `N` objects. In our example we count the subareas with 0, …, 5 objects. We also display the histogram of the frequency distribution as a two-dimensional histogram using `histogram`, after having first calculated the bin edges `E` from the bin centers `N` (Section 3.3) (Fig. 7.12).

```
N = 0 : 5;
E =  -0.5 : 1 : 5.5;

h = histogram(counts,E);
title('Histogram')
xlabel('Number of observations N')
ylabel('Subareas with N observations')

v = h.BinWidth * 0.5 + h.BinEdges(1:end-1);
n_obs = h.Values;
```

www.it-ebooks.info

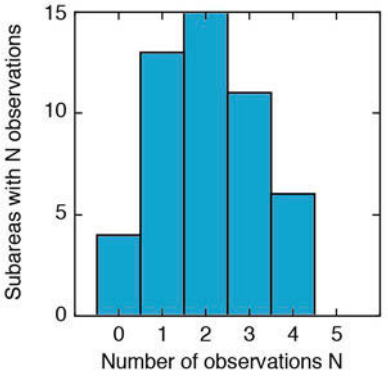


Fig. 7.12 Frequency distribution of subareas with *N* objects. In our example the subareas with 0, …, 5 objects are counted. The histogram of the frequency distribution is displayed as a two-dimensional histogram, using `histogram`.

Here, the midpoints of the histogram intervals `v` correspond to the `N`=0, …, 5 objects contained in the subareas. The expected number of subareas E_j with a certain number of objects j can be computed using

$$E_j = Te^{-n/T}\frac{(n/T)^j}{j!}$$

where n is the total number of objects and T is the number of subareas. For j=0, $j!$ is taken to be 1. We compute the expected number of subareas, i.e., the theoretical frequency distribution `n_exp`, using the equation shown above,

```
for i = 1 : 6
    n_exp(i) = 49*exp(-100/49)*(100/49)^N(i)/factorial(N(i));
end
n_exp = sum(n_obs)*n_exp/sum(n_exp);
```

and display both the empirical and theoretical frequency distributions in a single plot.

```
h1 = bar(v,n_obs);
hold on
h2 = bar(v,n_exp);
hold off

set(h1,'FaceColor','none','EdgeColor','r')
set(h2,'FaceColor','none','EdgeColor','b')
```

The χ^2-test is again used to compare the empirical and theoretical distributions. The test is performed at a p=95% significance level. Since the

www.it-ebooks.info

Poisson distribution is defined by only one parameter (Section 3.4), the number of degrees of freedom is Φ=6–(1+1)=4. The measured χ^2 of

```
chi2 = sum((n_obs - n_exp).^2 ./n_exp)

chi2 =
    3.7615
```

is well below the critical χ^2, which is

```
chi2inv(0.95,6-1-1)

ans =
    9.4877
```

We therefore cannot reject the null hypothesis and conclude that our data follow a Poisson distribution and the point distribution is random.

Test for Clustering

Point distributions in geosciences are often clustered. We use a *nearest-neighbor criterion* to test a spatial distribution for clustering. Davis (2002) has published an excellent summary of the nearest-neighbor analysis, summarizing the work of a number of other authors. Swan and Sandilands (1996) presented a simplified description of this analysis. The test for clustering computes the distances d_i separating all possible pairs of nearest points in the field. The *observed mean nearest-neighbor distance* is

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$$

where n is the total number of points or objects in the field. The arithmetic mean of all distances between possible pairs is related to the area covered by the map. This relationship is expressed by the *expected mean nearest-neighbor distance*, which is

$$\bar{\delta} = \frac{1}{2}\sqrt{A/n}$$

where A is the area covered by the map. Small values for this ratio then suggest significant clustering, whereas large values indicate regularity or uniformity. The test uses a Z statistic (Section 3.5), which is

www.it-ebooks.info

$$Z = \frac{\bar{d} - \bar{\delta}}{s_e}$$

where s_e is the standard error of the mean nearest-neighbor distance, which is defined as

$$s_e = \frac{0.26136}{\sqrt{n^2 / A}}$$

The null hypothesis *randomness* is tested against two alternative hypotheses, *clustering* and *uniformity or regularity*. The Z statistic has critical values of 1.96 and –1.96 at a significance level of 95%. If $-1.96<Z<+1.96$, we cannot reject the null hypothesis that the data are randomly distributed. If $Z<-1.96$, we reject the null hypothesis and accept the first alternative hypothesis of clustering. If $Z>+1.96$, we also reject the null hypothesis, but accept the second alternative hypothesis of uniformity or regularity.

As an example we again use the synthetic data analyzed in the previous examples.

```
clear

rng(5)
data = 10 * rand(100,2);
plot(data(:,1),data(:,2),'o')
```

We first compute the pairwise Euclidian distance between all pairs of observations using the function `pdist` (Section 9.5). The resulting distance matrix `distances` is then converted into a symmetric, square format, so that `distmatrix(i,j)` denotes the distance between `i` and `j` objects in the original data.

```
distances = pdist(data,'Euclidean');
distmatrix = squareform(distances);
```

The following `for` loop finds the nearest neighbors, stores the nearest-neighbor distances and computes the mean distance.

```
for i = 1 : 100
    distmatrix(i,i) = NaN;
    k = find(distmatrix(i,:) == min(distmatrix(i,:)));
    nearest(i) = distmatrix(i,k(1));
end
observednearest = mean(nearest)
```

www.it-ebooks.info

```
observednearest =
    0.5078
```

In our example the mean nearest distance `observednearest` comes to 0.5078. We next calculate the area of the map. The expected mean nearest-neighbor distance is half the square root of the map area divided by the number of observations.

```
maparea = (max(data(:,1)-min(data(:,1)))) ...
         *(max(data(:,2)-min(data(:,2))));
expectednearest = 0.5 * sqrt(maparea/length(data))

expectednearest =
    0.4875
```

In our example the expected mean nearest-neighbor distance `expectednearest` is 0.4875. Finally, we compute the standard error of the mean nearest-neighbor distance `se`

```
se = 0.26136/sqrt(length(data).^2/maparea)

se =
    0.0255
```

and the test statistic `z`.

```
Z = (observednearest - expectednearest)/se

Z =
    0.7954
```

Since –1.96<z<+1.96, we cannot reject the null hypothesis and conclude that the data are randomly distributed, but not clustered.

7.10 Analysis of Digital Elevation Models (by R. Gebbers)

Digital elevation models (DEMs) and their derivatives (e.g., slope and aspect) can indicate surface processes such as lateral water flow, solar irradiation, or erosion. The simplest derivatives of a DEM are the slope and the aspect. The *slope* (or *gradient*) is a measure of the steepness, the incline or the grade of a surface measured in either percentages or degrees. The *aspect* (or *exposure*) refers to the direction in which a slope faces.

We use the SRTM data set introduced in Section 7.5 to illustrate the analysis of a digital elevation model for slope, aspect and other derivatives. The data are loaded by

```
clear
```

www.it-ebooks.info

```
fid = fopen('S01E036.hgt','r');
SRTM = fread(fid,[1201,inf],'int16','b');
fclose(fid);

SRTM = SRTM';
SRTM = flipud(SRTM);
SRTM(find(SRTM==-32768)) = NaN;
```

These data are elevation values in meters above sea level, sampled on a 3 arc second or 90 meter grid. The SRTM data contain small-scale spatial disturbances and noise that could cause problems when computing a drainage pattern. We therefore filter the data with a two-dimensional moving-average filter, using the function `filter2`. The filter calculates a spatial running mean of 3-by-3 elements. We use only the subset `SRTM(400:600,650:850)` of the original data set, in order to reduce computation time. We also remove the data at the edges of the DEM to eliminate filter artifacts.

```
F = 1/9 * ones(3,3);
SRTM = filter2(F, SRTM(750:850,700:800));
SRTM = SRTM(2:99,2:99);
```

The DEM is displayed as a pseudocolor plot using `pcolor` and the colormap `demcmap` included in the Mapping Toolbox. The function `demcmap` creates and assigns a colormap appropriate for elevation data since it relates land and sea colors to hypsometry and bathymetry.

```
h = pcolor(SRTM);
demcmap(SRTM), colorbar
set(h,'LineStyle','none')
axis equal
title('Elevation [m]')
[r c] = size(SRTM);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

The DEM indicates a horseshoe-shaped mountain range surrounding a valley that slopes down towards the south-east (Fig. 7.15 a).

The SRTM subset is now analyzed for slope and aspect. When we are working with DEMs on a regular grid, slope and aspect can be estimated using centered finite differences in a local 3-by-3 neighborhood. Figure 7.13 shows a local neighborhood using the MATLAB cell indexing convention. For calculating slope and aspect, we need two finite differences in the DEM elements z, in x and y directions:

$$z_x = \frac{z_{r,c-1} - z_{r,c+1}}{2h}$$

www.it-ebooks.info

Z(1)	Z(4)	Z(7)
Z(2)	Z(5)	Z(8)
Z(3)	Z(6)	Z(9)

Fig. 7.13 Local neighborhood showing the MATLAB cell number convention.

and

$$z_y = \frac{z_{r-1,c} - z_{r+1,c}}{2h}$$

where h is the cell size, which has the same units as the elevation. Using the finite differences, the slope is then calculated by

$$SLP_{DF} = \sqrt{z_x{}^2 + z_y{}^2}$$

Other primary relief attributes such as the *aspect*, the *plan*, the *profile* and the *tangential curvature* can be derived in a similar way using finite differences (Wilson and Galant 2000). The function `gradientm` in the Mapping Toolbox calculates the slope and aspect of a data grid `z` in degrees above the horizontal and degrees clockwise from north. The function `gradientm(z,refvec)` requires a three-element reference vector `refvec`. The reference vector contains the number of cells per degree as well as the latitude and longitude of the upper-left (northwest) element of the data array. Since the SRTM digital elevation model is sampled on a 3 arc second grid, 60·60/3=1200 elements of the DEM correspond to one degree of longitude or latitude. For simplicity we ignore the actual coordinates of the SRTM subset in this example and use the indices of the DEM elements instead.

```
refvec = [1200 0 0];
[asp, slp] = gradientm(SRTM, refvec);
```

We display a pseudocolor map of the DEM slope in degrees (Fig 7.15 b).

```
h = pcolor(slp);
colormap(jet), colorbar
```

www.it-ebooks.info

```
set(h,'LineStyle','none')
axis equal
title('Slope [°]')
[r c] = size(slp);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

Flat areas are common on the summits and on the valley floors. The south-eastern and south-south-western sectors are also relatively flat. The steepest slopes are concentrated in the center of the area and in the south-western sector. Next, a pseudocolor map of the aspect is generated (Fig. 7.15 c).

```
h = pcolor(asp);
colormap(hsv), colorbar
set(h,'LineStyle','none')
axis equal
title('Aspect')
[r c] = size(asp);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

This plot displays the aspect in degrees, clockwise from north. For instance, mountain slopes facing north are displayed in red and east-facing slopes in green.

The aspect changes abruptly along the ridges of the mountain ranges where neighboring drainage basins are separated by *watersheds*. The Image Processing Toolbox includes the function `watershed` to detect these drainage divides and to ascribe numerical labels to each catchment area, starting with 1.

```
watersh = watershed(SRTM);
```

The catchment areas are displayed in a pseudocolor plot in which each area is assigned a color from the color table `hsv` (Fig. 7.15 d), according to its numerical label.

```
h = pcolor(watersh);
colormap(hsv), colorbar
set(h,'LineStyle','none')
axis equal
title('Watershed')
[r c] = size(watersh);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

The watersheds are represented by a series of red pixels. The largest catchment area corresponds to the medium blue region in the center of the map. To the north-west, this large catchment area appears to be bordered by

www.it-ebooks.info

three catchments areas (represented by green colors) with no outlets. As in this example, `watershed` often generates unrealistic results since watershed algorithms are sensitive to local minima that act as spurious sinks. We can detect such sinks in the SRTM data using the function `imregionalmin`. The output of this function is a binary image in which pixels containing local maxima are marked by a value of one and all other pixels are marked by a zero.

```
sinks = 1*imregionalmin(SRTM);

h = pcolor(sinks);
colormap(gray)
set(h,'LineStyle','none')
axis equal
title('Sinks')
[r c] = size(sinks);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

The pseudocolor plot of the binary image shows twelve local sinks, represented by white pixels, that are potential locations for spurious areas of internal drainage and should be borne in mind during any subsequent computation of hydrological characteristics from the DEM.

Flow accumulation, also called *specific catchment area* or *upslope contributing area*, is defined as the number of cells (or area) contributing runoff to a particular cell (Fig. 7.14). In contrast to the local parameters of slope and aspect, flow accumulation can only be determined from the global neighborhood. The principal operation is to add cell inflows from topographically higher neighboring cells, starting from the specified cell and working up to the watersheds. Before adding together the outflows from each cell we need to determine the gradient of each individual cell towards each neighboring cell, indexed by `N`. The array `N` contains indices for the eight adjacent cells, according to the MATLAB convention, as shown in Figure 7.13. We make use of the `circshift` function to access the neighboring cells. For a two-dimensional matrix `Z`, the function `circshift(Z,[r c])` circularly shifts the values in the matrix `Z` by `r` rows and `c` columns. For example `circshift(Z,[1 1])` will circularly shift `Z` one row down and one column to the right. The individual gradients are calculated by

$$grad = \frac{z_{r+y,c+x} - z_{r,c}}{h}$$

for the eastern, southern, western, and northern neighbors (known as the *rook's case*) and by

www.it-ebooks.info

$$grad = \frac{z_{r+y,c+x} - z_{r,c}}{\sqrt{2h}}$$

for the diagonal neighbors (known as the *bishop's case*). In these formulae, h is the cell size, $z_{r,c}$ is the elevation of the central cell, and $z_{r+y,c+x}$ is the elevation of the neighboring cells. The cell indices x and y are obtained from the matrix `N`. The gradients are stored in a three-dimensional matrix `grads`, where `grads(:,:,1)` contains the gradient towards the neighboring cells to the east, `grads(:,:,2)` contains the gradient towards the neighboring cells to the south-east, and so on. Negative gradients indicate outflow from the central cell towards the relevant neighboring cell. To obtain the surface flow between cells, gradients are transformed using the inverse tangent of `grads` divided by 0.5π.

```
N = [0 -1;-1 -1;-1 0;+1 -1;0 +1;+1 +1;+1 0;-1 +1];
[a b] = size(SRTM);
grads = zeros(a,b,8);
for c = 2 : 2 : 8
   grads(:,:,c) = (circshift(SRTM,[N(c,1) N(c,2)]) ...
      -SRTM)/sqrt(2*90);
end
for c = 1 : 2 : 7
   grads(:,:,c) = (circshift(SRTM,[N(c,1) N(c,2)]) ...
      -SRTM)/90;
end
grads = atan(grads)/pi*2;
```

Since a central cell can have several downslope neighbors, water can flow in several directions. This phenomenon is called *divergent flow*. Early flow accumulation algorithms were based on the single-flow-direction method (known as the D8 method, Fig. 7.14), which allows flow to only one of the cell's eight neighboring cells. This method cannot, however, model divergences in ridge areas and tends to produce parallel flow lines in some situations. In our example we illustrate the use of a multiple-flow-direction method, which allows flow from a central cell to multiple neighboring cells. The proportion of the total outflow that is assigned to a neighboring cell is dependent on the gradient between the central cell and that particular neighboring cell. Even though multiple-flow methods produce more realistic results in most situations, they tend to result in dispersion in valleys, where the flow should be more concentrated. A weighting factor w is therefore introduced, which controls the relationship between the outflows.

$$flow_i = \frac{grad_i^w}{\sum_{i=1}^{8} grad_i^w} \quad \text{for} \quad grad_i^w < 0$$

www.it-ebooks.info

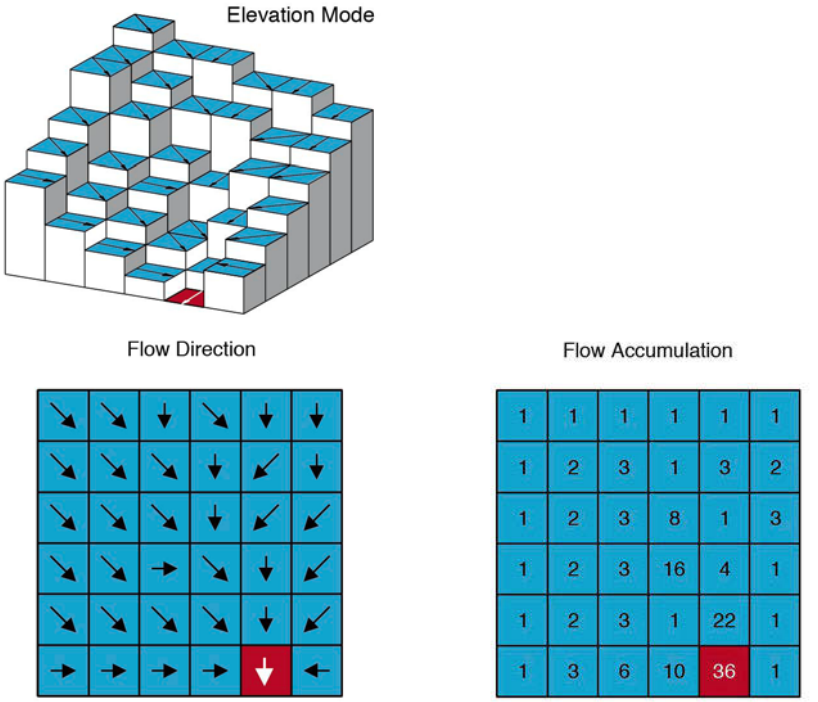


Fig. 7.14 Schematic of calculation of flow accumulation by the D8 method.

A recommended value for w is 1.1; higher values would concentrate the flow in the direction of the steepest slope, while w=0 would result in extreme dispersion. In the following sequence of commands, we first select those gradients that are less than zero and then multiply the gradients by the weighting.

```
w = 1.1;
flow = (grads.*(-1*grads<0)).^w;
```

We then sum up the upslope gradients along the third dimension of the `flow` matrix. Replacing all upslope gradient values of 0 by a value of 1 avoids the problems created by trying to divide by zero.

```
upssum = sum(flow,3);
upssum(upssum==0) = 1;
```

We divide the flows by `upssum` to obtain fractional weights that add up to a total of one. This is achieved separately for each layer of the 3D `flow` matrix using a `for` loop:

www.it-ebooks.info

```
for i = 1:8
   flow(:,:,i) = flow(:,:,i).*(flow(:,:,i)>0)./upssum;
end
```

The 2D matrix `inflowsum` will store the intermediate inflow totals for each iteration. These intermediate totals are then summed to reach a figure for the total accumulated flow `flowac` at the end of each iteration. The initial values for `inflowsum` and `flowac` are obtained through `upssum`.

```
inflowsum = upssum;
flowac = upssum;
```

Another 3D matrix `inflow` is now needed, in which to store the total intermediate inflow from all neighbors:

```
inflow = grads*0;
```

Flow accumulation is terminated when there is no inflow or, translating this into MATLAB code, we use a conditional `while` loop that terminates if `sum(inflowsum(:))==0`. The number of non-zero entries in `inflowsum` will decrease during each loop. This is achieved by alternately updating `inflow` and `inflowsum`. Here, `inflowsum` is updated with the intermediate `inflow` of the neighboring cells weighted by `flow`, under the condition that all of the neighboring cells are contributing cells, i.e., where `grads` are positive. Where not all neighboring cells are contributing cells, the intermediate `inflowsum` is reduced, as also is `inflow`. The flow accumulation `flowac` increases through consecutive summations of the intermediate `inflowsum`.

```
while sum(inflowsum(:))>0
   for i = 1:8
      inflow(:,:,i) = circshift(inflowsum,[N(i,1) N(i,2)]);
   end
   inflowsum = sum(inflow.*flow.*grads>0,3);
   flowac = flowac + inflowsum;
end
```

We display the result as a pseudocolor map with log-scaled values (Fig. 7.15 e).

```
h = pcolor(log(1+flowac));
colormap(flipud(jet)), colorbar
set(h,'LineStyle','none')
axis equal
title('Flow accumulation')
[r c] = size(flowac);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

www.it-ebooks.info

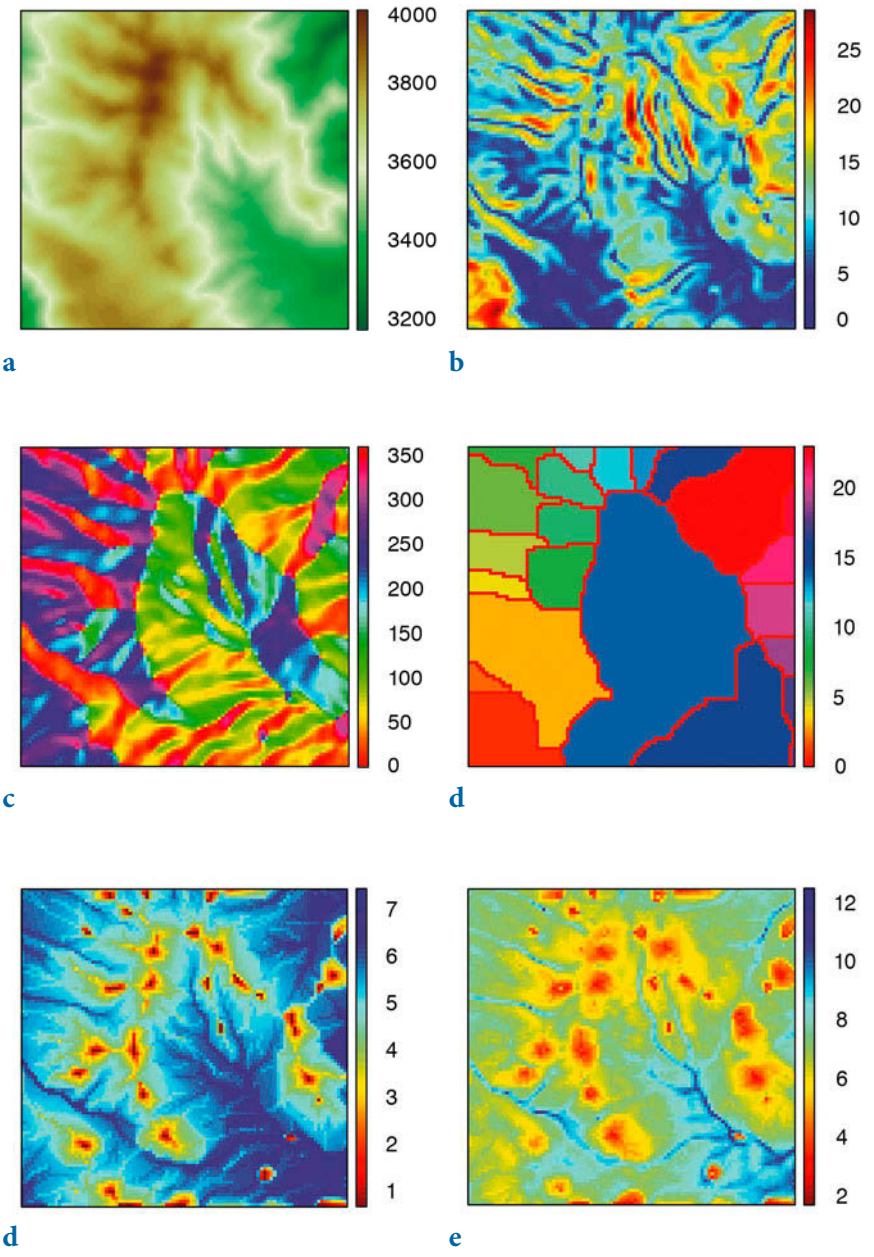


Fig. 7.15 Display of a subset of the SRTM data set used in Section 7.5 and primary and secondary attributes of the digital elevation model; **a** elevation, **b** slope, **c** aspect, **d** watershed, **e** flow accumulation, and **f** wetness index (data from Farr et al. 2000, 2008).

www.it-ebooks.info

The plot displays areas with high flow accumulation in shades of blue and areas with low flow accumulation, usually corresponding to ridges, in shades of red. We used a logarithmic scale for mapping the flow accumulation in order to obtain a better representation of the results. The simplified algorithm introduced here for calculating flow accumulation can be used to analyze sloping terrains in DEMs. In flat terrains, where the slope approaches zero, no flow direction can be generated by our algorithm and thus flow accumulation stops. Such situations require more sophisticated algorithms to perform analyses on completely flat terrain. These more advanced algorithms also include sink-filling routines to avoid spurious sinks that interrupt flow accumulation. Small depressions can be filled by smoothing, as we did at the beginning of this section.

The first part of this section was about primary relief attributes. Secondary attributes of a DEM are functions of two or more primary attributes. Examples of secondary attributes are the wetness index and the stream power index. The *wetness index* for a cell is the log of the ratio between the area of the catchment for that particular cell and the tangent of its slope:

$$weti = \log\left(\frac{1 + flowac}{\tan(slp)}\right)$$

The term 1+*flowac* avoids the problems associated with calculating the logarithm of zero when `flowac=0`. The wetness index is used to predict the soil water content (*saturation*) resulting from lateral water movement. The potential for waterlogging is usually highest in the lower parts of catchments, where the slopes are more gentle. Flat areas with a large upslope area have a high wetness index compared to steep areas with small catchments. The wetness index `weti` is computed and displayed by

```
weti = log((1+flowac)./tand(slp));

h = pcolor(weti);
colormap(flipud(jet)), colorbar
set(h,'LineStyle','none')
axis equal
title('Wetness index')
[r c] = size(weti);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

In this plot, blue colors indicate high values for the wetness index while red colors represent low values (Fig. 7.15 f). In our example soils in the south-east are the most likely to have a high water content due to the runoff from

www.it-ebooks.info

the large central valley and the flatness of the terrain.

The *stream power index* is another important secondary relief attribute that is frequently used in hillslope hydrology, geomorphology, soil science, and related disciplines. As a measure of stream power it provides an indication of the potential for sediment transport and erosion by water. It is defined as the product of the catchment area for a specific cell and the tangent of the slope of that cell:

$$spi = flowac \cdot \tan(slp)$$

The potential for erosion is high when large quantities of water (calculated by flow accumulation) are fast flowing due to an extreme slope. The following series of commands compute and display the stream power index:

```
spi = flowac.*tand(slp);

h = pcolor(log(1+spi));
colormap(jet), colorbar
set(h,'LineStyle','none')
axis equal
title('Stream power index')
[r c] = size(spi);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

The wetness and stream power indices are particularly useful in high resolution terrain analysis, i.e., digital elevation models sampled at intervals of less than 30 meters. In our terrain analysis example we have calculated `weti` and `spi` from a medium resolution DEM, and must expect a degree of scale dependency in these attributes.

This section has illustrated the use of basic tools for analyzing digital elevation models. A more detailed introduction to digital terrain modeling is given in the book by Wilson & Galant (2002). Furthermore, the article by Freeman (1991) provides a comprehensive summary of digital terrain analysis, including an introduction to the use of advanced algorithms for flow accumulation.

7.11 Geostatistics and Kriging (by R. Gebbers)

Geostatistics describes the autocorrelation of one or more variables in 1D, 2D, or 3D space, or even in 4D space-time, in order to make predictions for unobserved locations, to obtain information on the accuracy of the predictions, and to reproduce spatial variability and uncertainty. The shape, range, and direction of the spatial autocorrelation are described by

www.it-ebooks.info

a *variogram*, which is the main tool in linear geostatistics. The origins of geostatistics can be traced back to the early 1950s when the South African mining engineer Daniel G. Krige first published an interpolation method based on the spatial dependency of samples. In the 60s and 70s, the French mathematician George Matheron developed the *theory of regionalized variables*, which provides the theoretical foundations for Krige's more practical methods. This theory forms the basis of several procedures for the analysis and estimation of spatially dependent variables, which Matheron called *geostatistics*. Matheron also coined the term *kriging* for spatial interpolation by geostatistical methods.

Theorical Background

A basic assumption in geostatistics is that a spatiotemporal process is composed of both deterministic and stochastic components (Fig. 7.16). The deterministic components can be global and local trends (sometimes called *drifts*). The stochastic component comprises a purely random part and an autocorrelated part. The autocorrelated component suggests that on average, closer observations are more similar to each other than more widely separated observations. This behavior is described by the variogram in which squared differences between observations are plotted against their separation distances. Krige's fundamental idea was to use the variogram for interpolation, as a means of determining the amount of influence that neighboring observations have when predicting values for unobserved locations. Basic linear geostatistics includes two main procedures: variography for modeling the variogram, and kriging for interpolation.

Preceding Analysis

Because linear geostatistics as presented herein is a parametric method, the underlying assumptions need to be checked by a preceding analysis. As with other parametric methods, linear geostatistics is sensitive to outliers and deviations from a normal distribution. We first open the data file *geost_dat.mat* containing *xyz* data triplets, and then plot the sampling locations. This allows us to check the point distribution and detect any major errors in the data coordinates, *x* and *y*.

```
clear

load geost_dat.mat

plot(x,y,'.')
```

www.it-ebooks.info

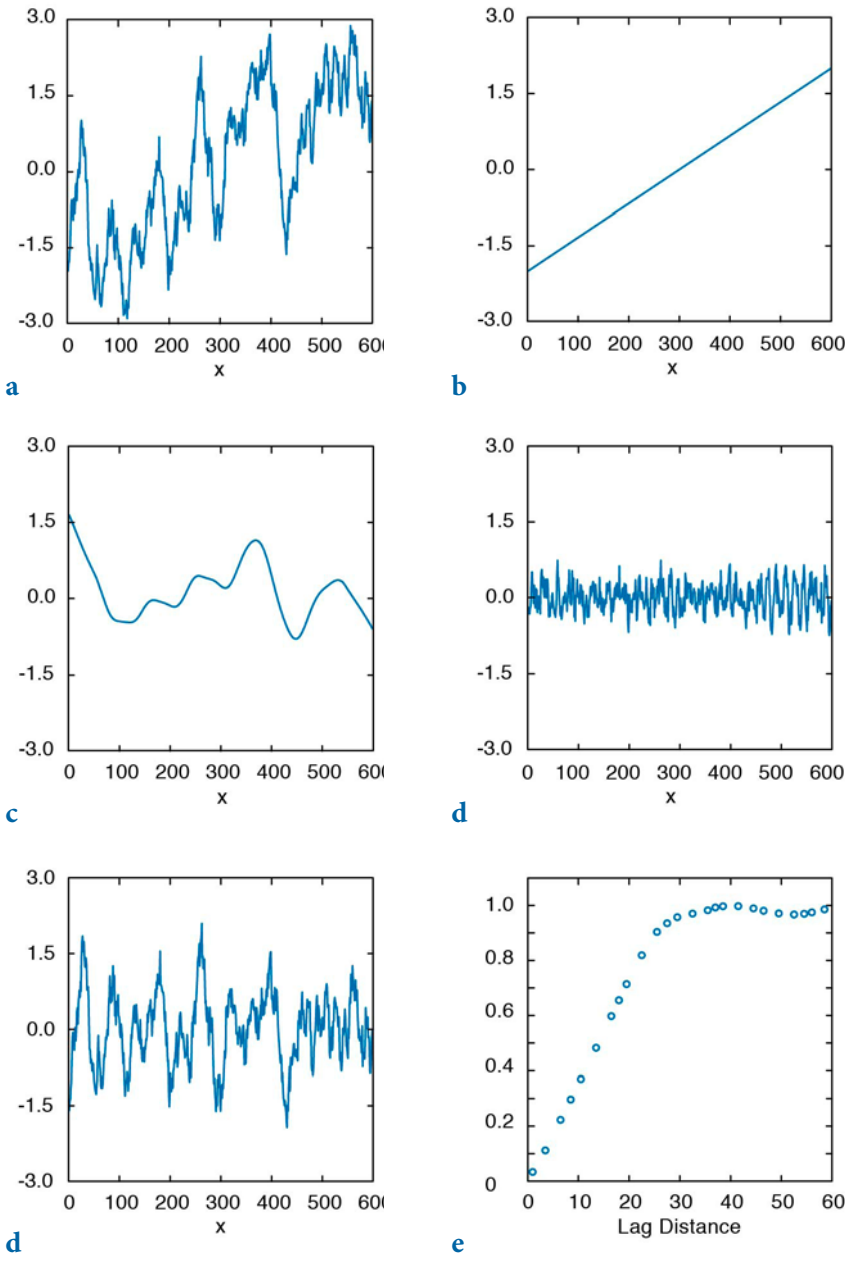


Fig. 7.16 Components of a spatiotemporal process and the variogram; **a** spatiotemporal process, **b** global trend component, **c** local trend component, **d** random component, **e** autocorrelation component, and **f** variogram. The variogram should only be derived from the autocorrelated component.

www.it-ebooks.info

The range of the observations z can be checked by

```
min(z)

ans =
    3.7199

max(z)

ans =
    7.8460
```

For linear geostatistics, the observations z should be Gaussian distributed. This is usually only tested by visual inspection of the histogram because statistical tests are often too sensitive if the number of samples exceeds about 100. One can also calculate the skewness and kurtosis of the data.

```
histogram(z)

skewness(z)

ans =
    0.2568

kurtosis(z)

ans =
    2.5220
```

A flat-topped or multiple-peaked distribution suggests that there is more than one population present in the data set. If these populations can be related to particular areas they should be treated separately. Another reason for multiple peaks can be preferential sampling of areas with high and/or low values. This usually happens as a result of some *a priori* knowledge and is known as a cluster effect. Dealing with a cluster effect is described in Deutsch and Journel (1998) and in Isaaks and Srivastava (1998).

Most problems arise from positive skewness, i.e., if the distribution has a long tail to the right. According to Webster and Oliver (2001), one should consider root transformation if the skewness is between 0.5 and 1, and logarithmic transformation if the skewness exceeds 1. A general transformation formula is:

$$z^* = \begin{cases} \dfrac{(z+m)^k - 1}{k} & \text{for } k \neq 0 \\ \log(z+m) & \text{for } k = 0 \end{cases}$$

www.it-ebooks.info

for $\min(z)+m>0$. This is known as the called Box-Cox transform, with the special case $k=0$ when a logarithm transformation is used. In the logarithm transformation, m should be added if z is zero or negative. Interpolation results of power-transformed values can be back-transformed directly after kriging. The back-transformation of log-transformed values is slightly more complicated and will be explained later. The procedure is known as *lognormal kriging*. It can be important because lognormal distributions are not uncommon in geology.

Variography with the Classical Variogram

A variogram describes the spatial dependency of referenced observations in a unidimensional or multidimensional space. Since the true variogram of the spatial process is usually unkown, it has to be estimated from observations. This procedure is called variography. Variography starts by calculating the *experimental variogram* from the raw data. In the next step, the experimental variogram is summarized by the *variogram estimator*. The variography then concludes by fitting a variogram model to the variogram estimator. The experimental variogram is calculated as the differences between pairs of observed values and is dependent on the *separation vector h* (Fig. 7.17). The classical experimental variogram is defined by the *semivariance*,

$$\gamma(h) = 0.5 \cdot (z_x - z_{x+h})^2$$

where z_x is the observed value at location x and z_{x+h} is the observed value at another point at a distance h from x. The length of the separation vector h is called the *lag distance*, or simply the *lag*. The correct term for $\gamma(h)$ is the *semivariogram* (or *semivariance*), where *semi* refers to the fact that it is half of the variance in the differences between z_x and z_{x+h}. It is, nevertheless, the variance per point when points are considered in pairs (Webster and Oliver 2001). Conventionally, $\gamma(h)$ is termed a *variogram* instead of a semivariogram, a convention that we shall follow for the rest of this section. To calculate the experimental variogram we first need to group pairs of observations. This is achieved by typing

```
[X1,X2] = meshgrid(x);
[Y1,Y2] = meshgrid(y);
[Z1,Z2] = meshgrid(z);
```

The matrix of separation distances `D` between the observation points is

```
D = sqrt((X1 - X2).^2 + (Y1 - Y2).^2);
```

www.it-ebooks.info

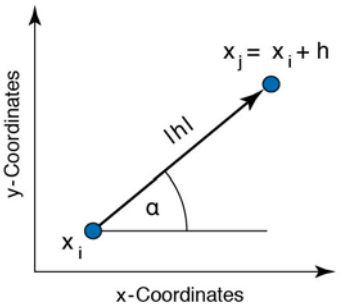


Fig. 7.17 Separation vector *h* between two points.

We then obtain the experimental variogram `G`, as half the squared differences between the observed values:

```
G = 0.5*(Z1 - Z2).^2;
```

In order to speed up the processing we use the MATLAB capability to vectorize commands instead of using *for* loops to run faster. However, we have computed *n2* pairs of observations although only $n(n-1)/2$ pairs are required. For large data sets (e.g., more than 3,000 data points) the software and physical memory of the computer may become limiting factors. In such cases, a more efficient method of programming is described in the user manual for the SURFER software (SURFER 2002). The plot of the experimental variogram is called the *variogram cloud* (Fig. 7.18), which we obtain by extracting the lower triangular portions of the `D` and `G` arrays.

```
indx = 1:length(z);
[C,R] = meshgrid(indx);
I = R > C;

plot(D(I),G(I),'.' )
xlabel('lag distance')
ylabel('variogram')
```

The variogram cloud provides a visual impression of the dispersion of values at the different lags. It can be useful for detecting outliers or anomalies, but it is hard to judge from this presentation whether there is any spatial correlation, and if so, what form it might have and how we could model it (Webster and Oliver 2001). To obtain a clearer view and to prepare a variogram model the experimental variogram is now replaced by the variogram estimator.

The variogram estimator is derived from the experimental variograms in order to summarize their central tendency (similar to the descriptive statistics derived from univariate observations, Section 3.2). The classical variogram

www.it-ebooks.info

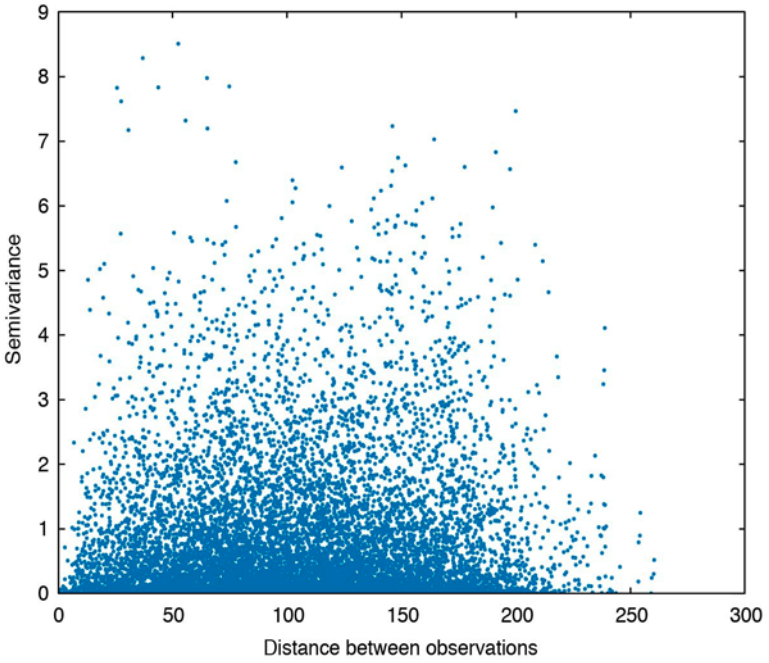


Fig. 7.18 Variogram cloud: plot of the experimental variogram (half the squared difference between pairs of observations) versus the lag distance (separation distance between the two components of a pair).

estimator is the averaged empirical variogram within certain distance classes or bins defined by multiples of the lag interval. The classification of separation distances is illustrated in Figure 7.19. The variogram estimator is calculated by:

$$\gamma_E(h) = \frac{1}{2 * N(h)} \cdot \sum_{i=1}^{N(h)} (z_{xi} - z_{xi+h})^2$$

where $N(h)$ is the number of pairs within the lag interval h.

We first need to decide on a suitable lag interval h. If sampling has been carried out on a regular grid, the length of a grid cell can be used. If the samples are unevenly spaced, as in our case, the mean minimum distance of pairs is a good starting point for the lag interval (Webster and Oliver 2001). To calculate the mean minimum distance of pairs we need to replace the zeros in the diagonal of the lag matrix `D` with `NaN`s, otherwise the mean minimum distance will be zero:

www.it-ebooks.info

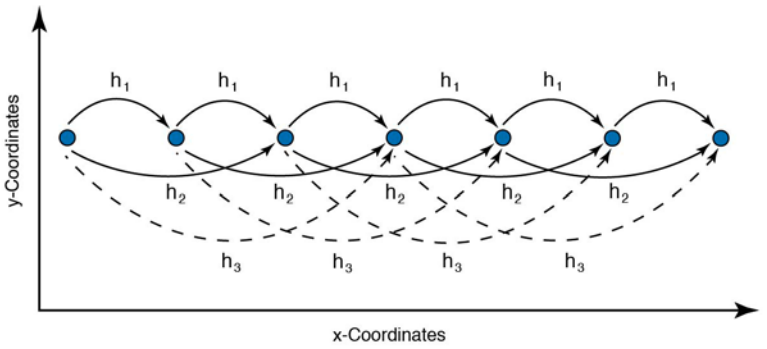


Fig. 7.19 Classification of separation distances for observations that are equally spaced. The lag interval is $h1$, and $h2$, $h3$ etc. are multiples of the lag interval.

```
D2 = D.*(diag(x*NaN)+1);
lag = mean(min(D2))

lag =
    8.0107
```

Since the estimated variogram values tend to become more erratic with increasing distances, it is important to place a maximum distance limit on the calculation. As a rule of thumb, half of the maximum distance is a suitable limit for variogram analysis. We obtain the half maximum distance and the maximum number of lags by:

```
hmd = max(D(:))/2

hmd =
  130.1901

max_lags = floor(hmd/lag)

max_lags =
    16
```

The separation distances are then classified and the classical variogram estimator is calculated:

```
LAGS = ceil(D/lag);

for i = 1 : max_lags
    SEL = (LAGS == i);
    DE(i) = mean(mean(D(SEL)));
    PN(i) = sum(sum(SEL == 1))/2;
    GE(i) = mean(mean(G(SEL)));
end
```

where `SEL` is the selection matrix defined by the lag classes in `LAG`, `DE` is the

www.it-ebooks.info

mean lag, PN is the number of pairs, and GE is the variogram estimator. We can now plot the classical variogram estimator (variogram versus mean separation distance), together with the population variance:

```
plot(DE,GE,'.' )
var_z = var(z);
b = [0 max(DE)];
c = [var_z var_z];
hold on

plot(b,c, '--r')
yl = 1.1 * max(GE);
ylim([0 yl])
xlabel('Averaged distance between observations')
ylabel('Averaged semivariance')
hold off
```

The variogram in Figure 7.20 exhibits a typical pattern of behavior. Values are low at small separation distances (near the origin), they increase with increasing distance until reaching a plateau (*sill*), which is close to the population variance. This indicates that the spatial process is correlated over short distances but there is no spatial dependency over longer distances. The extent of the spatial dependency is called the *range* and is defined as the separation distance at which the variogram reaches the sill.

The *variogram model* is a parametric curve fitted to the variogram estimator. This is similar to frequency distribution fitting (see Section 3.5), where the frequency distribution is modeled by a distribution type and its parameters (e.g., a normal distribution with its mean and variance). For theoretical reasons, only functions with certain properties should be used as variogram models. Common *authorized models* are the spherical, the exponential, and the linear model (more models can be found in the relevant published literature).

Spherical model:

$$\gamma_{\text{sph}}(h) = \begin{cases} c \cdot \left(1.5 \cdot \frac{h}{a} - 0.5 \left(\frac{h}{a} \right)^3 \right) & \text{for } 0 \le h \le a \\ c & \text{for } h > a \end{cases}$$

Exponential model:

$$\gamma_{\text{exp}}(h) = c \cdot \left(1 - \exp\left(-3 \cdot \frac{h}{a} \right) \right)$$

www.it-ebooks.info

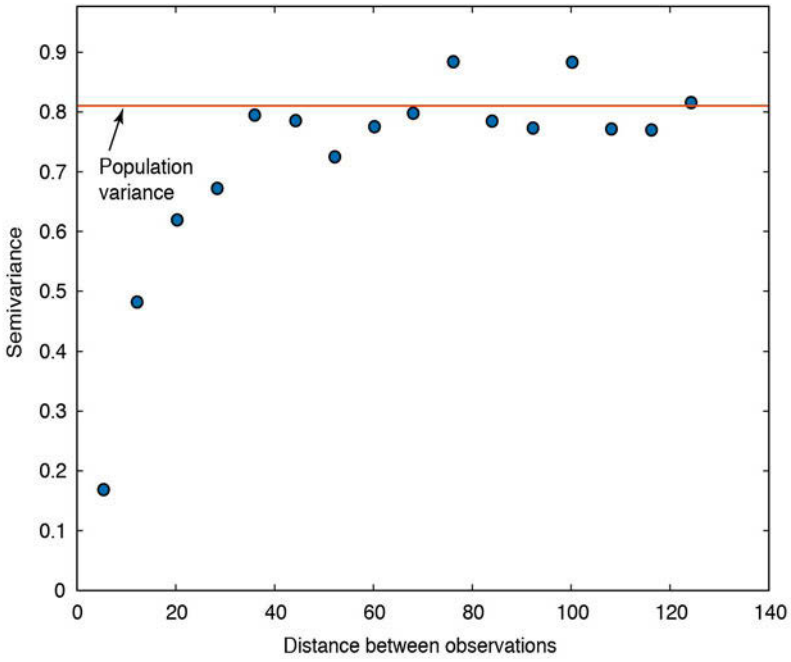


Fig. 7.20 The classical variogram estimator (blue circles) and the population variance (dashed line).

Linear model:

$$\gamma_{\text{lin}}(h) = b \cdot h$$

where c is the sill, a is the range, and b is the slope (for a linear model). The parameters c and either a or b must be modified if a variogram model is fitted to the variogram estimator. The *nugget effect* is a special type of variogram model. In practice, when extrapolating the variogram towards a separation distance of zero we often observe a positive intercept on the y-axis. This is called the nugget effect and it is explained by measurement errors and by small scale fluctuations (*nuggets*) that are not captured due to the sampling intervals being too large. We sometimes have expectations about the minimum nugget effect from the variance of repeated measurements in the laboratory, or from other prior knowledge. More details concerning the nugget effect can be found in Cressie (1993) and Kitanidis (1997). If there is a nugget effect, it can be added into the variogram model. An exponential model with a nugget effect looks like this:

www.it-ebooks.info

$$\gamma_{\exp+\mathrm{nug}}(h) = c_0 + c \cdot \left(1 - \exp\left(-3 \cdot \frac{h}{a}\right)\right)$$

where c_0 is the nugget effect.

We can even combine variogram models, e.g., two spherical models with different ranges and sills. These combinations are called *nested models*. During variogram modeling the components of a nested model are regarded as spatial structures that should be interpreted as being the results of geological processes. Before we discuss further aspects of variogram modeling let us just fit some models to our data. We begin with a spherical model with no nugget effect, and then add an exponential model and a linear model, both with nugget variances:

```
plot(DE,GE,'o','MarkerFaceColor',[.6 .6 .6])
var_z = var(z);
b = [0 max(DE)];
c = [var_z var_z];
hold on
plot(b,c,'--r')
xlim(b)
yl = 1.1*max(GE);
ylim([0 yl])

% Spherical model with nugget
nugget = 0;
sill = 0.803;
range = 45.9;
lags = 0:max(DE);
Gsph = nugget + (sill*(1.5*lags/range - 0.5*(lags/...
   range).^3).*(lags<=range) + sill*(lags>range));
plot(lags,Gsph,':g')

% Exponential model with nugget
nugget = 0.0239;
sill = 0.78;
range = 45;
Gexp = nugget + sill*(1 - exp(-3*lags/range));
plot(lags,Gexp,'-.b')

% Linear model with nugget
nugget = 0.153;
slope = 0.0203;
Glin = nugget + slope*lags;
plot(lags,Glin,'-m')
xlabel('Distance between observations')
ylabel('Semivariance')
legend('Variogram estimator','Population variance',...
   'Sperical model','Exponential model','Linear model')
hold off
```

www.it-ebooks.info

The techniques of variogram modeling are still very much under discussion. Some advocate *objective* variogram modeling by automated curve fitting, using a weighted least squares, maximum likelihood, or maximum entropy method. In contrast, it is often argued that geological knowledge should be included in the modeling process and visual fitting is therefore recommended. In many cases the problem with variogram modeling is much less a question of whether the appropriate procedure has been used than a question of the quality of the experimental variogram. If the experimental variogram is good, both procedures will yield similar results.

Another important question in variogram modeling is the intended use of the model. In our case the linear model does not at first appear to be appropriate (Fig. 7.21). Following a closer look, however, we can see that the linear model fits reasonably well over the first three lags. This can be sufficient if we use the variogram model only for kriging, because in kriging the nearby points are the most important points for the estimate (see discussion of kriging below). Different variogram models with similar fits close to the origin will therefore yield similar kriging results if the sampling points are regularly distributed. If, however, the objective is to describe the spatial structures then the situation is quite different. It then becomes important to find a model that is suitable over all lags and to accurately determine the sill and the range. A collection of geological case studies in Rendu and Readdy (1982) show how process knowledge and variography can be interlinked. Good guidelines for variogram modeling are provided by Gringarten and Deutsch (2001) and Webster and Oliver (2001).

We will now briefly discuss a number of other aspects of variography:

- *Sample size* – As in any statistical procedure, as large a sample as possible is required in order to obtain a reliable estimate. For variography it is recommended that the number of samples should be in excess of 100 to 150 (Webster and Oliver 2001). For smaller sample numbers a maximum likelihood variogram should be computed (Pardo-Igúzquiza and Dowd 1997).

- *Sampling design* – In order to obtain a good estimation close to the origin of the variogram, the sampling design should include observations over small distances. This can be achieved by means of a nested design (Webster and Oliver 2001). Other possible designs have been evaluated by Olea (1984).

- *Anisotropy* – Thus far we have assumed that the structure of spatial correlation is independent of direction. We have calculated *omnidirectional*

www.it-ebooks.info

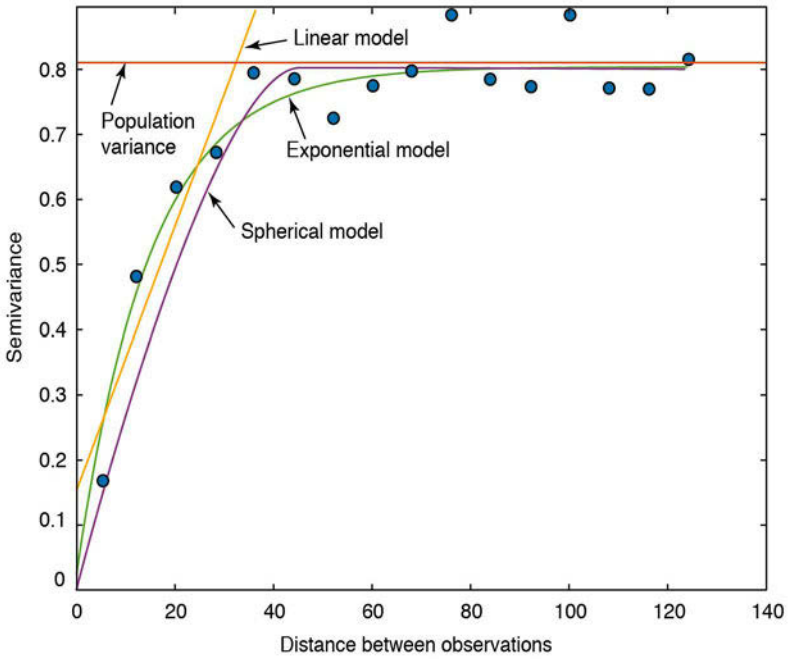


Fig. 7.21 Variogram estimator (blue circles), population variance (solid line), and spherical, exponential, and linear models (dotted and dashed lines).

variograms ignoring the direction of the separation vector h. In a more thorough analysis, the variogram should be discretized not only in distance but also in direction (directional bins). Plotting *directional variograms*, usually in four directions, we are sometimes able to observe different ranges (*geometric anisotropy*), different scales (*zonal anisotropy*), and different shapes (indicating a trend). The treatment of anisotropy requires a highly interactive graphical user interface, which is beyond the scope of this book (see the VarioWin software by Panatier 1996).

- *Number of pairs and the lag interval* – When calculating the classical variogram estimator it is recommended that more than 30 to 50 pairs of points be used per lag interval (Webster and Oliver 2001). This is due to the sensitivity to outliers. If there are fewer pairs, the lag interval should be increased. The lag spacing does not necessarily need to be uniform but can be chosen individually for each distance class. It is also possible to work with overlapping classes, in which case the *lag width* (*lag tolerance*) must be defined. However, increasing the lag width can cause unnecessary

www.it-ebooks.info

smoothing, with a resulting loss of detail. The separation distance and the lag width must therefore be chosen with care. Another option is to use a more robust variogram estimator (Cressie 1993, Deutsch and Journel 1998).

- *Calculation of separation distance* – If the observations cover a large area, for example more than 1,000 km^2, spherical distances should be calculated instead of Pythagorean distances from a planar Cartesian coordinate system.

Kriging

We will now interpolate the observations onto a regular grid by *ordinary point kriging*, which is the most popular kriging method. Ordinary point kriging uses a weighted average of the neighboring points to estimate the value of an unobserved point:

$$\hat{z}_{x0} = \sum_{i}^{N} \lambda_i \cdot z_{xi}$$

where λ_i are the weights that have to be estimated. The sum of the weights should be equal to one in order to guarantee that the estimates are unbiased:

$$\sum_{i}^{N} \lambda_i = 1$$

The expected (average) error for the estimation must be zero. That is:

$$E(\hat{z}_{x0} - z_{x0}) = 0$$

where z_{x0} is the true, but unknown value. We can use the above equations to compute algebraically the mean-squared error in terms of the variogram:

$$E\left(\left(\hat{z}_{x0} - z_{x0}\right)^2\right) = 2\sum_{i=1}^{N} \lambda_i \gamma(x_i, x_0) - \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j \gamma(x_i, x_j)$$

where E is the estimation or *kriging variance* (which must be minimized), $\gamma(x_i, x_0)$ is the variogram (semivariance) between the data points and the unobserved points, $\gamma(x_i, x_j)$ is the variogram between the data points x_i and x_j, and λ_i and λ_j are the weights of the ith and jth data points.

www.it-ebooks.info

For kriging we must minimize this equation (a quadratic objective function), satisfying the condition that the sum of the weights should be equal to one (linear constraint). This optimization problem can be solved using a Lagrange multiplier ν, resulting in a *linear kriging system* of N+1 equations and N+1 unknowns:

$$\sum_{i=1}^{N} \lambda_i \gamma(x_i, x_j) + \nu = \gamma(x_i, x_0)$$

After obtaining the weights λ_i, the kriging variance is given by

$$\sigma^2(x_0) = \sum_{i=1}^{N} \lambda_i \gamma(x_i, x_0) + \nu(x_0)$$

The kriging system can be presented in a matrix notation:

$$G_mod \cdot E = G_R$$

where

$$G_mod = \begin{bmatrix} 0 & \gamma(x_1,x_2) & \dots & \gamma(x_1,x_N) & 1 \\ \gamma(x_2,x_1) & 0 & \dots & \gamma(x_2,x_N) & 1 \\ & & & & \\ \gamma(x_N,x_1) & \gamma(x_N,x_2) & \dots & 0 & 1 \\ 1 & 1 & \dots & 1 & 0 \end{bmatrix}$$

is the matrix of the coefficients: these are the modeled variogram values for the pairs of observations. Note that on the diagonal of the matrix, where separation distance is zero, the value of γ disappears.

$$E = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \\ \nu \end{bmatrix}$$

is the vector of the unknown weights and the Lagrange multiplier.

www.it-ebooks.info

$$G_R = \begin{bmatrix} \gamma(x_1, x_0) \\ \gamma(x_2, x_0) \\ \vdots \\ \gamma(x_N, x_0) \\ 1 \end{bmatrix}$$

is the right-hand-side vector. To obtain the weights and the Lagrange multiplier the matrix *G_mod* is inverted:

$$E = G_mod^{-1} \cdot G_R$$

The kriging variance is given by

$$\sigma^2 = G_R^{-1} \cdot E$$

For our calculations using MATLAB we need the matrix of coefficients derived from the distance matrix *D* and a variogram model. *D* was calculated in the variography section above and we use the exponential variogram model with a nugget, sill, and range from the previous section:

```
G_mod = (nugget + sill*(1 - exp(-3*D/range))).*(D>0);
```

We then take the number of observations and add a column and row vector of all values of one to the `G_mod` matrix and a zero in the lower left corner:

```
n = length(x);
G_mod(:,n+1) = 1;
G_mod(n+1,:) = 1;
G_mod(n+1,n+1) = 0;
```

The `G_mod` matrix now has to be inverted:

```
G_inv = inv(G_mod);
```

A grid with the locations of the unknown values is needed. Here we use a grid cell size of five within a quadratic area ranging from 0 to 200 in *x* and *y* directions. The coordinates are created in matrix form by:

```
R = 0 : 5 : 200;
[Xg1,Xg2] = meshgrid(R,R);
```

and converted to vectors by:

```
Xg = reshape(Xg1,[],1);
```

www.it-ebooks.info

```
Yg = reshape(Xg2,[],1);
```

We then allocate memory for the kriging estimates `Zg` and the kriging variance `s2_k` by:

```
Zg = Xg * NaN;
s2_k = Xg * NaN;
```

We now krige the unknown values at each grid point:

```
for k = 1 : length(Xg)
    DOR = ((x - Xg(k)).^2 + (y - Yg(k)).^2).^0.5;
    G_R = (nugget + sill*(1 - exp(-3*DOR/range))).*(DOR>0);
    G_R(n+1) = 1;
    E = G_inv * G_R;
    Zg(k) = sum(E(1:n,1).*z);
    s2_k(k) = sum(E(1:n,1).*G_R(1:n,1))+E(n+1,1);
end
```

The first command computes the distance between the grid points `(Xg,Yg)` and the observation points`(x,y)`. We then create the right-hand-side vector of the kriging system using the variogram model `G_R` and adding one to the last row. We next obtain the matrix `E` with the weights and the Lagrange multiplier. The estimate `Zg` at each point `k` is the weighted sum of the observations `z`. Finally, the kriging variance `s2_k` of the grid point is computed and we can plot the results. We first create a grid of the kriging estimate and the kriging variance:

```
r = length(R);
Z = reshape(Zg,r,r);
SK = reshape(s2_k,r,r);
```

A subplot on the left presents the kriged values:

```
subplot(1,2,1)
h = pcolor(Xg1,Xg2,Z);
set(h,'LineStyle','none')
axis equal
ylim([0 200])
title('Kriging Estimate')
xlabel('x-Coordinates')
ylabel('y-Coordinates')
colormap(jet)
colorbar
```

The left hand subplot presents the kriging variance:

```
subplot(1,2,2)
h = pcolor(Xg1,Xg2,SK);
set(h,'LineStyle','none')
axis equal
```

www.it-ebooks.info

```
ylim([0 200])
title('Kriging Variance')
xlabel('x-Coordinates')
ylabel('y-Coordinates')
colormap(jet)
colorbar
hold on
```

and we overlay the sampling positions:

```
plot(x,y,'ok')
hold off
```

The kriged values are shown in Figure 7.22 a. The kriging variance depends only on the distance from the observations and not on the observed values (Fig. 7.22 b). Kriging reproduces the population mean when observations are beyond the range of the variogram; at the same time, the kriging variance increases (lower right corner of the maps in Figure 7.22). The kriging variance can be used as a criterion to improve sampling design and it is needed for back-transformation in lognormal kriging. Back-transformation for lognormal kriging is achieved by:

$$y(x_0) = \exp(z(x_0) + 0.5 \cdot \sigma^2(x_0) - \nu)$$

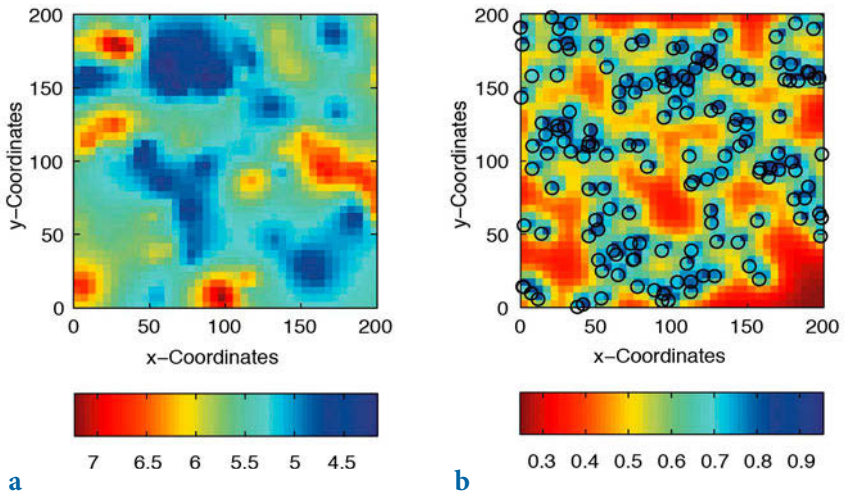


Fig. 7.22 Values interpolated on a regular grid by ordinary point kriging using **a** an exponential variogram model, and **b** the kriging variance as a function of the distance from the observation (empty circles).

www.it-ebooks.info

Discussion of Kriging

Point kriging as presented here is an exact interpolator. It reproduces exactly the values at an observation point, even though a variogram with a nugget effect is used. Smoothing can be achieved by including the variance of the measurement errors (see Kitanidis 1997), and by *block kriging*, which averages the observations within a certain neighborhood (or block). While kriging variance depends only on the distance between the observed and the unobserved locations, it is primarily a measure of the density of information (Wackernagel 2003). The accuracy of kriging is better evaluated by cross-validation using a resampling method or a surrogate test (Sections 4.6 and 4.7). The influence of the neighboring observations on the estimation depends on their configuration, as summarized by Webster and Oliver (2001): "Near points carry more weight than more distant ones; the relative weight of a point decreases when the number of points in the neighborhood increases; clustered points carry less weight individually than isolated ones at the same distance; data points can be screened by ones lying between them and the target." Sampling design for kriging is different from the design that might be optimal for variography. A regular grid, triangular or quadratic, can be regarded as optimal.

The MATLAB code presented here is a straightforward implementation of the above formulae. In professional programs the number of data points entering the *G_mod* matrix is restricted and the inversion of *G_mod* is avoided by working with the covariances instead of the variograms (Webster and Oliver 2001, Kitanidis 1997). For those who are interested in programming and in a deeper understanding of algorithms, the publication by Deutsch and Journel (1992) is essential reading. The best internet source is the homepage for AI-GEOSTATISTICS:

```
http://www.ai-geostats.org
```

Recommended Reading

Amante C, Eakins BW (2009) ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis. NOAA Technical Memorandum NESDIS NGDC-24

Bohlander, J, Scambos T (2007) Antarctic coastlines and grounding line derived from MODIS Mosaic of Antarctica (MOA). National Snow and Ice Data Center, Boulder, Colorado

Cressie N (1993) Statistics for Spatial Data, Revised Edition. John Wiley & Sons, New York

Davis JC (2002) Statistics and Data Analysis in Geology, third edition. John Wiley & Sons, New York

Deutsch CV, Journel AG (1998) GSLIB – Geostatistical Software Library and User's Guide, Second edition. Oxford University Press, Oxford

www.it-ebooks.info

Farr TG, Rosen P, Caro E, Crippen R, Duren R, Hensley S, Kobrick M, Paller M, Rodriguez E, Roth L, Seal D, Shaffer S, Shimada J, Umland J, Werner M, Oskin M, Burbank D, Alsdorf D (2008) The Shuttle Radar Topography Mission. Reviews in Geophysics 45, RG2004

Farr TG, Kobrick M (2000) Shuttle Radar Topography Mission produces a wealth of data. American Geophysical Union Eos 81:583–585

Freeman TG (1991) Calculating Catchment Area with Divergent Flow Based on a Regular Grid. Computers and Geosciences 17:413–422

Gorny AJ (1977) World Data Bank II General User Guide Rep. PB 271869. Central Intelligence Agency, Washington DC

Gringarten E, Deutsch CV (2001) Teacher's Aide Variogram Interpretation and Modeling. Mathematical Geology 33:507–534

Isaaks E, Srivastava M (1989) An Introduction to Applied Geostatistics. Oxford University Press, Oxford

Kitanidis P (1997) Introduction to Geostatistics – Applications in Hydrogeology. Cambridge University Press, Cambridge

Olea RA (1984) Systematic Sampling of Spatial Functions. Kansas Series on Spatial Analysis 7, Kansas Geological Survey, Lawrence, KS

MathWorks (2014a) Mapping Toolbox – User's Guide. The MathWorks, Inc., Natick, MA

MathWorks (2014b) Simulink 3D Animation – User's Guide. The MathWorks, Inc., Natick, MA

Pannatier Y (1996) VarioWin – Software for Spatial Data Analysis in 2D, Springer, Berlin Heidelberg New York

Pardo-Igúzquiza E, Dowd PA (1997) AMLE3D: A Computer Program for the Interference of Spatial Covariance Parameters by Approximate Maximum Likelihood Estimation. Computers and Geosciences 23:793–805

Rendu JM, Readdy L (1982) Geology and Semivariogram – A Critical Relationship. In: Johnson TB, Barns RJ (eds) Application of Computer & Operation Research in the Mineral Industry. 17th Intern. Symp. American Institute of Mining. Metallurgical and Petroleum Engineers, New York, pp. 771–783

Sandwell DT (1987) Biharmonic Spline Interpolation of GEOS-3 and SEASAT Altimeter data. Geophysical Research Letters 2:139–142

Soluri EA, Woodson VA (1990), World Vector Shoreline. International Hydrographic Review LXVII(1):27–35

Swan ARH, Sandilands M (1995) Introduction to Geological Data Analysis. Blackwell Sciences, Oxford

Golden Software, Inc. (2002) Surfer 8 (Surface Mapping System). Golden, Colorado

Wackernagel H (2003) Multivariate Geostatistics: An Introduction with Applications. Third, completely revised edition. Springer, Berlin Heidelberg New York

Webster R, Oliver MA (2001) Geostatistics for Environmental Scientists. John Wiley & Sons, New York

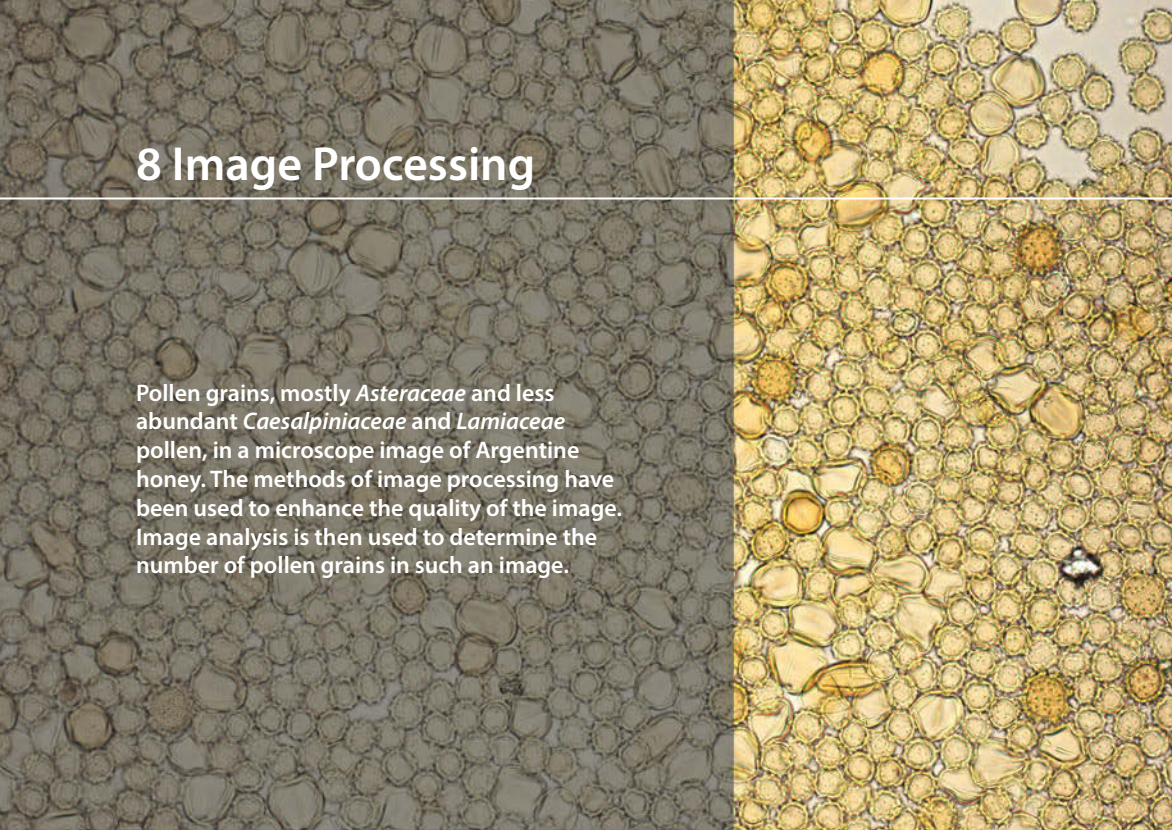
Wessel P, Bercovici D (1998) Gridding with Splines in Tension: A Green Function Approach. Mathematical Geology 30:77–93

Wessel P, Smith WHF (1996) A Global Self-consistent, Hierarchical, High-resolution Shoreline Database. Journal of Geophysical Research 101 B4: 8741-8743.

Wilson JP, Gallant JC (2000) Terrain Analysis, Principles and Applications. John Wiley & Sons, New York

www.it-ebooks.info

8 Image Processing



Pollen grains, mostly *Asteraceae* and less abundant *Caesalpiniaceae* and *Lamiaceae* pollen, in a microscope image of Argentine honey. The methods of image processing have been used to enhance the quality of the image. Image analysis is then used to determine the number of pollen grains in such an image.

8.1 Introduction

Computer graphics are stored and processed as either vector or raster data. Most of the data types that were encountered in the previous chapter were vector data, i.e., points, lines and polygons. Drainage networks, the outlines of geologic units, sampling locations, and topographic contours are all examples of vector data. In Chapter 7, coastlines are stored in a vector format while bathymetric and topographic data are saved in a raster format. Vector and raster data are often combined in a single data set, for instance to display the course of a river on a satellite image. Raster data are often converted to vector data by digitizing points, lines or polygons. Conversely, vector data are sometimes transformed to raster data.

Images are generally represented as raster data, i.e., as a 2D array of color intensities. Images are everywhere in geosciences. Field geologists use aerial photos and satellite images to identify lithologic units, tectonic structures, landslides and other features within a study area. Geomorphologists use such images to analyze drainage networks, river catchments, and vegetation or soil types. The analysis of images from thin sections, the automated identification of objects, and the measurement of varve thicknesses all make

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_8

www.it-ebooks.info

use of a great variety of image processing methods.

This chapter is concerned with the analysis and display of image data. The various ways that raster data can be stored on the computer are first explained (Section 8.2). The main tools for importing, manipulating and exporting image data are then presented in Section 8.3. This knowledge is then used to process and to georeference satellite images (Sections 8.4 to 8.6). On-screen digitization techniques are discussed in Section 8.7. Sections 8.8 and 8.9 deal with importing, enhancing, and analyzing images from laminated lake sediments, including color-intensity measurements on transects across the laminae. Finally, Sections 8.10 to 8.12 deal with automated grain size analysis, charcoal quantification in microscope images, and the detection of objects in microscope images on the basis of their shapes. The Image Processing Toolbox is used for the specific examples throughout this chapter (MathWorks 2014). While the MATLAB User's Guide to the Image Processing Toolbox provides an excellent general introduction to the analysis of images, this chapter provides an overview of typical applications in earth sciences.

8.2 Data Storage

Vector and raster graphics are the two fundamental methods for storing pictures. The typical format for storing *vector data* has already been introduced in the previous chapter. In the following example the two columns in the file *coastline.txt* represent the longitudes and latitudes of the points of a polygon.

```
NaN          NaN
42.892067    0.000000
42.893692    0.001760
NaN          NaN
42.891052    0.001467
42.898093    0.007921
42.904546    0.013201
42.907480    0.016721
42.910414    0.020828
42.913054    0.024642
(cont'd)
```

The `NaN`s help to identify break points in the data (Section 7.2).

The *raster data* are stored as 2D arrays. The elements of these arrays represent variables such as the altitude of a grid point above sea level, the annual rainfall or, in the case of an image, the color intensity values.

```
174 177 180 182 182 182
165 169 170 168 168 170
171 174 173 168 167 170
184 186 183 177 174 176
191 192 190 185 181 181
```

www.it-ebooks.info

```
189 190 190 188 186 183
```

Raster data can be visualized as 3D plots. The *x* and *y* figures are the indices of the 2D array or any other reference frame, and *z* is the numerical value of the elements of the array (see also Chapter 7). The numerical values contained in the 2D array can be displayed as a pseudocolor plot, which is a rectangular array of cells with colors determined by a colormap. A colormap is an *m*-by-3 array of real numbers between 0.0 and 1.0. Each row defines a red, green, or blue (RGB) color. An example is the above array, which could be interpreted as grayscale intensities ranging from 0 (black) to 255 (white). More complex examples include satellite images that are stored in 3D arrays.

As previously discussed, a computer stores data as bits that have one of two states, represented by either a one or a zero (Chapter 2). If the elements of the 2D array represent the color intensity values of the *pixels* (short for *picture elements*) of an image, 1-bit arrays contain only ones and zeros.

```
0   0   1   1   1   1
1   1   0   0   1   1
1   1   1   1   0   0
1   1   1   1   0   1
0   0   0   0   0   0
0   0   0   0   0   0
```

This 2D array of ones and zeros can be simply interpreted as a black-and-white image, where the value of one represents white and zero corresponds to black. Alternatively, the 1-bit array could be used to store an image consisting of any two different colors, such as red and blue.

In order to store more complex types of data, the bits are joined together to form larger groups, such as bytes consisting of eight bits. Since the earliest computers could only process eight bits at a time, early computer code was written in sets of eight bits, which came to be called bytes. Each element of the 2D array or pixel therefore contains a vector of eight ones or zeros.

```
1    0    1    0    0    0    0    1
```

These 8 bits (or 1 byte) allow 2^8=256 possible combinations of the eight ones or zeros, and are therefore able to represent 256 different intensities, such as grayscales. The 8 bits can be read in the following way, reading from right to left: a single bit represents two numbers, two bits represent four numbers, three bits represent eight numbers, and so forth up to a byte (or eight bits), which represents 256 numbers. Each added bit doubles the count of numbers. Here is a comparison of binary and decimal representations of the number 161:

www.it-ebooks.info

```
128   64   32   16    8    4    2    1         (value of the bit)
  1    0    1    0    0    0    0    1         (binary)

128 +  0 + 32  + 0 +  0 +  0 +  0 +  1 = 161   (decimal)
```

The end members of the binary representation of grayscales are

```
  0    0    0    0    0    0    0    0
```

which is black, and

```
  1    1    1    1    1    1    1    1
```

which is pure white. In contrast to the above 1-bit array, the 1-byte array allows a grayscale image of 256 different levels to be stored. Alternatively, the 256 numbers could be interpreted as 256 discrete colors. In either case, the display of such an image requires an additional source of information concerning how the 256 intensity values are converted into colors. Numerous global colormaps for the interpretation of 8-bit color images exist that allow the cross-platform exchange of raster images, while local colormaps are often embedded in a graphics file.

The disadvantage of 8-bit color images is that the 256 discrete colorsteps are not enough to simulate smooth transitions for the human eye. A 24-bit system is therefore used in many applications, with 8 bits of data for each RGB channel giving a total of 256^3=16,777,216 colors. Such a 24-bit image is stored in three 2D arrays, or one 3D array, of intensity values between 0 and 255.

```
195  189  203  217  217  221
218  209  187  192  204  206
207  219  212  198  188  190
203  205  202  202  191  201
190  192  193  191  184  190
186  179  178  182  180  169

209  203  217  232  232  236
234  225  203  208  220  220
224  235  229  214  204  205
223  222  222  219  208  216
209  212  213  211  203  206
206  199  199  203  201  187

174  168  182  199  199  203
198  189  167  172  184  185
188  199  193  178  168  172
186  186  185  183  174  185
177  177  178  176  171  177
179  171  168  170  170  163
```

www.it-ebooks.info

Compared to the 1-bit and 8-bit representations of raster data, 24-bit storage certainly requires a lot more computer memory. In the case of very large data sets such as satellite images and digital elevation models the user should therefore think carefully about the most suitable way to store the data. The default data type in MATLAB is the 64-bit array, which allows storage of the sign of a number (bit 63), the exponent (bits 62 to 52) and roughly 16 significant decimal digits between approximately 10^{-308} and 10^{+308} (bits 51 to 0). However, MATLAB also works with other data types such as 1-bit, 8-bit and 24-bit raster data, to save memory.

The amount of memory required for storing a raster image depends on the data type and the image's dimensions. The dimensions of an image can be described by the number of pixels, which is the number of rows multiplied by the number of columns of the 2D array. Let us assume an image of 729-by-713 pixels, such as the one we will use in the following section. If each pixel needs 8 bits to store a grayscale value, the memory required by the data is 729·713·8=4,158,216 bits or 4,158,216/8=519,777 bytes. This number is exactly what we obtain by typing `whos` in the command window. Common prefixes for bytes are kilo-, mega-, giga- and so forth.

```
bit = 1 or 0 (b)
8 bits = 1 byte (B)
1024 bytes = 1 kilobyte (KB)
1024 kilobytes = 1 megabyte (MB)
1024 megabytes = 1 gigabyte (GB)
1024 gigabytes = 1 terabyte (TB)
```

Note that in data communication 1 kilobit=1,000 bits, while in data storage 1 kilobyte=1,024 bytes. A 24-bit or *true color image* then requires three times the memory required to store an 8-bit image, or 1,559,331 bytes=1,559,331/1,024 kilobytes (KB)≈1,523 KB≈1,559,331/$1{,}024^2$=1.487 megabytes (MB).

However, the dimensions of an image are often given, not by the total number of pixels, but by the length and height of the image and its resolution. The resolution of an image is the number of *pixels per inch* (ppi) or *dots per inch* (dpi). The standard resolution of a computer monitor is 72 dpi although modern monitors often have a higher resolution such as 96 dpi. For instance, a 17 inch monitor with 72 dpi resolution displays 1,024-by-768 pixels. If the monitor is used to display images at a different (lower, higher) resolution, the image is resampled to match the monitor's resolution. For scanning and printing, a resolution of 300 or 600 dpi is enough in most applications. However, scanned images are often scaled for large printouts and therefore have higher resolutions such as 2,400 dpi. The image used in the next section has a width of 25.2 cm (or 9.92 inches) and a height of 25.7 cm (10.12 inches). The resolution of the image is 72 dpi. The total number of

www.it-ebooks.info

pixels is therefore 72·9.92≈713 in a horizontal direction, and 72·10.12≈729 in a vertical direction.

Numerous formats are available for saving vector and raster data into a file, each with their own particular advantages and disadvantages. Choosing one format over another in an application depends on the way the images are to be used in a project and whether or not the images are to be analyzed quantitatively. The most popular formats for storing vector and raster data are:

- *Compuserve Graphics Interchange Format* (GIF) – This format was developed in 1987 for raster images using a fixed 8-bit colormap of 256 colors. The GIF format uses compression without loss of data. It was designed for fast transfer rates over the Internet. The limited number of colors means that it is not the right format for the smooth color transitions that occur in aerial photos or satellite images. It is, however, often used for line art, maps, cartoons and logos (http://www.compuserve.com).

- *Portable Network Graphics* (PNG) – This is an image format developed in 1994 that is used as an alternative to the GIF. It is similar to the GIF in that it also uses a fixed 8-bit colormap of 256 colors. Alternatively, grayscale images of 1 to 16 bits can be stored, as well as 24 and 48 bit color images. The PNG format uses compression without loss of data, with the method employed being better than that used for GIF images.

- *Microsoft Windows Bitmap Format* (BMP) – This is the default image format for computers running Microsoft Windows as the operating system. However, numerous converters also exist to read and write BMP files on other platforms. Various modifications of the BMP format are available, some of them without compression and others with effective and fast compression (http://www.microsoft.com).

- *Tagged Image File Format* (TIFF) – This format was designed by the Aldus Corporation and Microsoft in 1986 to become an industry standard for image-file exchange. A TIFF file includes an image file header, a directory, and the data in all available graphics and image file formats. Some TIFF files even contain vector and raster versions of the same picture, as well as images at different resolutions and with different colormaps. The main advantage of TIFF files was originally their portability. A TIFF should perform on all computer platforms; unfortunately, however, numerous modifications of the TIFF have evolved in subsequent years, resulting in incompatibilities. The TIFF is therefore now often called the *Thousands of Incompatible File Formats*.

www.it-ebooks.info

- *PostScript* (PS) and *Encapsulated PostScript* (EPS) – The PS format was developed by John Warnock at PARC, the Xerox research institute. Warnock was also co-founder of Adobe Systems, where the EPS format was created. The PostScript vector format would never have become an industry standard without Apple Computers. In 1985 Apple needed a typesetter-quality controller for the new Apple LaserWriter printer and the Macintosh operating system and adopted the PostScript format. The third partner in the history of PostScript was the company Aldus, the developer of the software PageMaker and now a part of Adobe Systems. The combination of Aldus PageMaker software, the PS format and the Apple LaserWriter printer led to the creation of Desktop Publishing. The EPS format was then developed by Adobe Systems as a standard file format for importing and exporting PS files. Whereas a PS file is generally a single-page format containing either an illustration or a text, the purpose of an EPS file is to also allow the inclusion of other pages, i.e., a file that can contain any combination of text, graphics and images (http://www.adobe.com).

- In 1986 the *Joint Photographic Experts Group* (JPEG) was founded for the purpose of developing various standards for image compression. Although JPEG stands for the committee, it is now widely used as the name for an image compression and a file format. This compression involves grouping pixel values into 8-by-8 blocks and transforming each block with a discrete cosine transform. As a result, all unnecessary high-frequency information is deleted, which makes this compression method irreversible. The advantage of the JPEG format is the availability of a three-channel, 24-bit, true color version. This allows images with smooth color transitions to be stored. The new JPEG-2000 format uses a wavelet transform instead of the cosine transform (Section 5.8) (http://www.jpeg.org).

- *Portable Document Format* (PDF) – The PDF designed by Adobe Systems is now a true self-contained cross-platform document. PDF files contain the complete formatting of vector illustrations, raster images and text, or a combination of all these, including all necessary fonts. These files are highly compressed, allowing a fast internet download. Adobe Systems provides the free-of-charge Acrobat Reader for all computer platforms to read PDF files (http://www.adobe.com).

www.it-ebooks.info

8.3 Importing, Processing and Exporting Images

We first need to learn how to read an image from a graphics file into the workspace. As an example we use a satellite image showing a 10.5 km by 11 km subarea in northern Chile:

```
http://asterweb.jpl.nasa.gov/gallery/images/unconform.jpg
```

The file *unconform.jpg* is a processed TERRA-ASTER satellite image that can be downloaded free-of-charge from the NASA web page. We save this image in the working directory. The command

```
clear

I1 = imread('unconform.jpg');
```

reads and decompresses the JPEG file, imports the data as a 24-bit RGB image array and stores it in a variable `I1`. The command

```
whos
```

shows how the RGB array is stored in the workspace:

```
Name        Size                Bytes  Class    Attributes
I1        729x713x3           1559331  uint8
```

The details indicate that the image is stored as a 729-by-713-by-3 array, representing a 729-by-713 array for each of the colors red, green and blue. The listing of the current variables in the workspace also gives the information *uint8* array, i.e., each array element representing one pixel contains 8-bit integers. These integers represent intensity values between 0 (minimum intensity) and 255 (maximum). As an example, here is a sector in the upper-left corner of the data array for red:

```
I1(50:55,50:55,1)

ans =
   174 177 180 182 182 182
   165 169 170 168 168 170
   171 174 173 168 167 170
   184 186 183 177 174 176
   191 192 190 185 181 181
   189 190 190 188 186 183
```

We can now view the image using the command

```
imshow(I1)
```

www.it-ebooks.info

which opens a new Figure Window showing an RGB composite of the image (Fig. 8.1). In contrast to the RGB image, a grayscale image needs only a single array to store all the necessary information. We therefore convert the RGB image into a grayscale image using the command `rgb2gray` (RGB to gray):

```
I2 = rgb2gray(I1);
```

The new workspace listing now reads

```
Name        Size                Bytes  Class    Attributes
I1        729x713x3           1559331  uint8
```

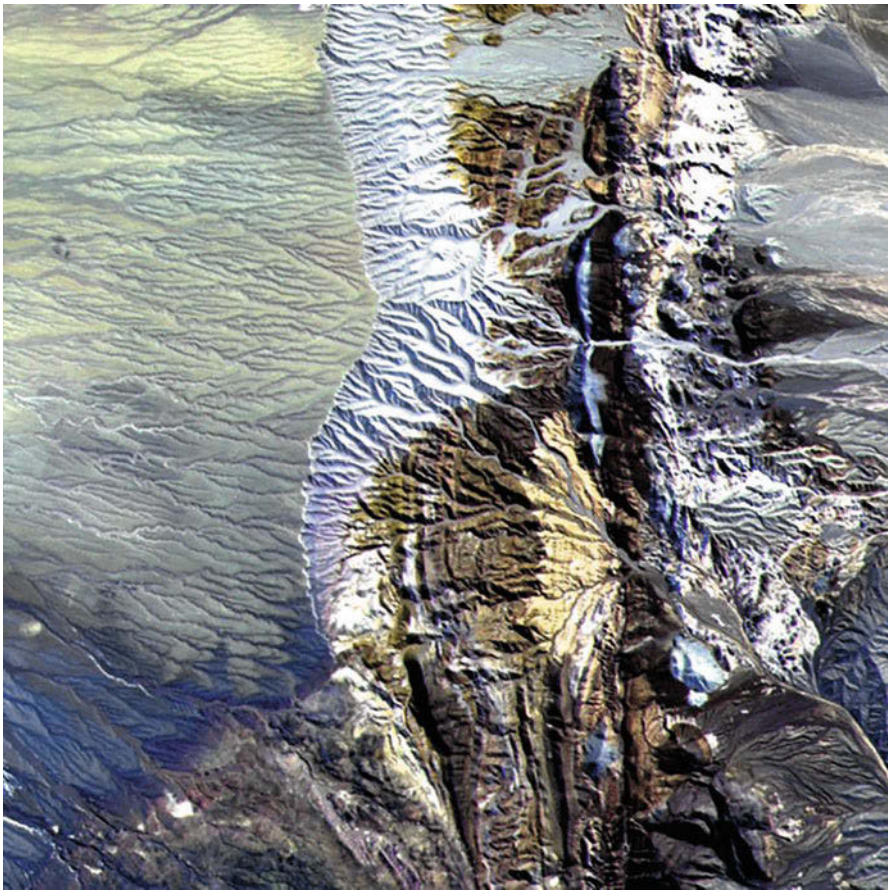


Fig. 8.1 RGB true color image contained in the file *unconform.jpg*. After decompressing and reading the JPEG file into a 729-by-713-by-3 array, MATLAB interprets and displays the RGB composite using the function `imshow`. See detailed description of the image on the NASA TERRA-ASTER webpage: http://asterweb.jpl.nasa.gov. Original image courtesy of NASA/GSFC/METI/ERSDAC/JAROS and U.S./Japan ASTER Science Team.

www.it-ebooks.info

```
I2         729x713                519777  uint8
ans          6x6                      36  uint8
```

in which the difference between the 24-bit RGB and the 8-bit grayscale arrays can be observed. The variable `ans` for *Most recent answer* was created above using `I1(50:55,50:55,1)`, without assigning the output to another variable. The commands

```
imshow(I2)
```

display the result. It is easy to see the difference between the two images in separate Figure Windows. Let us now process the grayscale image. First, we compute a histogram of the distribution of intensity values.

```
imhist(I2)
```

A simple technique to enhance the contrast in such an image is to transform this histogram to obtain an equal distribution of grayscales.

```
I3 = histeq(I2);
```

We can view the difference again using

```
imshow(I3)
```

and save the results in a new file.

```
imwrite(I3,'unconform_gray.jpg')
```

We can read the header of the new file by typing

```
imfinfo('unconform_gray.jpg')
```

which yields

```
ans =
           Filename: [1x40 char]
        FileModDate: '18-Dec-2013 11:26:53'
           FileSize: 138419
             Format: 'jpg'
      FormatVersion: ''
              Width: 713
             Height: 729
           BitDepth: 8
          ColorType: 'grayscale'
    FormatSignature: ''
    NumberOfSamples: 1
       CodingMethod: 'Huffman'
      CodingProcess: 'Sequential'
            Comment: {}
```

www.it-ebooks.info

Hence, the command `imfinfo` can be used to obtain useful information (name, size, format, and color type) concerning the newly-created image file.

There are many ways of transforming the original satellite image into a practical file format. The image data could, for instance, be stored as an *indexed color image*, which consists of two parts: a colormap array and a data array. The colormap array is an m-by-3 array containing floating-point values between 0 and 1. Each column specifies the intensity of the red, green and blue colors. The data array is an x-by-y array containing integer elements corresponding to the lines m of the colormap array, i.e., the specific RGB representation of a certain color. Let us transfer the above RGB image into an indexed image. The colormap of the image should contain 16 different colors. The result of

```
[I4,map] = rgb2ind(I1,16);
imshow(I1), figure, imshow(I4,map)
```

saved as another JPEG file using

```
imwrite(I4,map,'unconform_ind.jpg')
```

clearly shows the difference between the original 24-bit RGB image (256^3 or about 16.7 million different colors) and a color image of only 16 different colors. The display of the image uses the default colormap of MATLAB. Typing

```
imshow(I4,map)
cmap = colormap
```

actually retrieves the 16-by-3 array of the current colormap

```
cmap =
    0.0588    0.0275    0.0745
    0.5490    0.5255    0.4588
    0.7373    0.7922    0.7020
    0.3216    0.2706    0.2667
    0.6471    0.6784    0.6157
    0.7961    0.8549    0.9176
    0.4510    0.3922    0.3333
    0.2000    0.1451    0.1451
    0.4824    0.5412    0.5843
    0.4039    0.4078    0.4784
    0.6667    0.7020    0.7451
    0.8980    0.8745    0.7255
    0.2824    0.2902    0.4039
    0.9569    0.9647    0.9608
    0.1765    0.1686    0.2902
    0.5843    0.5843    0.6078
```

www.it-ebooks.info

We can replace the default colormap by any other built-in colormap. Typing

```
help graph3d
```

lists the available colormaps. As an example we can use

```
imshow(I4,map)
colormap(hot)
```

to display the image with a black-red-yellow-white colormap. Typing

```
edit hot
```

reveals that `hot` is a function creating the *m*-by-3 array containing floating-point values between 0 and 1. We can also design our own colormaps, either by manually creating an *m*-by-3 array or by creating another function similar to `hot`. As an example the colormap `precip.m` (which is a yellow-blue colormap included in the book's file collection) was created to display precipitation data, with yellow representing low rainfall and blue representing high rainfall. Alternatively, we can also use random numbers

```
rng(0)
map = rand(16,3);
imshow(I4,map)
```

Gallery 8.1

to display the image with random colors. Finally, we can create an indexed color image of three different colors, displayed with a simple colormap of full intensity red, green and blue.

```
[I5,map] = rgb2ind(I1,3);
imshow(I5,[1 0 0;0 1 0;0 0 1])
```

Typing

```
imwrite(I4,map,'unconform_rgb.jpg')
```

saves the result as another JPEG file.

8.4 Importing, Processing and Exporting LANDSAT Images

The Landsat project is a satellite remote sensing program jointly managed by the US National Aeronautics and Space Administration (NASA) and the US Geological Survey (USGS), which began with the launch of the Landsat 1 satellite (originally known as the Earth Resources Technology Satellite 1) on 23rd July 1972. The latest in a series of successors is the Landsat 8 satellite, launched on 11th February 2013 (Ochs et al. 2009, Irons et al. 2011). It has two sensors, the Operational Land Imager (OLI) and the Thermal Infrared

www.it-ebooks.info

Sensor (TIRS). These two sensors provide coverage of the global landmass at spatial resolutions of 30 meters (visible, NIR, SWIR), 100 meters (thermal), and 15 meters (panchromatic) (Ochs et al. 2009, Irons et al. 2011). General information concerning the Landsat program can be obtained from the webpage

```
http://landsat.gsfc.nasa.gov/?page_id=7195
```

Landsat data, together with data from other NASA satellites, can be obtained from the webpage

```
http://earthexplorer.usgs.gov
```

On this webpage we first select the desired map section in the *Search Criteria*, either by entering the coordinates of the four corners of the map or by zooming into the area of interest and selecting *Use Map*. As an example we enter the coordinates 4°42'40.72"N 36°51'10.47"E of the Chew Bahir Basin in the Southern Ethiopian Rift. We then choose *L8 OLI/TIRS* from the *Landsat Archive* as the *Data Set* and click *Results*. Clicking *Results* produces a list of records, together with a toolbar for previewing and downloading data. By clicking the *Show Browse Overlay* button we can examine the images for cloud cover. We find the cloud-free image

```
Entity ID: LC81690572013358LGN00
Coordinates: 4.33915,36.76225
Acquisition Date: 24-DEC-13
Path: 169
Row: 57
```

taken on 24th December 2013. We need to register with the USGS website, log on, and then download the Level 1 GeoTIFF Data Product (897.5 MB), which is then stored on the hard drive in the file *LC81690572013358LGN00.tar.gz*. The *.tar.gz* archive contains separate files for each spectral band as well as a metadata file containing information about the data. We use band 4 (Red, 640–670 nm), band 3 (Green, 530–590 nm) and band 2 (Blue, 450–510 nm), each of which has a 30 m resolution. We can import the 118.4 MB TIFF files using

```
clear

I1 = imread('LC81690602013150LGN00_B4.TIF');
I2 = imread('LC81690602013150LGN00_B3.TIF');
I3 = imread('LC81690602013150LGN00_B2.TIF');
```

Typing

www.it-ebooks.info

```
whos
```

reveals that the data are in a unsigned 16-bit format `uint16`, i.e., the maximum range of the data is from 0 to 2^{16}=65,536.

```
  I1        7771x7611           118290162  uint16
  I2        7771x7611           118290162  uint16
  I3        7771x7611           118290162  uint16
```

For quantitative analyses these digital number (DN) values need to be converted to radiance and reflectance values, which is beyond the scope of the book. The radiance is the power density scattered from the earth in a particular direction and has the units of watts per square meter per steradian (Wm^{-2} sr^{-1}) (Richards 2013). The radiance values need to be corrected for atmospheric and topographic effects to obtain earth surface reflectance percentages. The Landsat 8 Handbook provides the necessary information on these conversions:

```
https://landsat.usgs.gov/Landsat8_Using_Product.php
```

We will instead use the Landsat 8 data to create an RGB composite of bands 4, 3, and 2 to be used for fieldwork. Since the image has a relatively low level of contrast, we use `adapthisteq` to perform a contrast-limited adaptive histogram equalization (CLAHE) (Zuiderveld 1994). Unlike `histeq` used in the previous section, the `adapthisteq` algorithm works on small regions (or tiles) of the image, rather than on the entire image. The neighboring tiles are then combined using bilinear interpolation to eliminate edge effects.

```
I1 = adapthisteq(I1,'ClipLimit',0.1,'Distribution','Rayleigh');
I2 = adapthisteq(I2,'ClipLimit',0.1,'Distribution','Rayleigh');
I3 = adapthisteq(I3,'ClipLimit',0.1,'Distribution','Rayleigh');
```

Using `ClipLimit` with a real scalar between 0 and 1 limits the contrast enhancement, while higher numbers result in increased contrast; the default value is 0.01. The `Distribution` parameter sets the desired histogram shape for the tiles by specifying a distribution type, such as `Uniform`, `Rayleigh`, or `Exponential`. Using a `ClipLimit` of 0.1 and a `Rayleigh` distribution yields good results. The three bands are concatenated to a 24-bit RGB images using `cat`.

```
I = cat(3,I1,I2,I3);
```

We only display that section of the image containing the Chew Bahir Basin (using axes limits) and hide the coordinate axes. We scale the images to 10% of the original size to fit the computer screen.

```
axes('XLim',[3000 5000],'YLim',[1000 4000],'Visible','Off'), hold on
```

www.it-ebooks.info

```
imshow(I,'InitialMagnification',10)
```

Exporting the processed image from the Figure Window, we only save the image at the monitor's resolution. To obtain an image of the basins at a higher resolution, we use the command

```
imwrite(I(1000:4000,3000:5000,:),'chewbahirbasin.tif','tif')
```

This command saves the RGB composite as a TIFF-file *chewbahirbasin.tif* (about 36.3 MB) in the working directory, which can then be processed using other software such as Adobe Photoshop.

According to the USGS Landsat webpage, Landsat data are amongst the most geometrically and radiometrically corrected data available. Data anomalies do occasionally occur, however, of which the most common types are listed on the USGS webpage:

```
http://landsat.usgs.gov/science_an_anomalies.php
```

We explore one of these types of anomaly as an example, i.e., artifacts known as *Single Event Upsets* (SEUs) that cause anomalously high values in the image, similar to the *Impulse Noise* (IN) that is also described on the same webpage. These anomalies occur in some, but not all, Landsat images and similarly anomalous high or low values can also occur in other satellite images. We therefore use a part of a Landsat 7 image covering an area in the southern Ethiopian Rift, acquired by the Enhanced Thematic Mapper (ETM+) instrument of that satellite. We can load and display the image using

```
clear

I1 = imread('ethiopianrift_blue.tif');
imshow(I1,'InitialMagnification',200), title('Original Image')
```

The parameter `InitialMagnification` is a numeric scalar that scales the image to, as an example, 200% magnification. The image `I1` shows numerous randomly-distributed anomalously high or low values, as well as a parallel track of paired anomalies in the right half of the image. We first apply a 10-by-10 pixel median filter to the image (see Section 8.8):

```
I2 = medfilt2(I1,[10,10],'symmetric');
imshow(I2,'InitialMagnification',200)
title('Median Filtered Image')
```

Using the option `symmetric` with the function `medfilt2` extends `I2` symmetrically at the image boundaries by reflecting it across its boundaries, instead of padding the image with zeros at the boundaries (by default). The

www.it-ebooks.info

median-filtered version of the I2 image is, of course, very smooth compared to the original I1 image. We would, however, lose a lot of detail if we used this version of the image. We next subtract the median-filtered image I2 from original image I1, which yields the image I3.

```
I3 = imsubtract(I1,I2);
imshow(I3,'InitialMagnification',200)
title('I1-I2')
```

We then subtract the original image I1 from the median-filtered image I2, which yields the image I4.

```
I4 = imsubtract(I2,I1);
imshow(I4,'InitialMagnification',200)
title('I2-I1')
```

We next replace the original pixels with their median-filtered versions if the difference between the median-filtered image I2 and the original image I1 is great than 10 in both directions (as it is in our example).

```
I5 = I1;
I5(I3>10 | I4>10) = I2(I3>10 | I4>10);
imshow(I5,'InitialMagnification',200)
title('Despeckled Image')
```

The image I5 obtained using this approach is the despeckled version of the image I1. We can also explore the pixel values of both versions of the image (I1 and I5) in a 3D surface plot, using

```
subplot(1,2,1)
I1S = im2double(I1);
surface(I1S), colormap jet, caxis([0 1])
shading interp, view(120,33), axis off
axis([1 size(I1,1) 1 size(I1,2) min(I1S(:)) max(I1S(:))])

subplot(1,2,2)
I5S = im2double(I5);
surface(I5S), colormap jet, caxis([0 1])
shading interp, view(120,33), axis off
axis([1 size(I1,1) 1 size(I1,2) min(I1S(:)) max(I1S(:))])
```

We need convert the image data to class double using im2double in order to be able to display the data using surface. Finally, we can display both images in the same figure window

```
subplot(1,2,1), imshow(I1), title('Original Image')
subplot(1,2,2), imshow(I5), title('Despeckled Image')
```

to see the result of despeckling the image I1 (Fig. 8.2).

www.it-ebooks.info

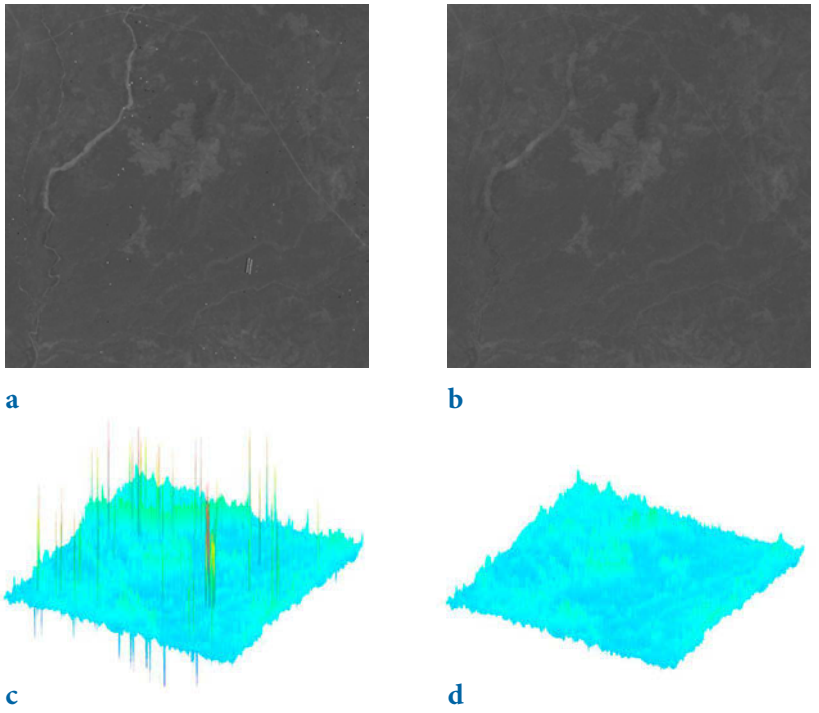


Fig. 8.2 Despeckled section of the blue band of a Landsat image covering the Chew Bahir catchment in Southern Ethiopia; **a** original image, b despeckled image, c surface plots of the original image, and d surface plot of the image after despeckling. Original image courtesy of the Landsat Program of the US National Aeronautics and Space Administration (NASA) and the US Geological Survey (USGS).

8.5 Importing and Georeferencing TERRA ASTER Images

In Section 8.3 we used a processed ASTER image that we downloaded from the ASTER webpage. In this section we will use raw data from this sensor. The ASTER sensor is mounted on the TERRA satellite launched in 1999, part of the Earth Observing System (EOS) series of multi-national NASA satellites (Abrams and Hook 2002). ASTER stands for *Advanced Spaceborne Thermal Emission and Reflection Radiometer*, providing high-resolution (15 to 90 meter) images of the earth in 14 bands, including three visible to near infrared bands (VNIR bands 1 to 3), six short-wave infrared bands (SWIR bands 4 to 9), and five thermal (or long-wave) infrared bands (TIR bands 10 to 14). ASTER images are used to map the surface temperature, emissivity, and reflectance of the earth's surface. The 3rd near infrared band is recorded twice: once with the sensor pointing directly downwards (band 3N, where

www.it-ebooks.info

N stands for *nadir* from the Arabic word for *opposite*), as it does for all other channels, and a second time with the sensor angled backwards at 27.6° (band 3B, where *B* stands for backward looking). These two bands are used to generate ASTER digital elevation models (DEMs).

The ASTER instrument produces two types of data: Level-1A (L1A) and Level-1B (L1B) data (Abrams and Hook 2002). Whereas the L1A data are reconstructed, unprocessed instrument data, the L1B data are radiometrically and geometrically corrected. Any data that ASTER has already acquired are available; they can be located by searching the Japan Space Systems GDS ASTER/PALSAR Unified Search Site and can be ordered from

```
http://gds.ersdac.jspacesystems.or.jp/?lang=en
```

or from NASA Reverb

```
http://reverb.echo.nasa.gov/reverb/
```

As an example we process an image from an area in Kenya showing Lake Naivasha (0°46'31.38"S 36°22'17.31"E). The Level-1A data are stored in two files

```
AST_L1A_003_03082003080706_03242003202838.hdf
AST_L1A_003_03082003080706_03242003202838.hdf.met
```

The first file (116 MB) contains the actual raw data, whereas the second file (102 KB) contains the header, together with all sorts of information about the data. We save both files in our working directory. Since the file name is very long, we first save it in the `filename` variable and then use `filename` instead of the long file name. We then need to modify only this single line of MATLAB code if we want to import and process other satellite images.

```
filename = 'AST_L1A_003_03082003080706_03242003202838.hdf';
```

The Image Processing Toolbox contains various tools for importing and processing files stored in the hierarchical data format (HDF). The graphical user interface (GUI) based import tool for importing certain parts of the raw data is

```
hdftool('filename')
```

This command opens a GUI that allows us to browse the content of the HDF-file *naivasha.hdf*, obtains all information on the contents, and imports certain frequency bands of the satellite image. Alternatively, the command `hdfread` can be used as a quicker way of accessing image data. The *vnir_Band3n*, *vnir_Band2*, and *vnir_Band1* typically contain much information

www.it-ebooks.info

about lithology (including soils), vegetation and water on the earth's surface. These bands are therefore usually combined into 24-bit RGB images. We first read the data

```
I1 = hdfread(filename,'VNIR_Band3N','Fields','ImageData');
I2 = hdfread(filename,'VNIR_Band2','Fields','ImageData');
I3 = hdfread(filename,'VNIR_Band1','Fields','ImageData');
```

These commands generate three 8-bit image arrays, each representing the intensity within a certain infrared (IR) frequency band of a 4200-by-4100 pixel image. We are not using the data for quantitative analyses and therefore do not need to convert the digital number (DN) values into radiance and reflectance values. The *ASTER User Handbook* provides the necessary information on these conversions (Abrams and Hook 2002). Instead, we will process the ASTER image to create a georeferenced RGB composite of bands 3N, 2 and 1, to be used in fieldwork. We first use a contrast-limited adaptive histogram equalization method to enhance the contrast in the image by typing

```
I1 = adapthisteq(I1);
I2 = adapthisteq(I2);
I3 = adapthisteq(I3);
```

and then concatenate the result to a 24-bit RGB image using `cat`.

```
naivasha_rgb = cat(3,I1,I2,I3);
```

As with the previous examples, the 4200-by-4100-by-3 array can now be displayed using

```
imshow(naivasha_rgb,'InitialMagnification',10)
```

We set the initial magnification of this very large image to 10%. MATLAB scales images to fit the computer screen. Exporting the processed image from the Figure Window, we only save the image at the monitor's resolution. To obtain an image at a higher resolution, we use the command

```
imwrite(naivasha_rgb,'naivasha.tif','tif')
```

This command saves the RGB composite as a TIFF-file *naivasha.tif* (ca. 52 MB) in the working directory, which can then be processed using other software such as Adobe Photoshop. The processed ASTER image does not yet have a coordinate system and therefore needs to be tied to a geographical reference frame (*georeferencing*). The HDF browser

```
hdftool('naivasha.hdf')
```

www.it-ebooks.info

can be used to extract the geodetic coordinates of the four corners of the image. This information is contained in the header of the HDF file. Having launched the HDF tool, we select on the uppermost directory called *naivasha.hdf* and find a long list of file attributes in the upper right panel of the GUI, one of which is *productmetadata.0*, which includes the attribute *scenefourcorners*. We collect the coordinates of the four scene corners into a single array `inputpoints`:

```
inputpoints(1,:) = [36.214332 -0.319922];  % upper left corner
inputpoints(2,:) = [36.096003 -0.878267];  % lower left corner
inputpoints(3,:) = [36.770406 -0.400443];  % upper right corner
inputpoints(4,:) = [36.652213 -0.958743];  % lower right corner
```

It is important to note that the coordinates contained in *productmetadata.0* need to be flipped in order to have *x=longitudes* and *y=latitudes*. The four corners of the image correspond to the pixels in the four corners of the image, which we store in a variable named `basepoints`.

```
basepoints(1,:) = [1,1];                   % upper left pixel
basepoints(2,:) = [1,4200];                % lower left pixel
basepoints(3,:) = [4100,1];                % upper right pixel
basepoints(4,:) = [4100,4200];             % lower right pixel
```

The function `fitgeotrans` now takes the pairs of control points, `inputpoints` and `basepoints`, and uses them to infer a spatial transformation matrix `tform`.

```
tform = fitgeotrans(inputpoints,basepoints,'affine');
```

We next determine the limits of the input for georeferencing (i.e., of the original image `naivasha_rgb`) using `size`, which yields `xLimitsIn` and `yLimitsIn`. Adding a value of 0.5 to both `xLimitsIn` and `yLimitsIn` prevents the edges of the image from being truncated during the affine transformation. We then determine the limits of the output (i.e. of the georeferenced image, which is subsequently called `newnaivasha_rgb`) using `outputLimits`, which yields `XBounds` and `YBounds`.

```
xLimitsIn = 0.5 + [0 size(naivasha_rgb,2)];
yLimitsIn = 0.5 + [0 size(naivasha_rgb,1)];
[XBounds,YBounds] = outputLimits(tform,xLimitsIn,yLimitsIn);
```

We then use `imref2d` to reference the image to a world (or global) coordinate system.

```
Rout = imref2d(size(naivasha_rgb),XBounds,YBounds);
```

An `imref2d` object encapsulates the relationship between the intrinsic coordinates anchored to the rows and columns of the image, and the spatial

www.it-ebooks.info

location of the same row and column locations within a world coordinate system. Finally, the affine transformation can be applied to the original RGB composite `naivasha_rgb` in order to obtain a georeferenced version of the satellite image `newnaivasha_rgb` with the same size as `naivasha_rgb`.

```
newnaivasha_rgb = imwarp(naivasha_rgb,tform,'OutputView',Rout);
```

An appropriate grid for the image can now be computed. The grid is typically defined by the minimum and maximum values for the longitude and latitude. The vector increments are then obtained by dividing the ranges of the longitude and latitude by the array's dimensions and then subtracting one from the results. Note the difference between the MATLAB numbering convention and the common coding of maps used in published literature. The north/south suffix is generally replaced by a negative sign for south,

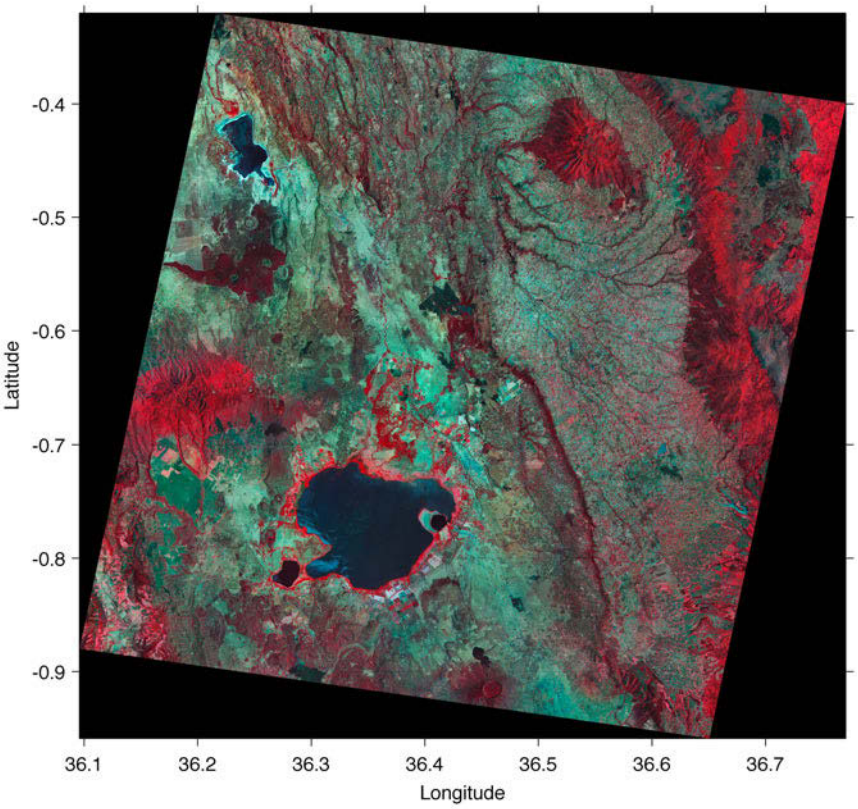


Fig. 8.3 Geoferenced RGB composite of a TERRA-ASTER image using the infrared bands *vnir_Band3n, 2* and *1*. The result is displayed using `imshow`. Original image courtesy of NASA/GSFC/METI/ERSDAC/JAROS and U.S./Japan ASTER Science Team.

www.it-ebooks.info

whereas MATLAB coding conventions require negative signs for north.

```
X = 36.096003 : (36.770406 - 36.096003)/4100 : 36.770406;
Y = -0.958743 : ( 0.958743 -  0.319922)/4200 : -0.319922;
```

The georeferenced image is displayed with coordinates on the axes and a superimposed grid (Fig. 8.3). By default, the function `imshow` inverts the latitude axis when images are displayed by setting the `YDir` property to `Reverse`. To invert the latitude axis direction back to normal, we need to set the `YDir` property to `Normal` by typing

```
imshow(newnaivasha_rgb,'XData',X,'YData',Y,'InitialMagnification',10)
axis on, grid on, set(gca,'YDir','Normal')
xlabel('Longitude'), ylabel('Latitude')
title('Georeferenced ASTER Image')
```

Exporting the image is possible in many different ways, for example using

```
print -djpeg70 -r600 naivasha_georef.jpg
```

to export it as a JPEG file *naivasha_georef.jpg*, compressed to 70% and with a resolution of 600 dpi.

In the previous example we used the geodetic coordinates of the four corners to georeference the ASTER image. The Image Processing Toolbox also includes functions to automatically align two images that are shifted and/or rotated with respect to each other, cover slightly different areas, or have a different resolutions. We use two ASTER images of the Suguta Valley in the Northern Kenya Rift as an example. The images have been processed in the same way as described for the image of Lake Naivasha and exported as TIFF files using `imwrite`. The image in the file *sugutavalley_1.tif* was taken on 20th February 2003 and the second image in *sugutavalley_2.tif* was taken on 31st August 2003, both just after 8 o'clock in the morning. Lake Logipi, in the center of the images, is much larger in the second image than in the first image. The original images are otherwise almost identical, except for the second image being shifted slightly towards the east. To demonstrate the automatic alignment of the images, the second image has been rotated counterclockwise by five degrees. Furthermore, both images have been cropped to the area of the Suguta Valley, including a small section of the rift shoulders to the west and to the east. We import both images using

```
clear

image1 = imread('sugutavalley_1.tif');
image2 = imread('sugutavalley_2.tif');
```

www.it-ebooks.info

The size of `image1` is 666-by-329-by-3, while `image2` is slightly smaller: 614-by-270-by-3. We display the images by typing

```
subplot(1,2,1), imshow(image1)
subplot(1,2,2), imshow(image2)
```

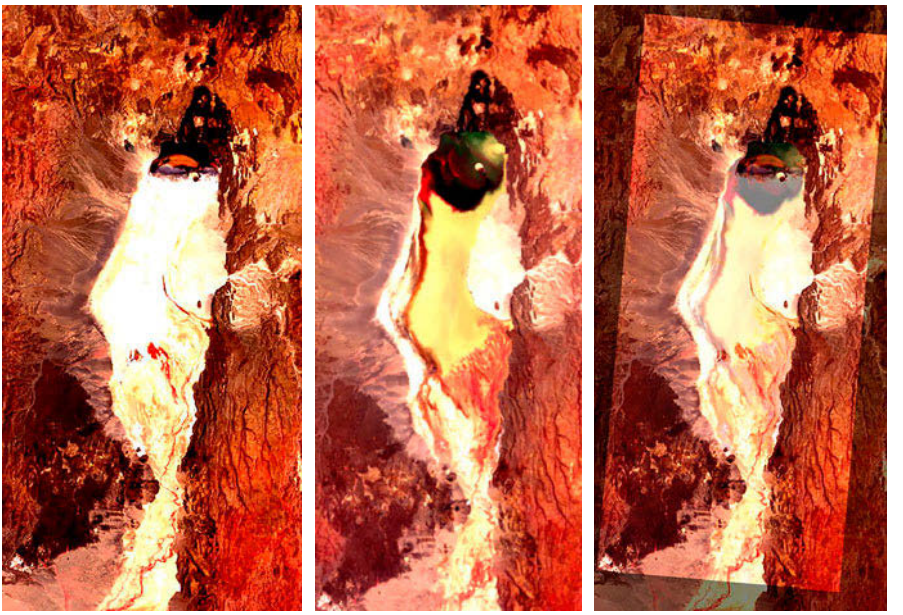
The function `imregconfig` creates optimizer and metric configurations that we transfer into `imregister` to perform intensity-based image registration,

```
[optimizer, metric] = imregconfig('monomodal');
```

where `monomodal` assumes that the images were captured by the same sensor. We can use this configurations to calculate the spatial transformation matrix `tform` using the transformation type `affine`, as in the previous example.

```
tform = imregtform(image2(:,:,1),image1(:,:,1), ...
    'affine',optimizer,metric);
```

This transformation can be applied to `image2` in order to automatically align it with `image1`.



a **b** **c**

Fig. 8.4 Automatically aligned TERRA-ASTER images of the Suguta Valley in the Northern Kenya Rift; **a** first image taken on 20th February 2003, **b** second image taken on 31st August 2003, and **c** overlay of the second image aligned with the first image. Original image courtesy of NASA/GSFC/METI/ERSDAC/JAROS and U.S./Japan ASTER Science Team.

www.it-ebooks.info

```
image2_reg = imwarp(image2,tform,'OutputView', ...
    imref2d(size(image1)));
```

We can compare the result with the original images using

```
subplot(1,3,1), imshow(image1)
subplot(1,3,2), imshow(image2)
subplot(1,3,3), imshowpair(image1,image2_reg,'blend')

print -djpeg70 -r600 sugutavalley_aligned.jpg
```

As we can see, the second image is now nicely aligned with the first image (Fig. 8.4). The two images can now be used to map changes in the area (e.g., in the size of the lake) between 20th February and 31st August 2003. This script can also be used to automatically align other images, in particular those captured by different sensors.

8.6 Processing and Exporting EO-1 Hyperion Images

The Earth Observing-1 Mission (EO-1) satellite is part of the New Millennium Program of the US National Aeronautics and Space Administration (NASA) and the US Geological Survey (USGS), which began with the launch of this satellite on 21st November 2000. EO-1 has two sensors: the Advanced Land Image (ALI) has nine multispectral bands with a 30 m spatial resolution and a panchromatic band with a 10-m resolution, and the hyperspectral sensor (Hyperion) has 220 bands between 430 and 2,400 nm (Mandl et al. 2002, Line 2012). General information about the EO-1 program can be obtained from the webpage

```
http://eo1.gsfc.nasa.gov
```

Hyperion data (together with data from of other NASA satellites) are freely available from the webpage

```
http://earthexplorer.usgs.gov
```

On this webpage we first select the desired map section in the *Search Criteria*, either by entering the coordinates of the four corners of the map or by zooming into the area of interest and selecting *Use Map*. As an example we enter the coordinates 2°8'37.58"N 36°33'47.06"E of the Suguta Valley in the Northern Kenya Rift. We then choose *Hyperion* from the *EO-1* collection as the *Data Set* and click *Results*. Clicking *Results* produces a list of records, together with a toolbar for previewing and downloading data. Clicking the *Show Browse Overlay* button allows us to examine the images for cloud cover. We find the cloud-free image

www.it-ebooks.info

```
Entity ID: EO1H1690582013197110KF_PF2_01
Acquisition Date: 16-JUL-13
Target Path: 169
Target Row: 58
```

taken on 16th July 2013. As before, we need to register with the USGS website, log on, and then download the radiometrically corrected (but not geometrically corrected) Level 1R (L1R) product (215.3 MB), which is then stored on the hard drive in the file *LEO1H1690582013197110KF_1R.ZIP*. The *.ZIP* archive consists of a metadata file (*.MET*), a Federal Geographic Data Committee (FGDC) metadata file (*.fgdc*), an HDF data set file (*.L1R*), and multiple auxiliary files. The EO-1 User's Guides provides some useful information on the data formats of these files (Barry 2001, Beck 2003). We can import the data from the *EO1H1690582013197110KF.L1R* file using

```
clear

HYP = hdfread('EO1H1690582013197110KF.L1R',...
    '/EO1H1690582013197110KF.L1R', ...
    'Index', {[1 1 1],[1 1 1],[3189 242 256]});
```

The parameter `Index` is a three-element cell array, `{start,stride,edge}`, specifying the location, range, and values to be read from the data set. The value of `start` specifies the position in the file to begin reading. In our example it starts reading at the beginning of the file, i.e., starting from the `[1 1 1]` element. The value of `stride` defines the interval between the values to be read, which in our example is `[1 1 1]`, i.e., every element of the data set is to be read. The value of `edge` specifies the size of the data in the file; in our example the dimensions of the data are 3,189-by-242-by-256. Typing

```
whos
```

shows how the hyperspectral image is stored in the workspace:

```
HYP       3189x242x256            395129856  int16
```

The details indicate that the image is stored as a 3,189-by-242-by-256 array, representing a 3,189-by-256 array for each of the 242 spectral bands. The listing of the current variables in the workspace also gives the information *int16* array, i.e., each array element representing one pixel contains signed 16-bit integers. We need to permute the array to move the bands to the third dimension by typing

```
HYP = permute(HYP,[1 3 2]);
```

We next need to determine the radiance values from the digital number

www.it-ebooks.info

(DN) values in HYP. The radiance is the power density scattered from the earth in a particular direction and has the units of watts per square meter per steradian (Wm^{-2} sr^{-1}) (Richards 2013). The EO-1 User Guide (v. 2.3) provides the necessary information on these conversions in its *Frequently Asked Questions* (FAQ) section (Beck 2003). According to this document, the radiance HYPR for the visible and near-infrared (VNIR) bands (bands 1 to 70) is calculated by dividing the digital number in HYP by 40. The radiance for the shortwave infrared (SWIR) bands (bands 71 to 242) is calculated by dividing HYP by 80.

```
HYPR(:,:,1:70) = HYP(:,:,1:70)/40;
HYPR(:,:,71:242) = HYP(:,:,71:242)/80;
```

For quantitative analyses, the radiance values HYPR need to be corrected for atmospheric and topographic effects. This correction, which yields earth surface reflectance values (in percentages), is beyond the scope of the book. The EO-1 User Guide (v. 2.3) again explains several methods to convert radiance to reflectance values (Beck 2003). A simple way to convert radiance to reflectance for relatively clear Hyperion images is given in

```
https://eo1.usgs.gov/faq/question?id=21
```

We will instead process the Hyperion image to create a georeferenced RGB composite of bands 29, 23 and 16, to be used in fieldwork. The header file *O1H169058201319711OKF.HDR* contains (among other things) the wavelengths corresponding to the 242 spectral bands. We can read the wavelengths from the file using textscan:

```
fid = fopen('EO1H1690582013197110KF.hdr');
C = textscan(fid,'%f %f %f %f %f %f %f %f',...
    'Delimiter',',','Headerlines',257,'CollectOutput',1)
fclose(fid);
```

This script opens the header file for read only access using fopen and defines the file identifier fid, which is then used to read the text from the file with textscan, and to write it into the array C. The character string %f %f %f %f %f %f %f %f defines the conversion specifiers enclosed in single quotation marks, where %f stands for the double-precision floating-point 64-bit output class. The parameter Headerlines is set to 257, which means that the first 257 lines are ignored when reading the file. If the parameter CollectOutput is 1 (true), textscan concatenates output cells of the same data type into a single array. Function fclose closes the file defined by the file identifier fid. The array C is a cell array, which is a data type with indexed containers called cells. We can easily obtain the wavelengths from C using

www.it-ebooks.info

```
wavelengths = C{1};
wavelengths = wavelengths';
wavelengths = wavelengths(isnan(wavelengths(:))==0);
```

We can now plot the radiance `HYPR` of the VNIR bands (blue) and SWIR bands (in red) in a single plot.

```
plot(wavelengths(1:60),squeeze(HYPR(536,136,1:60)),'b',...
     wavelengths(71:242),squeeze(HYPR(536,136,71:242)),'r')
```

According to v. 2.3 of the EO-1 User Guide (Beck 2003), Hyperion records 220 unique spectral channels collected from a complete spectrum covering 357 to 2,576 nm. The L1R product has 242 bands but 198 bands are calibrated. Because of an overlap between the VNIR and SWIR focal planes, there are only 196 unique channels. Calibrated channels are 8–57 for the VNIR, and 77–224 for the SWIR. The bands that are not calibrated are set to zero in those channels.

In order to create an RGB composite of bands 29, 23 and 16, we can extract the bands from the radiance values data `HYPR` by typing

```
HYP1 = HYPR(:,:,29);
HYP2 = HYPR(:,:,23);
HYP3 = HYPR(:,:,16);
```

To display the data with `imshow` we need convert the signed integer 16-bit (`int16`) data to unsigned integer 8-bit data (`uint8`). For this purpose, we first obtain an overview of the range of the data using a histogram plot with 100 classes.

```
subplot(1,3,1), histogram(double(HYP1(:)),100), title('Band 29')
subplot(1,3,2), histogram(double(HYP2(:)),100), title('Band 23')
subplot(1,3,3), histogram(double(HYP3(:)),100), title('Band 16')
```

As we see, the radiance values of most pixels from the spectral bands 29, 23 and 16 lie between 0 and 200 Wm^{-2} sr^{-1}. Several functions are available for converting the data from `int16` to `uint8`. The function `im2uint8` rescales the data to the range [0,255] and offsets it if necessary. The function `uint8` simply rounds all values in the range [0,255] to the nearest integer; all values less than 0 are set to 0 and all values greater than 255 are set to 255. The function `mat2gray(A,[AMIN AMAX])` converts an arbitrary array `A` to an intensity image `I` containing values in the range 0 (black) to 1.0 (full intensity or white), where `[AMIN AMAX]` can be used to limit the range of the original data, which is scaled to the range [0,1]. Since most of our radiance values are within the range [0,255], we use `uint8` to convert our data to the `uint8` data type without losing much information.

www.it-ebooks.info

```
HYP1 = uint8(HYP1);
HYP2 = uint8(HYP2);
HYP3 = uint8(HYP3);
```

Again, displaying the radiance values of the three bands in a histogram using

```
subplot(1,3,1), histogram(single(HYP1(:)),30), title('Band 29')
subplot(1,3,2), histogram(single(HYP2(:)),30), title('Band 23')
subplot(1,3,3), histogram(single(HYP3(:)),30), title('Band 16')
```

reveals that most radiance values are actually within the range [20,80]. Instead of using `histogram` we can also use `imhist` to display the `uint8` data.

```
subplot(1,3,1), imhist(HYP1(:)), title('Band 29')
subplot(1,3,2), imhist(HYP2(:)), title('Band 23')
subplot(1,3,3), imhist(HYP3(:)), title('Band 16')
```

We then use `histeq` to enhance the contrast in the image and concatenate the three bands to a 3,189-by-242-by-3 array.

```
HYP1 = histeq(HYP1);
HYP2 = histeq(HYP2);
HYP3 = histeq(HYP3);

subplot(1,3,1), imhist(HYP1(:)), title('Band 29')
subplot(1,3,2), imhist(HYP2(:)), title('Band 23')
subplot(1,3,3), imhist(HYP3(:)), title('Band 16')

HYPC = cat(3,HYP1,HYP2,HYP3);
```

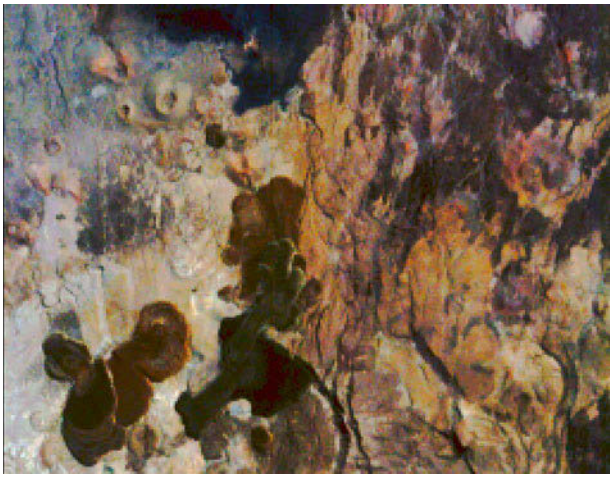


Fig. 8.5 RGB composite of an EO-1 Hyperion image using VNIR bands 29, 23 and 16, showing the Barrier Volcanic Complex in the Suguta Valley of the Northern Kenya Rift. The image was acquired on 16th July 2013. Original image courtesy of NASA EO-1 Mission.

www.it-ebooks.info

Finally, we can display the entire image using

```
imshow(HYPC)
```

or, as an alternative, that part of the image showing the Barrier Volcanic Complex in the northern Suguta Valley (Fig. 8.5).

```
imshow(HYPC(900:1100,:,:))
```

Exporting the image is possible in many different ways, for example using

```
print -r600 -dtiff barrier.tif
```

to export it as a TIFF file *barrier.tif* with a resolution of 600 dpi.

8.7 Digitizing from the Screen

On-screen digitizing is a widely-used image processing technique. While practical digitizer tablets exist in all formats and sizes, most people prefer digitizing vector data from the screen. Examples of this type of application include the digitizing of river networks and catchment areas on topographic maps, of the outlines of lithologic units on geological maps, of landslide distributions on satellite images, and of mineral grain distributions in microscope images. The digitizing procedure consists of the following steps. The image is first imported into the workspace. A coordinate system is then defined, allowing the objects of interest to be entered by moving a cursor or cross hair onto it and clicking the mouse button. The result is a two-dimensional array of *xy* data, such as longitudes and latitudes of the corner points of a polygon, or the coordinates of the objects of interest in a particular area.

The function `ginput` included in the standard MATLAB toolbox allows graphical input from the screen, using a mouse. It is generally used to select points, such as specific data points, from a figure created by an arbitrary graphics function such as `plot`. The function `ginput` is often used for interactive plotting, i.e., the digitized points appear on the screen after they have been selected. The disadvantage of the function is that it does not provide coordinate referencing on an image. We therefore use a modified version of the function, which allows an image to be referenced to an arbitrary rectangular coordinate system. Save the following code for this modified version of the function `ginput` in a text file *minput.m*.

```
function data = minput(imagefile)
% Specify the limits of the image
xmin = input('Specify xmin! ');
```

www.it-ebooks.info

```
xmax = input('Specify xmax! ');
ymin = input('Specify ymin! ');
ymax = input('Specify ymax! ');

% Read image and display
B = imread(imagefile);
a = size(B,2); b = size(B,1);
imshow(B);

% Define lower left and upper right corner of image
disp('Click on lower left and upper right corner, then <return>')
[xcr,ycr] = ginput;
XMIN = xmin-((xmax-xmin)*xcr(1,1)/(xcr(2,1)-xcr(1,1)));
XMAX = xmax+((xmax-xmin)*(a-xcr(2,1))/(xcr(2,1)-xcr(1,1)));
YMIN = ymin-((ymax-ymin)*ycr(1,1)/(ycr(2,1)-ycr(1,1)));
YMAX = ymax+((ymax-ymin)*(b-ycr(2,1))/(ycr(2,1)-ycr(1,1)));

% Digitize data points
disp('Click on data points to digitize, then <return>')
[xdata,ydata] = ginput;
XDATA = XMIN + ((XMAX-XMIN)*xdata/size(B,2));
YDATA = YMIN + ((YMAX-YMIN)*ydata/size(B,1));
data(:,1) = XDATA; data(:,2) = YDATA;
```

The function `minput` has four stages. In the first stage the user enters the limits of the coordinate axes as reference points for the image. The image is then imported into the workspace and displayed on the screen. The third stage uses `ginput` to define the upper left and lower right corners of the image. In the fourth stage the relationship between the coordinates of the two corners on the figure window and the reference coordinate system is then used to compute the transformation for all of the digitized points.

As an example we use the image stored in the file *naivasha_georef.jpg* and digitize the outline of Lake Naivasha in the center of the image. We activate the new function `minput` from the Command Window using the commands

```
clear

data = minput('naivasha_georef.jpg')
```

The function first asks for the coordinates of the limits of the x-axis and the y-axis, for the reference frame. We enter the corresponding numbers and press *return* after each input.

```
Specify xmin! 36.1
Specify xmax! 36.7
Specify ymin! -1
Specify ymax! -0.3
```

The function then reads the file *naivasha_georef.jpg* and displays the image. We ignore the warning

www.it-ebooks.info

```
Warning: Image is too big to fit on screen; displaying at 33%
```

and wait for the next response

```
Click on lower left and upper right corner, then <return>
```

The image window can be scaled according to user preference. Clicking on the lower left and upper right corners defines the dimensions of the image. These changes are registered by pressing *return*. The routine then references the image to the coordinate system and waits for the input of the points we wish to digitize from the image.

```
Click on data points to digitize, then <return>
```

We finish the input by again pressing *return*. The *xy* coordinates of our digitized points are now stored in the variable `data`. We can now use these vector data for other applications.

8.8 Image Enhancement, Correction and Rectification

This section introduces some fundamental tools for image enhancement, correction and rectification. As an example we use an image of varved sediments deposited around 33 kyrs ago in a landslide-dammed lake in the Quebrada de Cafayate of Argentina (25°58.900'S 65°45.676'W) (Trauth et al. 1999, 2003). The diapositive was taken on 1st October 1996 with a film-based single-lens reflex (SLR) camera. A 30-by-20 cm print was made from the slide, which has been scanned using a flatbed scanner and saved as a 394 KB JPEG file. We use this as an example because it demonstrates some problems that we can solve with the help of image enhancement (Fig. 8.6). We then use the image to demonstrate how to measure color-intensity transects for use in time series analysis (Section 8.9).

We can read and decompress the file *varves_original.jpg* by typing

```
clear

I1 = imread('varves_original.jpg');
```

which yields a 24-bit RGB image array `I1` in the MATLAB workspace. Typing

```
whos
```

yields

```
Name         Size                 Bytes  Class    Attributes
I1        1096x1674x3           5504112  uint8
```

www.it-ebooks.info

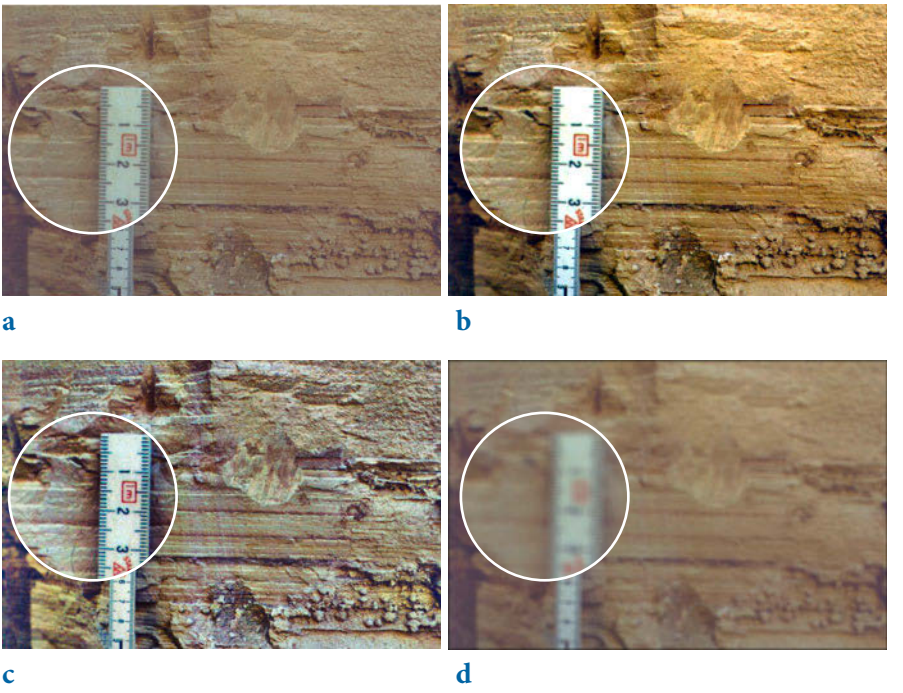


Fig. 8.6 Results of image enhancements; **a** original image, **b** image with intensity values adjusted using `imadjust`, Gamma=1.5, **c** image with contrast enhanced using `adapthisteq`, **d** image after filtering with a 20-by-20 pixel filter with the shape of a Gaussian probability density function with a mean of zero and a standard deviation of 10, using `fspecial` and `imfilter`.

revealing that the image is stored as an `uint8` array of the size 1,096-by-1,674-by-3, i.e., 1,096-by-1,674 arrays for each color (red, green and blue). We can display the image using the command

```
imshow(I1)
```

which opens a new Figure Window showing an RGB composite of the image. As we see, the image has a low level of contrast and very pale colors, and the sediment layers are not exactly horizontal. These are characteristics of the image that we want to improve in the following steps.

First, we adjust the image intensity values or colormap. The function `imadjust(I1,[li; hi],[lo ho])` maps the values of the image `I1` to new values in `I2`, such that values between `li` and `hi` are adjusted to values between `lo` and `ho`. Values below `li` and above `hi` are clipped, i.e., these values are adjusted to `lo` and `ho`, respectively. We can determine the range of the pixel

www.it-ebooks.info

values using

```
lh = stretchlim(I1)
```

which yields

```
lh =
    0.3255    0.2627    0.2784
    0.7020    0.7216    0.7020
```

indicating that the red color ranges from 0.3255 to 0.7020, green ranges from 0.2627 to 0.7216, and blue ranges from 0.2784 to 0.7020. We can utilize this information to automatically adjust the image with `imadjust` by typing

```
I2 = imadjust(I1,lh,[]);
```

which adjusts the ranges to the full range of [0,1], and then display the result.

```
imshow(I2)
```

We can clearly see the difference between the very pale image `I1` and the more saturated image `I2`. The parameter `gamma` in `imadjust(I1,[li;hi],[lo;ho],gamma)` specifies the shape of the curve describing the relationship between `I1` and `I2`. If `gamma<1` the mapping is weighted toward higher (brighter) output values. If `gamma>1` the mapping is weighted toward lower (darker) output values. The default value of `gamma=1` causes linear mapping of the values in `I1` to new values in `I2`.

```
I3 = imadjust(I1,lh,[],0.5);
I4 = imadjust(I1,lh,[],1.5);

subplot(2,2,1), imshow(I1), title('Original Image')
subplot(2,2,2), imshow(I2), title('Adjusted Image, Gamma=1.0')
subplot(2,2,3), imshow(I3), title('Adjusted Image, Gamma=0.5')
subplot(2,2,4), imshow(I4), title('Adjusted Image, Gamma=1.5')
```

We can use `imhist` to display a histogram showing the distribution of intensity values for the image. Since `imhist` only works for two-dimensional images, we examine the histogram of the red color only.

```
subplot(2,2,1), imhist(I1(:,:,1)), title('Original Image')
subplot(2,2,2), imhist(I2(:,:,1)), title('Adjusted Image, Gamma=1.0')
subplot(2,2,3), imhist(I3(:,:,1)), title('Adjusted Image, Gamma=0.5')
subplot(2,2,4), imhist(I4(:,:,1)), title('Adjusted Image, Gamma=1.5')
```

The result obtained using `imadjust` differs from that obtained using `histeq` (which we used in Section 8.3 to enhance the contrast in the image). The function `histeq(I1,n)` transforms the intensity of image `I1`, returning in

www.it-ebooks.info

I5 an intensity image with n discrete levels. A roughly equal number of pixels is ascribed to each of the n levels in I5, so that the histogram of I5 is approximately flat. Histogram equalization using histeq has to carried out separately for each color, since histeq only works for two-dimensional images. We use n=50 in our exercise, which is slightly below the default value of n=64.

```
I5(:,:,1) = histeq(I1(:,:,1),50);
I5(:,:,2) = histeq(I1(:,:,2),50);
I5(:,:,3) = histeq(I1(:,:,3),50);

subplot(2,2,1), imshow(I1), title('Original Image')
subplot(2,2,3), imhist(I1(:,:,1)), title('Original Image')
subplot(2,2,2), imshow(I5), title('Enhanced Image')
subplot(2,2,4), imhist(I5(:,:,1)), title('Enhanced Image')
```

The resulting image looks quite disappointing and we therefore use the improved function adapthisteq instead of histeq. This function uses the contrast-limited adaptive histogram equalization (CLAHE) by Zuiderveld (1994). Unlike histeq and imadjust, the algorithm works on small regions (or tiles) of the image, rather than on the entire image. The neighboring tiles are then combined using bilinear interpolation to eliminate edge effects.

```
I6(:,:,1) = adapthisteq(I1(:,:,1));
I6(:,:,2) = adapthisteq(I1(:,:,2));
I6(:,:,3) = adapthisteq(I1(:,:,3));

subplot(2,2,1), imshow(I1), title('Original Image')
subplot(2,2,3), imhist(I1(:,:,1)), title('Original Image')
subplot(2,2,2), imshow(I6), title('Enhanced Image')
subplot(2,2,4), imhist(I6(:,:,1)), title('Enhanced Image')
```

The result looks slightly better than that obtained using histeq. However, all three functions for image enhancement, imadjust, histeq and adapthisteq, provide numerous ways to manipulate the final outcome. The *Image Processing Toolbox – User's Guide* (MathWorks 2014) and the excellent book by Gonzalez and others (2009) provide more detailed introductions to the use of the various parameters available and the corresponding values of the image enhancement functions.

The Image Processing Toolbox also includes numerous functions for 2D filtering of images. Many of the methods we have looked at in Chapter 6 for one-dimensional data also work with two-dimensional data, as we have already seen in Chapter 7 when filtering digital terrain models. The most popular 2D filters for images are Gaussian filters and median filters, as well as filters for image sharpening. Both Gaussian and median filters are used to smooth an image, mostly with the aim of reducing the amount of noise.

www.it-ebooks.info

In most examples the signal-to-noise ratio is unknown and adaptive filters (similar to those introduced in Section 6.10) are therefore used for noise reduction. A Gaussian filter can be designed using

```
h = fspecial('gaussian',20,10);
I7 = imfilter(I1,h);
```

where `fspecial` creates predefined 2D filters, such as moving average, disk, or Gaussian filters. The Gaussian filter weights `h` are calculated using `fspecial('gaussian',20,10)`, where `20` corresponds the size of a 20-by-20 pixel filter following the shape of a Gaussian probability density function with a standard devation of `10`. Next, we calculate `I8`, which is a median-filtered version of `I1`.

```
I8(:,:,1) = medfilt2(I1(:,:,1),[20 20]);
I8(:,:,2) = medfilt2(I1(:,:,2),[20 20]);
I8(:,:,3) = medfilt2(I1(:,:,3),[20 20]);
```

Since `medfilt2` only works for two-dimensional data, we again apply the filter separately to each color (red, green and blue). The filter output pixels are the medians of the 20-by-20 neighborhoods around the corresponding pixels in the input image.

The third filter example deals with sharpening an image using `imsharpen`.

```
I9 = imsharpen(I1);
```

This function calculates the Gaussian lowpass filtered version of the image that is used as an unsharp mask, i.e., the sharpened version of the image is calculated by subtracting the blurred filtered version from the original image. The function comes with several parameters that control the ability of the Gaussian filter to blur the image and the strength of the sharpening effect, and a threshold specifying the minimum contrast required for a pixel to be considered an edge pixel and sharpened by unsharp masking. Comparing the results of the three filtering exercises with the original image

```
subplot(2,2,1), imshow(I1), title('Original Image')
subplot(2,2,2), imshow(I7), title('Gaussian Filter')
subplot(2,2,3), imshow(I8), title('Median Filter')
subplot(2,2,4), imshow(I9), title('Sharpening Filter')
```

clearly demonstrates the effect of the 2D filters. As an alternative to these time-domain filters, we can also design 2D filters with a specific frequency response, such as the 1D filters described in Section 6.9. Again, the book by Gonzalez and others (2009) provides an overview of 2D frequency-selective filtering for images, including functions used to generate such filters. The authors also demonstrate the use of a 2D Butterworth lowpass filter in image

www.it-ebooks.info

processing applications.

We next rectify the image, i.e., we correct the image distortion by transforming it to a rectangular coordinate system using a script that is similar to that used for georeferencing satellite images in Section 8.5. This we achieve by defining four points within the image, which are actually at the corners of a rectangular area (which is our reference area). We first define the *upper left*, *lower left*, *upper right*, and *lower right* corners of the reference area, and then press return. Note that it is important to pick the coordinates of the corners in this particular order. In this instance we use the original image `I1`, but we could also use any other enhanced version of the image from the previous exercises. As an example we can click the left side of the ruler at 1.5 cm and at 4.5 cm, where two thin white sediment layers cross the ruler, for use as the upper-left and lower-left corners. We then choose the upper-right and lower-right corners, further to the right of the ruler but also lying on the same two white sediment layers,

```
imshow(I1)
basepoints = ginput
```

and click return which yields

```
basepoints =
  517.0644  508.9059
  511.5396  733.5792
  863.2822  519.9554
  859.5990  739.1040
```

or any similar values. The image and the reference points are then displayed in the same figure window.

```
close all
imshow(I1)
hold on
line(basepoints(:,1),basepoints(:,2),...
       'LineStyle','none',...
       'Marker','+',...
       'MarkerSize',48,...
       'Color','b')
hold off
```

We arbitrarily choose new coordinates for the four reference points, which are now on the corners of a rectangle. To preserve the aspect ratio of the image, we select numbers that are the means of the differences between the x- and y-values of the reference points in `basepoints`.

```
dx = (basepoints(3,1)+basepoints(4,1))/2- ...
     (basepoints(1,1)+basepoints(2,1))/2
```

www.it-ebooks.info

```
dy = (basepoints(2,2)+basepoints(4,2))/2- ...
     (basepoints(1,2)+basepoints(3,2))/2
```

which yields

```
dx =
  347.1386

dy =
  221.9109
```

We therefore choose

```
inputpoints(1,:) = [0 0];
inputpoints(2,:) = [0 dy];
inputpoints(3,:) = [dx 0];
inputpoints(4,:) = [dx dy];
```

The function `fitgeotrans` now takes the pairs of control points, `inputpoints` and `basepoints`, and uses them to infer a spatial transformation matrix `tform` using the transformation type `projective`.

```
tform = fitgeotrans(inputpoints,basepoints,'projective');
```

We next need to estimate the spatial limits for the output, `XBounds` and `YBounds`, corresponding to the projective transformation `tform`, and a set of spatial limits for the input `xLimitsIn` and `yLimitsIn`.

```
xLimitsIn = 0.5 + [0 size(I1,2)]
yLimitsIn = 0.5 + [0 size(I1,1)]

[XBounds,YBounds] = outputLimits(tform,xLimitsIn,yLimitsIn)
```

Then we use `imref2d` to reference the image to world coordinates.

```
Rout = imref2d(size(I1),XBounds,YBounds)
```

An `imref2d` object `Rout` encapsulates the relationship between the intrinsic coordinates anchored to the rows and columns of the image and the spatial location of the same row and column locations in a world coordinate system. Finally, the projective transformation can be applied to the original RGB composite `I1` in order to obtain a rectified version of the image (`I10`).

```
I10 = imwarp(I1,tform,'OutputView',Rout);
```

We now compare the original image `I1` with the rectified version `I10`.

```
subplot(2,1,1), imshow(I1), title('Original Image')
subplot(2,1,2), imshow(I10), title('Rectified Image')
```

www.it-ebooks.info

We see that the rectified image has black areas at the corners. We can remove these black areas by cropping the image using `imcrop`.

```
I11 = imcrop(I10);

subplot(2,1,1), imshow(I1), title('Original Image')
subplot(2,1,2), imshow(I11), title('Rectified and Cropped Image')
```

The function `imcrop` creates displays of the image with a resizable rectangular tool that can be interactively positioned and manipulated using the mouse. After manipulating the tool into the desired position, the image is cropped by either double clicking on the tool or choosing *Crop Image* from the tool's context menu. The result of our image enhancement experiment can now be used in the next section to analyze the colors of individual sediment layers.

8.9 Color-Intensity Transects Across Varved Sediments

High-resolution core logging has, since the early 1990s, become popular as an inexpensive tool for investigating the physical and chemical properties of marine and lacustrine sediments. During the early days of nondestructive core logging, magnetic susceptibility and grayscale intensity transects were measured on board research vessels to generate a preliminary stratigraphy of marine cores, since the cyclic recurrence of light and dark layers seemed to reflect glacial-interglacial cycles during the Pleistocene. Paleolimnologists adopted these techniques to analyze annually-layered (varved) lake sediments and to statistically detect short-term variabilities such as the 11 year sunspot cycle, the 3-7 year El Niño cycle, or the 78 year Gleissberg cycle. Modern multi-sensor core loggers are now designed to log a great variety of physical and chemical properties using optical scanners, radiograph imaging, and x-ray fluorescence elemental analyzers.

As an example we explore varved sediments deposited around 33 kyrs ago in a landslide-dammed lake in the Quebrada de Cafayate of Argentina (Trauth et al. 1999, 2003). These lake sediments have been intensively studied for paleoclimate reconstructions since they document episodes of a wetter and more variable climate that eventually fostered mass movements in the NW Argentine Andes during the Late Pleistocene and Holocene. Aside from various sedimentological, geochemical and micropaleontological analyses, the colors of the sediments have been studied as a proxy for rainfall intensities at the time of deposition. Color-intensity transects were analyzed to detect interannual variations in precipitation caused by the El Niño/Southern Oscillation (ENSO, 3–7 year cycles) and the Tropical Atlantic Sea-Surface Temperature Variability (TAV, 10–15 year cycles) using linear and nonlinear

www.it-ebooks.info

methods of time-series analysis (e.g., Trauth et al. 2000, Marwan et al. 2003).

The El Paso section in the Quebrada de Cafayate contains well-developed annual layers in most parts of the profile (Fig. 8.7). The base of each of these mixed clastic and biogenic varves consists of reddish silt and clay, with a sharp lower boundary. Towards the top of the varves, reddish clay and silt are gradually replaced by light-brown to greenish clay. The change from reddish hues correlates with a slight decrease in grain size. This clastic portion of the varves is capped by a thin layer of pure white diatomite. Diatomites are sediments comprised mainly of porous siliceous skeletons of single-cell algae, i.e. diatoms. This internal structure of the laminae is typical of annual-layered sediments. The recurrence of these layers and the distribution of diatoms, together with the sediment coloration and provenance, all provide additional evidence that rhythmic sedimentation in this region was controlled by a well-defined annual cycle. The provenance of the sediments contained in the varved layers can be traced using index minerals characteristic of the various possible source areas within the catchment. A comparison of the mineral assemblages in the sediments with those of potential source rocks within the catchment area indicates that Fe-rich Tertiary sedimentary rocks

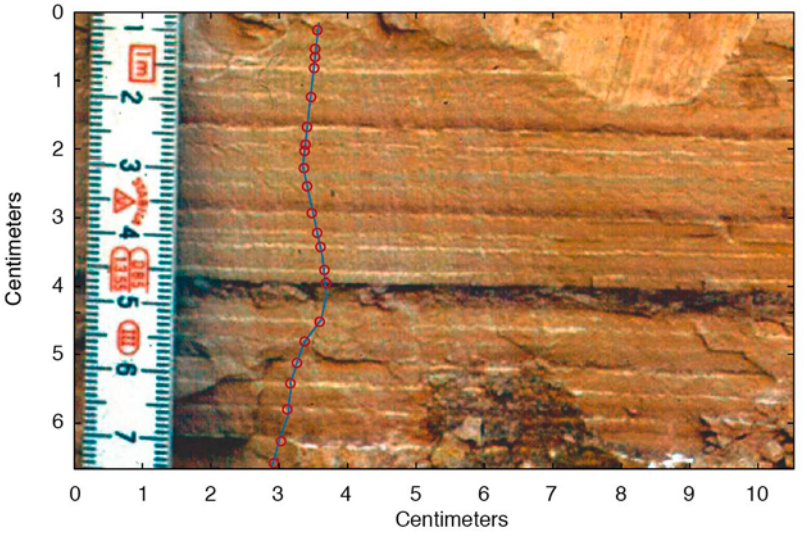


Fig. 8.7 Photograph of varved lake sediments from the Quebrada de Cafayate in the Santa Maria Basin, with cyclic occurrences of intense dark-red coloration reflecting enhanced precipitation and sediment input during ENSO- and TAV-type periodicities (350 cm above the base of the El Paso section). The solid blue line denotes the course of the digitized color-intensity transect. The red circles note the position of the diatomite layers, representing annual layers.

www.it-ebooks.info

exposed in the Santa Maria Basin were the source of the red-colored basal portion of the varves. In contrast, metamorphic rocks in the mountainous parts of the catchment area were the most likely source of the drab-colored upper part of the varves.

In nearly every second to fifth, and every tenth to fourteenth varve, the varve thickness increases by a factor of 1.5 to 2 and the basal reddish coloration is more intense, suggesting a greater fluvial input of Fe-rich Tertiary material. Exceptionally well-preserved sections containing 70–250 varves were used for more detailed statistical analysis of the observed cyclicities (see Chapter 5). High-quality photographs from these sections were scanned and subjected to standardized color and illumination corrections. Pixel-wide representative red-color intensities were subsequently extracted from transects across the images of these varves. The resolution of these time series was of the order of ten intensity values per varve.

We will now analyze a 22-year package of varved lake sediments from the Quebrada de Cafayate as an example (Fig. 8.6). The photo was taking during a field expedition in the late 1990s using an analog camera. It was then scanned and the contrast levels adjusted to heighten details using standard photo processing software. We import the image from the file *varves_original.tif* as a 24-bit RGB image array and store the data in a variable `I`.

```
clear

I = imread('varves_original');
```

We display the image using `imshow` and turn the axis labeling, tick marks and background back on.

```
imshow(I), axis on
```

The image is scaled to pixel indices or coordinates, so we first need to scale the image to a centimeter scale. While keeping the figure window open we use `ginput` to count the number of pixels per centimeter. The function `ginput` provides a crosshair with which to gather an unlimited number of points forming a polygon, until the *return* key is pressed. We place the crosshair at 1 cm and 6 cm on the scale in the image and click to gather the pixel coordinates of the 5-cm interval.

```
[x,y] = ginput;
```

The image is `size(I,2)=1830` pixels wide and `size(I,1)=1159` pixels high. We convert the width and the height of the image into centimeters using the conversion `5/sqrt((y(2,1)-y(1,1))^2+(x(2,1)-x(1,1))^2)`, where `5` corres-

www.it-ebooks.info

ponds to the 5 cm interval equivalent to the `sqrt((y(2,1)-y(1,1))^2+(x(2,1)-x(1,1))^2)` pixels gathered using `ginput`.

```
ix = 5 * size(I,2) / sqrt((y(2,1)-y(1,1))^2+(x(2,1)-x(1,1))^2);
iy = 5 * size(I,1) / sqrt((y(2,1)-y(1,1))^2+(x(2,1)-x(1,1))^2);
```

We can now display the image using the new coordinate system where `ix` and `iy` are the width and height of the image, respectively (in centimeters).

```
imshow(I,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeters'), ylabel('Centimeters')
```

We now digitize the color-intensity transect from the top of the image to bottom. The function `improfile` determines the RGB pixel values `C` along line segments defined by the coordinates `[CX,CY]`.

```
[CX,CY,C] = improfile;
```

The scaled image and the polygon are displayed in the same figure window. The three color-intensity curves are plotted in a separate window.

```
imshow(I,'XData',[0 ix],'YData',[0 iy]), hold on
plot(CX,CY), hold off

figure
plot(CY,C(:,1),'r',CY,C(:,2),'g',CY,C(:,3),'b')
xlabel('Centimeters'), ylabel('Intensity')
```

The image and the color-intensity profiles are on a centimeter scale. To detect the interannual precipitation variability, as recorded in the color intensity of the sediments, we need to convert the length scale to a time scale. We use the 22 white diatomite layers as time markers to define individual years in the sedimentary history. We again use `ginput` to mark the diatomite layers from top to bottom along the color-intensity transect and store the coordinates of the laminae in the new variable `laminae`.

```
imshow(I,'XData',[0 ix],'YData',[0 iy]), hold on
plot(CX,CY), hold off
laminae = ginput;
```

To inspect the quality of the age model we plot the image, together with the polygon and the marked diatomite layers.

```
imshow(I,'XData',[0 ix],'YData',[0 iy])
hold on
plot(CX,CY)
plot(laminae(:,1),laminae(:,2),'ro')
xlabel('Centimeters'), ylabel('Centimeters')
hold off
```

www.it-ebooks.info

We define a new variable `newlaminae` that contains the vertical y-component of `laminae` as the first column and the years 1 to 22 (counting backwards in time) as the second colum. The 22 years are equivalent to the length of `laminae`. The function `interp1` is used to interpolate the color-intensity transects over an evenly-spaced time axis stored in the variable `age`.

```
newlaminae(:,1) = laminae(:,2);
newlaminae(:,2) = 1 : length(laminae(:,2));
age = interp1(newlaminae(:,1),newlaminae(:,2),CY);
```

We complete the analysis by plotting the color-intensity curves on both a length and a time scale for comparison (Fig. 8.8).

```
subplot(2,1,1), plot(CY,C(:,1),CY,C(:,2),CY,C(:,3))
xlabel('Centimeters'), ylabel('Intensity'), title('Color vs. Length')
subplot(2,1,2), plot(age,C(:,1),age,C(:,2),age,C(:,3))
xlabel('Years'), ylabel('Intensity'), title('Color vs. Age')
```

The interpolated color-intensity transects can now be further analyzed using the time-series analysis tools. The analysis of a representative red-color intensity transect across 70–250 varves during the project described

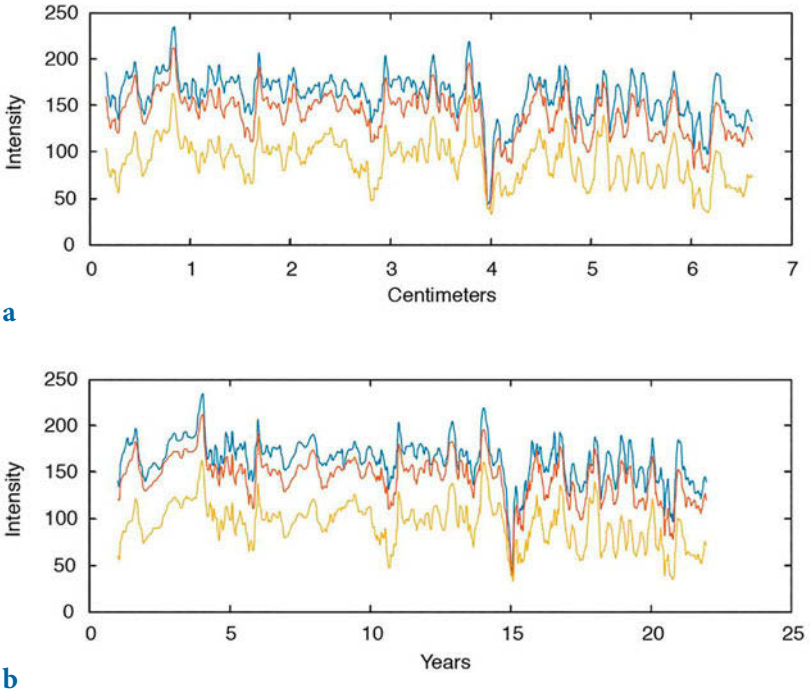


Fig. 8.8 Color-intensity curves (red, green and blue) plotted against **a** depth and **b** age.

www.it-ebooks.info

above revealed significant peaks at about 13.1, 3.2, 2.2 and 1.0 yrs, suggesting both ENSO and TAV influences in the area at around 33,000 years ago (see Chapter 5 and Fig. 5.1).

8.10 Grain Size Analysis from Microscope Images

Identifying, measuring and counting particles in an image are the classic applications of image analysis. Examples from the geosciences include grain size analysis, counting pollen grains, and determining the mineral composition of rocks from thin sections. For grain size analysis the task is to identify individual particles, measure their sizes, and then count the number of particles per size class. The motivation to use image analysis is the ability to perform automated analyses of large sets of samples in a short period of time and at relatively low costs. Three different approaches are commonly used to identify and count objects in an image: (1) region-based segmentation using the watershed segmentation algorithm, (2) object detection using the Hough transform, and (3) thresholding using color differences to separate objects. Gonzalez, Woods and Eddins (2009) describe these methods in great detail in the 2nd edition of their excellent book, which also provides numerous MATLAB recipes for image processing. The book has a companion webpage at

```
http://www.imageprocessingplace.com/
```

that offers additional support in a number of important areas (including classroom presentations, M-files, and sample images) as well as providing numerous links to other educational resources. We will use two examples to demonstrate the use of image processing for identifying, measuring, and counting particles. In this section we will demonstrate an application of watershed segmentation in grain size analysis and then in Section 8.9 we will introduce thresholding as a method for quantifying charcoal in microscope images. Both applications are implemented in the MATLAB-based RADIUS software developed by Klemens Seelos from the University of Mainz (Seelos and Sirocko 2005). RADIUS is a particle-size measurement technique, based on the evaluation of digital images from thin sections that offers a sub-mm sample resolution and allows sedimentation processes to be studied within the medium silt to coarse sand size range. It is coupled with an automatic pattern recognition system for identifying sedimentation processes within undisturbed samples. The MATLAB code for RADIUS can be downloaded from

```
http://www.particle-analysis.info/
```

www.it-ebooks.info

The following example for object segmentation illustrates the segmentation, measuring, and counting of objects using the watershed segmentation algorithm (Fig. 8.9). We first read an image of coarse lithic grains of different sizes and store it in the variable `I1`. The size of the image is 284-by-367 pixels and, since the width is 3 cm, the height is 3 cm·284/367=2.32 cm.

```
clear

I1 = imread('grainsize.tif');
ix = 3; iy = 284 * 3 / 367;
imshow(I1,'XData',[0 ix],'YData',[0 iy])
title('Original Image')
```

Here, `ix` and `iy` denote the coordinate axes used to calibrate the image `I1` to a centimeter scale. The true number of objects counted in this image is 236, including three grains that overlap the borders of the image and will therefore be ignored in the following exercise. We reject the color information of the image and convert `I1` to grayscale using the function `rgb2gray`.

```
I2 = rgb2gray(I1);
imshow(I2,'XData',[0 ix],'YData',[0 iy])
title('Grayscale Image')
```

This grayscale image `I2` has a relatively low level of contrast. We therefore use the function `imadjust` to adjust the image intensity values. The function `imadjust` maps the values in the intensity image `I2` to new values in `I3`, such that 1% of the data is saturated at low and high intensities. This increases the contrast in the new image `I3`.

```
I3 = imadjust(I2);
imshow(I3,'XData',[0 ix],'YData',[0 iy])
title('Adjusted Intensity Values')
```

We next determine the background of the `I3` image, which means basically the texture of the black foil on which the grains are located. The function `imopen(im,se)` determines objects in an image `im` with a specific shape `se` (a flat structuring element such as a circular disk) and size (expressed as a specific number of pixels), as defined by the function `strel`. We then produce a background-free image, `I4`.

```
I4 = imopen(I3,strel('disk',1));
imshow(I4,'XData',[0 ix],'YData',[0 iy])
title('No Background')
```

We subtract the background-free image `I4` from the original grayscale image `I3` to observe the background `I5` that has been eliminated.

www.it-ebooks.info

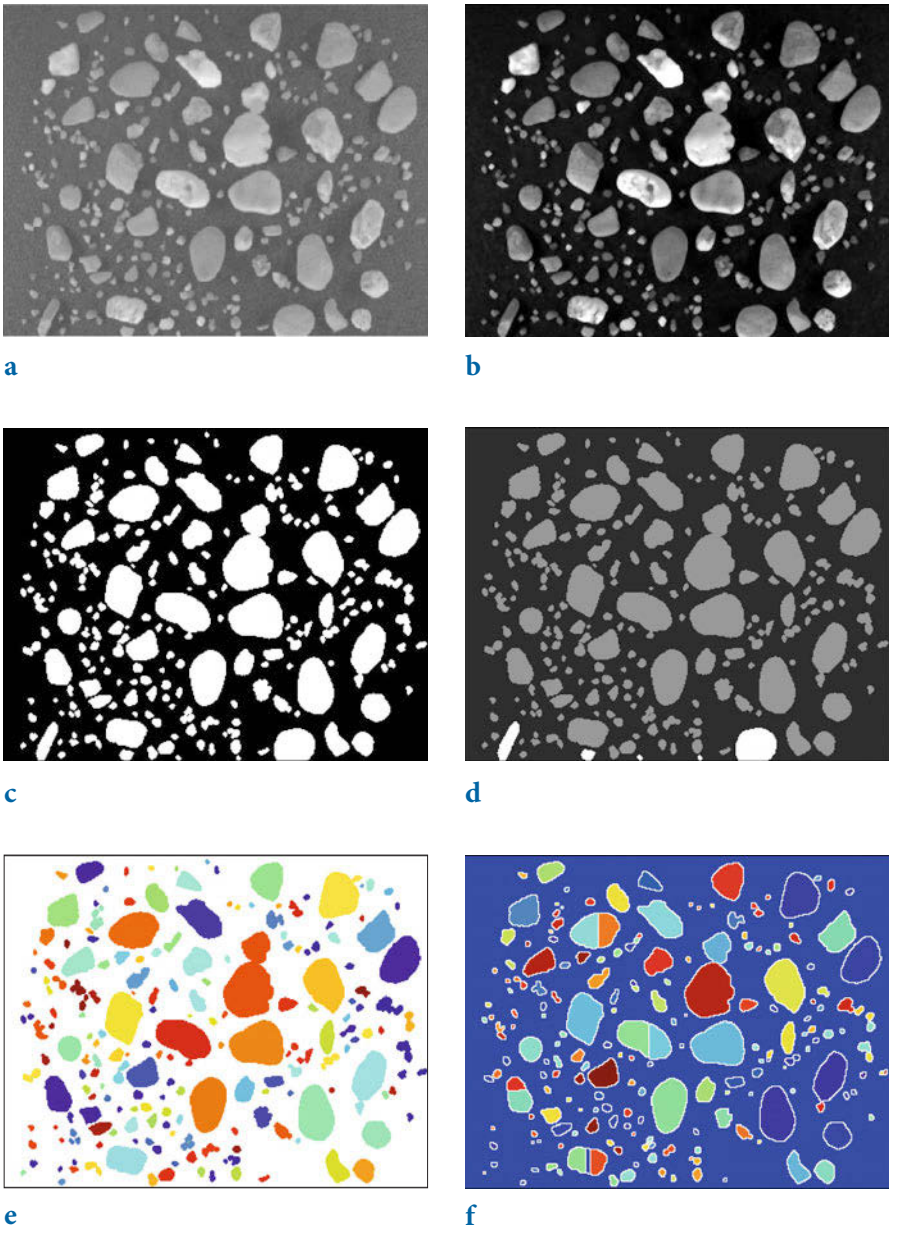


Fig. 8.9 Results from automated grain size analysis of a microscope image; **a** original grayscale image, **b** image after removal of background, **c** image after conversion to binary image, **d** image after eliminating objects overlapping the image border, **e** image with objects detected by tracing the boundaries of connected pixels, and **f** image with objects detected using a watershed segmentation algorithm.

www.it-ebooks.info

```
I5 = imsubtract(I3,I4);
imshow(I5,'XData',[0 ix],'YData',[0 iy])
title('Background')
```

The function `im2bw` converts the background-free image `I4` to a binary image `I6` by thresholding. If the threshold is 1.0 the image is all black, corresponding to the pixel value of 0. If the threshold is 0.0 the image is all white, corresponding to a pixel value of 1. We manually change the threshold value until we get a reasonable result and find 0.2 to be a suitable threshold.

```
I6 = im2bw(I4,0.2);
imshow(I6,'XData',[0 ix],'YData',[0 iy])
title('Binary Image')
```

We next eliminate objects in `I6` that overlap the image border, since they are actually larger than shown in the image and will result in false estimates. We eliminate these using `imclearborder` and generate image `I7`.

```
I7 = imclearborder(I6);
himage1 = imshow(I6,'XData',[0 ix],'YData',[0 iy]); hold on
set(himage1, 'AlphaData', 0.7);
himage2 = imshow(imsubtract(I6,I7),'XData',[0 ix],'YData',[0 iy]);
set(himage2, 'AlphaData', 0.4);
title('Image Border'), hold off
```

We then trace the boundaries using `bwboundaries` in a binary image where non-zero pixels belong to an object and zero pixels are background. By default, the function also traces the boundaries of holes in the `I7` image. We therefore choose the option `noholes` to suppress the tracing of the holes. Function `label2grb` converts the label matrix `L` resulting from `bwboundaries` to an RGB image. We use the colormap `jet`, the zerocolor `w` for `white`, and the color order `shuffle` (which simply shuffles the colors of `jet` in a pseudorandom manner).

```
[B,L] = bwboundaries(I7,'noholes');
imshow(label2rgb(L,@jet,'w','shuffle'),...
   'XData',[0 ix],'YData',[0 iy])
title('Define Objects')
```

The function `bwlabeln` is used to obtain the number of connected objects found in the binary image. The integer 8 defines the desired connectivity, which can be either 4 or 8 in two-dimensional neighborhoods. The elements of `L` are integer values greater than or equal to 0. The pixels labeled 0 are the background. The pixels labeled 1 make up one object, the pixels labeled 2 make up a second object, and so on.

```
[labeled,numObjects] = bwlabeln(I7,8);
```

www.it-ebooks.info

```
numObjects
```

In our example the method identified 192 grains, which is significantly less than the 236 grains counted manually, reduced by the three objects that overlap the borders of the image. Visual inspection of the color-coded image generated by `bwboundaries` reveals the reason for the underestimated number of grains. Two large grains in the middle of the image have been observed as being connected, giving a single, very large grain in the final result. Reducing the disk size with `strel` from `disk=1` to `disk=5` can help separate connected grains. Larger disks, on the other hand, reduce the number of grains because smaller grains are lost by filtering. We now determine the areas of each of the grains.

```
graindata = regionprops(labeled,'basic');
grainareas= [graindata(:).Area];
objectareas = 3^2 * grainareas * 367^(-2);
```

We then find the maximum, minimum and mean areas for all grains in the image, in cm^2.

```
max_area = max(objectareas)
min_area = min(objectareas)
mean_area = mean(objectareas)
```

The connected grain in the middle of the image has a size of 0.16 cm^2, which represents the maximum size of all grains in the image. Finally, we plot the histogram of all the grain areas.

```
clf
e = 0 : 0.0005 : 0.15;
histogram(objectareas,e)
xlabel('Grain Size in Millimeters^2')
ylabel('Number of Grains')
axis([0 0.1 0 30])
```

Several methods exist that partly overcome the artifact of connected grains in grain size analyses. The most popular technique for region-based segmentation is the watershed segmentation algorithm. Watersheds in geomorphology are ridges that divide areas contributing to the hydrological budget of adjacent catchments (see Section 7.10). Watershed segmentation applies to grayscale images the same methods used to separate catchments in digital elevation models. In this application, the grayscale values are interpreted as elevations in a digital elevation model, where the watershed then separates the two objects of interest.

The criterion commonly used to identify pixels that belong to one object is the nearest-neighbor distance. We use the distance transform performed

www.it-ebooks.info

by `bwdist`, which assigns to each pixel a number that is the distance between a pixel and the nearest non-zero pixel in `I7`. In an image in which objects are identified by pixel values of zero and the background by pixel values of one, the distance transform has zeros in the background areas and non-zero values that increase progressively with increasing distances from the edges of the objects. In our example however, the objects have pixel values of one and the background has pixels with with values of zero. We therefore have to apply `bwdist` to the complement of the binary image `I7` instead of to the image itself.

```
D = bwdist(~I7,'cityblock');
```

The function `bwdist` provides several methods for computing the nearest-neighbor distances, including *Euclidean distances*, *cityblock distances*, *chessboard distances* and *quasi-Euclidean distances*. We choose the *cityblock* option in this particular example, but other methods might be more appropriate for separating objects in other images. The distance matrix now contains positive non-zero values in the object pixels and zeros elsewhere. We then complement the distance transform, and ascribe a value of `-Inf` to each pixel that does not belong to an object.

```
D = -D;
D(~I7) = -Inf;
```

We compute the watershed transform for the distance matrix, and display the resulting label matrix.

```
L2 = watershed(D);
imshow(label2rgb(L2,@jet,'w','shuffle'),...
   'XData',[0 ix],'YData',[0 iy])
title('Watershed Segmentation')
```

After having displayed the results from watershed segmentation, we determine the number of pixels for each object using the recipe from above, except for index `i` running from `2` to `max(objects)` since the value `1` denotes the background and `0` denotes the boundaries of the objects. The first true object is therefore marked by the value of `2`.

```
objects = sortrows(L2(:),1);
max(objects)
clear objectsizes
for i = 2 : max(objects)
    clear individualobject
    individualobject = objects(objects == i);
    objectsizes(i) = length(individualobject);
end
objectsizes = objectsizes';
```

www.it-ebooks.info

```
objectsizes = sortrows(objectsizes,1);
objectsizes = objectsizes(objectsizes~=0);
```

We have now recognized 205 objects, i.e., more objects than were identified in the previous exercise without watershed segmentation. Visual inspection of the result, however, reveals some oversegmentation (due to noise or other irregularities in the image) in which larger grains are divided into smaller pieces. On the other hand, very small grains have been eliminated by filtering the image with the morphological structuring element `strel`. We scale the object sizes. The area of one pixel is $(3\ \text{cm}/367)^2$.

```
objectareas = 3^2 * objectsizes * 367^(-2);
```

We now determine the areas for each of the grains. We again find the maximum, minimum and mean areas for all grains in the image, in cm^2.

```
max_area = max(objectareas)
min_area = min(objectareas)
mean_area = mean(objectareas)
```

The largest grain in the center of the image has a size of 0.09 cm^2, which represents the maximum size of all grains in the image. Finally, we plot the histogram of all the grain areas.

```
clf
e = 0 : 0.0005 : 0.15;
histogram(objectareas,e)
xlabel('Grain Size in Millimeters^2'),...
   ylabel('Number of Grains')
axis([0 0.1 0 70])
```

As a check of the final result we digitize the outline of one of the larger grains and store the polygon in the variable `data`.

```
figure
imshow(I1,'XData',[0 ix],'YData',[0 iy])
data = ginput;
```

We close the polygon by copying the first row of coordinates to the end of the array. We then display the polygon on the original image.

```
data(end+1,:) = data(1,:)

imshow(I1,'XData',[0 ix],'YData',[0 iy]), hold on
plot(data(:,1),data(:,2)), hold off
```

The function `polyarea` yields the area of the large grain.

```
polyarea(data(:,1),data(:,2))
```

www.it-ebooks.info

```
ans =
    0.0951
```

The calculated area corresponds approximately to the result from the grain size analysis. If oversegmentation is a major problem when using segmentation to count objects in an image, the reader is referred to the book by Gonzalez, Woods and Eddins (2009) that describes marker-controlled watershed segmentation as an alternative method to avoid oversegmentation.

8.11 Quantifying Charcoal in Microscope Images

Quantifying the composition of substances in geosciences, such as the mineral composition of a rock in thin sections, or the amount of charcoal in sieved sediment samples, is facilitated by the use of image processing methods. Thresholding provides a simple solution to segmenting objects within an image that have different coloration or grayscale values. During the thresholding process, pixels with an intensity value greater than a threshold value are marked as object pixels (e.g., pixels representing charcoal in an image) and the rest as background pixels (e.g., all other substances). The threshold value is usually defined manually through visual inspection of the image histogram, but numerous automated algorithms are also available.

As an example we analyze an image of a sieved lake-sediment sample from Lake Nakuru, Kenya (Fig. 8.10). The image shows abundant light-gray oval ostracod shells and some mineral grains, as well as gray plant remains and black charcoal fragments. We use thresholding to separate the dark charcoal particles and count the pixels of these particles after segmentation. After having determined the number of pixels for all objects distinguished from the background by thresholding, we use a lower threshhold value to determine the ratio of the number of pixels representing charcoal to the number of pixels representing all particles in the sample, i.e., to determine the percentage of charcoal in the sample.

We read the image of size 1500-by-1500 pixels and assume that the width and the height of the square image are both one centimeter.

```
clear

I1 = imread('lakesediment.jpg');
ix = 1; iy = 1;
imshow(I1,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeter'), ylabel('Centimeter')
title('Original Image')
```

The RGB color image is then converted to a grayscale image using the function `rgb2gray`.

www.it-ebooks.info

```
I2 = rgb2gray(I1);
imshow(I2,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeters'), ylabel('Centimeters')
title('Grayscale')
```

Since the image contrast is relatively low, we use the function `imadjust` to adjust the image intensity values. The function `imadjust` maps the values in the intensity image `I1` to new values in `I2`, such that 1% of the data is saturated at low and high intensities of `I2`. This increases the contrast in the new image `I2`.

```
I3 = imadjust(I2);
imshow(I3,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeters'), ylabel('Centimeters')
title('Better Contrast')
```

We next determine the background of the lithic grains, which basically means the texture of the black foil on which the grains are located. The function `imopen(im,se)` determines objects in an image `im` below a certain pixel size and a flat structuring element `se`, such as a disk with a radius of 5 pixels generated by the function `strel`. The variable `I4` is the background-free image resulting from this operation.

```
I4 = imopen(I3,strel('disk',5));
imshow(I4,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeters'), ylabel('Centimeters')
title('W/O Background')
```

We subtract the background-free image `I4` from the original grayscale image `I3` to observe the background `I5` that has been eliminated.

```
I5 = imsubtract(I3,I4);
imshow(I5,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeters'), ylabel('Centimeters')
title('Background')
```

The function `im2bw` converts the `I4` image to a binary image (`I6`) by thresholding. If the threshold is 1.0 the image is all black, corresponding to a pixel value of 0. If the threshold is 0.0 the image is all white, corresponding to a pixel value of 1. We manually change the threshold value until we get a reasonable result. In our example a threshold of 0.03 gives good results for identifying charcoal fragments.

```
I6 = im2bw(I4,0.03);
imshow(I6,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeters'), ylabel('Centimeters')
title('Only Charcoal')
```

www.it-ebooks.info

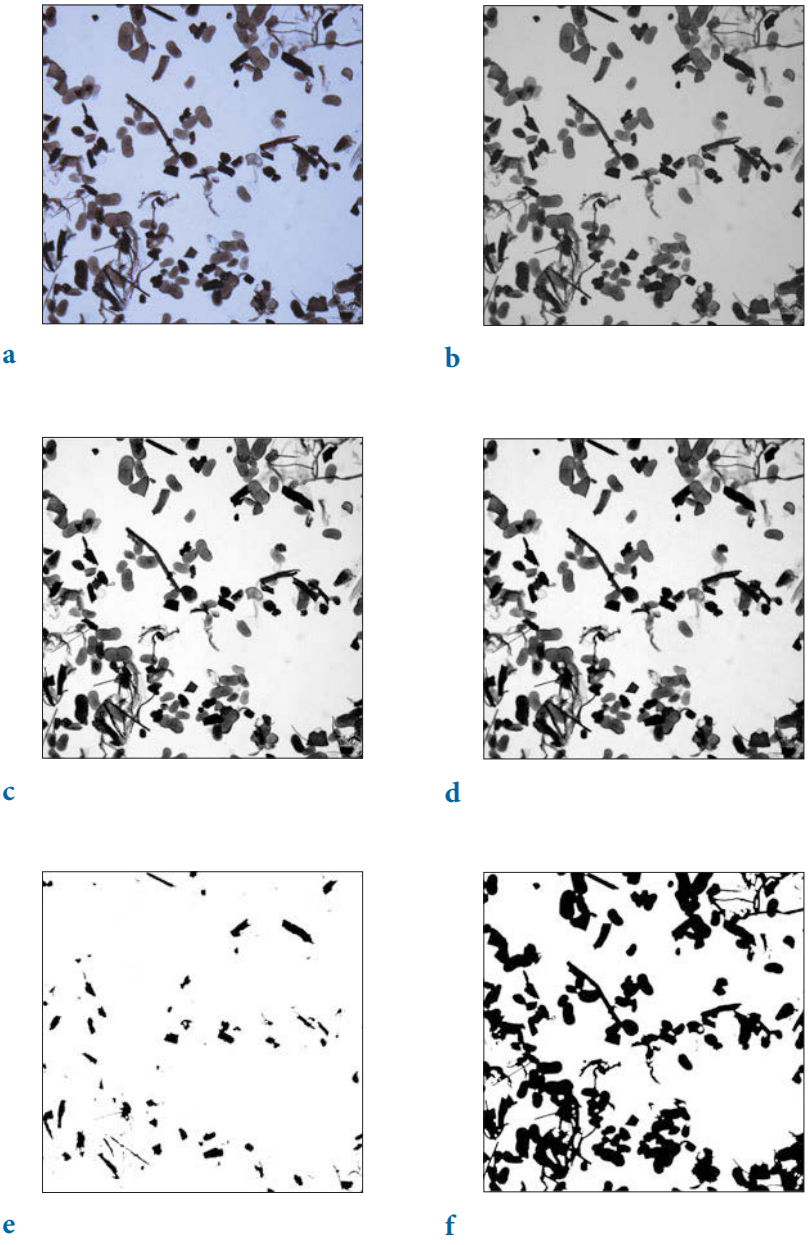


Fig. 8.10 Display of results from automatic quantification of charcoal in a microscope image; **a** original color image, **b** grayscale image, **c** image after enhancement of contrast, **d** image after removal of background, **e** image after thresholding to separate charcoal particles, and **f** image after thresholding to separate all objects.

www.it-ebooks.info

Since we know the size of a pixel we can now simply count the number of pixels to estimate the total amount of charcoal in the image. Finally, we compute the area of all objects, including charcoal.

```
I7 = im2bw(I4,0.6);
imshow(I7,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeters'), ylabel('Centimeters')
title('All Objects')
```

We are not interested in the absolute areas of charcoal in the image but in the percentage of charcoal in the sample.

```
100*sum(sum(I6==0))/sum(sum(I7==0))

ans =
   13.4063
```

The result suggests that approximately 13% of the sieved sample is charcoal. As a next step, we could quantify the other components in the sample, such as ostracods or mineral grains, by choosing different threshold values.

8.12 Shape-Based Object Detection in Images

The counting of objects within images on the basis of their shapes is a very time-consuming task. Examples of where this is carried out for round objects include the counting of planktonic foraminifera shells to infer past sea-surface temperatures, of diatom frustules to infer past chemical composition of lake water, and of pollen grains to determine assemblages that can be used to reconstruct regional air temperature and precipitation. Linear objects that are determined include faults in aerial photos and satellite images (to derive the present-day stress field of an area) and annual layers (varves) in thin sections (to establish an annually-resolved sedimentary history).

The Hough transform, named after the related 1962 patent of Paul VC Hough, is a popular technique with which to detect objects within images, based on their shapes. The Hough transform was orginally used to detect linear features, but soon after being patented it was generalized to identify objects of any shape (Duda and Hart 1972, Ballard 1981). The book by Gonzalez and others (2009) contains a comprehensive introduction to the Hough transform and its applications for detecting objects within images. According to their introduction to the method, the Hough transform is performed in two steps. In the first step an edge detector is used to extract edge features, such as distinct sediment layers or the outlines of pollen grains, from an image (Fig. 8.13). In the second step lines (or objects of any other shape) that trace these edge features are identified. The Image Processing

www.it-ebooks.info

Toolbox (MathWorks 2014) contains functions that use the Hough transform to detect lines or circular objects.

The classic Hough transform is used to detect lines in images. After applying an edge detector of any kind we end up with a binary image that has black pixels on the edges and white pixels in between. We next describe the lines through a given black pixel by the Euklidean distance ρ between the line and the origin, and by the angle θ of the vector from the origin to the closest point on the line (Fig. 8.11 a):

$$\rho = x\cos\theta + y\sin\theta$$

The family of all lines passing through this particular pixel (x_i,y_i) of an edge feature are displayed as a sinusoidal curve in the (θ,ρ) parameter space (Fig. 8.11 b). The intersection point (θ',ρ') of two such sinusoidal curves corresponds to the line that passes through two different pixels, (x_1,y_1) and (x_2,y_2), of an edge feature. Next, we search for n points (x_i,y_i) in the Hough transform where many lines intersect, since these are points defining the line tracing an edge feature. Detecting circles instead of lines works in a similar manner, using the coordinates of the center of the circle and its radius instead of ρ and θ.

For our a first example we use these functions to detect the thin layers of pure white diatomite within varved sediments exposed in the Quebrada de Cafayate of Argentina, which have already been used as examples in previous sections (Trauth et al. 1999, 2003) (Fig. 8.12). The quality of the image is not perfect, which is why we can not expect optimal results. We first read the cropped version of the laminated sediment from Section 8.8 and store it in the variable `I1`. The size of the image is 1,047-by-1,691 pixels, consisting of three colors (red, green and blue).

```
clear

I1 = imread('varves_cropped.tif');
imshow(I1,'InitialMagnification',30)
```

We reject the color information of the image and convert `I1` to grayscale using the function `rgb2gray`.

```
I2 = rgb2gray(I1);
imshow(I2,'InitialMagnification',30)
```

We then use `adapthisteq` to perform a contrast-limited adaptive histogram equalization (CLAHE), in order to enhance the contrast in the image (Zuiderveld 1994).

www.it-ebooks.info

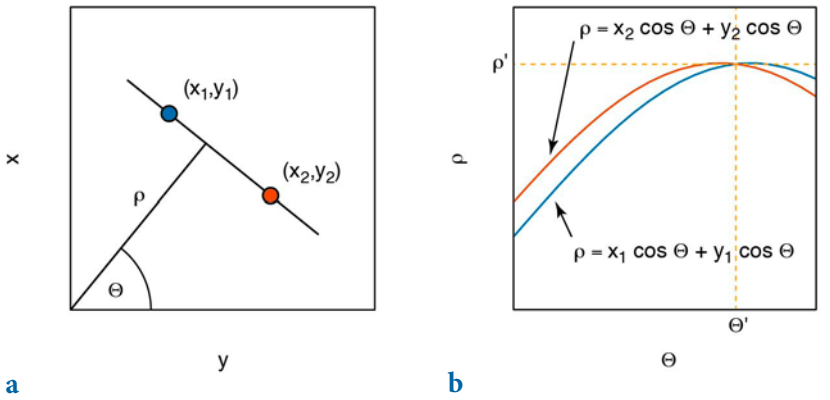


Fig. 8.11 The concept of the Hough transform: **a** parametrization of lines in the *xy*-plane, and **b** sinusoidal curves in the $p\theta$ parameter space, with the point of intersection corresponding to the line that passes through two different pixels of an edge feature (modified from Gonzales et al. 2009).

```
I3 = adapthisteq(I2,'ClipLimit',0.1,'Distribution','Rayleigh');
imshow(I3,'InitialMagnification',30)
```

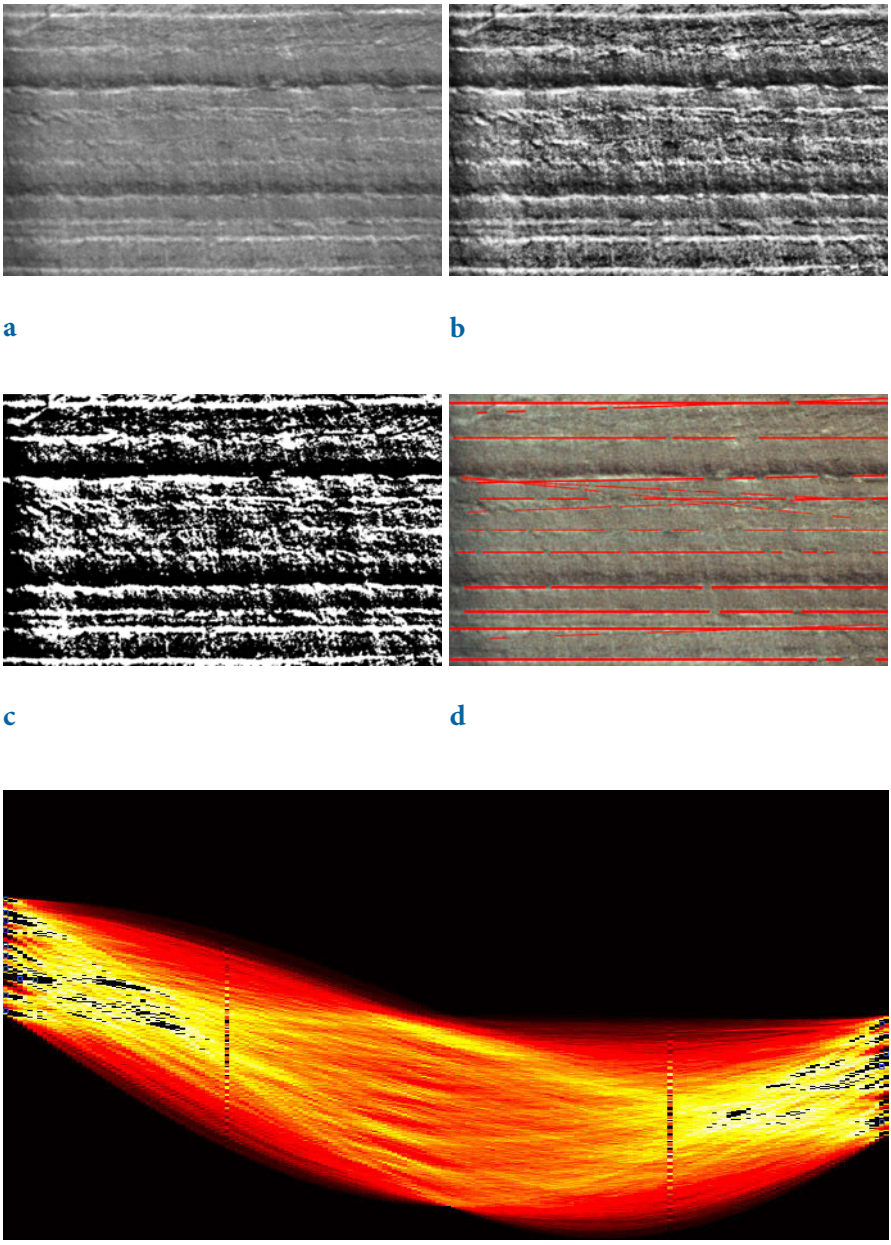
Here, `ClipLimit` limits the contrast enhancement using a real scalar from 0 to 1, with higher numbers resulting in greater contrast; the default value is 0.01. The `Distribution` parameter defines the desired histogram shape for the tiles by specifying a distribution type, such as `Uniform`, `Rayleigh` and `Exponential`. Using a `ClipLimit` of 0.1 and a `Rayleigh` distribution yields good results. Using the function `im2bw` then converts the `I3` image to a binary image (`I4`) by thresholding. If the threshold is 1.0 the image is all black, corresponding to the pixel value of 0. If the threshold is 0.0 the image is all white, corresponding to a pixel value of 1. We manually change the threshold value until we get a reasonable result and find 0.55 to be a suitable threshold.

```
I4 = im2bw(I3, 0.55);
imshow(I4,'InitialMagnification',30)
```

The function `hough` implements the Hough transform, `houghpeaks` finds the high-count accumulator cells in the Hough transform, and `houghlines` extracts lines in the original image, based on the other two functions. We determine the n=15 lines corresponding to the first 15 maxima of the Hough transform and store fifteen of the lines (lines 1 to 5, 6 to 10, and 11 to 15) in three separate variables `lines1`, `lines2` and `lines3`.

```
[H,theta,rho] = hough(I4);
peaks  = houghpeaks(H,15);
```

www.it-ebooks.info



a **b**

c **d**

e

Fig. 8.12 Automated detection of thin layers of pure white diatomite within varved sediments exposed in the Quebrada de Cafayate of Argentina, using `houghlines` (Trauth et al. 1999, 2003); **a** grayscale image, **b** enhanced image, **c** binary image, **d** image with diatomite layers marked by red lines, and **e** Hough transform of the image.

www.it-ebooks.info

```
lines1 = houghlines(I4,theta,rho,peaks(1:5,:));
lines2 = houghlines(I4,theta,rho,peaks(6:10,:));
lines3 = houghlines(I4,theta,rho,peaks(11:15,:));
```

We then display the Hough transform and mark the 15 maxima of the Hough transform with blue squares.

```
imshow(imadjust(mat2gray(H)),[], ...
    'XData',theta, ...
    'YData',rho, ...
    'InitialMagnification','fit')
colormap(hot), axis square, hold on
plot(theta(peaks(:,2)),rho(peaks(:,1)), ...
    'LineStyle','none', ...
    'Marker','s', ...
    'Color','b')
xlabel('\theta')
ylabel('\rho')
title('Hough Transform')
```

The variables `lines1`, `lines2` and `lines3` can now be used to display the lines on the image, with the line thickness decreasing from `lines3` to `lines1` depending on the rank of the lines in the Hough transform.

```
imshow(I1,'InitialMagnification',30), hold on
for k = 1:length(lines1)
xy = [lines1(k).point1; lines1(k).point2];
plot(xy(:,1),xy(:,2),'LineWidth',3,'Color',[1 0 0]);
end
hold on
for k = 1:length(lines2)
xy = [lines2(k).point1; lines2(k).point2];
plot(xy(:,1),xy(:,2),'LineWidth',2,'Color',[1 0 0]);
end
for k = 1:length(lines3)
xy = [lines3(k).point1; lines3(k).point2];
plot(xy(:,1),xy(:,2),'LineWidth',1,'Color',[1 0 0]);
end
```

The result shows that the clearly recognizable white layers are well detected whereas the less pronounced layers are not identified. The method also mistakenly marks non-existing lines on the image because of the low quality of the image. Using a better quality image and carefully adjusting the parameters used with the Hough transform will yield better results.

In the second example the Hough transform is used to automatically count pollen grains in a microscope image of Argentine honey (Fig. 8.13). The quality of the image is again not perfect, which is why we can not expect optimum results. In particular, the image of three-dimensional objects was taken with a very large magnification, so it is slightly blurred. We first read the pollen image and store it in the variable `I1`. The size of the image is 968-

www.it-ebooks.info

by-1,060 pixels of three colors (red, green and blue). Since the image is relatively large, we reduce the image size by a factor of two.

```
clear

I1 = imread('pollen.jpg');
I1 = I1(1:2:end,1:2:end,:);
imshow(I1,'InitialMagnification',100)
```

We reject the color information of the image and use the red color only.

```
I2 = I1(:,:,1);
imshow(I2,'InitialMagnification',100)
```

Next, we use `adapthisteq` to perform a contrast-limited adaptive histogram equalization (CLAHE) in order to enhance the contrast in the image (Zuiderveld 1994).

```
I3 = adapthisteq(I2);
imshow(I3,'InitialMagnification',100)
```

The function `imfindcircles` implements the Hough transform and extracts circles from the original image.

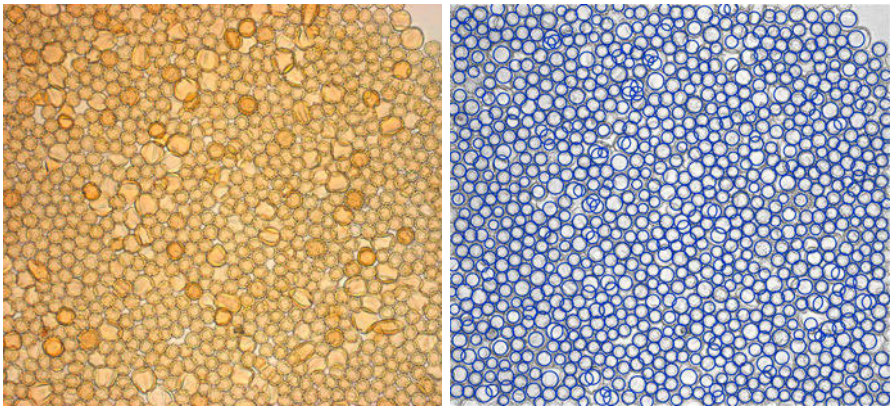
```
[centers,radii] = imfindcircles(I3,[12 20],...
    'Method','TwoStage',...
    'ObjectPolarity','Bright',...
    'Sensitivity',0.92,...
    'EdgeThreshold',0.20);
num = length(centers);
nstr = ['Number of Pollen Grains: ',num2str(num)];
```

Herein we use the `TwoStage` detection method for a two-stage circular Hough transform, following the procedure described by Yuen et al. (1990) and Davies (2005). The object polarity is set to `bright` as we are looking for bright rather than dark objects in our image. The sensitivity of `0.92` and the edge threshold of `0.20` are found by trial and error. We then use `viscircles` to display the circles on the grayscale image `I2`

```
imshow(I2,'InitialMagnification',100)
viscircles(centers, radii,'EdgeColor','b')
title(nstr)
```

using the output `centers` and `radii` from `imfindcircles`. The edge color of the circles in the graphics is set to `b` for blue. The result shows that we have counted 884 pollen grains with the method. The algorithm identifies the majority of the objects, but some are not recognized and some of the larger objects are mistakenly identified as two or more pollen grains. Using a better

www.it-ebooks.info



a **b**

Fig. 8.13 Automated detection of pollen grains (mostly *Asteraceae* and less abundant *Caesalpiniaceae* and *Lamiaceae* pollen) in a microscope image of Argentine honey, using `imfindcircles` (original image courtesy K Schittek and F Flores); **a** original RGB image, **b** pollen grains detected using the Hough transform.

quality image and carefully adjusting the parameters used with the Hough transform will yield better results. Plotting the histogram of the pollen radii using `histogram`

```
histogram(radii)
```

reveals that most of the grains have a radius of around 15 pixels.

Recommended Reading

Abrams M, Hook S (2002) ASTER User Handbook – Version 2. Jet Propulsion Laboratory and EROS Data Center, Sioux Falls

Ballard DH (1981) Generalizing the Houghtransform to detectarbitraryshapes. Pattern Recognition 13:111–122

Barry P (2001) EO-1/Hyperion Science Data User's Guide. TRW Space, Defense & Information Systems, Redondo Beach, CA

Beck R (2003) EO-1 User Guide. USGS Earth Resources Observation Systems Data Center (EDC), Sioux Falls, SD

Campbell JB (2002) Introduction to Remote Sensing. Taylor & Francis, London

Davies ER (2005) Machine Vision: Theory, Algorithms, Practicalities – 3rd Edition. Morgan Kauffman Publishers, Burlington MA

Duda RO, Hart PE (1972) Use of the Hough transform to Detect Lines and Curves in Pictures. Communications of the ACM 15:11–15

Francus P (2005) Image Analysis, Sediments and Paleoenvironments – Developments in Paleoenvironmental Research. Springer, Berlin Heidelberg New York

www.it-ebooks.info

Gonzalez RC, Woods RE, Eddins SL (2009) Digital Image Processing Using MATLAB – 2nd Edition. Gatesmark Publishing, LLC

Hough PVC (1962) Method and Means for Recognizing Complex Patterns. US Patent No. 3069654

Irons J, Riebeek H, Loveland T (2011) Landsat Data Continuity Mission – Continuously Observing Your World. NASA and USGS (available online)

Jensen JR (2013) Remote Sensing of the Environment: Pearson New International Edition. Pearson, London

Lein JK (2012) Environmental Sensing – Analytical Techniques for Earth Observation. Springer, Berlin Heidelberg New York

Mandl D, P Cruz, S Frye, Howard, J (2002) A NASA/USGS Collaboration to Transform Earth Observing-1 Into a Commercially Viable Mission to Maximize Technology Infusion. SpaceOps 2002, Houston, TX, October 9–12, 2002

Marwan N, Trauth MH, Vuille M, Kurths J (2003) Nonlinear time-series analysis on present-day and Pleistocene precipitation data from the NW Argentine Andes. Climate Dynamics 21:317–326

MathWorks (2014) Image Processing Toolbox – User's Guide. The MathWorks, Inc., Natick, MA

Ochs B., Hair D, Irons J, Loveland T (2009) Landsat Data Continuity Mission, NASA and USGS (available online)

Richards JA (2013) Remote Sensing Digital Image Analysis – 4th Edition. Springer, Berlin Heidelberg New York

Seelos K, Sirocko F (2005) RADIUS – Rapid Particle Analysis of digital images by ultra-high-resolution scanning of thin sections. Sedimentology 52:669–681

Trauth MH, Bookhagen B, Marwan N, Strecker MR (2003) Multiple landslide clusters record Quaternary climate changes in the NW Argentine Andes. Palaeogeography Palaeoclimatology Palaeoecology 194:109–121

Trauth MH, Alonso RA, Haselton KR, Hermanns RL, Strecker MR (2000) Climate change and mass movements in the northwest Argentine Andes. Earth and Planetary Science Letters 179:243–256

Trauth MH, Strecker MR (1999) Formation of landslide-dammed lakes during a wet period between 40,000–25,000 yr B.P. in northwestern Argentina. Palaeogeography Palaeoclimatology Palaeoecology 153:277–287

Yuen HK, Princen J, Illingworth J, Kittler J (1990) Comparative study of Hough transform methods for circle finding. Image and Vision Computing 8:71–77

Zuiderveld K (1994) Contrast Limited Adaptive Histograph Equalization. Graphic Gems IV. San Diego: Academic Press Professional, 474–485

www.it-ebooks.info

9 Multivariate Statistics

Miocene sedimentary rocks near the village of Nachola, to the east of the Suguta Valley in northern Kenya. Unmixing the composition of such deposits using multivariate statistical methods helps to determine the provenance of the sediments and their temporal variations, and hence changes in the relief, the drainage network and the rainfall patterns.

9.1 Introduction

Multivariate analysis aims to understand and describe the relationships between an arbitrary number of variables. Earth scientists often deal with multivariate data sets such as microfossil assemblages, geochemical fingerprints of volcanic ash layers, or the clay mineral content of sedimentary sequences. Such multivariate data sets consist of measurements of p variables on n objects, which are usually stored in n-by-p arrays:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

The rows of the array represent the n objects and the columns represent the p variables. The characteristics of the 2nd object in the suite of samples are described by the vector in the second row of the data array:

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_9

www.it-ebooks.info

$$X_2 = (\, x_{21} \quad x_{22} \quad \ldots \quad x_{2p} \,)$$

As an example consider a set of microprobe analyses on glass shards from volcanic ash layers in a tephrochronology project. The variables then represent the p chemical elements and the objects are the n ash samples. The aim of the study is to correlate ash layers by means of their geochemical fingerprints.

Most multi-parameter methods simply try to overcome the main difficulty associated with multivariate data sets, which relates to data visualization. Whereas the character of univariate or bivariate data sets can easily be explored by visual inspection of a 2D histogram or an *xy* plot (Chapter 3), the graphical display of a three variable data set requires a projection of the 3D distribution of data points onto a two-dimensional display. It is impossible to imagine or display a higher number of variables. One solution to the problem of visualization of high-dimensional data sets is to reduce the number of dimensions. A number of methods group together highly-correlated variables contained within the data set and then explore the reduced number of groups.

The classic methods for reducing the number of dimensions are *principal component analysis* (PCA) and *factor analysis* (FA). These methods seek the directions of maximum variance in a data set and use these as new coordinate axes. The advantage of replacing the variables by new groups of variables is that the groups are uncorrelated. Moreover, these groups can often assist in the interpretation of a multivariate data set since they often contain valuable information on the processes responsible for the distribution of the data points. In a geochemical analysis of magmatic rocks the groups defined by the method usually contain chemical elements with similar sized ions in similar locations within the lattices of certain minerals. Examples include Si^{4+} and Al^{3+}, and Fe^{2+} and Mg^{2+}, in silicates.

A second important suite of multivariate methods aims to group objects by their similarity. As an example *cluster analysis* (CA) is often used to correlate volcanic ash layers such as that used in the above example. Tephrochronology attempts to correlate tephra by means of their geochemical fingerprints. When combined with a few radiometric age determinations from the key ash layers this method allows correlation between different sedimentary sequences that contain these ash layers (e.g., Westgate 1998, Hermanns et al. 2000). Cluster analysis is also used in the field of micropaleontology, for example, to compare the pollen, foraminifera, or diatom content of different microfossil assemblages (e.g., Birks and Gordon 1985).

A third group of methods is concerned with the classification of observations. Humans tend to want to classify the things around them,

www.it-ebooks.info

even though nature rarely falls into discrete classes. Classification (or categorization) is useful as it can, for example, help decision makers to take necessary precautions to reduce risk, to drill an oil well, or to assign fossils to a particular genus or species. Most classification methods make decisions based on Boolean logic with two options, true or false; an example is the use of a threshold value for identifying charcoal in microscope images (Section 8.11). Alternatively, fuzzy logic (which is not explained in this book) is a generalization of the binary Boolean logic with respect to many real world problems in decision-making, where gradual transitions are reasonable (Zadeh 1965, MathWorks 2014a).

The following sections introduce the most important techniques of multivariate statistics: principal component analysis (PCA) and cluster analysis (CA) in Sections 9.2 and 9.5, and *independent component analysis* (ICA), which is a nonlinear extension of PCA, in Section 9.3. Section 9.4 introduces *discriminant analysis* (DA), which is a popular method of classification in earth sciences. Section 9.6. introduces *multiple linear regression*. These sections first provide an introduction to the theory behind the various techniques and then demonstrate their use for analyzing earth sciences data, using MATLAB functions (MathWorks 2014b).

9.2 Principal Component Analysis

Principal component analysis (PCA) detects linear dependencies between variables and replaces groups of correlated variables with new, uncorrelated variables referred to as the *principal components* (PCs). PCA was introduced by Karl Pearson (1901) and further developed by Harold Hotelling (1931). The performance of PCA is better illustrated with a bivariate data set than with a multivariate data set. Figure 9.1 shows a bivariate data set that exhibits a strong linear correlation between the two variables x and y in an orthogonal xy coordinate system. The two variables have their individual univariate means and variances (Chapter 3). The bivariate data set can be described by the bivariate sample mean and the covariance (Chapter 4). The xy coordinate system can be replaced by a new orthogonal coordinate system, where the first axis passes through the long axis of the data scatter and the new origin is the bivariate mean. This new reference frame has the advantage that the first axis can be used to describe most of the variance, while the second axis contributes only a small amount of additional information. Prior to this transformation two axes were required to describe the data set, but it is now possible to reduce the dimensions of the data by dropping the second axis without losing very much information, as shown in Figure 9.1.

This process is now expanded to an arbitrary number of variables and

www.it-ebooks.info

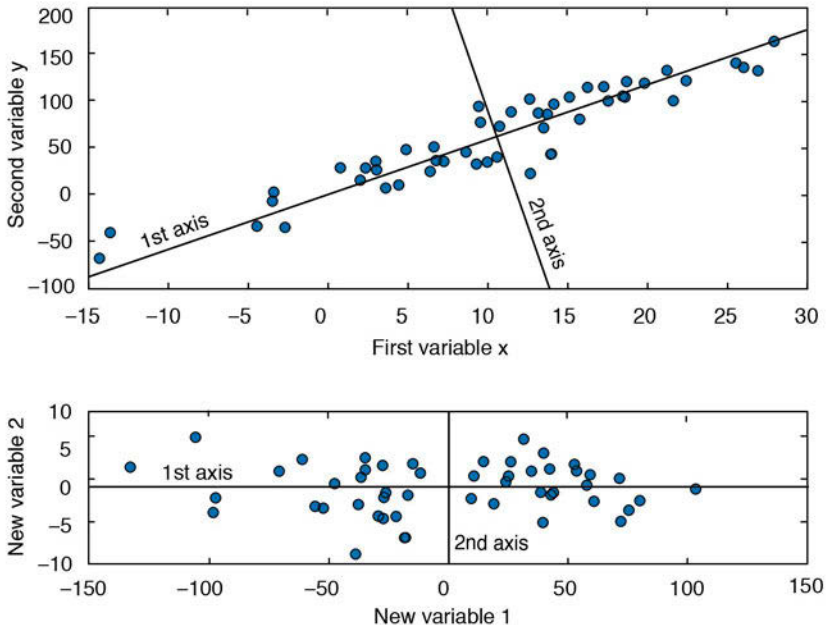


Fig. 9.1 Principal component analysis (PCA) illustrated for a bivariate scatter. The original *xy* coordinate system is replaced by a new orthogonal system, where the first axis passes through the long axis of the data scatter and the new origin is the bivariate mean. We can now reduce the number of dimensions by dropping the second axis with little loss of information.

samples. Assume a data set that consists of measurements of p variables on n samples, stored in an n-by-p array.

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

The columns of the array represent the p variables and the rows represent the n samples. After rotating the axis and moving the origin, the new coordinates Y_j can be computed by

$$Y_1 = a_{11}X_1 + a_{12}X_2 + \ldots + a_{1p}X_p$$
$$Y_2 = a_{21}X_1 + a_{22}X_2 + \ldots + a_{2p}X_p$$
$$\vdots$$
$$Y_p = a_{p1}X_1 + a_{p2}X_2 + \ldots + a_{pp}X_p$$

www.it-ebooks.info

The first principle component (PC_1), which is denoted by Y_1, contains the highest variance, PC_2 contains the second highest variance, and so forth. All the PCs together contain the full variance of the data set. This variance is, however, largely concentrated in the first few PCs, which include most of the information content of the data set. The last PCs are therefore generally ignored to reduce the dimensions of the data. The factors a_{ij} in the above equations are the *principal component loads*; their values represent the relative contributions of the original variables to the new PCs. If the load a_{ij} of a variable X_j in PC_1 is close to zero then influence of this variable is low, whereas a high positive or negative a_{ij} suggests a strong contribution. The new values Y_j of the variables computed from the linear combinations of the original variables X_j, weighted by the loads, are called the *principal component scores*.

PCA is commonly used as a method for unmixing (or separating) variables X, which are a linear combination of independent source variables S

$$X = A^T \cdot S$$

where A is the mixing matrix. PCA tries to determine (although not quantitatively) both the source variables S (represented by the principal components scores) and the mixing matrix A (represented by the principal component loads). Unmixing such variables works best if the probability distribution of the original variables X is a Gaussian distribution, and only in such cases are the principal components completely decorrelated. However, data in earth sciences are often not Gaussian distributed and alternative methods, such as independent component analysis (ICA), should therefore be used instead (Section 9.3). For example, radiance and reflectance values from hyperspectral data are often not Gaussian distributed and ICA is therefore widely used in remote sensing applications to decorrelate the spectral bands, rather than PCA. Examples in which PCA is used include the assessment of sediment provenance (as described in the example below), the unmixing of peridotite mantle sources of basalts, and multispectral classification of satellite images.

In the following example a synthetic data set is used to illustrate the use of the function `pca` included in the Statistics Toolbox. Thirty samples were taken from thirty different levels in a sedimentary sequence containing varying proportions of the three different minerals stored in the columns of the array `x`. The sediments were derived from three distinct rock types (with unknown mineral compositions) whose relative contributions to each of the thirty sediment samples are represented by `s1`, `s2` and `s3`. Variations in these relative contributions (as represented by the thirty values in `s1`, `s2` and

www.it-ebooks.info

s3) could, for example, reflect climatic variability within the catchment area of the sedimentary basin. It may therefore be possible to use the sediment compositions in the array x (from which we calculate s1, s2 and s3 using a PCA) to derive information on past variations in climate

We need to create a synthetic data set consisting of three measurements representing the proportions of each of the three minerals in the each of the thirty sediment samples. We first clear the workspace, reset the random number generator with rng(0) and create thirty values s1, s2 and s3. We use random numbers with a Gaussian distribution generated using randn, with means of zero and standard deviations of 10, 7 and 12.

```
clear

rng(0)
s1 = 10*randn(30,1);
s2 =  7*randn(30,1);
s3 = 12*randn(30,1);
```

We then calculate the varying proportions of each of the three minerals in the thirty sediment samples by summing up the values in s1, s2 and s3, after first multiplying them by a weighting factor.

```
x(:,1) =  15.4+ 7.2*s1+10.5*s2+2.5*s3;
x(:,2) = 124.0-8.73*s1+ 0.1*s2+2.6*s3;
x(:,3) = 100.0+5.25*s1- 6.5*s2+3.5*s3;
```

The weighting factors, which together represent the mixing matrix in our exercise, reflect not only differences in the mineral compositions of the source rocks, but also differences in the weathering, mobilization, and deposition of minerals within sedimentary basins. Hence if two minerals have weighting factors with different signs, one could be (for example) the weathering product of the other mineral, which would explain why their proportions in the thirty sediment samples were anti-correlated. Alternatively, the different signs could indicate a dilution effect, i.e., if the proportions of one of the minerals in the sediment samples remain constant but the proportions of all other minerals vary in a similar way, they will be anti-correlated with the proportions of the first mineral. To complete the generation of the data set, we add some Gaussian noise with a standard deviation of 3.8 to the proportions of the minerals and save the result as *sediment_1.txt*.

```
x = x + 3.8*randn(size(x));

save sediments_1.txt x -ascii
```

Having seen how the sedimentary record was created and the dependencies

www.it-ebooks.info

that exist within the data set, we will now pretend that we do not know the relative contributions that the three source rocks made to the thirty sediment samples. Instead, we have a record of the mineral contents of thirty sediment samples stored in the file *sediment_1.txt*.

```
clear

x = load('sediments_1.txt');
```

The aim of the PCA is now to decipher the statistically independent contribution of the three source rocks to the sediment compositions. We can display the histograms of the data and see that they are not perfectly Gaussian distributed, which means that we cannot expect a perfect unmixing result.

```
subplot(1,3,1), histogram(x(:,1))
subplot(1,3,2), histogram(x(:,2))
subplot(1,3,3), histogram(x(:,3))
```

We display the proportions of the three minerals in the thirty samples along the sedimentary section. In this graphic we can see weak correlations and anti-correlations between the proportions of the three minerals.

```
plot(x(:,1:3)), grid
legend('Min1','Min2','Min3')
xlabel('Sample ID')
ylabel('Quantity')
```

Before running the PCA we define labels for the various graphics created during the exercise. We number the samples 1 to 30, with the minerals being identified by four-character abbreviations `Min1`, `Min2` and `Min3`.

```
for i = 1 : 30
   samples(i,:) = [sprintf('%02.0f',i)];
end
minerals = ['Min1';'Min2';'Min3'];
```

We can explore the correlations between the minerals in pairwise bivariate scatter plots. We observe a strong negative correlation between the first and second mineral, a weak positive correlation between the first and third mineral, and a moderate negative correlation between the second and third mineral.

```
subplot(1,3,1),
plot(x(:,1),x(:,2),'o')
xlabel('Mineral 1')
ylabel('Mineral 2')
subplot(1,3,2),
plot(x(:,1),x(:,3),'o')
```

www.it-ebooks.info

```
xlabel('Mineral 1')
ylabel('Mineral 3')
subplot(1,3,3),
plot(x(:,2),x(:,3),'o')
xlabel('Mineral 2')
ylabel('Mineral 3')
```

The *correlation matrix* provides a technique for exploring such dependencies between the variables in the data set (i.e., the three minerals in our example). The elements of the correlation matrix are Pearson's correlation coefficients (Chapter 4) for each pair of variables, as shown in Figure 9.2.

```
corrmatrix = corrcoef(x);
corrmatrix = flipud(corrmatrix);

imagesc(corrmatrix), colormap(hot), caxis([-1 1])
title('Correlation Matrix')
axis square, colorbar, hold
set(gca,'XTick',[1 2 3],...
   'XTickLabel',minerals,...
   'YTick',[1 2 3],...
   'YTickLabel',flipud(minerals))
```

This pseudocolor plot of the correlation coefficients confirms the correlations

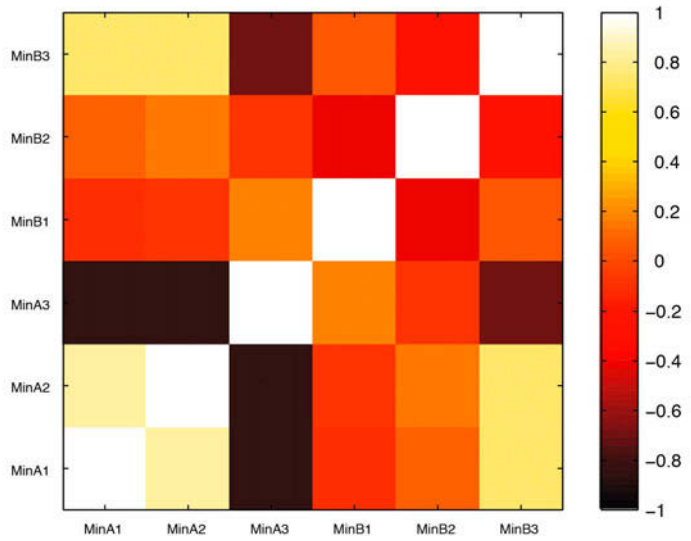


Fig. 9.2 Correlation matrix containing Pearson's correlation coefficients for each pair of variables (for example, minerals in a sediment sample). Light colors represent strong positive linear correlations, while dark colors represent negative correlations. Orange suggests no correlation.

www.it-ebooks.info

between the minerals revealed in the pairwise bivariate scatter plots, again showing a strong negative correlation between the first and second mineral (r=–0.8184), a weak positive correlation between minerals 1 and 3 (r=0.3483), and a moderate negative correlation between minerals 2 and 3 (r=–0.5557). These observed dependencies would lead us to expect interesting results from the application of a PCA.

Various methods exist for scaling the original data before applying a PCA, such as *mean centering* (using a mean equal to zero) or *standardizing* (using a mean equal to zero and a standard deviation equal to one). We will, however use the original data for computing the PCA. The output of the function `pca` includes the principal component loads `pcs`, the scores `newx`, and the variances `variances`. The loads `pcs` are weights (or weighting factors) that indicate the extent to which the old variables (the minerals) contribute to the new variables (the principal components, or PCs). The principal component scores are the coordinates of the thirty samples in the new coordinate system defined by the three principal components, PC_1 to PC_3 (stored in the three columns of `pcs`), which we interpret as the three source rocks.

```
[pcs,newx,variances] = pca(x);
```

The loads of the three principal components PC_1 to PC_3 can be shown by typing

```
pcs(:,1:3)

ans =
    0.6342   -0.5085    0.5825
   -0.6215    0.1130    0.7753
    0.4600    0.8536    0.2444
```

We observe that PC_1 (first column) has high positive loads in variables 1 and 3 (first and third rows), and a high negative load in variable 2 (second row). PC_2 (second column) has a high negative load in variable 1 and a high positive load in variable 3, while the load in variable 2 is close to zero. PC_3 (third column) has high loads in variables 1 and 2, with the load in variable 3 being relatively low but also positive. We create a number of plots to visualize the PCs:

```
subplot(1,3,1)
plot(1:3,pcs(:,1),'o'), axis([0.5 3.5 -1 1])
text((1:3)+0.2,pcs(:,1),minerals,'FontSize',14), hold
plot(0.5:3.5,zeros(4,1),'r'), title('PC 1')
subplot(1,3,2)
plot(1:3,pcs(:,2),'o'), axis([0.5 3.5 -1 1])
text((1:3)+0.2,pcs(:,2),minerals,'FontSize',14), hold
plot(0.5:3.5,zeros(4,1),'r'), title('PC 2')
```

www.it-ebooks.info

```
subplot(1,3,3)
plot(1:3,pcs(:,3),'o'), axis([0.5 3.5 -1 1])
text((1:3)+0.2,pcs(:,3),minerals,'FontSize',14), hold
plot(0.5:3.5,zeros(4,1),'r'), title('PC 3')
```

The loads of the minerals and their relationships to the PCs can be used to interpret the relative (not absolute) influences of the different source rocks. PC_1 is characterized by strong positive contributions of minerals 1 and 3, reflecting a relatively strong influence of the first rock type as a source of the sediments. An opposite sign for the contribution of mineral 2 to the sediment reflects either different mobilization processes for this mineral within the catchment area, or a dilution effect. The second principal component PC_2 is also dominated by minerals 1 and 3, but with opposite signs, while mineral 2 has relatively little influence. The third principal component PC_3 is influenced by all three minerals, with the same sign. An alternative way to plot of the loads is as a bivariate plot of two principal components. We ignore PC_3 at this point and concentrate on PC_1 and PC_2. Remember to either close the figure window before plotting the loads or clear the figure window using `clf`, in order to avoid integrating the new plot as a fourth subplot in the previous figure window.

```
plot(pcs(:,1),pcs(:,2),'o'), hold on
text(pcs(:,1)+0.02,pcs(:,2),minerals,'FontSize',14)
plot([-1 1],[0 0],'r')
plot([0 0],[-1 1],'r')
xlabel('First Principal Component Loads')
ylabel('Second Principal Component Loads')
hold off
```

We can now observe in a single plot the same relationships that have previously been shown in several graphics (Fig. 9.3). It is also possible to plot the data set as functions of the new variables (the source rocks). This requires the second output of `pca`, containing the principal component scores.

```
plot(newx(:,1),newx(:,2),'+'), hold on
text(newx(:,1)+2,newx(:,2),samples,'FontSize',14)
plot([-400 400],[0 0],'r')
plot([0 0],[-150 300],'r')
xlabel('First Principal Component Scores')
ylabel('Second Principal Component Scores')
hold off
```

This plot clearly defines groups of samples with similar influences, e.g., samples 19, 20 and 26, and possibly also sample 18. We next use the third output from the function `pca` to compute the variances of the PCs.

```
percent_explained = 100*variances/sum(variances)
```

www.it-ebooks.info

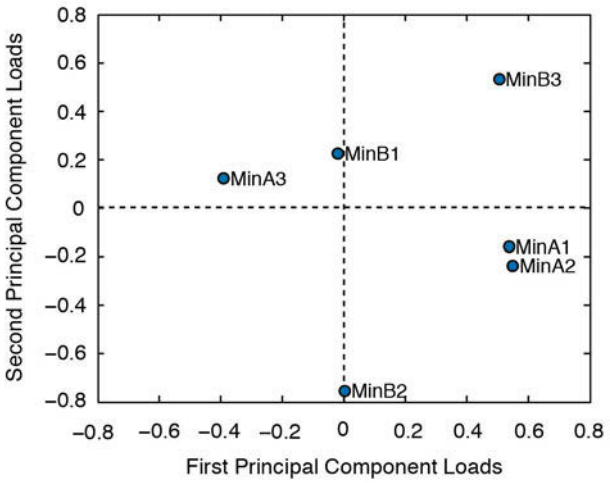


Fig. 9.3 Principal component loads suggesting that the PCs are influenced by different minerals. See text for detailed interpretation of the PCs.

```
percent_explained =
   72.4362
   22.7174
    4.8463
```

We see that more than 72% of the total variance is contained in PC_1, and about 22% is contained in PC_2, while PC_3 contributes very little to the total variance of the data set (~5%). This means that most of the variability in the data set can be described by just two new variables. As would be expected, the two new variables do not correlate with each other, as illustrated by a correlation coefficient between `newx(:,1)` and `newx(:,2)` that is close to zero.

```
corrcoef(newx(:,1),newx(:,2))

ans =
    1.0000    0.0000
    0.0000    1.0000
```

We can therefore plot the time series of the thirty samples as two independent variables, PC_1 and PC_2, in a single plot.

```
plot(1:30,newx(:,1),1:30,newx(:,2))
grid, legend('PC1','PC2')
xlabel('Sample ID'), ylabel('Value')
```

This plot displays approximately 94% (72%+22%) of the variance contained in the multivariate data set. According to our interpretation of PC_1 and

www.it-ebooks.info

PC_2, this plot shows the variability in the relative contributions from the two sources to the sedimentary column under investigation. Since we have worked with a synthetic data set and the actual contribution of the three source rocks to the sediment is known from

```
rng(0)
s1 = 10*randn(30,1);
s2 =  7*randn(30,1);
s3 = 12*randn(30,1);
```

we can estimate the quality of the result by comparing the initial variations in `s1`, `s2` and `s3` with the (more or less) independent variables PC_1, PC_2 and PC_3 stored in the three columns of `newx`.

```
subplot(3,1,1)
plotyy(1:30,newx(:,1),1:30,s1), title('PC1')
subplot(3,1,2)
plotyy(1:30,-newx(:,2),1:30,s2), title('PC2')
subplot(3,1,3)
plotyy(1:30,newx(:,3),1:30,s3), title('PC3')
```

The sign and the amplitude cannot be determined quantitatively and therefore, in this case, we change the sign of the second PC and use `plotyy` to display the data on different axes in order to compare the results. As we can see, we have successfully unmixed the varying contributions of the source rocks `s1`, `s2` and `s3` to the mineral composition of the sedimentary sequence.

The approach described above has been used to study the provenance of the varved lake sediments described in the previous chapter (Section 8.9), which were deposited around 33 kyrs ago in a landslide-dammed lake in the Quebrada de Cafayate (Trauth et al. 2003). The provenance of the sediments contained in the varved layers can be traced using index minerals characteristic of the various possible source areas within the catchment. A comparison of the mineral assemblages in the sediments with those of potential source rocks within the catchment area indicates that Fe-rich Tertiary sedimentary rocks exposed in the Santa Maria Basin were the source of the red-colored basal portion of the varves. In contrast, metamorphic rocks in the mountainous parts of the catchment area were the most likely source of the relatively drab-colored upper part of the varves.

9.3 Independent Component Analysis (by N. Marwan)

Principal component analysis (PCA) is the standard method for unmixing (or separating) mixed variables (Section 9.2). Such analyses produce signals that are linearly uncorrelated, and this method is also called *whitening* since

www.it-ebooks.info

this property is characteristic of white noise. Although the separated signals are uncorrelated, they can still be interdependent, i.e., they may retain a nonlinear correlation. This phenomenon arises when, for example, the data are not Gaussian distributed and the PCA consequently does not yield good results. The *independent component analysis* (ICA) was developed for this type of task; it separates the variables X into independent variables S, which are then nonlinearly uncorrelated. The basis of an ICA, according to the central limit theorem, is that the mixture of standardized random numbers is Gaussian distributed. The ICA algorithms therefore use a criterion that estimates how Gaussian the combined distribution of the independent components is (Hyvärinen 1999). The less Gaussian this distribution, the more independent the individual components.

According to the linear mixing model, p independent variables X are linearly mixed in n measurements,

$$X = A^T \cdot S$$

in which we are interested in the source variables S and the mixing matrix A (see Section 9.2). For example we can imagine that we are at a party in which a lot of people are carrying on independent conversations. We can hear a mixture of these conversations but perhaps cannot distinguish them individually. We could install some microphones and use these to separate out the individual conversations: hence, this dilemma is sometimes known as the *cocktail party problem*. Its correct term is *blind source separation*, which is defined by

$$S = W^T X$$

where W^T is the separation matrix required to reverse the mixing and obtain the original signals. In earth sciences we encounter similar problems, for example, if we want to determine the relative contributions of different source rocks to basin sediments, as we did with the PCA but this time with the possibility that there are nonlinear dependencies in the data and that these are not Gaussian distributed (Section 9.2).

We again create a synthetic data set consisting of thirty measurements (the proportions of each of the three minerals) from each of the thirty sediment samples. In contrast to the PCA example, however, the temporal variation in the source rocks is not Gaussian distributed but is uniformly distributed, since we use `rand` instead of `randn` to create the pseudorandom numbers.

```
clear
```

www.it-ebooks.info

```
rng(0)
s1 = 10*rand(30,1);
s2 =  7*rand(30,1);
s3 = 12*rand(30,1);
```

We use the same mixing equation as in the PCA example to create the three columns of `x` (corresponding to the three minerals) by linearly mixing the source rocks `s1`, `s2` and `s3`

```
x(:,1) =  15.4+ 7.2*s1+10.5*s2+2.5*s3;
x(:,2) = 124.0-8.73*s1+ 0.1*s2+2.6*s3;
x(:,3) = 100.0+5.25*s1- 6.5*s2+3.5*s3;

x = x + 3.8*randn(size(x));

save sediments_2.txt x -ascii
```

and save the data in the file *sediments_2.txt* (Fig. 9.4). Having created the synthetic data set, the data stored in *sediment_2.txt* are loaded by typing

```
clear

x = load('sediments_1.txt');
```

We can display the histograms of the data to see that they are not Gaussian distributed.

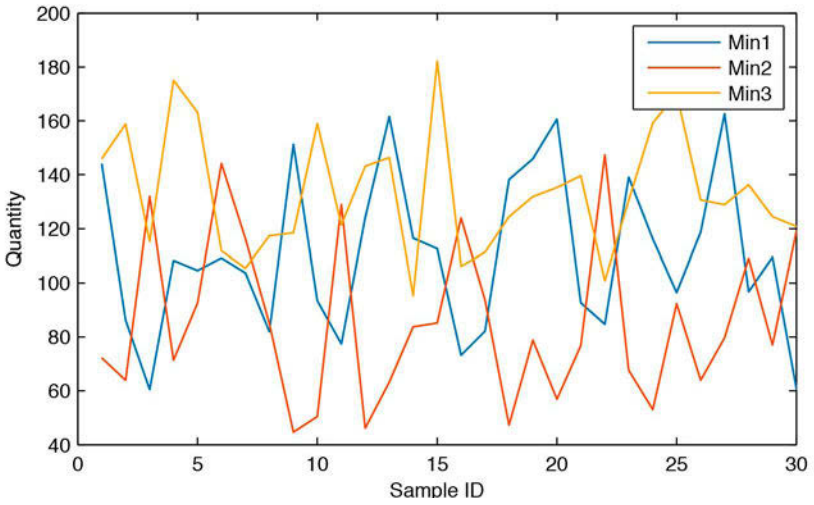


Fig. 9.4 Sample input for the independent component analysis. The relative proportions of three minerals in 30 sediment samples reflect temporal variations in the contributions from three source rocks within the catchment of a sedimentary basin.

www.it-ebooks.info

```
subplot(1,3,1), hist(x(:,1))
subplot(1,3,2), hist(x(:,2))
subplot(1,3,3), hist(x(:,3))
```

We then display the proportions of the minerals in the thirty samples as a time series, in two separate plots.

```
plot(x(:,1:3)), grid
legend('Min1','Min2','Min3')
xlabel('Sample ID')
ylabel('Quantity')
```

We begin with the unmixing of the variables using the PCA. We calculate the principal components `pcs`, the mixing matrix `a_pca`, and the whitening matrix `w_pca` using

```
[pcs,newx,variances] = pca(x);
newx = newx./repmat(std(newx),length(newx),1);

a_pca = pcs*sqrt(diag(variances));
w_pca = inv(sqrt(diag(variances)))*pcs';
```

The pre-whitening reduces the unmixing into independent components (S) to a problem of finding a suitable rotation matrix B that can be applied to the variables X_{PC}:

$$S = B^T X_{PC}$$

We need to find a rotation matrix B such that the variables S have a completely non-Gaussian distribution. There are several possibilities for implementing such a non-Gaussian criterion, such as minimizing or maximizing the excess kurtosis,

$$\gamma = E(x^4) - 3(E(x^2))^2$$

because the excess kurtosis γ for normally distributed data is zero. Please note that the excess kurtosis differs from the kurtosis: the excess kurtosis is the kurtosis minus three, since the kurtosis of Gaussian-distributed data is three (Section 3.2) but the excess kurtosis of such a distribution is zero. To find a B that ensures a minimum or maximum excess kurtosis, a learning algorithm can be used that finds the fixed points of the learning rule

$$E(x(B^T x)^3) - 3\left|\left|B\right|\right|^2 B + f(\left|\left|B\right|\right|^2)B = 0$$

www.it-ebooks.info

The FastICA algorithm by Hyvärinen (1999), which is based on a fixed-point iteration scheme, is an efficient algorithm for solving this problem. The learning rule is reduced in this example to the form

$$B(k) = E(x(B(k-1)^T x)^3) - 3B(k-1)$$

We begin with an initial rotation matrix B, which consists only of random numbers.

```
B = orth(rand(3,3)-0.5);
BOld = zeros(size(B));
```

The iteration is continued until a divergence criterion `div` is reached. We choose a value of zero for this criterion.

```
div = 0;
```

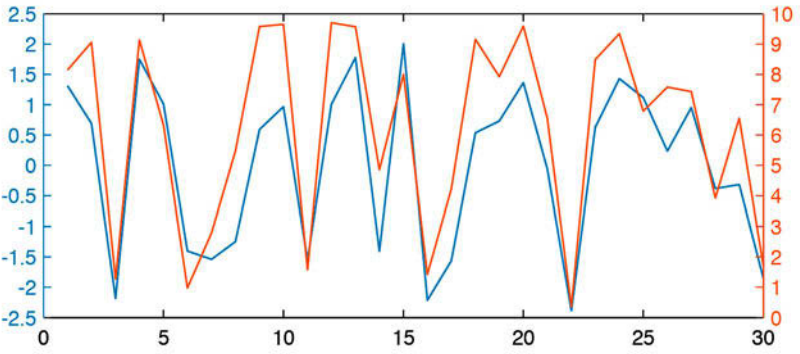
The fixed-point iteration scheme consists of two steps: a symmetric orthogonalization, and the application of the learning rule. The divergence criterion `div` is updated with each iteration step and checked in order to stop the iteration process as soon as the divergence criterion is reached. Since MATLAB works with floating-point numbers, `div` will not actually reach zero and we therefore allow `(1-div)` to fall below the floating-point relative accuracy `eps(1)`, or `eps` without input parameter `1` (since the default input for `eps` is `1`) which is about `2.2204e-16` and therefore close to zero (see Section 2.5).

```
while (1-div) > eps
   B = B*real(inv(B'*B)^(1/2));
   div = min(abs(diag(B'*BOld)));
   BOld = B;
   B = (newx'*(newx*B).^3)/length(newx)-3*B;
   sica = newx*B;
end
```

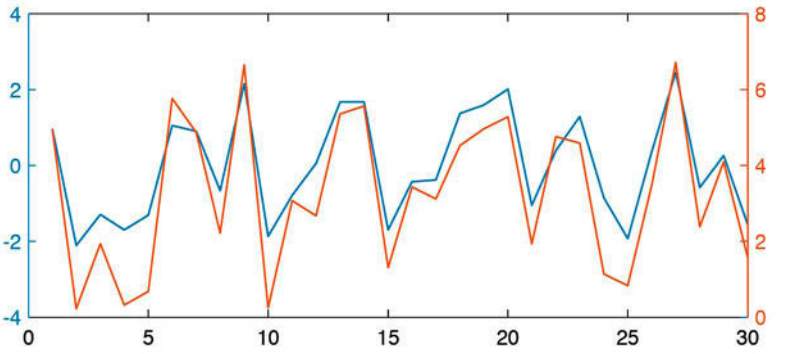
Finally, we compare the synthetic source rocks `s1`, `s2` and `s3` with the unmixed variables `IC1`, `IC2` and `IC3` (Fig. 9.5).

```
rng(0)
s1 = 10*rand(30,1);
s2 =  7*rand(30,1);
s3 = 12*rand(30,1);

subplot(3,1,1)
plotyy(1:30,-sica(:,3),1:30,s1)
title('IC1')
subplot(3,1,2)
plotyy(1:30,sica(:,2),1:30,s2)
```

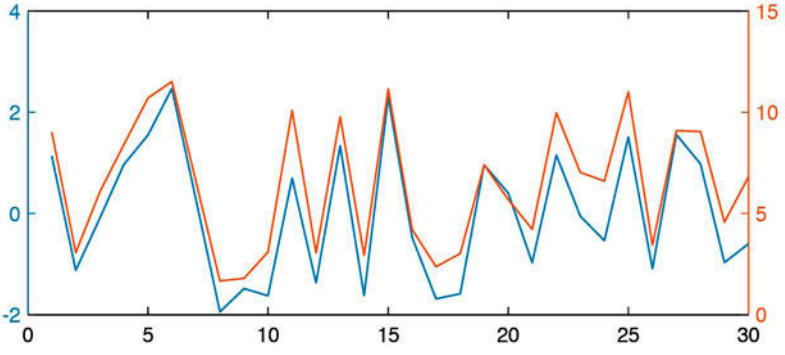
www.it-ebooks.info



a



b



c

Fig. 9.5 Independent component analysis output. The ICA has identified the source signals almost perfectly, as the pairwise similarity of the original inputs and the independent components suggests.

www.it-ebooks.info

```
title('IC2')
subplot(3,1,3)
plotyy(1:30,sica(:,1),1:30,s3)
title('IC3')
```

The ICA has identified the source signals almost perfectly. We notice that the descending order of the ICs is different from the initial order of `s1`, `s2` and `s3`, which is due to the commutativity of addition. In real-world examples, of course, the order of the ICs is not relevant. Furthermore, the exact sign and amplitude does not match the original values and the ICA therefore yields only semi-quantitative results, as was also the case for the PCA. Finally, we can actually compute the mixing matrix `a_ica` and the separation matrix `w_ica` using

```
a_ica = a_pca*B;
w_ica = B'*w_pca;
```

The mixing matrix `a_ica` can be used to estimate the proportions of the separated variables in our measurements. The components a_{ij} of the mixing matrix `a_ica` correspond to the principal component loads, as introduced in Section 9.2. The FastICA package is available for MATLAB and can be found on A. Hyvärinen's webpage:

```
http://research.ics.aalto.fi/ica/fastica/
```

9.4 Discriminant Analysis

Discriminant analysis helps to assign objects to established categories or groups. Examples include the assignment of fossil specimens to established genera or species, the identification of rock types following mineralogical (or chemical) analysis, and the mapping of vegetation types from satellite images. Discriminant analysis is different from simple classification, which does not define the number of groups or categories prior to the analysis.

The classic example of a discriminant analysis in petrography is the QAPF or Streckeisen diagram (Streckeisen 1974, 1976). This diagram categorizes igneous rocks (especially plutonic rocks) by their percentages of quartz (Q), alkali feldspar (including albite) (A), plagioclase (P), and feldsparthoids (F), normalized a total of 100%. The QAPF diagram displays the percentages of the four minerals in a double ternary plot, with QAP percentages in the upper half of the graphics and FAP percentages in the lower half. The QAPF diagram is commonly used by the International Union of Geological Sciences (IUGS) to classify plutonic rocks: e.g., a plutonic rock with 50% quartz, 30% alkali feldspar, 20% plagioclase and 0% feldsparthoids is termed a granite. Whereas Albert Streckeisen's definition of plutonic rocks represents compromises

www.it-ebooks.info

between established usages in different parts of the world (Streckeisen 1974, 1976), discriminant analysis is based solely on mathematical constraints for the classification of objects. Furthermore, discriminant analysis assumes normality for the measured values within a class, which is probably not a valid assumption for the Streckeisen classification. Since normality is important for the success of the method, the user must be careful to ensure that this condition is actually met, especially when analyzing compositional (closed) data (see also Section 9.5).

Discriminant analysis was first introduced by Sir Ronald A. Fisher (1936) to discriminate between two or more populations of flowering plant. The first step involves determining the discriminant function Y_i that best separate two groups of objects described by the normally-distributed variables X_i

$$Y = a_0 + a_1 X_1 + a_2 X_2 + \ldots + a_n X_n$$

The parameters a_i are determined to maximize the distance between the multivariate means of the individual groups. In other words, we determine a_i such that the ratio of the distances between the means of the groups to the distances between group members is high. In the second step, having determined the discriminant function from a training set of objects, new objects can be assigned to one group or the other. Using the Streckeisen diagram, a rock sample is assigned to an established rock type such as granite on the basis of the percentages of Q, A, P and F.

As an example we first create a synthetic data set of granite rock samples, described by two variables, x_1 and x_2. These two variables could represent the percentages of two chemical elements expressed as oxides (in weight percent). Let us assume that we know from preliminary studies that these rock samples come from three different granites that were formed at different times during three separate magmatic events. Apart from natural inhomogeneities within a granite intrusion, we can assume that the measured values from the granite samples are normally distributed. In this example we will first determine the discriminant functions separating the three groups (or types of granite). We then use the discriminant functions to assign rock samples (which were collected during a subsequent field campaign) to one of the three types of granite.

We first clear the workspace.

```
clear
```

We then reset the random number generator.

```
rng(0)
```

www.it-ebooks.info

Next, we generate a data set from the chemical compositions of the three types of granite. Sixty granite samples were collected from each rock type and chemically analyzed. The percentages of two chemical elements, `x1` and `x2`, we store in three variables `data1`, `data2` and `data3`.

```
data1 = randn(60,2);
data1(:,1) = 3.4 + 1.2*data1(:,1);
data1(:,2) = 1.7 + 0.4*data1(:,2);

data2 = randn(60,2);
data2(:,1) = 5.5 + 1.8*data2(:,1);
data2(:,2) = 2.9 + 0.6*data2(:,2);

data3 = randn(60,2);
data3(:,1) = 3+ 1.8*data2(:,1);
data3(:,2) = 0.3 + 1.2*data2(:,2);
```

In order to define the established categories (or classes) we create a character array containing the labels for each pair of measurements using `remap`, which creates 60 copies of each string `Granite_1`, `Granite_2`, and `Granite_3` and stores them in `classes`.

```
classes(1:60,1:9) = repmat('Granite_1',60,1);
classes(61:120,1:9) = repmat('Granite_2',60,1);
classes(121:180,1:9) = repmat('Granite_3',60,1);
```

We then vertically concatenate the three variables `data1`, `data2` and `data3` to a single variable `data`.

```
data = [data1(:,1:2);data2(:,1:2);data3(:,1:2)];
```

We save the resulting synthetic data set in the binary file *granite.mat*.

```
save granite.mat
```

We have thus generated a synthetic data set from three groups of normally distributed data. We can load the data set (as we would with any other real data set) from the file *granite.mat* by typing

```
clear

load granite.mat
```

We then create a linear discriminant analysis classifier using `ClassificationDiscriminant.fit`.

```
cls = ClassificationDiscriminant.fit(data,classes);
```

The function returns a discriminant analysis model for predictors `data` and

www.it-ebooks.info

class labels `classes`. The layers of `cls` can be listed by typing

```
cls
```

resulting in the output

```
cls =
  ClassificationDiscriminant
    PredictorNames: {'x1'  'x2'}
      ResponseName: 'Y'
        ClassNames: [3x9 char]
    ScoreTransform: 'none'
     NObservations: 180
       DiscrimType: 'linear'
                Mu: [3x2 double]
            Coeffs: [3x3 struct]
```

The meaning of most fields in this output is obvious, except for `ScoreTransform` which is either a function handle for transforming scores or a string representing a built-in transformation function. We are not using the score transform which is why it is set to `none`. The field `Mu` contains the means of the two variables from the three classes. The 3-by-3 structure `Coeffs` of the three classes contains the coefficients of the linear classification boundaries between classes 1 and 2, and between classes 2 and 3. Typing

```
cls.Coeffs
```

results in the output

```
ans =
3x3 struct array with fields:
    DiscrimType
    Const
    Linear
    Class1
    Class2
```

which lists the discriminant type (`linear` in our example), the class names of the pairs of classes `i` and `j`, and the two parameters `Const` and `Linear` describing the classification boundaries separating the two pairs of classes. We can extract the parameters of the classification boundaries by typing

```
K1 = cls.Coeffs(2,3).Const;
L1 = cls.Coeffs(2,3).Linear;
K2 = cls.Coeffs(1,2).Const;
L2 = cls.Coeffs(1,2).Linear;
```

and store them in new variables `K1`, `L1`, `K2` and `L2`. We then store the bivariate means by typing

www.it-ebooks.info

```
Mu1 = cls.Mu(1,:);
Mu2 = cls.Mu(2,:);
Mu3 = cls.Mu(3,:);
```

Finally, we display the result in a graphic using

```
h1 = axes('Box','On');
hold on

line(data1(:,1),data1(:,2),...
    'Marker','.','MarkerSize',8,...
    'LineStyle','None','MarkerEdgeColor','r')
line(data2(:,1),data2(:,2),...
    'Marker','.','MarkerSize',8,...
    'LineStyle','None','MarkerEdgeColor','b')
line(data3(:,1),data3(:,2),...
    'Marker','.','MarkerSize',8,...
    'LineStyle','None','MarkerEdgeColor','m')

line(Mu1(:,1),Mu1(:,2),...
    'Marker','o','MarkerEdgeColor','k',...
    'MarkerSize',8,'MarkerFaceColor','k')
line(Mu2(:,1),Mu2(:,2),...
    'Marker','o','MarkerEdgeColor','k',...
    'MarkerSize',8,'MarkerFaceColor','k')
line(Mu3(:,1),Mu3(:,2),...
    'Marker','o','MarkerEdgeColor','k',...
    'MarkerSize',8,'MarkerFaceColor','k')

h2 = legend('Granite 1','Granite 2','Granite 3',...
    'Location','SouthEast');

set(h2,'Box','Off')

f1 = @(x1,x2) K1 + L1(1)*x1 + L1(2)*x2;
h3 = ezplot(f1,[-5 12 0 5]);
set(h3,'Color','k')

f2 = @(x1,x2) K2 + L2(1)*x1 + L2(2)*x2;
h4 = ezplot(f2,[-5 10 0 5]);
set(h4,'Color','k')

title('Discriminant Analysis')

hold off
```

The graphic shows the members of the three classes (or types of granite) in three different colors, the two red lines separating the three classes and the bivariate means of the classes (marked as red plus signs) (Fig. 9.6). A new sample with the composition `x1`=5.2 and `x2`=3.5 can easily be assigned to the class Granite 2.

```
h1 = axes('XLim',[-5 25],'YLim',[0 5],...
```

www.it-ebooks.info

```
    'Box','On');
hold on

line(5.2,3.5,...
    'Marker','o','MarkerEdgeColor','b',...
    'MarkerSize',8,'MarkerFaceColor','b')

f1 = @(x1,x2) K1 + L1(1)*x1 + L1(2)*x2;
h3 = ezplot(f1,[-5 12 0 5]);
set(h3,'Color','k')

f2 = @(x1,x2) K2 + L2(1)*x1 + L2(2)*x2;
h3 = ezplot(f2,[-5 10 0 5]);
set(h3,'Color','k')

text(0,2,'Granite 1','Color','r')
text(-0.5,4,'Granite 2','Color','b')
text(10,4.5,'Granite 3','Color','m')

title('Discriminant Analysis')

hold off
```

The function `ClassificationDiscriminant.fit` also allows a quadratic classifier to be fitted instead of the linear classifier used in the example with three types of granite (MathWorks 2014b).

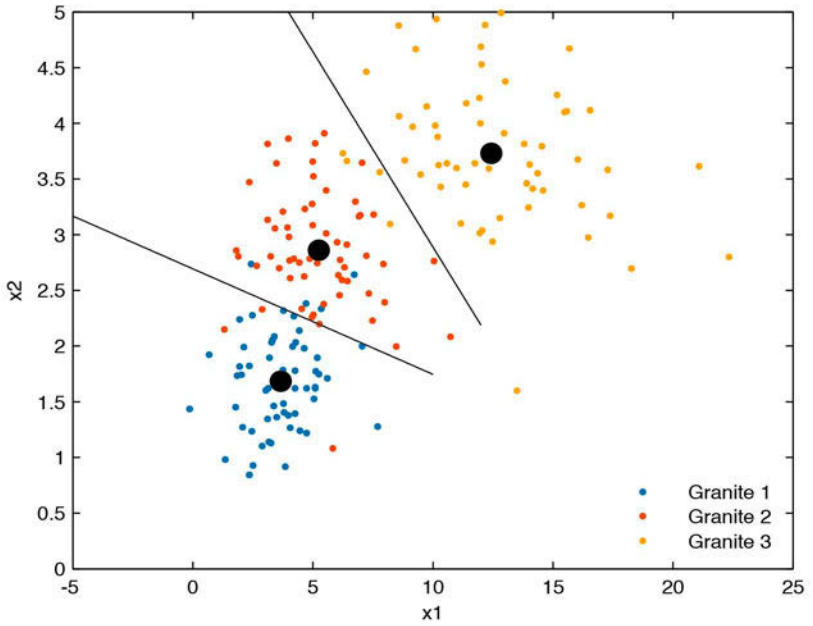


Fig. 9.6 Result of linear discriminant analysis, separating three types of granites (1, 2 and 3).

www.it-ebooks.info

9.5 Cluster Analysis

Cluster analysis creates groups of objects that are very similar to each other, compared to other individual objects or groups of objects. It first computes the similarity or (alternatively) the dissimilarity (or distance) between all pairs of objects and then ranks the groups according to their similarity or distance, finally creating a hierarchical tree visualized as a dendrogram. The grouping of objects can be useful in the earth sciences, for example when making correlations within volcanic ash layers (Hermanns et al. 2000) or comparing different microfossil assemblages (Birks and Gordon 1985).

There are numerous methods for calculating the similarity or (alternatively) the dissimilarity (or distance) between two data vectors. Let us define two data sets consisting of multiple measurements on the same object. These data can be described by vectors:

$$X_1 = (x_{11} \quad x_{12} \quad \ldots \quad x_{1p})$$
$$X_2 = (x_{21} \quad x_{22} \quad \ldots \quad x_{2p})$$

The most popular measures of dissimilarity (or distance) between the two sample vectors are:

- the *Euclidian distance* – This is simply the shortest distance between the two points describing two measurements in the multivariate space:

  $$\Delta_{12} = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 + \ldots + (x_{1p} - x_{2p})^2}$$

  The Euclidian distance is certainly the most intuitive measure for similarity. However, in heterogeneous data sets consisting of a number of different types of variables, a better alternative would be

- the *Manhattan* (or *city block*) *distance* – In the city of Manhattan, one must follow perpendicular avenues rather than crossing blocks diagonally. The Manhattan distance is therefore the sum of all differences:

  $$\Delta_{12} = |x_{11} - x_{21}| + |x_{12} - x_{22}| + \ldots + |x_{1p} - x_{2p}|$$

Measures of similarity include

- the *correlation similarity coefficient* – This uses Pearson's linear product-moment correlation coefficient to compute the similarity of two objects:

www.it-ebooks.info

$$r_{x1x2} = \frac{\sum_{i=1}^{n}(x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)}{(n-1)s_{x1}s_{x2}}$$

This measure is used if one is interested in the ratios between the variables measured on the objects. However, Pearson's correlation coefficient is highly sensitive to outliers and should be used with care (see also Section 4.2).

- the *inner-product similarity index* – Normalizing the length of the data vectors to a value of one and computing their inner product yields another important similarity index that is often used in transfer function applications. In this example a set of modern flora or fauna assemblages with known environmental preferences is compared with a fossil sample, in order to reconstruct past environmental conditions.

$$s_{12} = \frac{1}{|X_1|}\frac{1}{|X_2|}\left(x_{11}\ x_{12} \ldots x_{1p}\right)\begin{pmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2p} \end{pmatrix}$$

The inner-product similarity varies between 0 and 1. A zero value suggests no similarity and a value of one represents maximum similarity.

The second step in performing a cluster analysis is to rank the groups by their similarity and to build a hierarchical tree, visualized as a dendrogram. Most clustering algorithms simply link the two objects with the highest level of similarity or dissimilarity (or distance). In the following steps, the most similar pairs of objects or clusters are linked iteratively. The difference between clusters, each made up of groups of objects, is described in different ways depending on the type of data and the application:

- *K-means clustering* uses the Euclidean distance between the multivariate means of a number of *K* clusters as a measure of the difference between the groups of objects. This distance is used if the data suggest that there is a true mean value surrounded by random noise. Alternatively,

- *K-nearest-neighbors clustering* uses the Euclidean distance of the nearest neighbors as measure of this difference. This is used if there is a natural heterogeneity in the data set that is not attributed to random noise.

www.it-ebooks.info

It is important to evaluate the data properties prior to the application of a clustering algorithm. The absolute values of the variables should first be considered. For example a geochemical sample from volcanic ash might show an SiO_2 content of around 77% and a Na_2O contents of only 3.5%, but the Na_2O content may be considered to be of greater importance. In such a case the data need to be transformed so that they have means equal to zero (*mean centering*). Differences in both the variances and the means are corrected by *standardizing*, i.e., the data are standardized to means equal to zero and variances equal to one. Artifacts arising from closed data, such as artificial negative correlations, are avoided by using *Aitchison's log-ratio transformation* (Aitchison 1984, 1986). This ensures data independence and avoids the constant sum normalization constraints. The log-ratio transformation is

$$x_{tr} = \log(x_i / x_d)$$

where x_{tr} denotes the transformed score (i=1, 2, 3, …, d–1) of some raw data x_i. The procedure is invariant under the group of permutations of the variables, and any variable can be used as the divisor x_d.

As an exercise in performing a cluster analysis, the sediment data stored in *sediment_3.txt* are loaded. This data set contains the percentages of various minerals contained in sediment samples. The sediments are sourced from three rock types: a magmatic rock containing amphibole (*amp*), pyroxene (*pyr*) and plagioclase (*pla*), a hydrothermal vein characterized by the presence of fluorite (*flu*), sphalerite (*sph*) and galena (*gal*), some feldspars (plagioclase and potassium feldspars, *ksp*) and quartz, and a sandstone unit containing feldspars, quartz and clay minerals (*cla*). Ten samples were taken from various levels in this sedimentary sequence, each containing varying proportions of these minerals. First, the distances between pairs of samples can be computed. The function `pdist` provides many different measures of distance, such as the Euclidian or Manhattan (or city block) distance. We use the default setting which is the Euclidian distance.

```
clear

data = load('sediments_3.txt');
Y = pdist(data);
```

The function `pdist` returns a vector `Y` containing the distances between each pair of observations in the original data matrix. We can visualize the distances in another pseudocolor plot.

```
imagesc(squareform(Y)), colormap(hot)
```

www.it-ebooks.info

```
title('Euclidean distance between pairs of samples')
xlabel('First Sample No.')
ylabel('Second Sample No.')
colorbar
```

The function `squareform` converts `Y` into a symmetric, square format, so that the elements `(i,j)` of the matrix denote the distance between the `i` and `j` objects in the original data. We next rank and link the samples with respect to the inverse of their separation distances using the function `linkage`.

```
Z = linkage(Y)

Z =
    2.0000    9.0000    0.0564
    8.0000   10.0000    0.0730
    1.0000   12.0000    0.0923
    6.0000    7.0000    0.1022
   11.0000   13.0000    0.1129
    3.0000    4.0000    0.1604
   15.0000   16.0000    0.1737
    5.0000   17.0000    0.1764
   14.0000   18.0000    0.2146
```

In this 3-column array `Z`, each row identifies a link. The first two columns identify the objects (or samples) that have been linked, while the third column contains the separation distance between these two objects. The first row (link) between objects (or samples) 1 and 2 has the smallest distance, corresponding to the greatest similarity. In our example samples 2 and 9 have the smallest separation distance of 0.0564 and are therefore grouped together and given the label 11, i.e., the next available index higher than the highest sample index 10. Next, samples 8 and 10 are grouped to 12 since they have the second smallest separation difference of 0.0730. The next row shows that the new group 12 is then grouped with sample 1, which have a separation difference of 0.0923, and so forth. Finally, we visualize the hierarchical clusters as a dendrogram, which is shown in Figure 9.7.

```
dendrogram(Z);
xlabel('Sample No.')
ylabel('Distance')
box on
```

Clustering finds the same groups as the principal component analysis. We observe clear groups consisting of samples 1, 2, 8, 9 and 10 (the magmatic source rocks), samples 3, 4 and 5 (the hydrothermal vein), and samples 6 and 7 (the sandstone). One way to test the validity of our clustering result is to use the *cophenet correlation coefficient*:

```
cophenet(Z,Y)
```

www.it-ebooks.info

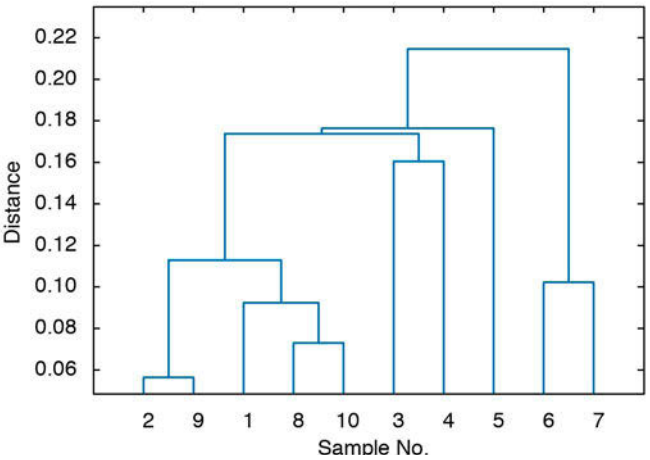


Fig. 9.7 Cluster analysis output. The dendrogram shows clear groups consisting of samples 1, 2, 8, 9 and 10 (the magmatic source rocks), samples 3, 4 and 5 (the magmatic dyke containing ore minerals), and samples 6 and 7 (the sandstone unit).

```
ans =
    0.7579
```

The result is convincing since the closer this coefficient is to one, the better the cluster solution.

9.6 Multiple Linear Regression

In Chapter 4 linear regression models were introduced as a way of describing the relationship between a dependent variable y and an independent variable x. The dependent variable is also known as the *response variable*, and the independent variable as the *predictor variable*. A multiple linear regression model describes the relationship between a dependent (or response) variable y, and n independent (or predictor) variables x_i

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

where b_i are the n+1 regression coefficients of the linear model. The linearity in the term multiple linear regression refers to the fact that the response variable is a linear function of the predictor variables. The regression coefficients are estimated by minimizing the mean-squared difference between the predicted and true values of the response variable y. As an example that is commonly used in the earth sciences is the quality of crude oil, which is assumed to be linearly dependent on the age of the sediment, the

www.it-ebooks.info

burial depth, and temperature. In practice, the plausibility of the assumption of linearity must first be examined. If this assumption is probably true then there are several methods of multiple linear regression available, some of which are included in the Statistics Toolbox (Mathworks 2014b).

As a first example we create a noise-free synthetic data set with three variables `Var1`, `Var2` and `Var3`. We wish to find the influence of variables `Var1` and `Var2` on variable `Var3`. The variables `Var1` and `Var2` are therefore the predictor variables and the variable `Var3` is the response variable. The linear relationship between the response variable and the predictor variables is `Var3=0.2-52.0*Var1+276.0*Var2`. The three variables `Var1`, `Var2` and `Var3` are stored as columns 1, 2 and 3 in a single array `data`.

```
clear

rng(0)
data(:,1) = 0.3 + 0.03*randn(50,1);
data(:,2) = 0.2 + 0.01*randn(50,1);
data(:,3) =     0.2 ...
            -  52.0*data(:,1) ...
            + 276.0*data(:,2);
```

We create labels for the names of the samples and the names of the variables, as we did in Section 9.2.

```
for i = 1 : size(data,1)
    samples(i,:) = [sprintf('%02.0f',i)];
end

variables = ['Var1';
             'Var2';
             'Var3'];
```

Then we calculate the coefficients `beta` of the multiple linear regression model using `fitlm`.

```
beta = fitlm(data(:,1:2),data(:,3),...
    'ResponseVar',variables(3,:),...
    'PredictorVars',variables(1:2,:))
```

The function `fitlm` uses a least mean-squares criterion to calculate `beta`. The method calculates an *F*-statistic to test the null hypothesis that all regression coefficients are zero and there is no relationship between the response and predictor variables. The output of the function `fitlm`

```
beta =

Linear regression model:
    Var3 ~ 1 + Var1 + Var2
```

www.it-ebooks.info

```
Estimated Coefficients:
                   Estimate    SE    tStat    pValue
                   ________    __    _____    ______

    (Intercept)    0.2         0      Inf     0
    Var1           -52         0     -Inf     0
    Var2           276         0      Inf     0


Number of observations: 50, Error degrees of freedom: 47
Root Mean Squared Error: 0
R-squared: 1,  Adjusted R-Squared 1
F-statistic vs. constant model: Inf, p-value = 0
```

first recalls the linear regression model using the names of the response variable and the predictor variables. It then displays a table with the regression coefficient estimates for each predictor variable in the first column, the standard error, the *F*-statistic, and the *p*-values of the coefficient estimates. The quality of the model is excellent as the *p*-value of the constant model, i.e., with no influence of predictor variables on the response variable, is zero, as suggested in the last line of the comments below the table.

Since we have only three variables we can display the results in a three-dimensional plot. We first create a rectangular grid for the first two variables and then calculate the predicted values for the second variable using the estimated regression coefficients. We then use `surf` to display the linear regression plane of the model and `line` to plot the measurements as red points, with a marker size of 30 points.

```
[Var1,Var2] = meshgrid(0.20:0.01:0.45,0.17:0.01:0.23);
Var3 = beta.Coefficients.Estimate(1) + ...
       beta.Coefficients.Estimate(2)*Var1 + ...
       beta.Coefficients.Estimate(3)*Var2;

surf(Var1,Var2,Var3,'FaceAlpha',0.2), hold on
line(data(:,1), data(:,2), data(:,3),...
     'LineStyle','none',...
     'Marker','.',...
     'MarkerSize',30)
grid on
view(70,30)
hold off
```

Since the data set is noise free the data points all lie on the linear regression plane. This changes if we introduce normally-distributed noise with a standard deviation of 2.4.

```
clear

rng(0)
```

www.it-ebooks.info

```
data(:,1) = 0.3 + 0.03*randn(50,1);
data(:,2) = 0.2 + 0.01*randn(50,1);
data(:,3) =     0.2 ...
            -  52.0*data(:,1) ...
            + 276.0*data(:,2) ...
            +   2.4*randn(50,1);

for i = 1 : size(data,1)
    samples(i,:) = [sprintf('%02.0f',i)];
end

variables = ['Var1';
             'Var2';
             'Var3'];
```

Using `fitlm` again

```
beta = fitlm(data(:,1:2),data(:,3),...
    'ResponseVar',variables(3,:),...
    'PredictorVars',variables(1:2,:))
```

we see from the output

```
beta =

Linear regression model:
    Var3 ~ 1 + Var1 + Var2

Estimated Coefficients:
                   Estimate      SE       tStat       pValue
                   ________    ______    _______    __________

    (Intercept)     5.0222     6.9235    0.72538       0.47181
    Var1           -45.307     9.1424    -4.9557    9.7694e-06
    Var2             239.8     33.273     7.2071     3.969e-09


Number of observations: 50, Error degrees of freedom: 47
Root Mean Squared Error: 2.41
R-squared: 0.596,  Adjusted R-Squared 0.579
F-statistic vs. constant model: 34.6, p-value = 5.68e-10
```

that the estimates of the regression coefficients (5.0222, -45.307, and 239.8) do not exactly match those that were used to create the synthetic data (0.2, -52.0, and 276.0). Furthermore, the *p*-value of the constant model is significantly higher than in the previous example, although it is still very good, as suggested in the last line of the comments below the table. The *p*-values of the individual regression coefficient remain low, however, suggesting that both predictor variables `Var1` and `Var2` contribute equally to the response variable `Var3` at a 5% significance level. We can again display the results in a three-dimensional plot (Fig. 9.8)

www.it-ebooks.info

```
[Var1,Var2] = meshgrid(0.20:0.01:0.45,0.17:0.01:0.23);
Var3 = 5.0222 - 45.307*Var1 + 239.8*Var2;

surf(Var1,Var2,Var3,'FaceAlpha',0.2), hold on
line(data(:,1), data(:,2), data(:,3),...
     'LineStyle','none',...
     'Marker','.',...
     'MarkerSize',30)
grid on
view(70,30)
hold off
```

Movie 9.1

and observe some differences between the data and the regression plane if we rotate the graph in three dimensions using the *Rotate 3D* tool on the toolbar of the Figure Window.

As a second example we use laboratory data from the Sherwood Sandstone, England, that suggest that intergranular permeability (as response variable) is mainly influenced by porosity and matrix conductivity (as predictor variables), while other variables have less influence. This example has been previously discussed in detail in an article by Olorunfemi (1985) and was used as an example in the textbook by Swan and Sandilands (1995). The data for 40 rock samples from the Sherwood Sandstone were taken from M.O. Olorunfemi's publication and are stored in the file *sherwoodsandstone.txt*. We load the data from the file by typing

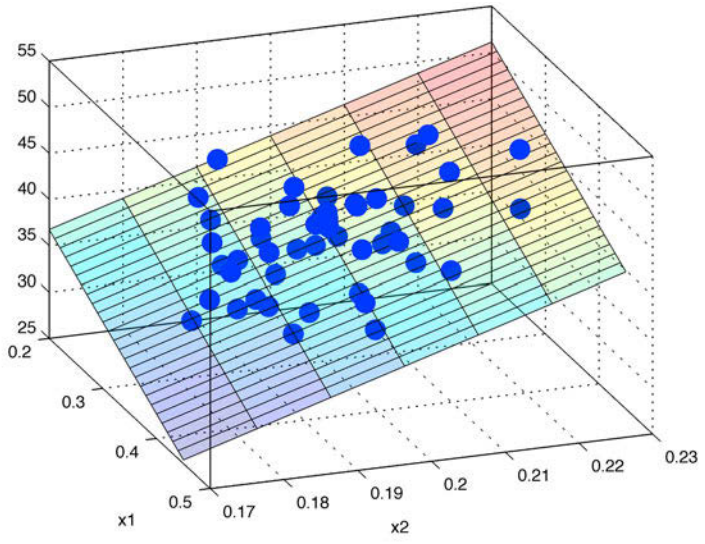


Fig. 9.8 Linear regression model for a synthetic data set with three variables and random noise.

www.it-ebooks.info

```
clear

data = load('sherwoodsandstone.txt');
```

The five columns of the array `data` contain the numerical values of the petrophysical parameters permeability (in m/s), porosity (without physical unit), matrix conductivity (in S/m), true formation factor (without physical unit), and induced polarization (in %). We also load the sample IDs (from Olorunfemi 1985) from the file *sherwoodsandstone_samples.txt* using `textscan`:

```
fid = fopen('sherwoodsandstone_samples.txt');
samples = textscan(fid,'%s','Headerlines',8);
fclose(fid);
```

The *sherwoodsandstone_samples.txt* file contains 40 sample IDs, each with five characters including spaces that are used to make the length of the sample IDs identical so that they fit into the cell array `samples`. This cell array contains only one cell, `samples{1}`, which in turn contains a 40-by-5 character array of the sample IDs. We next create another character array that contains the variable names: `Perm` for permeability, `Poro` for porosity, `MaCo` for matrix conductivity, `TrFF` for true formation factor, and `InPo` for induced polarization.

```
variables = ['Perm';
             'Poro';
             'MaCo';
             'TrFF';
             'InPo'];
```

We then display the variables in a plot that includes a legend and has the data points labeled with the sample IDs. We use random colors for the data points, with the random number generator first set to a seed value of 0. Since the values of the variables are very small we plot the logarithm of the values instead of the true values.

```
rng(0)
for i = 1:size(data,2)
    rcolor = rand(1,3);
    line(log(data(:,i)),1:40,...
        'Marker','Square','MarkerEdgeColor',rcolor,...
        'MarkerSize',8,'MarkerFaceColor',rcolor,...
        'LineStyle',':','Color',rcolor)
    text(log(data(:,i))+0.1,1:40,samples{1})
end
legend(variables)
```

We then calculate the coefficients `beta` of the multiple linear regression

www.it-ebooks.info

model using `fitlm`.

```
beta = fitlm(data(:,2:5),data(:,1),...
    'ResponseVar',variables(1,:),...
    'PredictorVars',variables(2:5,:))
```

The function `fitlm` uses a least mean-squares criterion to calculate `beta`. The output of the function `fitlm`

```
beta =

Linear regression model:
    Perm ~ 1 + Poro + MaCo + TrFF + InPo

Estimated Coefficients:
                    Estimate      SE        tStat       pValue
                    ________    _______    ________    ________

    (Intercept)      0.15868     7.5546    0.021005     0.98336
    Poro               50.636     18.714      2.7058    0.010458
    MaCo               -330.9     150.51     -2.1986    0.034609
    TrFF              -0.17857    0.18537    -0.96329     0.34201
    InPo              -0.45263    0.50101    -0.90344     0.37247


Number of observations: 40, Error degrees of freedom: 35
Root Mean Squared Error: 2.68
R-squared: 0.483,  Adjusted R-Squared 0.424
F-statistic vs. constant model: 8.18, p-value = 9.07e-05
```

first recalls the linear regression model using the names of the response variable and the predictor variables. It then displays a table with the regression coefficient estimates for each predictor variable in the first column, the standard error, the *t*-statistic, and the *p*-values of the coefficient estimates. The quality of the model is good as the *p*-value of the constant model (i.e., with no influence of predictor variables on the response variable) is very low, as the last line of the comments below the table suggests. The *p*-values of the individual regression coefficient estimates suggests that only the porosity and matrix conductivity influence the permeability of the Sherwood Sandstone, at a 5% significance level.

There are multiple ways to examine the quality of the result. The Statistics Toolbox includes several functions for displaying diagnostic plots to identify and remove outliers from the data set. We can, for instance, produce a leverage plot of the data and the model by typing

```
plotDiagnostics(beta)
```

The leverage plot displays the leverage that an individual measurement has when added to the regression model, assuming that all other measurements

www.it-ebooks.info

are already in the model. The plot also displays a horizontal dashed line that is not explained in the User's Guide of the Statistics Toolbox (MathWorks 2014b). Information regarding this dashed line can, however, be found in the R2013 Release Notes and the M-file of the function `LinearModel`

```
edit LinearModel.fit
```

where the regression diagnostics are explained in detail from line 126 onward. According to this information, the average value of leverage is the ratio between the number of regression coefficients and the number of measurements

```
size(data,2)/size(data,1)
```

which in this particular example yields

```
ans =
    0.1250
```

Observations with leverage values larger than twice this ratio

```
2*size(data,2)/size(data,1)
```

which yields

```
ans =
    0.2500
```

are considered to have high leverage, and are all measurements above the dashed line at 0.250. In our example there are two points with leverages greater than 0.250, one in row 11 of the array and the other in row 22. We can obtain the IDs of these samples by typing

```
samples{1}(11)
samples{1}(22)
```

which yields

```
ans =
    'HF14'

ans =
    'HF49'
```

The leverage plot does not reveal whether or not the high-leverage points are outliers. We can instead calculate and display the Cook's distance of each data point

```
plotDiagnostics(beta,'cookd')
```

www.it-ebooks.info

which is the change in the residuals of a model if this specific data point is removed. The larger the Cook's distance of a particular data point the more likely it is to be an outlier. The diagnostic plot again displays a dashed line, which marks a recommended threshold of three times the mean Cook's distance

```
3*mean(mdl.Diagnostics.CooksDistance)
```

which yields

```
ans =
    0.1144
```

According to this plot there are three data points above the critical value marked by the dashed line, one each in rows 22, 27 and 35. Since the data point `HF49` in the 22nd row also has a high leverage on the model it should be removed as an outlier and the improved model then run. We can also examine the residuals of the model using

```
plotResiduals(beta)
```

which suggests that the measurements above 9 are outliers. As the last diagnostic plot in this exercise we display the normal probability plot of the residuals

```
plotResiduals(beta,'probability')
```

where the black line represents the probability of the residuals in the case of an ideal normal distribution. In our example, the tails of the normal distribution plot of the residuals do not match a normal distribution and our linear regression model is therefore good (but not perfect) without having removed the outliers. The regression coefficients calculated from the regression analysis (including the outliers) are, however, similar to those reported on page 347 of Swan and Sandilands (1995).

The linear regression model can now be used to predict the permeability of a new sample from the values of the other petrophysical parameters. The effect that each predictor variable has on the model can be examined using one of several functions included in the Statistics Toolbox. As an example

```
plotSlice(beta)
```

yields a slice plot of the responses displaying each predictor variable separately. Within this plot we can drag the lines or insert values for the individual predictor variables and examine the resulting value of the response variable.

www.it-ebooks.info

Recommended Reading

Aitchison J (1984) The Statistical Analysis of Geochemical Composition. Mathematical Geology 16(6):531–564

Aitchison J (1999) Logratios and Natural Laws in Compositional Data Analysis. Mathematical Geology 31(5):563–580

Birks HJB, Gordon AD (1985) Numerical Methods in Quaternary Pollen Analysis. Academic Press, London

Brown CE (1998) Applied Multivariate Statistics in Geohydrology and Related Sciences. Springer, Berlin Heidelberg New York

Fisher RA (1936) The use of multiple measurements in taxonomic problems. Annals of Eugenics 7:179–188

Härdle WK, Simar L (2012) Applied Multivariate Statistical Analysis. Springer, Berlin Heidelberg New York

Hermanns R, Trauth MH, McWilliams M, Strecker M (2000) Tephrochronologic Constraints on Temporal Distribution of Large Landslides in NW-Argentina. Journal of Geology 108:35–52

Hotelling H (1931) Analysis of a Complex of Statistical Variables with Principal Components. Journal of Educational Psychology 24(6):417–441

Hyvärinen A (1999) Fast and Robust Fixed-Point Algorithms for Independent Component Analysis. IEEE Transactions on Neural Networks 10(3):626–634

MathWorks (2014a) Fuzzy Logic Toolbox – User's Guide. The MathWorks, Inc., Natick, MA

MathWorks (2014b) Statistics Toolbox – User's Guide. The MathWorks, Inc., Natick, MA

Olorunfemi MO (1985) Statistical Relationships Among Some Formation Parameters for Sherwood Sandstone, England. Mathematical Geology 17:845–852

Pawlowsky-Glahn V (2004) Geostatistical Analysis of Compositional Data – Studies in Mathematical Geology. Oxford University Press, Oxford

Pearson K (1901) On lines and planes of closest fit to a system of points in space. Philosophical Magazine and Journal of Science 6(2):559–572

Reyment RA, Savazzi E (1999) Aspects of Multivariate Statistical Analysis in Geology. Elsevier Science, Amsterdam

Streckeisen A (1976) To each plutonic rock its proper name. Earth-Science Reviews 12:1–33

Swan ARH, Sandilands M (1995) Introduction to Geological Data Analysis. Blackwell Sciences, Oxford

Trauth MH, Bookhagen B, Mueller A, Strecker MR (2003) Erosion and climate change in the Santa Maria Basin, NW Argentina during the last 40,000 yrs. Journal of Sedimentary Research 73 (1):82–90

Westgate JA, Shane PAR, Pearce NJG, Perkins WT, Korisettar R, Chesner CA, Williams MAJ, Acharyya SK (1998) All Toba Tephra Occurrences Across Peninsular India Belong to the 75,000 yr BP Eruption. Quaternary Research 50:107–112

Zadeh L (1965) Fuzzy sets. Information Control 8:338–353

www.it-ebooks.info

10 Directional Data

Orthoceras fossils from an outcrop at Neptuni Acrar near Byxelkrok on Öland, Sweden. *Orthoceras* is a cephalopod with a straight shell that lived in the Ordovician era, about 450 million years ago. A statistical analysis of cephalopod orientations at Neptuni Acrar has revealed a significant southerly paleocurrent direction, which is in agreement with paleogeographic reconstructions.

10.1 Introduction

Methods for analyzing circular and spherical data are widely used in earth sciences. For instance, structural geologists measure and analyze the orientation of slickensides (or striae) on fault planes. Circular statistics is also common in paleomagnetic applications. Microstructural investigations include the analysis of grain shapes and quartz c-axis orientations in thin sections. Paleoenvironmentalists also reconstruct paleocurrent directions from fossil alignments (Fig. 10.1). There are theoretically two types of directional data in earth sciences: directional data *sensu stricto*, and oriented data. Directional data, such as the paleocurrent direction of a river as documented by flute marks, or the flow direction of a glacier as indicated by glacial striae, have a true polarity. Oriented data describe axial data and lines, such as the orientation of joints, without any sense of direction.

There are a number of useful publications available on the statistical analysis of directional data, such as the books by Fisher (1993) and Mardia and Jupp (2000), and the chapters on the subject in books by Swan and Sandilands (2000), Davis (2002), and Borradaile (2003). Furthermore, Chapter 9 of the book by Middleton (1999) and two journal articles by

© Springer-Verlag Berlin Heidelberg 2015

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*,
DOI 10.1007/978-3-662-46244-7_10

www.it-ebooks.info

Fig. 10.1 *Orthoceras* fossils from an outcrop at Neptuni Acrar near Byxelkrok on Öland, Sweden. *Orthoceras* is a cephalopod with a straight shell that lived in the Ordovician era, about 450 million years ago. Such elongated, asymmetric objects tend to orient themselves in the hydrodynamically most stable position and therefore indicate paleocurrent directions. The statistical analysis of cephalopod orientations at Neptuni Acrar reveals a significant southerly paleocurrent direction, which is in agreement with the paleogeographic reconstructions.

Jones (2006a, b) discuss the use of MATLAB® for the statistical analysis of directional data. MATLAB is not the first choice for analyzing directional data since it does not provide the relevant functions, such as algorithms to compute the probability distribution function of a von Mises distribution, or to run a Rayleigh's test for the significance of a mean direction. Earth scientists have therefore developed numerous stand-alone programs with which to analyze such data, e.g., the excellent software developed by Rick Allmendinger:

```
http://www.geo.cornell.edu/geology/faculty/RWA/programs.html
```

The following tutorial on the analysis of directional data is independent of these tools. It provides simple MATLAB codes to display directional data, to compute the von Mises distribution, and to run simple statistical tests. The first section introduces rose diagrams as the most widely used method to display directional data (Section 10.2). With a similar concept to Chapter

www.it-ebooks.info

3 on univariate statistics, the next sections are then on the use of empirical and theoretical distributions to describe directional data (Sections 10.3 and 10.4). The last three sections then describe the three most important tests for directional data, these being the tests for randomness of directional data (Section 10.5), for the significance of a mean direction (Section 10.6), and for the difference between two sets of directional data (Section 10.7).

10.2 Graphical Representation

The classic way to display directional data is the rose diagram. A rose diagram is a histogram for measurements of angles. In contrast to a bar histogram with the height of the bars proportional to frequency, the rose diagram comprises segments of a circle, with the radius of each sector being proportional to the frequency. We use synthetic data to illustrate two types of rose diagram that are used to display directional data. We load a set of directional data from the file *directional_1.txt*.

```
clear

data_degrees_1 = load('directional_1.txt');
```

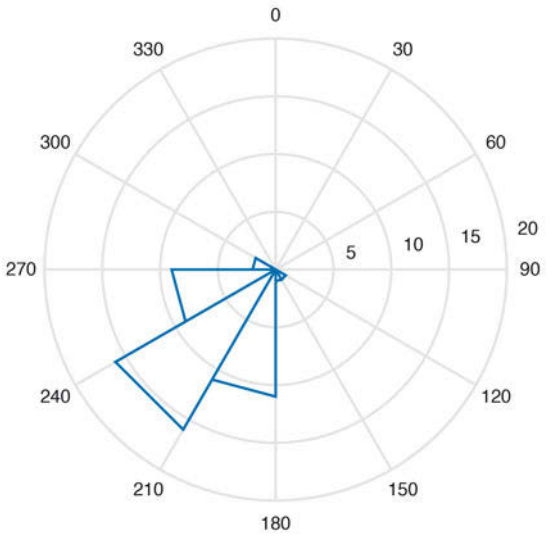


Fig. 10.2 Rose diagram to display directional data using the function `rose`. The radii of the area segments are proportional to the frequencies for each class. We using `rose` with `view(90,-90)`, so that 0° points due north and 90° points due east, i.e., the angles increase in a clockwise direction.

www.it-ebooks.info

The data set contains forty measurements of angles, in degrees. We use the function `rose(az,nb)` to display the data. The function plots an angle histogram for the angles `az` in radians, where `nb` is the number of classes. However, since the original data are in degrees we need to convert all measurements to radians before we plot the data.

```
data_radians_1 = pi*data_degrees_1/180;
rose(data_radians_1,12)
```

The function `rose` counts in a counterclockwise direction, with zero degrees lying along the *x*-axis of the coordinate graph. In geosciences, however, 0° points due north, 90° points due east, and the angles increase in a clockwise direction. The command `view` rotates the plot by +90° (the azimuth) and mirrors the plot by –90° (the elevation) (Fig. 10.2).

```
rose(data_radians_1,12)
view(90,-90)
```

The area of the arc segments increases with the square of the frequency. In a modification the rose diagram is therefore scaled to the square root of the class frequency. The function `rose` does not allow plotting of the square root of the frequencies by default, but the corresponding file `rose.m` can be easily

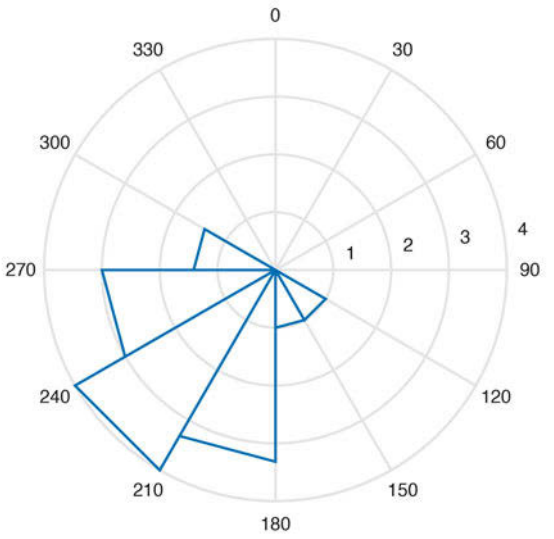


Fig. 10.3 Modified rose diagram to display directional data using the function `rose_sqrt`. The plot scales the rose diagram to the square root of the class frequency. The area of the arc segments then increases with frequency.

www.it-ebooks.info

modified as follows. After the histogram of the angles is computed in line 57 using the function `histc`, add a line with the command `nn = sqrt(nn)`, which computes the square root of the frequencies `nn`. Save the modified function as file `rose_sqrt.m` and apply the new function to the data set.

Movie 10.1

```
rose_sqrt(data_radians_1,12)
view(90,-90)
```

This plot satisfies all conventions in geosciences (Fig. 10.3).

10.3 Empirical Distributions

This section introduces statistical measures used to describe empirical distributions of directional data. The characteristics of directional data are described by measures of central tendency and dispersion, similar to the statistical characterization of univariate data sets (Chapter 3). Assume that we have collected a number of angular measurements such as fossil alignments. The collection of data can be written as

$$\theta_1, \theta_2, \theta_3, \ldots, \theta_N$$

containing N observations θ_i. Sine and cosine values are computed for each direction θ_i to compute the *resultant* or *mean direction* for the set of angular data (Fisher 1993, Mardia and Jupp 2000).

$$x_r = \sum \sin\theta_i$$
$$y_r = \sum \cos\theta_i$$

The resultant direction of the data set is

$$\overline{\theta} = \tan^{-1}(x_r / y_r)$$

The length of the resultant is

$$R = \sqrt{(x_r^{\,2} + y_r^{\,2})}$$

The resultant length clearly depends on the dispersion of the data. Normalizing the resultant length to the number of observations yields the mean resultant length.

www.it-ebooks.info

$$\overline{R} = R / N$$

Movie 10.2

The value of the mean resultant length decreases with increasing dispersion (Fig. 10.4). The difference between one and the mean resultant length is therefore often used as a measure of dispersion for directional data,

$$\sigma_0 = 1 - \overline{R}$$

which is the *circular variance*.

The following example illustrates the use of these parameters by means of synthetic directional data. We first load the data from the file *directional_1.txt* and convert all measurement to radians.

```
clear

data_degrees_1 = load('directional_1.txt');
data_radians_1 = pi*data_degrees_1/180;
```

We now calculate the resultant vector `R`. Firstly, we compute the x and y components of the resultant vector.

```
x_1 = sum(sin(data_radians_1))
y_1 = sum(cos(data_radians_1))

x_1 =
  -24.3898

y_1 =
  -25.9401
```

The mean direction is the inverse tangent of the ratio of x and y.

```
mean_radians_1 = atan(x_1/y_1)
mean_degrees_1 = 180*mean_radians_1/pi
```

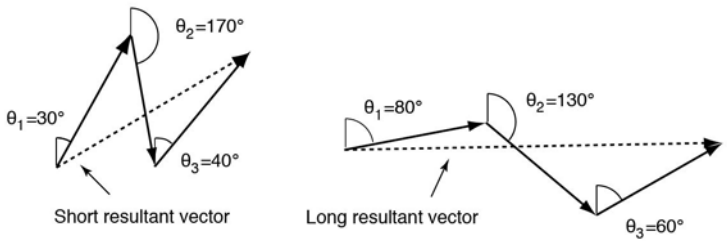


Fig. 10.4 The resultant length R of a sample decreases with increasing dispersion of the data θ_i.

www.it-ebooks.info

```
mean_radians_1 =
    0.7546

mean_degrees_1 =
   43.2357
```

This result suggests that the resultant vector R is around 0.75 radians or 43°. However, since both x and y are negative, the true value of `mean_degrees` is located in the third quadrant and we therefore add 180°

```
mean_degrees_1 = mean_degrees_1 + 180

mean_degrees_1 =
  223.2357
```

which results in a mean direction of around 223°. The length of this vector is the absolute value of the vector, which is

```
R_1 = sqrt(x_1^2 + y_1^2)

R_1 =
   35.6055
```

The resultant length depends on the dispersion of the directional data. Normalizing the resultant length to the sample size yields the mean resultant length `Rm` of

```
Rm_1 = R_1 / (length(data_radians_1))

Rm_1 =
    0.8901
```

A high `Rm` value suggests less variance. We then compute the circular variance `sigma`, which is

```
sigma_1 = 1 - Rm_1

sigma_1 =
    0.1099
```

10.4 Theoretical Distributions

As in Chapter 3, the next step in a statistical analysis is to find a suitable theoretical distribution that we fit to the empirical distribution visualized and described in the previous section. The classic theoretical distribution to describe directional data is the *von Mises distribution*, named after the Austrian mathematician Richard Edler von Mises (1883–1953). The probability density function of a von Mises distribution is

www.it-ebooks.info

$$f(\theta) = \frac{1}{2\pi I_0(\kappa)} e^{\kappa \cos(\theta - \mu)}$$

where μ is the mean direction and κ is the concentration parameter (Mardia and Jupp 2000) (Fig. 10.5). $I_0(\kappa)$ is the modified Bessel function of first kind and order zero. The Bessel functions are solutions of a second-order differential equation (Bessel's differential equation) and are important in many problems of wave propagation in a cylindrical waveguide, and of heat conduction in a cylindrical object. The von Mises distribution is also known as the circular normal distribution since it has similar characteristics to a normal distribution (Section 3.4). The von Mises distribution is used when the mean direction is the most frequent direction. The probability of deviations is equal on either side of the mean direction and decreases with increasing distance from the mean direction.

As an example let us assume a mean direction of `mu=0` and five different values for the concentration parameter `kappa`.

```
clear
```

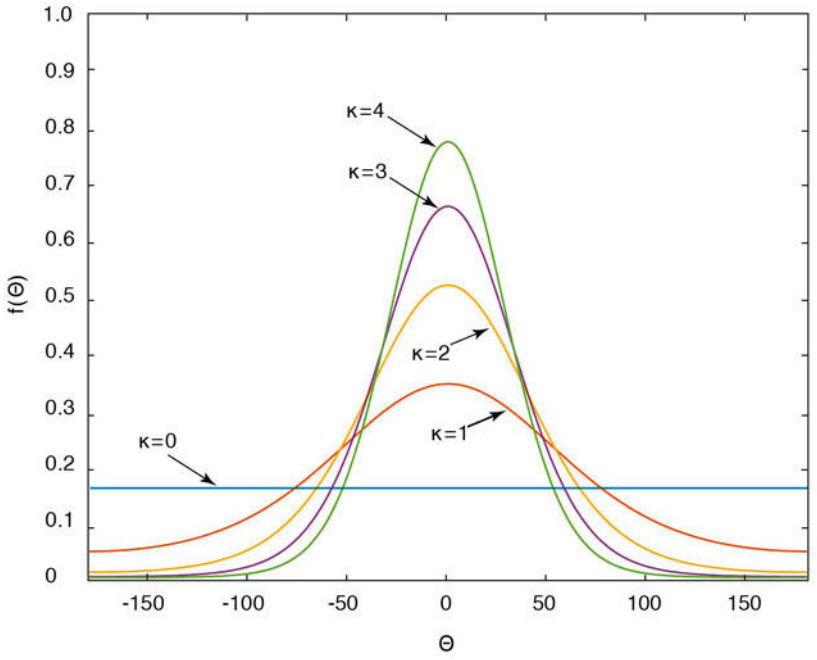


Fig. 10.5 Probability density function $f(x)$ of a von Mises distribution with μ=0 and five different values for κ.

www.it-ebooks.info

```
mu = 0; kappa = [0 1 2 3 4]';
```

We first define an angle scale for a plot that runs from –180 to 180 degrees, with intervals of one degree.

```
theta = -180:1:180;
```

All angles are converted from degrees to radians.

```
mu_radians = pi*mu/180;
theta_radians = pi*theta/180;
```

We then compute the von Mises distribution for these values. The formula uses the modified Bessel function of first kind and order zero that can be calculated using the function `besseli`. We compute the probability density function for the five values of `kappa`.

```
for i = 1:5
   mises(i,:) = (1/(2*pi*besseli(0,kappa(i))))* ...
   exp(kappa(i)*cos(theta_radians-mu_radians));
   theta(i,:) = theta(1,:);
end
```

The results are plotted by

```
for i = 1:5
   plot(theta(i,:),mises(i,:))
   axis([-180 180 0 max(mises(i,:))])
   hold on
end
```

The mean direction and concentration parameters of such theoretical distributions can be easily modified for comparison with empirical distributions.

10.5 Test for Randomness of Directional Data

The first test for directional data compares the data set with a uniform distribution. Directional data following a uniform distribution are purely random, i.e., there is no preference for any direction. We use the χ^2-test (Section 3.8) to compare the empirical frequency distribution with the theoretical uniform distribution. We first load our sample data.

```
clear

data_degrees_1 = load('directional_1.txt');
```

We then use the function `histogram` to count the number of observations

www.it-ebooks.info

within 12 classes, each with a width of 30 degrees.

```
h = histogram(data_degrees_1,0:30:360);
counts = h.Values;
```

The expected number of observations is 40/12, where 40 is the total number of observations and 12 is the number of classes.

```
expect = 40/12 * ones(1,12);
```

The χ^2-test explores the squared differences between the observed and expected frequencies. The quantity χ^2 is defined as the sum of these squared differences divided by the expected frequencies.

```
chi2 = sum((counts - expect).^2 ./expect)

chi2 =
    102.8000
```

The critical χ^2 value can be calculated using `chi2inv`. The χ^2-test requires the degrees of freedom Φ. In our example we test the hypothesis that the data are uniformly distributed, i.e., we estimate one parameter, which is the number of possible values N. Since the number of classes is 12, the number of degrees of freedom is Φ=12–1–1=10. We test our hypothesis on a p=95% significance level. The function `chi2inv` computes the inverse of the cumulative distribution function (CDF) of the χ^2 distribution with parameters specified by Φ for the corresponding probabilities in p.

```
chi2inv(0.95,12-1-1)

ans =
    18.3070
```

Since the critical χ^2 of 18.3070 is well below the measured χ^2 of 102.8000 we reject the null hypothesis and conclude that our data do not follow a uniform distribution, i.e., they are not randomly distributed.

10.6 Test for the Significance of a Mean Direction

Having measured a set of directional data in the field, we may wish to know whether there is a prevailing direction documented in the data. We use the Rayleigh's test for the significance of a mean direction (Mardia 1972, Mardia and Jupp 2000). This test uses the mean resultant length introduced in Section 10.3, which increases as the preferred direction becomes more significant.

www.it-ebooks.info

$$\overline{R} = \frac{1}{n}\sqrt{(\sum \sin\theta_i)^2 + (\sum \cos\theta_i)^2}$$

The data show a preferred direction if the calculated mean resultant length is below the critical value (Mardia 1972). As an example we again load the data contained in the file *directional_1.txt*.

```
clear

data_degrees_1 = load('directional_1.txt');
data_radians_1 = pi*data_degrees_1/180;
```

We then calculate the mean resultant vector `Rm`.

Table 10.1 Critical values of mean resultant length for Rayleigh's test for the significance of a mean direction of *N* samples (Mardia 1972).

	Level of Significance, α				
N	0.100	0.050	0.025	0.010	0.001
5	0.677	0.754	0.816	0.879	0.991
6	0.618	0.690	0.753	0.825	0.940
7	0.572	0.642	0.702	0.771	0.891
8	0.535	0.602	0.660	0.725	0.847
9	0.504	0.569	0.624	0.687	0.808
10	0.478	0.540	0.594	0.655	0.775
11	0.456	0.516	0.567	0.627	0.743
12	0.437	0.494	0.544	0.602	0.716
13	0.420	0.475	0.524	0.580	0.692
14	0.405	0.458	0.505	0.560	0.669
15	0.391	0.443	0.489	0.542	0.649
16	0.379	0.429	0.474	0.525	0.630
17	0.367	0.417	0.460	0.510	0.613
18	0.357	0.405	0.447	0.496	0.597
19	0.348	0.394	0.436	0.484	0.583
20	0.339	0.385	0.425	0.472	0.569
21	0.331	0.375	0.415	0.461	0.556
22	0.323	0.367	0.405	0.451	0.544
23	0.316	0.359	0.397	0.441	0.533
24	0.309	0.351	0.389	0.432	0.522
25	0.303	0.344	0.381	0.423	0.512
30	0.277	0.315	0.348	0.387	0.470
35	0.256	0.292	0.323	0.359	0.436
40	0.240	0.273	0.302	0.336	0.409
45	0.226	0.257	0.285	0.318	0.386
50	0.214	0.244	0.270	0.301	0.367
100	0.150	0.170	0.190	0.210	0.260

www.it-ebooks.info

```
x_1 = sum(sin(data_radians_1));
y_1 = sum(cos(data_radians_1));

mean_radians_1 = atan(x_1/y_1);
mean_degrees_1 = 180*mean_radians_1/pi;
mean_degrees_1 = mean_degrees_1 + 180;

Rm_1 = 1/length(data_degrees_1) .*(x_1.^2+y_1.^2).^0.5

Rm_1 =
    0.8901
```

The mean resultant length in our example is 0.8901. The critical `Rm` (α=0.05, *n*=40) is 0.273 (Table 10.1 from Mardia 1972). Since this value is lower than the `Rm` from the data we reject the null hypothesis and conclude that there is a preferred single direction, which is

```
theta_1 = 180 * atan(x_1/y_1) / pi

theta_1 =
   43.2357
```

The negative signs of the sine and cosine, however, suggest that the true result is in the third sector (180–270°), and the correct result is therefore 180+43.2357=223.2357.

10.7 Test for the Difference between Two Sets of Directions

Let us consider two sets of measurements in two files *directional_1.txt* and *directional_2.txt*. We wish to compare the two sets of directions and test the hypothesis that these are significantly different. We use the Watson-William test to test the similarity between two mean directions

$$F = \left(1 + \frac{3}{8\kappa}\right)\frac{(n-2)(R_A + R_B - R_T)}{n - R_A - R_B}$$

where κ is the concentration parameter, R_A and R_B are the resultant lengths of samples *A* and *B*, respectively, and R_T is the resultant lengths of the combined samples (Watson and Williams 1956, Mardia and Jupp 2000). The concentration parameter can be obtained from tables using R_T (Batschelet 1965, Gumbel et al. 1953, Table 10.2). The calculated *F* is compared with critical values from the standard *F* tables (Section 3.8). The two mean directions are not significantly different if the calculated *F*-value is less than the critical *F*-value, which depends on the degrees of freedom Φ_a=1 and Φ_b=*n*–2, and also on the significance level α. Both samples must follow a von

www.it-ebooks.info

Mises distribution (Section 10.4).

We use two synthetic data sets of directional data to illustrate the application of this test. We first load the data and convert the degrees to radians.

```
clear

data_degrees_1 = load('directional_1.txt');
data_degrees_2 = load('directional_2.txt');

data_radians_1 = pi*data_degrees_1/180;
data_radians_2 = pi*data_degrees_2/180;
```

We then compute the lengths of the resultant vectors.

```
x_1 = sum(sin(data_radians_1));
y_1 = sum(cos(data_radians_1));
x_2 = sum(sin(data_radians_2));
y_2 = sum(cos(data_radians_2));

mean_radians_1 = atan(x_1/y_1);
mean_degrees_1 = 180*mean_radians_1/pi;
mean_radians_2 = atan(x_2/y_2);
mean_degrees_2 = 180*mean_radians_2/pi;

mean_degrees_1 = mean_degrees_1 + 180
mean_degrees_2 = mean_degrees_2 + 180

R_1 = sqrt(x_1^2 + y_1^2);
R_2 = sqrt(x_2^2 + y_2^2);

mean_degrees_1 =
  223.2357

mean_degrees_2 =
  200.8121
```

The orientations of the resultant vectors are approximately 223° and 201°. We also need the resultant length for both samples combined, so we combine both data sets and again compute the resultant length.

```
data_radians_T = [data_radians_1;data_radians_2];

x_T = sum(sin(data_radians_T));
y_T = sum(cos(data_radians_T));

mean_radians_T = atan(x_T/y_T);
mean_degrees_T = 180*mean_radians_T/pi;

mean_degrees_T = mean_degrees_T + 180;

R_T = sqrt(x_T^2 + y_T^2)
Rm_T = R_T / (length(data_radians_T))
```

www.it-ebooks.info

```
R_T =
   69.5125

Rm_T =
    0.8689
```

We apply the test statistic to the data for `kappa`=4.177 for `Rm_T`=0.8689 (Table 10.2). The computed value for `F` is

```
n = length(data_radians_T);

F = (1+3/(8*4.177)) * (((n-2)*(R_1+R_2-R_T))/(n-R_1-R_2))

F =
   12.5844
```

Using the *F*-statistic, we find that for 1 and 80–2 degrees of freedom and α=0.05, the critical value is

```
finv(0.95,1,78)

ans =
    3.9635
```

Table 10.2 Maximum likelihood estimates of concentration parameter κ for calculated mean resultant length *R* (adapted from Batschelet 1965 and Gumbel et al. 1953).

R	κ	*R*	κ	*R*	κ	*R*	κ
0.000	0.000	0.260	0.539	0.520	1.224	0.780	2.646
0.010	0.020	0.270	0.561	0.530	1.257	0.790	2.754
0.020	0.040	0.280	0.584	0.540	1.291	0.800	2.871
0.030	0.060	0.290	0.606	0.550	1.326	0.810	3.000
0.040	0.080	0.300	0.629	0.560	1.362	0.820	3.143
0.050	0.100	0.310	0.652	0.570	1.398	0.830	3.301
0.060	0.120	0.320	0.676	0.580	1.436	0.840	3.479
0.070	0.140	0.330	0.700	0.590	1.475	0.850	3.680
0.080	0.161	0.340	0.724	0.600	1.516	0.860	3.911
0.090	0.181	0.350	0.748	0.610	1.557	0.870	4.177
0.100	0.201	0.360	0.772	0.620	1.600	0.880	4.489
0.110	0.221	0.370	0.797	0.630	1.645	0.890	4.859
0.120	0.242	0.380	0.823	0.640	1.691	0.900	5.305
0.130	0.262	0.390	0.848	0.650	1.740	0.910	5.852
0.140	0.283	0.400	0.874	0.660	1.790	0.920	6.539
0.150	0.303	0.410	0.900	0.670	1.842	0.930	7.426
0.160	0.324	0.420	0.927	0.680	1.896	0.940	8.610
0.170	0.345	0.430	0.954	0.690	1.954	0.950	10.272
0.180	0.366	0.440	0.982	0.700	2.014	0.960	12.766
0.190	0.387	0.450	1.010	0.710	2.077	0.970	16.927
0.200	0.408	0.460	1.039	0.720	2.144	0.980	25.252
0.210	0.430	0.470	1.068	0.730	2.214	0.990	50.242
0.220	0.451	0.480	1.098	0.740	2.289	0.995	100.000
0.230	0.473	0.490	1.128	0.750	2.369	0.999	500.000
0.240	0.495	0.500	1.159	0.760	2.455	1.000	5000.000
0.250	0.516	0.510	1.191	0.770	2.547		

www.it-ebooks.info

which is well below the observed value of F=12.5844 (Section 3.8). We therefore reject the null hypothesis and conclude that the two samples could not have been drawn from populations with the same mean direction.

Recommended Reading

Batschelet E (1965) Statistical Methods for the Analysis of Problems in Animal Orientation and Certain Biological Rhythms. American Institute of Biological Sciences Monograph, Washington DC

Borradaile G (2003) Statistics of Earth Science Data – Their Distribution in Time, Space and Orientation. Springer, Berlin Heidelberg New York

Davis JC (2002) Statistics and Data Analysis in Geology, Third Edition. John Wiley and Sons, New York

Fisher NI (1993) Statistical Analysis of Circular Data. Cambridge University Press, New York

Gumbel EJ, Greenwood JA, Durand D (1953) The Circular Normal Distribution: Tables and Theory. Journal of the American Statistical Association 48:131–152

Jones TA (2006a) MATLAB functions to analyze directional (azimuthal) data–I: Single sample inference. Computers and Geosciences 32:166–175

Jones TA (2006b) MATLAB functions to analyze directional (azimuthal) data–II: Correlation. Computers and Geosciences 32:176–183

Mardia KV (1972) Statistics of Directional Data. Academic Press, London

Mardia KV, Jupp PE (2000) Directional Statistics. Wiley, Chichester

Middleton GV (1999) Data Analysis in the Earth Sciences Using MATLAB. Prentice Hall, New Jersey

Swan ARH, Sandilands M (1995) Introduction to geological data analysis. Blackwell Sciences, Oxford

Watson GS, Williams EJ (1956) On the construction of significance tests on a circle and the sphere. Biometrika 43:344–352

www.it-ebooks.info