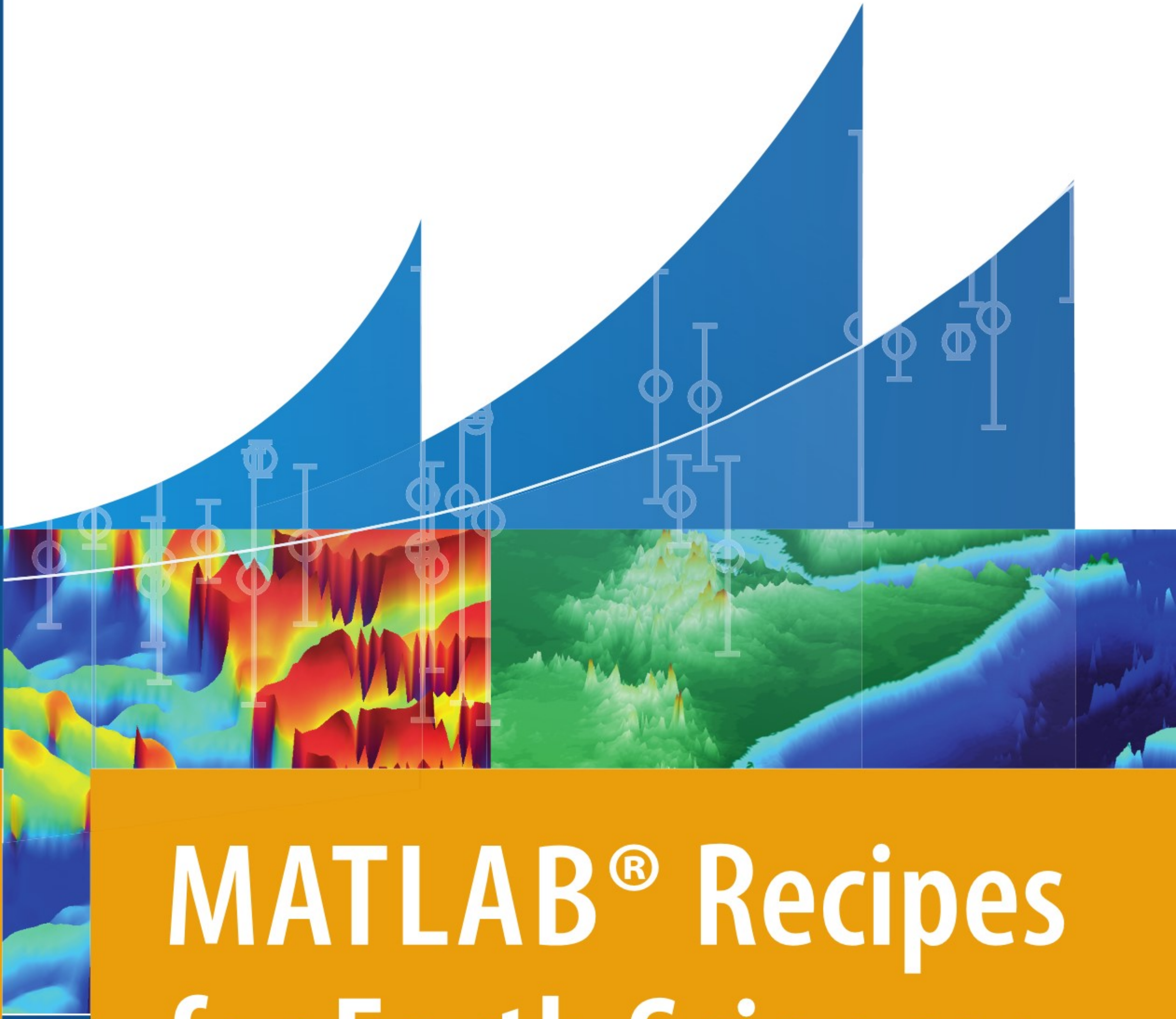


# MATLAB® Recipes for Earth Sciences

Third Edition

EXTRA MATERIALS
extras.springer.com

Springer

Martin H. Trauth

# MATLAB® Recipes for Earth Sciences

Third Edition

Martin H. Trauth

# MATLAB® Recipes for Earth Sciences

Third Edition

With Contributions by
Robin Gebbers and Norbert Marwan
and illustrations by Elisabeth Sillmann

Springer

Privatdozent Dr. rer. nat. habil. Martin H. Trauth
University of Potsdam
Department of Earth and Environmental Sciences
Karl-Liebknecht-Str. 24
14476 Potsdam
Germany
trauth@geo.uni-potsdam.de

ISBN 978-3-642-12761-8 e-ISBN 978-3-642-12762-5
DOI 10.1007/978-3-642-12762-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2010930277

Typesetting and book design by Elisabeth Sillmann, www.blaetterwaldDesign.de Landau, Germany

*Cover design*: deblik, Berlin

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

# Preface

The book *MATLAB Recipes for Earth Sciences* is designed to help undergraduates, and PhD students, post-doctoral researchers, and professionals find quick solutions for common problems in data analysis in earth sciences. It provides a minimum 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*, to solve typical problems in earth sciences, such as 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. It comes with a compact disk that contains all MATLAB recipes and example data files as well as presentation files for instructors. The MATLAB codes can be easily modified for application to the reader's data and projects.

This revised and updated Third Edition includes new sections on software-related issues (Sections 2.4, 2.5, 2.8 and 2.9). Chapter 2 was difficult to update since MATLAB has expanded so much over the years, and I have deliberately tried to restrict this chapter to demonstrating of those tools actually used in the book. A second difficulty arose from the current move by *The MathWorks Inc.* to use and incorporate objects and classes in some areas of their MATLAB routines, although there does not seem to be any intention of abandoning the existing procedural code. Again, I have restricted the introduction and use of objects and classes to the absolute minimum, even at the expense of omitting one of the new features of MATLAB. Some functions, however, such as those for distribution fitting use this new concept of object-oriented programming, and I hope that the reader will forgive me for not explaining all the details of the MATLAB code when using it. The other new sections are on distribution fitting (Section 3.9), and on nonlinear and weighted regression (Section 4.10), as these techniques are widely used in, for instance, isotope geochemistry and geochronology. Sections 8.7

to 8.9 introduce some advanced methods in image analysis such the extraction of color-intensity transects from laminated sediments, automatic grain size analysis, and the quantification of charcoal in microscope images. These techniques are frequently used in my research projects and are always in demand during the short courses that I teach.

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 7 Release R2010a, 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, the more experienced reader may use only one particular method to solve a specific problem. The concept of the book and the contents of its chapters 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 Third Edition now includes new sections on data structures and classes of objects, on generating M-files to regenerate graphs and on publishing M-files.

- *Chapters 3 and 4* – Fundamentals in univariate and bivariate statistics. These two chapters contain basic concepts in statistics, and also introduces advanced topics such as resampling schemes and cross validation. The reader already familiar with basic statistics might skip these two chapters. The Third Edition now includes new sections on fitting normal distributions to observations and on nonlinear and weighted regression analysis.

- *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.

- *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 Third Edition now includes new sections on color-intensity transects of laminated sediments, automated grain size analysis from photos and quantifying charcoal in microscope images.

- *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.

- *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 the grain shapes and quartz c-axis orientations in thin sections.

While the book *MATLAB Recipes for Earth Sciences* is about data analysis it does not attempt to cover modeling. For this subject, I recommend the excellent book *Environmental Modeling Using MATLAB* by Ekkehard Holzbecher (Springer 2007), which first introduces basic concepts of modeling and then provides a great overview of modeling examples using MATLAB. Holzbecher's book uses a very similar concept to *MATLAB Recipes for Earth Sciences* as it gives a brief introduction to the theory, and then explains MATLAB examples. Neither book provides a complete introduction to all available techniques, but they both provide a quick overview of basic concepts for data analysis and modeling in earth sciences. One of the few critical reviewers of the First Edition of *MATLAB Recipes for Earth Sciences* raised the question of why I had not included a chapter on finite-element and finite-difference modeling, and on solving differential equations – in his opinion a major omission in the book. However, this is far beyond of the scope of the book and my own expertise. Students and colleagues interested in this topic are directed to the book

*MATLAB Guide to Finite Elements: An Interactive Approach* by Peter I. Kattan (Springer 2007). While my book may be considered by some to be a little light on image processing, I have included in Chapter 8 three new sections on the analysis of sediment images. I would also strongly recommend to anyone interested in this topic the very successful book *Digital Image Processing Using MATLAB* by Gonzales, Woods and Eddins (Gatesmark Publishing 2009), for which a 2nd edition has just been published.

I have taken all other critiques quite seriously and invite all readers to also comment on the Third Edition: the book is constantly changing and evolving. As the Third Edition appears on the bookshelves I will create a new folder on the hard disk of my computer named *Fourth Edition*, where new ideas will be collected. The book has benefited from the comments of many people, in particular my contributing authors Robin Gebbers and Norbert Marwan, my colleagues Ira Ojala, Lydia Olaka, Jim Renwick, Jochen Rössler, Rolf Romer, Annette Witt, and the students Matthias Gerber, Mathis Hain, Martin Homann, Stefanie von Lonski, Oliver Rach, Marius Walter and Max Zitzmann. 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 Fernandez from the *Book Program* and Kate Fiore from *Academic Support* at *The MathWorks Inc.*, Claudia Olrogge and Annegret Schumann at *The MathWorks GmbH Deutschland*, Christian Witschel, Chris Bendall and their team at *Springer*, and Andreas Bohlen, Brunhilde Schulz and their team at *UP Transfer GmbH*. I also thank the *NASA/GSFC/METI/ERSDAC/JAROS* and the *U. S./Japan ASTER Science Team* and the director Mike Abrams for allowing me to include the ASTER images in this book.

Potsdam, April 2010

Martin Trauth

# Contents

**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

**2 Introduction to MATLAB ..... 11**

2.1 MATLAB in Earth Sciences ..... 11
2.2 Getting Started ..... 12
2.3 The Syntax ..... 14
2.4 Data Storage and Handling ..... 18
2.5 Data Structures and Classes of Objects ..... 21
2.6 Scripts and Functions ..... 26
2.7 Basic Visualization Tools ..... 29
2.8 Generating M-Files to Regenerate Graphs ..... 32
2.9 Publishing M-Files ..... 35

**3 Univariate Statistics ..... 37**

3.1 Introduction ..... 37
3.2 Empirical Distributions ..... 37
3.3 Example of Empirical Distributions ..... 44
3.4 Theoretical Distributions ..... 51
3.5 Example of Theoretical Distributions ..... 59
3.6 The t-Test ..... 61
3.7 The F-Test ..... 66
3.8 The $\chi^2$-Test ..... 70
3.9 Distribution Fitting ..... 73

4 **Bivariate Statistics** ........ **79**

4.1 Introduction ........ 79
4.2 Pearson's Correlation Coefficient ........ 80
4.3 Classical Linear Regression Analysis and Prediction ........ 88
4.4 Analyzing the Residuals ........ 92
4.5 Bootstrap Estimates of the Regression Coefficients ........ 94
4.6 Jackknife Estimates of the Regression Coefficients ........ 95
4.7 Cross Validation ........ 98
4.8 Reduced Major Axis Regression ........ 99
4.9 Curvilinear Regression ........ 100
4.10 Nonlinear and Weighted Regression ........ 103

5 **Time-Series Analysis** ........ **107**

5.1 Introduction ........ 107
5.2 Generating Signals ........ 108
5.3 Auto-Spectral and Cross-Spectral Analysis ........ 112
5.4 Examples of Auto-Spectral and Cross-Spectral Analysis ........ 117
5.5 Interpolating and Analyzing Unevenly-Spaced Data ........ 126
5.6 Evolutionary Power Spectrum ........ 131
5.7 Lomb-Scargle Power Spectrum ........ 135
5.8 Wavelet Power Spectrum ........ 139
5.9 Nonlinear Time-Series Analysis (*by N. Marwan*) ........ 146

6 **Signal Processing** ........ **161**

6.1 Introduction ........ 161
6.2 Generating Signals ........ 162
6.3 Linear Time-Invariant Systems ........ 164
6.4 Convolution and Filtering ........ 166
6.5 Comparing Functions for Filtering Data Series ........ 169
6.6 Recursive and Nonrecursive Filters ........ 172
6.7 Impulse Response ........ 173
6.8 Frequency Response ........ 176
6.9 Filter Design ........ 182
6.10 Adaptive Filtering ........ 185

7 **Spatial Data** ........ **193**

7.1 Types of Spatial Data ........ 193
7.2 The GSHHS Shoreline Data Set ........ 194

7.3 The 2-Minute Gridded Global Relief Data ETOPO2 ........ 196
7.4 The 30-Arc Seconds Elevation Model GTOPO30 ........ 199
7.5 The Shuttle Radar Topography Mission SRTM ........ 201
7.6 Gridding and Contouring Background ........ 204
7.7 Gridding Example ........ 207
7.8 Comparison of Methods and Potential Artifacts ........ 211
7.9 Statistics of Point Distributions ........ 216
7.10 Analysis of Digital Elevation Models (*by R. Gebbers*) ........ 224
7.11 Geostatistics and Kriging (*by R. Gebbers*) ........ 235

**8 Image Processing ........ 255**

8.1 Introduction ........ 255
8.2 Data Storage ........ 256
8.3 Importing, Processing and Exporting Images ........ 261
8.4 Importing, Processing and Exporting Satellite Images ........ 266
8.5 Georeferencing Satellite Images ........ 268
8.6 Digitizing from the Screen ........ 271
8.7 Color-Intensity Transects of Varved Sediments ........ 274
8.8 Grain Size Analysis from Microscope Images ........ 279
8.9 Quantifying Charcoal in Microscope Images ........ 286

**9 Multivariate Statistics ........ 291**

9.1 Introduction ........ 291
9.2 Principal Component Analysis ........ 293
9.3 Independent Component Analysis (*by N. Marwan*) ........ 300
9.4 Cluster Analysis ........ 304

**10 Statistics on Directional Data ........ 311**

10.1 Introduction ........ 311
10.2 Graphical Representation ........ 312
10.3 Empirical Distributions ........ 313
10.4 Theoretical Distributions ........ 318
10.5 Test for Randomness of Directional Data ........ 320
10.6 Test for the Significance of a Mean Direction ........ 321
10.7 Test for the Difference Between Two Sets of Directions ........ 322

**General Index ........ 327**

# 1 Data Analysis in Earth Sciences

## 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.

## 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

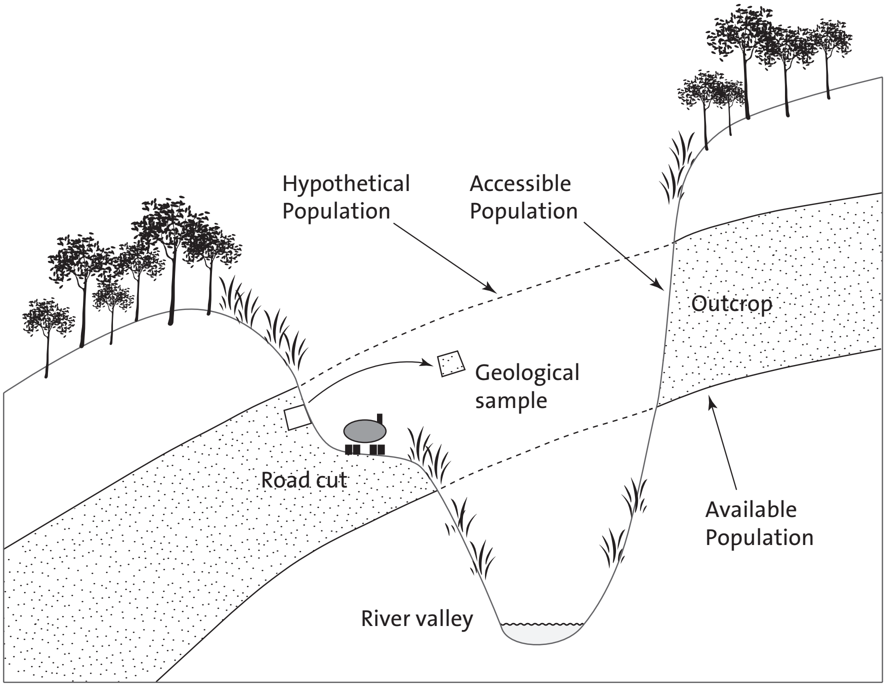


**Fig. 1.1** Samples and populations. Deep valley incision has eroded parts of a sandstone unit (*hypothetical population*). The remaining sandstone (*available population*) can only be sampled from outcrops, i.e., road cuts and quarries (*accessible population*). Note the difference between a statistical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

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 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 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 spaced. Mohs' hardness scale is one example of an ordinal scale. The hard-

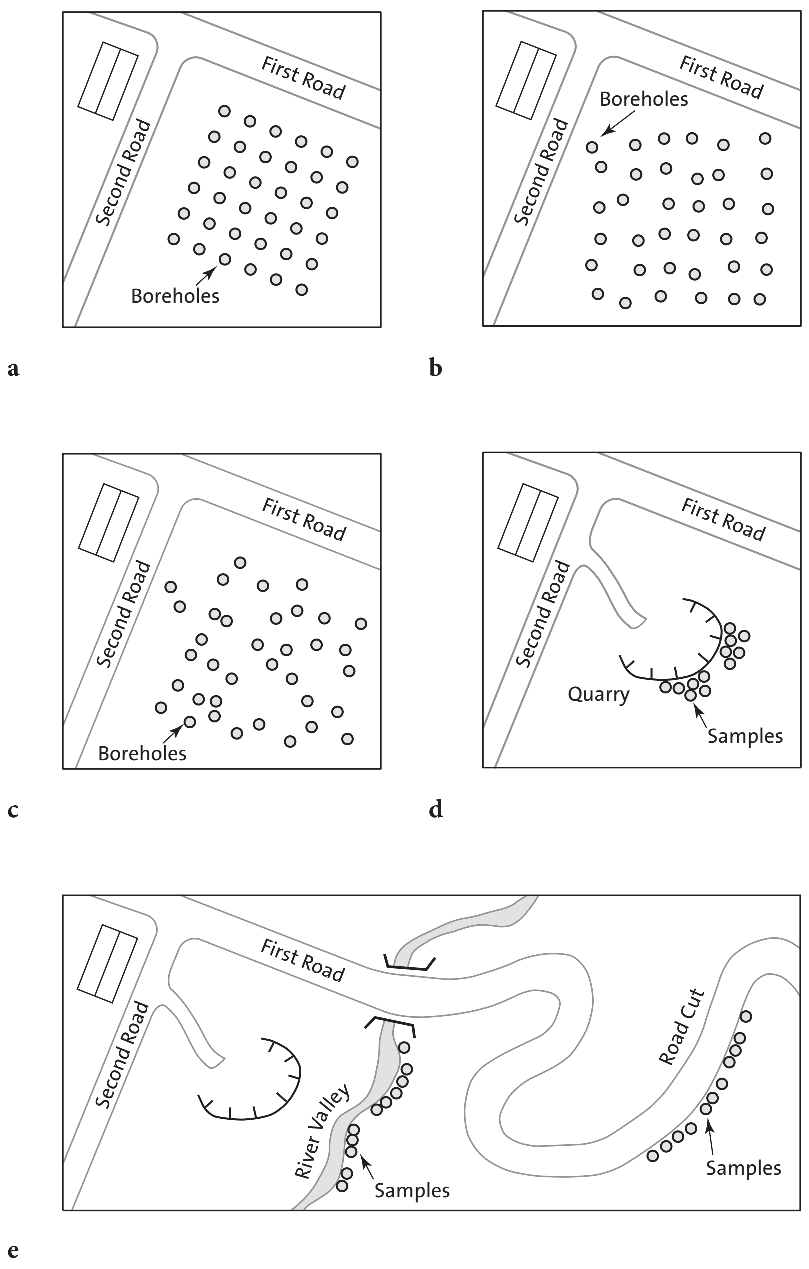


**Fig. 1.2** Sampling schemes. **a** *Regular sampling* on an evenly-spaced rectangular grid, **b** *uniform sampling* by obtaining samples randomly located within regular grid squares, **c** *random sampling* using uniformly-distributed *xy* coordinates, **d** *clustered sampling* constrained by limited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

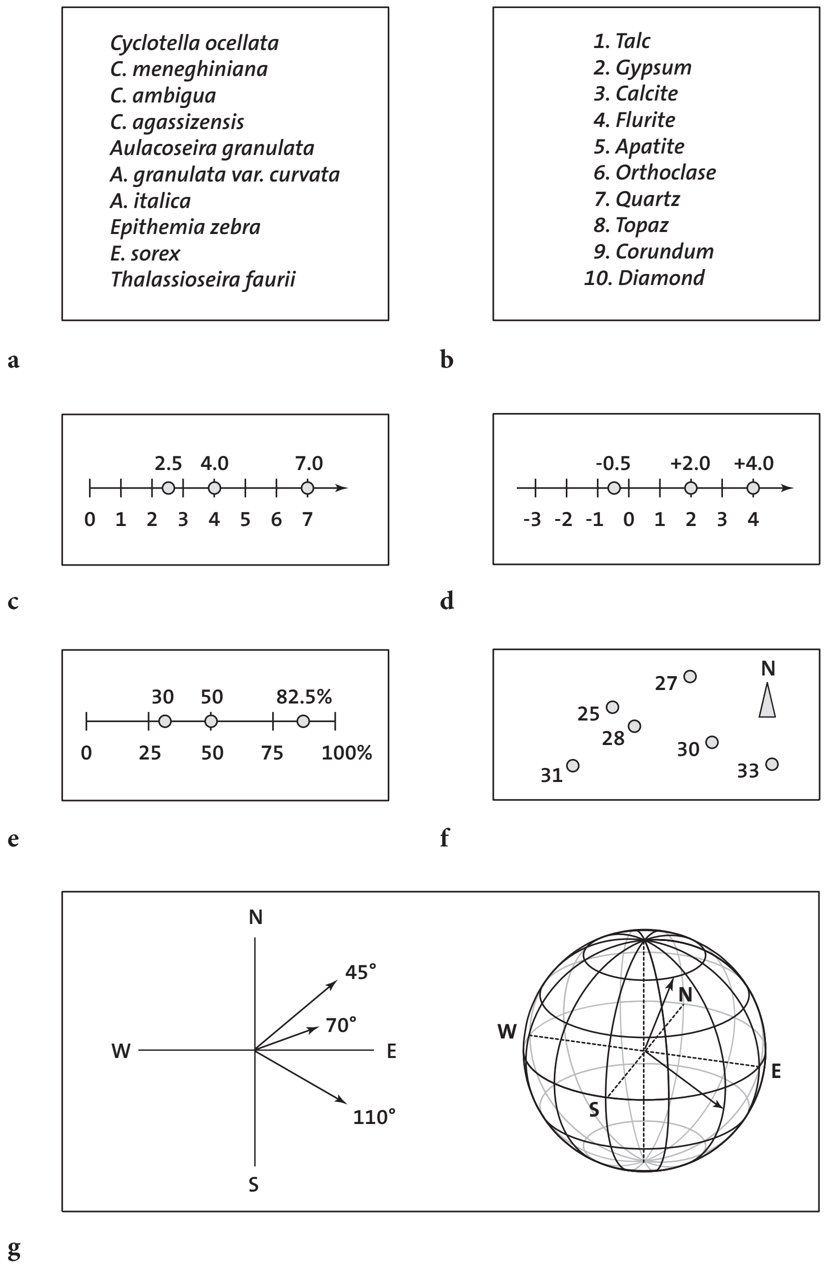


**Fig. 1.3** Types of earth science data. **a** Nominal data, **b** ordinal data, **c** ratio data, **d** interval data, **e** closed data, **f** spatial data, and **g** directional data. All of these data types are described in this book.

ness 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).

- *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.

Most of these different types of data require specialized methods of analysis, 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. Examples are the sodium content of volcanic glass shards that have been affected by chemical weathering, or 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 $SiO_2$.

- *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 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 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

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

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

Holzbecher E (2007) Environmental Modeling using 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

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

# 2 Introduction to MATLAB

## 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 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, to the setup on the computer (Section 2.2), the syntax (Section 2.3), data input and output (Sections 2.4 and 2.5), programming (Section 2.6), and visualization (Section 2.7). Advanced sections are also included on generating M-files to regenerate graphs (Section 2.8) and on publishing M-files (Section 2.9). It is recommended to go through the entire chapters in order to obtain a good knowledge of the software before proceeding to the following chapter. A more detailed introduction can be found in the *MATLAB 7 Getting Started Guide* (The MathWorks 2010) which is available in print form, online and as PDF file.

In this book we use MATLAB Version 7.10 (Release 2010a), the Image Processing Toolbox Version 7.0, the Mapping Toolbox Version 3.1, the Signal Processing Toolbox Version 6.13, the Statistics Toolbox Version 7.3 and the Wavelet Toolbox Version 4.5.

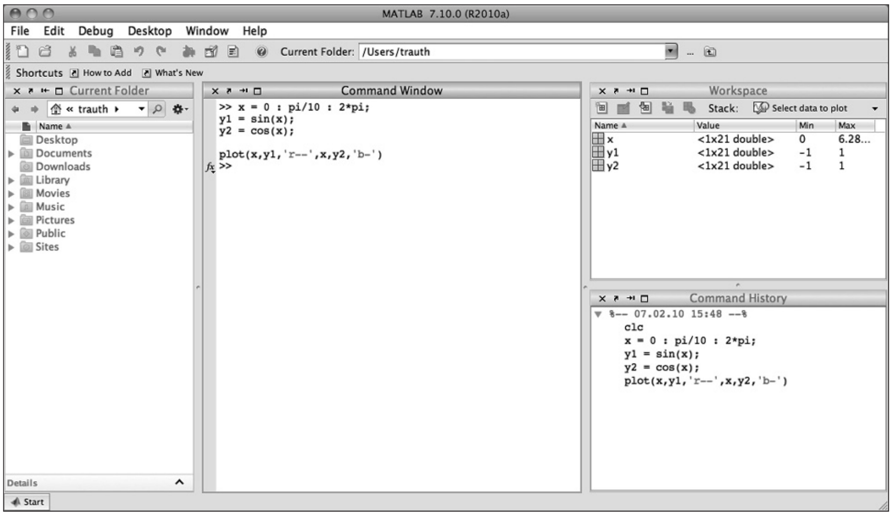
## 2.2 Getting Started

The software package comes with extensive documentation, tutorials and examples. The first three chapters of the book *MATLAB 7 Getting Started Guide* (The MathWorks 2010) are directed at beginners. The chapters on programming, creating graphical user interfaces (GUI) and development environments are aimed at more advanced users. Since *MATLAB 7 Getting Started Guide* 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
```

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. The *Command History* panel records all operations previously typed into the Command Window and enables them to be recalled by the user. In this book we mainly use the Command Window and the built-in *Editor*, which can be launched by typing



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

plot(x,y1,'r--',x,y2,'b-')
>>
```

| Name | Value | Min | Max |
|---|---|---|---|
| x | <1x21 double> | 0 | 6.28... |
| y1 | <1x21 double> | -1 | 1 |
| y2 | <1x21 double> | -1 | 1 |

```
%-- 07.02.10 15:48 --%
   clc
   x = 0 : pi/10 : 2*pi;
   y1 = sin(x);
   y2 = cos(x);
   plot(x,y1,'r--',x,y2,'b-')
```

**Fig. 2.1** Screenshot of the MATLAB default desktop layout including the *Current Folder* (left in the figure), the *Command Window* (center), the *Workspace* (upper right) and *Command History* (lower 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.

```
edit
```

or by selecting the *Editor* from the *Desktop* menu. 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 organized in directories on the hard disk. The default search path includes only the MATLAB directory that has been created by the installer in the applications folder and the default working directory. To see which directories are in the search path or to add new directories, select *Set Path* from the *File* menu, 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 matrix or array is a scalar. Matrices with one column or row are vectors, time series or other one-dimensional data fields. An *m*-by-*n* matrix or 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. 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 matrix 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 matrix

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

Displaying the elements of `A` could be problematic for very large matrices, such as digital elevation models consisting of thousands or millions of elements. To suppress the display of a matrix 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 matrix `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 matrix 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 matrices. 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

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

which first sums the columns of the matrix 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
ans       1x4                32  double
b         1x1                 8  double
```

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 matrices and lower-case letters for vectors and scalars. We could now delete the contents of the variable `ans` by typing

```
clear ans
```

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

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

simply yields the matrix element located in the third row and second column, which is `9`. The matrix indexing therefore follows the rule (*row, col-*

*umn*). We can use this to replace single or multiple matrix elements. As an example, we type

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

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

```
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 matrix `A`. The colon operator also has several other uses in MATLAB, for instance as a shortcut for entering matrix elements such as

```
c = 0 : 10
```

which creates a row vector 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 matrices 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 vector 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 axes 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`

```
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
    75   136   -76    39
    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, right, `/`, and left, `\`, representing different transformations. 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 an array 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 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 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 data for each sample. The absurd value `-999` 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 file and assigns the matrix to a variable `geochem` specified by the filename *geochem.txt*. Typing

```
whos
```

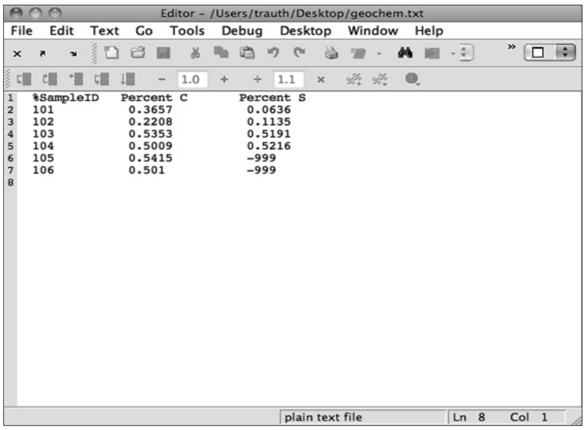
yields

```
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 com-


Editor - /Users/trauth/Desktop/geochem.txt
File Edit Text Go Tools Debug Desktop Window Help
1.0
1.1
plain text file
Ln 8 Col 1

```
%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
```

**Fig. 2.2** Screenshot of MATLAB *Editor* showing the content of the file *geochem.txt*. The first line of the text is commented by a percent sign at the beginning of the line, followed by the actual data matrix.

puter platforms running different floating-point formats. The command

```
save geochem_new.mat geochem
```

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 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.

## 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. This double precision array allows storage of the sign of a number (first bit), the exponent (bits 2 to 12) and roughly 16 significant decimal digits between approximately $10^{-308}$ and $10^{+308}$ (bits 13 to 64). As an example, typing

```
clear

rand('seed',0)
A = rand(3,4)
```

creates a 3-by-4 array of random numbers with double precision. We use the function `rand` that generates uniformly distributed pseudorandom numbers within the open interval (0,1). To obtain identical data values, we 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 =
    0.2190    0.6793    0.5194    0.0535
    0.0470    0.9347    0.8310    0.5297
    0.6789    0.3835    0.0346    0.6711
```

By default, the output is in a scaled fixed point format with 5 digits, e.g., 0.2190 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
```

yields the output

```
A =
  Columns 1 through 3
   0.218959186328090   0.679296405836612   0.519416372067955
   0.047044616214486   0.934692895940828   0.830965346112365
   0.678864716868319   0.383502077489859   0.034572110527461
  Column 4
   0.053461635044525
   0.529700193335163
   0.671149384077242
```

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 is not changed. The precision is, however, affected by converting the data type from *double* to 32-bit *single precision*. Typing

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

yields

```
B =
   0.2189592   0.6792964   0.5194164   0.0534616
   0.0470446   0.9346929   0.8309653   0.5297002
   0.6788647   0.3835021   0.0345721   0.6711494
```

Although we have switched to `format long`, only 8 digits are displayed. The command `who` 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 applications where 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 3×4 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

```
A =
   0.2190           0.6793           3.0000 + 4.0000i    0.0535
   0.0470           0.9347           0.8310              0.5297
   0.6789           0.3835           0.0346              0.6711
```

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 24-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 of 0 to 1. We then multiply the random numbers by 256 and round the results towards plus infinity using the function `ceil` to get values between 1 and 256.

```
clear

I1 = 256 * rand(100,100); I1 = ceil(I1);
I2 = 256 * rand(100,100); I2 = ceil(I2);
I3 = 256 * rand(100,100); I3 = ceil(I3);
```

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

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

Since RGB images are represented by integer values between 1 and 256 for each color, we convert the 64-bit double precision values to unsigned 8-bit integers using `uint8`.

```
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 1 and 256, whereas using `int8` to create signed 8-bit integers generates values between -128 and +127. The value of zero requires one bit and therefore there is no space 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
```

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 $10\times16=160$ bits or $160/8=20$ bytes of memory. Also 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 =
   06-Oct-2009
```

The third layer of `sample_1` contains the image created in the previous example, whereas 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
```

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*. Structures are often used in applications where organization of the data is of high importance. Cell arrays are often used when working with data that is intended for processing by index in a programming control loop.

## 2.6 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-files 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 vector `x`.

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

The first line of the *if loop* yields the dimensions of the variable `x` using the command `size`. In our example, `x` should be either a column vector with dimensions `(m,1)` or a row vector with 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` loop picks either `m` or `n` depending on whether `m==1` is false or true. Here, the double equal sign `'=='` makes element by element comparisons between the variables (or numbers) to the left and right of the equal signs and returns a matrix of the same size, made up of elements set to logical `1` where the relation is true and elements set to logical `0` where it is not. 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` loop 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 vector `x`. We then type

```
average
```

without the extension *.m* to run our script and obtain the average of the elements of the vector `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 vector `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.
```

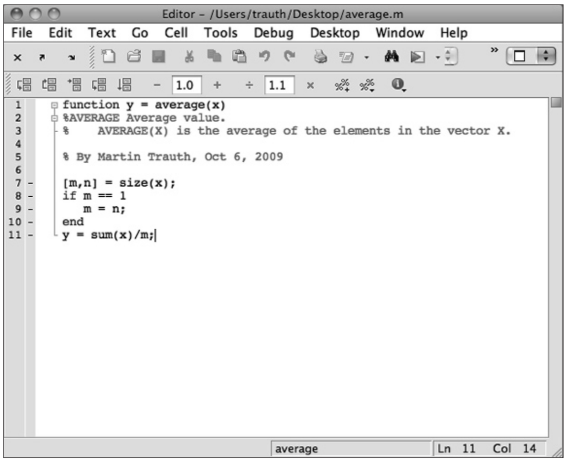
Obviously, we cannot assign a variable to the output of a script. Moreover, all variables defined and used in the script appear in the workspace; in our example these are the variables `m` and `n`. Scripts contain sequences of commands that are applied to variables in the workspace. MATLAB functions, 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 have 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 vector X.

% By Martin Trauth, Oct 6, 2009

[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


Editor – /Users/trauth/Desktop/average.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
average
Ln 11 Col 14

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

% By Martin Trauth, Oct 6, 2009

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

**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.

to suppress the display of the result in the Command Window. Next we type

```
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];
```

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.

## 2.7 Basic Visualization Tools

MATLAB provides numerous routines for displaying data as graphs. This section introduces the most important graphics functions. The graphs 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 vectors `x` and `y`, where `y` is the sine of `x`. The vector `x` contains values between 0 and $2\pi$ with $\pi/10$

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 vectors with 21 elements each, i. e., two 1-by-21 arrays. Since the two vectors `x` and `y` have the same length, we can use `plot` to produce a linear 2d graph `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,'r--',x,y2,'b-')
```

creates a dashed red line displaying the sine of `x` and a solid blue line representing the cosine of this vector (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.

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

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 second graph as a bar plot.

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

This command plots `y1` versus `x` as dashed line, whereas `y2` versus `x` is shown as a group of blue vertical bars. Alternatively, we can plot both graphs in the same Figure Window but in different plots using `subplot`.

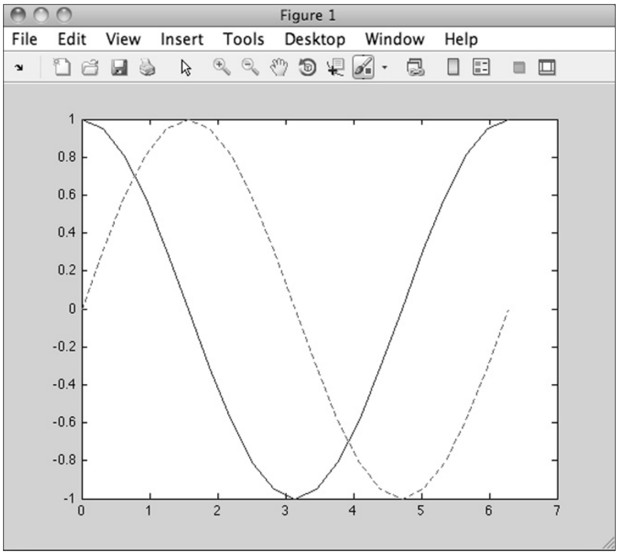


**Fig. 2.4** Screenshot of the MATLAB *Figure Window* showing two curves in different colors and line types. The Figure Window allows editing of all elements of the graph after selecting *Edit Plot* from the *Tools* menu. Double clicking on the graphics elements opens an options window for modifying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

The syntax `subplot(m,n,p)` divides the Figure Window into an *m*-by-*n* matrix of display regions and makes the `p`th display region active.

```
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, whereas the bar plot appears in the lower half of the Figure Window. 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 graphs 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 $\pi$, 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 in the following chapters of this book.

## 2.8 Generating M-Files to Regenerate Graphs

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 has 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 properties of the graph such as axes, lines, patches and text objects.

The *Generate M-Files* option enables us to automatically generate the MATLAB code of a figure to recreate a graph with different data. We use a simple plot to illustrate the use of the Property Editor and the Generate M-Files 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 3rd column of the color chart. Moving the mouse over this square displays the RGB color code [0.7 0.78 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 6-point square marker. We can close the Property Editor by clicking on the `X` in the upper right corner of the Property Editor panel below the graph. Finally, 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 M-File* from the *File* menu of the Figure Window. The generated code displays in the MATLAB Editor.

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

%  Auto-generated by MATLAB on 06-Oct-2009 17:37:20

% Create figure
figure1 = figure('XVisual',...
  '0x24 (TrueColor, depth 24, RGB mask 0xff0000 0xff00 0x00ff)',...
  'Color',[0.6784 0.9216 1]);
```

```
% Create axes
axes('Parent',figure1);
box('on');
hold('all');

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

We can then rename the function *createfigure* to *mygraph* and save the file as *mygraph.m* using *Save As* from the *Editor File* menu.

```
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 `[0.702 0.7804 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 M-File function of MATLAB can therefore be used to create templates for graphs that can be used to generate plots of multiple data sets using the same layout.

Even though the MATLAB provides enormous editing facilities and the Generate M-File function even allows the generation of complex templates for graphs, 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 graphs are exported by selecting the command *Save as* from the *File* menu or by using the command `print`. This function exports the graph either as raster image (e.g., JPEG or GIF) or 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 graphs.

## 2.9 Publishing M-Files

A relatively new 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 graphs 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. Before we start developing the MATLAB code, we activate *Enable Cell Mode* in the *Cell* menu of the Editor. Whereas single percent signs `%` are known (from Section 2.6) to initiate comments in MATLAB, we now use double percent signs `%%` that indicate the start of new cells in the Editor. The *Cell Mode* is a feature in MATLAB that enables you to evaluate blocks of commands by using the buttons *Evaluate Cell*, *Evaluate Cell and Advance* and *Evaluate Entire File* on the *Editor Cell Mode* toolbar. The *Save and Publish* button, which was situated next to the Cell Mode buttons in earlier versions of MATLAB, is now included in the Editor Toolbar emphasizing the importance and popularity of this feature.

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 cells of MATLAB code and comments separated by the double percent signs `%%`.

```
%% Example for Publishing M-Files
% This M-file illustrates the use of the publishing feature
% of MATLAB.
% By Martin Trauth, Feb 8, 2009.

%% 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 myproject.m* button in the Editor Toolbar. 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 dark 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 graphs 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 graphs can be saved as a PDF-file and shared with students or colleagues.

## Recommended Reading

Davis TA, Sigmon K (2005) The MATLAB Primer, Seventh Edition. Chapman & Hall/CRC, London
Etter DM, Kuncicky DC, Moore H (2004) Introduction to MATLAB 7. Prentice Hall, New Jersey
Gilat A (2007) MATLAB: An Introduction with Applications. John Wiley & Sons, New York
Hanselman DC, Littlefield BL (2004) Mastering MATLAB 7. Prentice Hall, New Jersey
Palm WJ (2004) Introduction to MATLAB 7 for Engineers. McGraw-Hill, New York
The MathWorks (2010) MATLAB 7 Getting Started Guide. The MathWorks Inc., Natick, MA

# 3 Univariate Statistics

## 3.1 Introduction

The statistical properties of a single parameter are investigated by means of univariate analysis. Such a parameter could be, for example, the organic carbon content of a sedimentary unit, the thickness of a sandstone layer, the age of sanidine crystals in a volcanic ash or the volume of landslides. 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.

We first need to describe the characteristics of the sample using statistical parameters, and compute an *empirical distribution* (*descriptive statistics*) (Sections 3.2 and 3.3). A brief introduction to the most important statistical parameters such as the *measures of central tendency and dispersion* is provided, followed by MATLAB examples. Next, we 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 to explore these distributions. We then try to draw conclusions from the sample that can be applied to the larger phenomenon of interest (*hypothesis testing*). The relevant sections below (Sections 3.6 to 3.8) introduce the three most important statistical tests for applications in earth sciences: the t-test to compare the means of two data sets, the F-test to compare variances and the $\chi^2$-test to compare distributions. The final section in this chapter introduces methods used to fit distributions to our own data sets (Section 3.9).

## 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 popula-

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*, 3rd ed.,
DOI 10.1007/978-3-642-12762-5_3, © Springer-Verlag Berlin Heidelberg 2010

tion of interest, can be written as a vector $x$

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

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

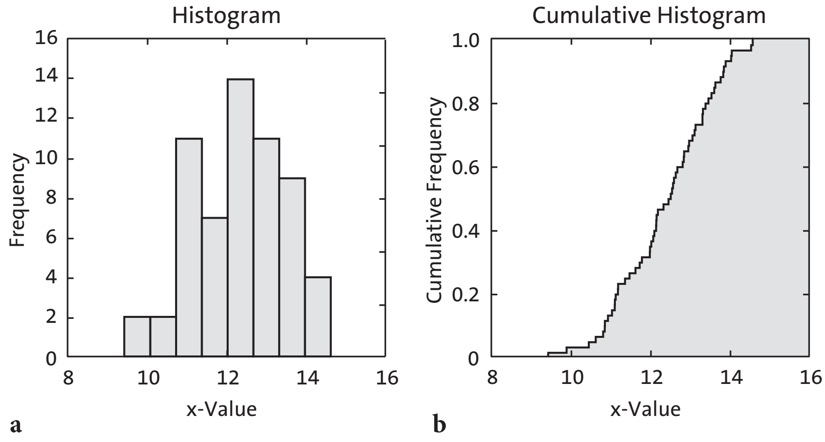


**Fig. 3.1** Graphical representation of an empirical frequency distribution. **a** In a histogram, the frequencies are organized in classes and plotted as a bar plot. **b** The *cumulative histogram* of a frequency distribution displays the totals of all classes lower than and equal to a certain value.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

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 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:

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

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 $\mu$ 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

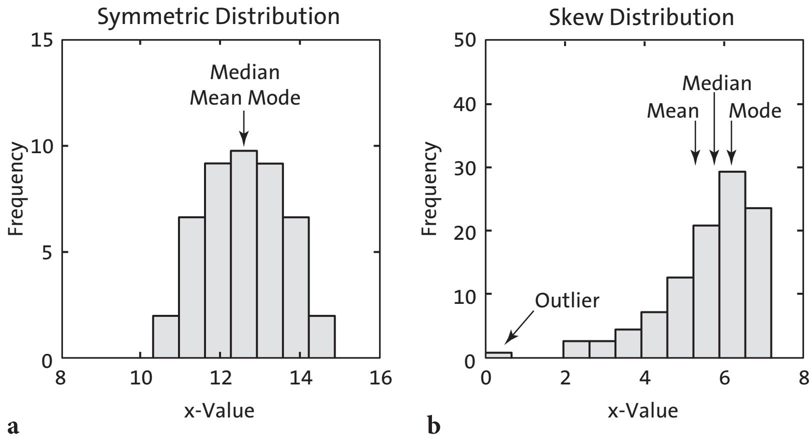


**Fig. 3.2** Measures of *central tendency*. **a** In an unimodal symmetric distribution, the mean, the median and the mode are identical. **b** In a skewed distribution, the median lies between the mean and the mode. The mean is highly sensitive to outliers, whereas the median and the mode are little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

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 larger than the median and 50 % are smaller. The median of a data set sorted in ascending order is defined as

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

if $N$ is odd and

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

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*).

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 1.10 after one year, 1.375 after two years, or 2.20 after three years. The average growth of the population is therefore calculated by the geometric mean:

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

The average growth of these values is 1.4929 suggesting an approximate per annum growth in the population of 49 %. The arithmetic mean would result in an erroneous value of 1.5583 or approximately 56 % 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*

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

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*.

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

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 $\sigma$. 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

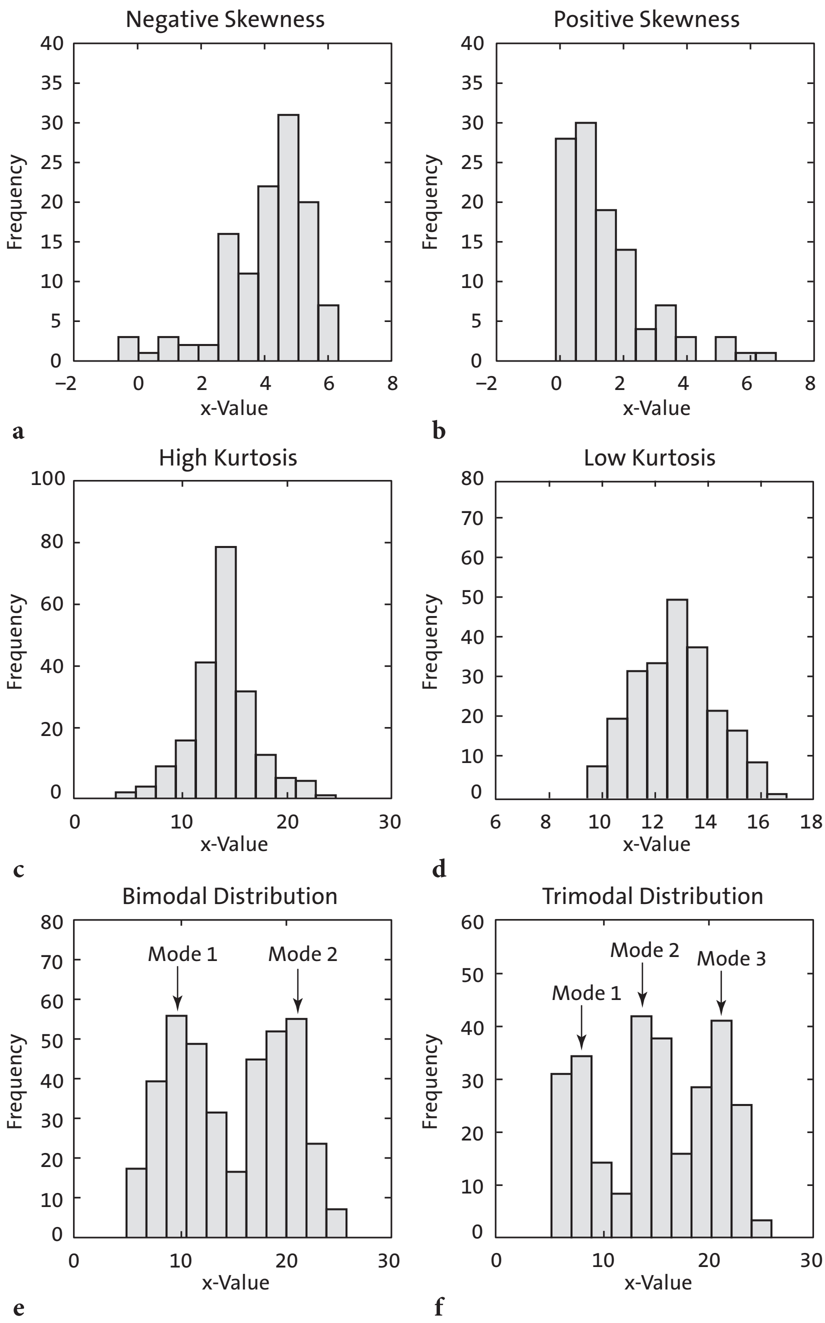


**Fig. 3.3** *Dispersion* and *shape* of a distribution. **a-b** Unimodal distributions showing a negative or positive skew. **c-d** Distributions showing a high or low kurtosis. **e-f** Bimodal and trimodal distributions showing two or three modes.

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 $\sigma$.

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

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

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. 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.

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

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

Having defined the most important parameters used to describe an empirical distribution, the measures of central tendency and dispersion are now illustrated by examples.

## 3.3 Example of Empirical Distributions

As an example, we can analyze the data contained in the file *organicmatter_one.txt*. This file contains the organic matter content 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 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.

```
hist(corg)
```

By default, the function `hist` divides the range of the data into ten equal intervals, bins or classes, counts the number of observations within each interval and displays the frequency distribution as a bar plot. The midpoints of the default intervals `v` and the number of observations per interval `n` can be accessed using

```
[n,v] = hist(corg)

n =
  Columns 1 through 8
     2     1     5     8     5    10    10     9
  Columns 9 through 10
     8     2

v =
  Columns 1 through 5
    9.6740   10.1885   10.7029   11.2174   11.7319
  Columns 6 through 10
   12.2463   12.7608   13.2753   13.7898   14.3042
```

The number of classes should not be lower than six or higher than fifteen for practical purposes. In practice, the square root of the number of observations, rounded to the nearest integer, is often used as the number of classes. In our example, we use eight classes instead of the default ten classes.

```
hist(corg,8)
```

We can even define the midpoint values of the histogram classes. Here, it is recommended to choose interval endpoints that avoid data points falling between two intervals. We can use the minimum, the maximum and the range of the data to define the midpoints of the classes. We can then add half of the interval width to the lowest value in `corg` to calculate the midpoint of the lowest class. The midpoint of the highest class is the highest value in `corg` reduced by half of the interval width.

```
min(corg) + 0.5*range(corg)/8

ans =
   9.7383

max(corg) - 0.5*range(corg)/8

ans =
   14.2399

range(corg)/8

ans =
    0.6431
```

We can now round these values and define

```
v = 9.75 : 0.65 : 14.30;
```

as midpoints of the histogram intervals. This method for defining the midpoints is equivalent to the one used by the function `hist` if `v` is not specified by the user. The commands for displaying the histogram and calculating the frequency distribution are

```
hist(corg,v)
```

```
n = hist(corg,v)

n =
     2     2    11     7    14    11     9     4
```

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)

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 =
    12.3500
```

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)
```

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, standard deviation and variance. We have already used the range to define the midpoints of the classes. The variance is the average of the squared deviations 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 lower than three.

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

```
hist(sodium,11)

[n,v] = hist(sodium,11)

n =
  Columns 1 through 10
     3     3     5     2     6     8    15    14    22    29
  Column 11
    13

v =
  Columns 1 through 6
    2.6738    3.1034    3.5331    3.9628    4.3924    4.8221
  Columns 7 through 11
    5.2518    5.6814    6.1111    6.5407    6.9704
```

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)

ans =
    5.9741

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

ans =
    6.5407
```

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, which is better suited for 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 compute the number of observations `n` with respect to the classes `v`.

```
[n,v] = hist(sodium,7);

mean(sodium)

ans =
   16.6379

median(sodium)

ans =
   16.9739

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

ans =

   17.7569
```

The mean of the data is 16.6379, the median is 16.9739 and the mode is 17.7569. We now introduce a single, very low value of 1.5 wt % in addition to the 50 measurements contained in the original data set.

```
sodium(51,1) = 1.5;

hist(sodium,7)
```

The histogram of this data set illustrates the distortion of 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 =
   16.3411

median(sodium)

ans =
   16.9722

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

```
ans =
   17.7569
```

The most significant change observed is in the mean (16.3411), 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

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

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.

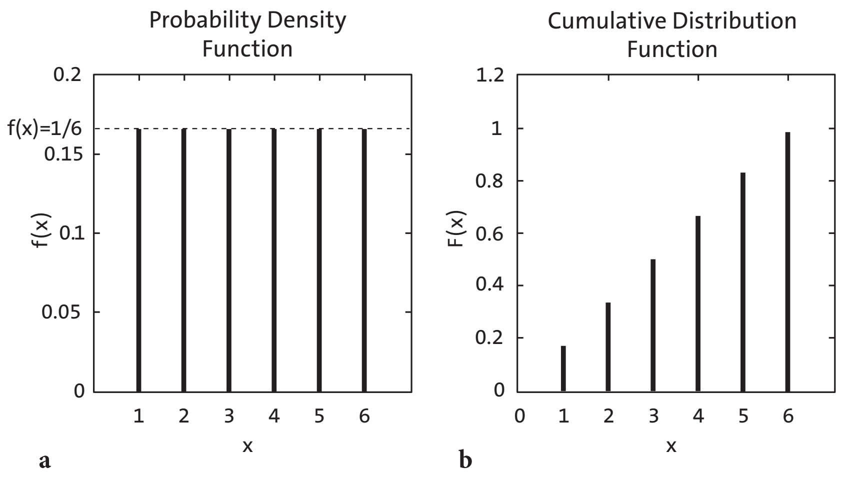


**Fig. 3.4** **a** Probability density function $f(x)$ and **b** cumulative distribution function $F(x)$ of a uniform distribution with $N$=6. The 6 discrete values of the variable $x$ have the same probability of 1/6.

## 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

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

The cumulative distribution function is

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

where

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

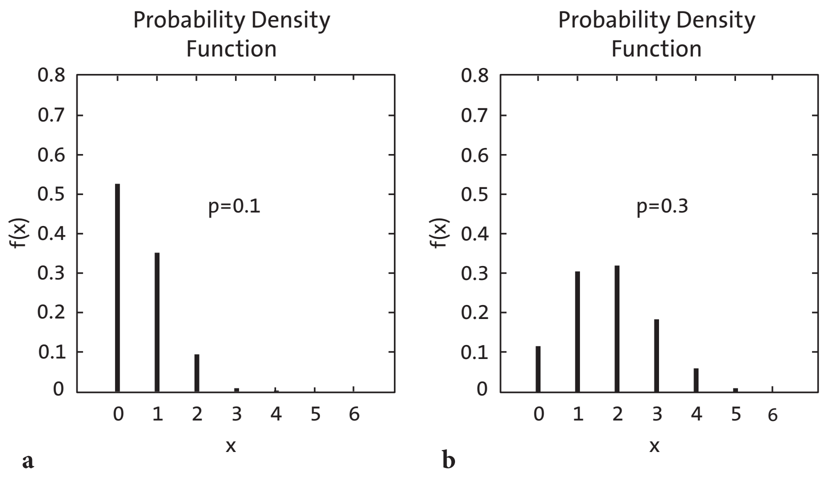


**Fig. 3.5** Probability density function $f(x)$ of a binomial distribution, which gives the probability $p$ of $x$ successes out of $N$=6 trials, with probability **a** $p$=0.1 and **b** $p$=0.3 of success in any given trial.

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.

## Poisson Distribution

When the number of trials is $N\rightarrow\infty$ and the success probability is $p\rightarrow 0$, 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

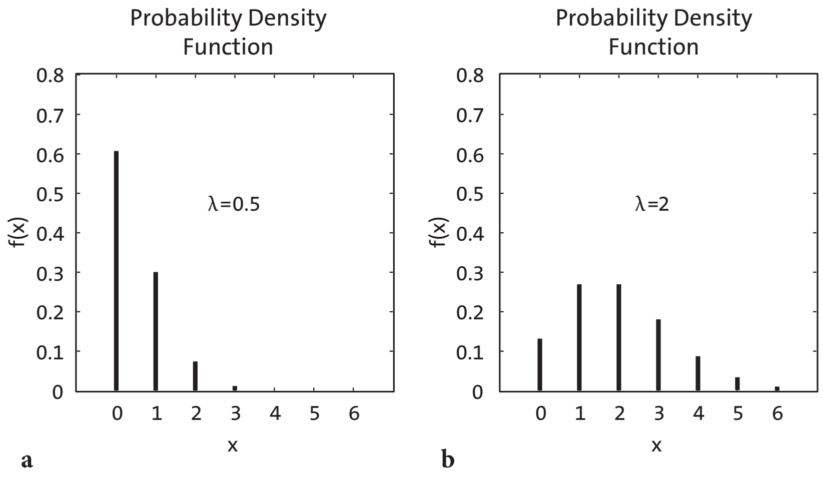


**Fig. 3.6** Probability density function $f(x)$ of a Poisson distribution with different values for $\lambda$. **a** $\lambda$=0.5 and **b** $\lambda$=2.

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

and the cumulative distribution function is

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

The single parameter $\lambda$ 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 $\mu$ and standard deviation $\sigma$ (Fig. 3.7). The probability density function of a normal distribution is

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

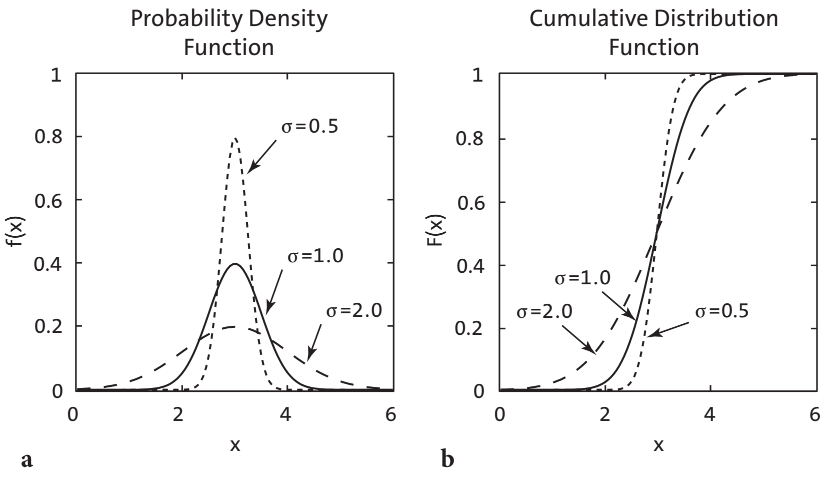


**Fig. 3.7** **a** Probability density function $f(x)$ and **b** cumulative distribution function $F(x)$ of a Gaussian or normal distribution with a mean $\mu=3$ and various values for standard deviation $\sigma$.

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*.

## 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 $\mu$ and the standard deviation $\sigma$. 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.

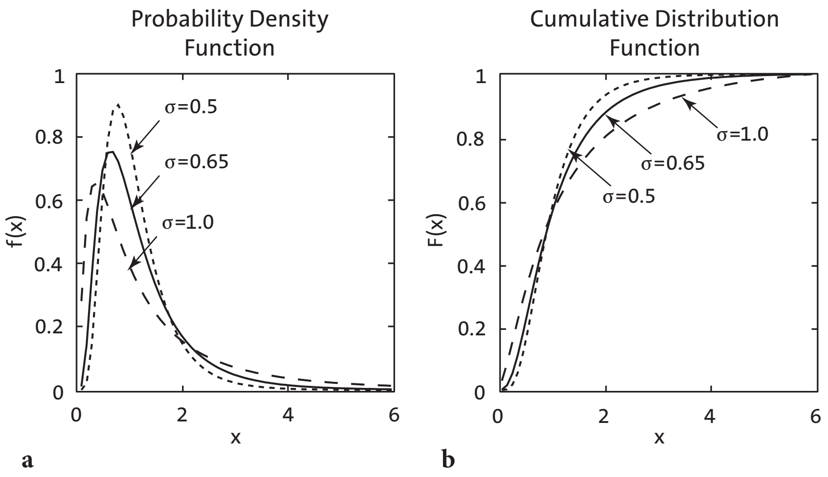


**Fig. 3.8** **a** Probability density function *f*(*x*) and **b** cumulative distribution function *F*(*x*) of a logarithmic normal distribution with a mean μ=0 and with various values for *σ*.

## 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

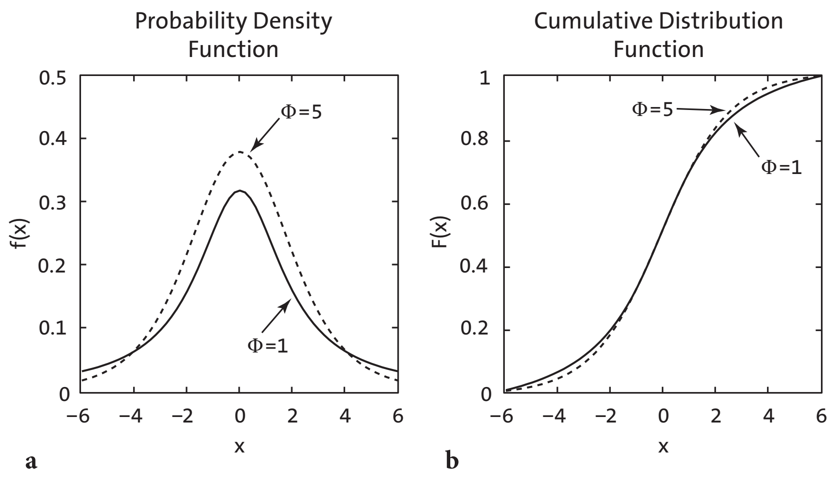


**Fig. 3.9** **a** Probability density function $f(x)$ and **b** cumulative distribution function $F(x)$ of a Student's t distribution with two different values for $\Phi$.

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

if $x>0$. The single parameter $\Phi$ of the t distribution is the number of degrees of freedom. In the analysis of univariate data, this distribution has $n-1$ degrees of freedom, where $n$ is the sample size. As $\Phi \to \infty$, 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 3.6).

## 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 3.7). The F distribution has a relatively complex probability density function (Fig. 3.10):

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

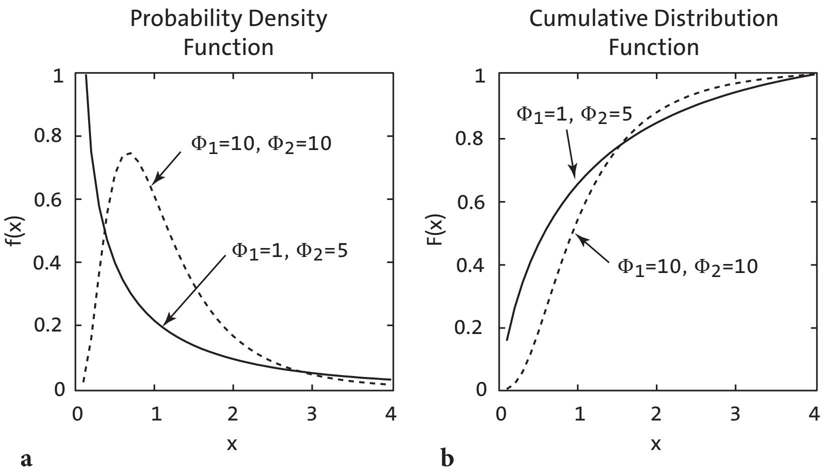


**Fig. 3.10** **a** Probability density function $f(x)$ and **b** cumulative distribution function $F(x)$ of a Fisher's F distribution with different values for $\Phi_1$ and $\Phi_2$.

where $x>0$ and $\Gamma$ is again the Gamma function. The two parameters $\Phi_1$ and $\Phi_2$ are the numbers of degrees of freedom.

### $\chi^2$ or Chi-Squared Distribution

The $\chi^2$ 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 $\chi^2$-test (Section 3.8). The probability density function of the $\chi^2$ 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 $f(x)=0$; $\Gamma$ is again the Gamma function. Once again, $\Phi$ is the number of degrees of freedom (Fig. 3.11).

## 3.5 Example of Theoretical Distributions

The function `randtool` is a tool to simulate discrete data with statistics similar to our data. This function creates a histogram of *random numbers* from the distributions in the Statistics Toolbox. The random numbers that have

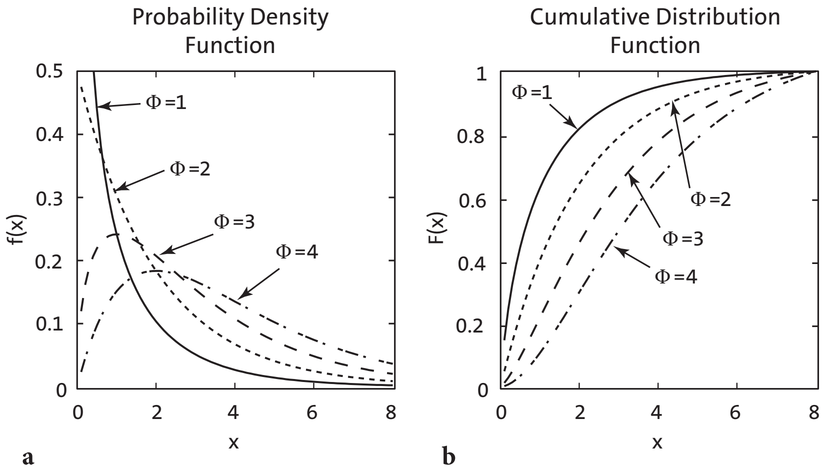


**Fig. 3.11** **a** Probability density function $f(x)$ and **b** cumulative distribution function $F(x)$ of a $\chi^2$ distribution with different values for $\Phi$.

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

```
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 $\mu$ (the mean of a population) and *Sigma* for $\sigma$ (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 samples represents a rough estimate of the 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 cumulative distribution function (CDF) to compute a theoretical distribution. 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

x = 9 : 0.1 : 15;
pdf = normpdf(x,12.3448,1.1660);
cdf = normcdf(x,12.3448,1.1660);
plot(x,pdf,x,cdf)
```

MATLAB also provides a GUI-based function for generating PDFs and CDFs with specific statistics, which is called `disttool`.

```
disttool
```

We choose `PDF` as 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 the non-GUI functions for calculating probability density functions and cumulative distribution functions, such as `normpdf` and `normcdf`.

The remaining sections in this chapter are concerned with methods of drawing conclusions from the sample that can then applied to the larger phenomenon of interest (*hypothesis testing*). The three most important statistical tests for earth science applications are introduced, these being 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 $\chi^2$-test to compare distributions. The last section introduces methods that can be used to fit distributions to our data sets.

## 3.6 The t-Test

The Student's t-test by William Gossett 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 sets of 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 variance for the two sets of measurements should be similar. The appropriate test statistic for the difference between the two means is then

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

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 $\alpha$. The significance level $\alpha$ of a test is the maximum probability of accidentally rejecting a true null hypothesis. Note that we cannot prove the null hypothesis, in other words *not guilty* is not the same as *innocent* (Fig. 3.12). The 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 of the data to the far left and far right of a probability density function, as, for example, in the t distribution. 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 by the 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 is 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 an example data set of 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, while the standard deviations are 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

```
[n1,x1] = hist(corg1);
[n2,x2] = hist(corg2);

h1 = bar(x1,n1);
hold on
h2 = bar(x2,n2);
hold off
```

```
set(h1,'FaceColor','none','EdgeColor','r')
set(h2,'FaceColor','none','EdgeColor','b')
```

Here we use the command `set` to change graphic objects of the bar plots `h1` and `h2`, such as the face and edge colors of the bars. We then compute the frequency distributions of both samples, together with the sample sizes, the means and the standard deviations.

```
[n1,x1] = hist(corg1);
[n2,x2] = hist(corg2);

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` of 1.7995 with the critical `tcrit`. 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 the 5 % significance level. This is a two-sample t-test, i. e., the means are not equal. Computing the two-tailed critical `tcrit` by entering 1–0.05/2 yields the upper (positive) `tcrit` 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` calculated from the data is smaller than the critical `tcrit`, 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 suggests that the chances of observing more extreme *t* values than the values in this example, from similar experiments, would be 4681 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, 24.3 and 25.5, while the standard deviations are again 1.3 and 1.5, respectively.

```
clear

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

This file again contains two data sets `corg1` and `corg2`. As before, we first compute the frequency distributions of both samples, together with the sample sizes, the means and the standard deviations.

```
[n1,x1] = hist(corg1);
[n2,x2] = hist(corg2);

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` of 4.7364 with the critical `tcrit`. 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` by entering 1 – 0.05/2 yields the upper (positive) `tcrit` 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` calculated from the data is now larger than the critical `tcrit`, 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

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 (and hypothesized) difference between the means of 25.5 – 24.3 = 1.2.

## 3.7 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 in turn will depend on the number of degrees of freedom $\Phi_a = n_a - 1$ and $\Phi_b = n_b - 1$, respectively, and the significance level $\alpha$. The one-sample F-test, in contrast, virtually performs a $\chi^2$-test of the hypothesis that the data come from a normal distribution with a specific variance (see Section 3.8). 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

```
s1 = std(corg1)

s2 = std(corg2)
```

which yields

```
s1 =
    1.2550

s2 =
    1.2097
```

The F-distribution has two parameters, `df1` and `df2`, which are the numbers of observations of both 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` with the critical `F`. 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 therefore must divide the *p*-value of 0.05 by two. Typing

```
Fcalc = slarger^2 / ssmaller^2

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

yields

```
Fcalc =
    1.0762

Fcrit =
    1.6741
```

Since the `F` calculated from the data is smaller than the critical `F`, we cannot reject the null hypothesis without another cause. We conclude therefore that the variances are identical at 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 $\chi^2$-test of the hypothesis that the data in the vector x come from a normal distribution with a variance defined by variance. The $\chi^2$-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 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.

```
clear

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

We again compare the calculated Fcalc with the critical Fcrit 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` calculated from the data is now larger than the critical `Fcrit`, 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    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 (and hypothesized) ratio `var(corg1)/var(corg2)` of $1.8799^2/1.2939^2=1.0762$.

## 3.8 The $\chi^2$-Test

The $\chi^2$-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 $\chi^2$-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. 3.12). The null hypothesis can be rejected if the measured $\chi^2$ is higher than the critical $\chi^2$, 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, $Z$=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.

```
clear

corg = load('organicmatter_one.txt');

v = 9.40 : 0.74 : 14.58;
n_obs = hist(corg,v);
```

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')
```

Visual inspection of these plots reveals that they are similar. It is, however, advisable to use a more quantitative approach. The $\chi^2$-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 =
    6.0383
```

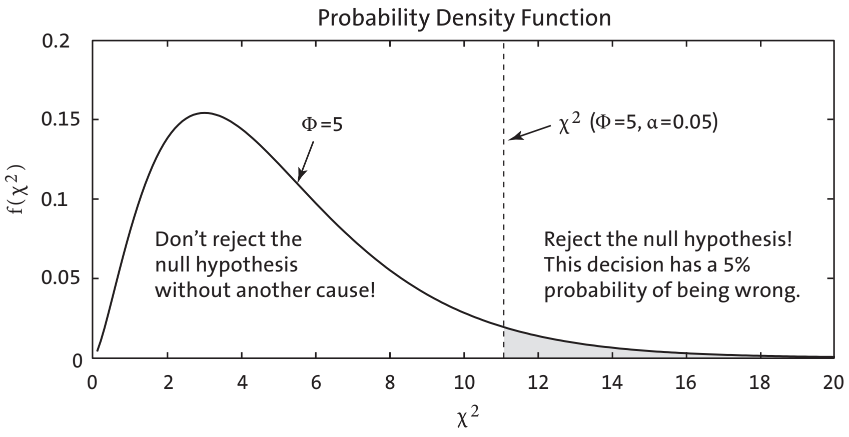


**Fig. 3.12** Principles of a $\chi^2$-test. The alternative hypothesis that the two distributions are different can be rejected if the measured $\chi^2$ is lower than the critical $\chi^2$. $\chi^2$ depends on $\Phi = K - Z$, where $K$ is the number of classes and $Z$ is the number of parameters describing 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 $\chi^2 = 6.0383$ is below the critical $\chi^2$, we cannot reject the null hypothesis. In our example, we can conclude that the sample distribution is not significantly different from a Gaussian distribution.

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

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

chi2crit =
    11.0705
```

Since the critical `chi2crit` of 11.0705 is well above the measured `chi2calc` of 5.4256, 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,stats] = chi2gof(corg)
```

yields

```
h =
     0

p =
    0.6244

stats =
 chi2stat: 2.6136
       df: 4
    edges: [9.417 10.960 11.475 11.990 12.504 13.018 13.533 14.5615]
        O: [8 8 5 10 10 9 10]
        E: [7.0506 6.6141 9.1449 10.4399 9.8406 7.6587 9.2511]
```

The function automatically defines seven classes instead of the eight classes that we used in our experiment. 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 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 $\chi^2$, which is 2.6136 and differs from our result of 5.2456 due to the different number of classes. The array `stats` also contains the number of degrees of freedom $\Phi=7-(2+1)=4$, the eight edges of the seven classes automatically defined by the function `chi2gof`, and the observed and expected frequencies of the distribution.

## 3.9 Distribution Fitting

In the previous section we computed the mean and standard deviation of our sample and designed a normal distribution based on these two parameters. We then used the $\chi^2$-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 the data is `normfit`. To demonstrate the use of this function we first generate 100 synthetic, Gaussian-distributed values, with a mean of 6.4 and a standard deviation of 1.4.

```
clear

randn('seed',0)
data = 6.4 + 1.4*randn(100,1);
```

We then define the midpoints `v` of nine histogram intervals, display the result and calculate the frequency distribution `n`.

```
v = 2 : 10;
hist(data,v)
n = hist(data,v);
```

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 =
    6.5018

sigmahat =
    1.3350
```

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, but 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 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), or multimodal chemical ages of 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. In our examples, the methods described in this section help to determine the episodes of deformation in the 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

randn('seed',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 `hist`.

```
v = 0 : 30;
hist(data,v)
```

```
n = hist(data,v);
```

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 of the distribution. The maximization step computes the parameters which maximize the expected logarithmic likelihood computed in the expectation step. The function `mgdistribution.fit` constructs 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: 0.509171
Mean:      6.5478

Component 2:
Mixing proportion: 0.490829
Mean:     13.4277
```

Thus we obtain the means and relative mixing proportion of both distributions. In our example, both normal distributions with means of 6.5492 and 13.4300, respectively, contribute ca. 50 % (0.51 and 0.49, 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.

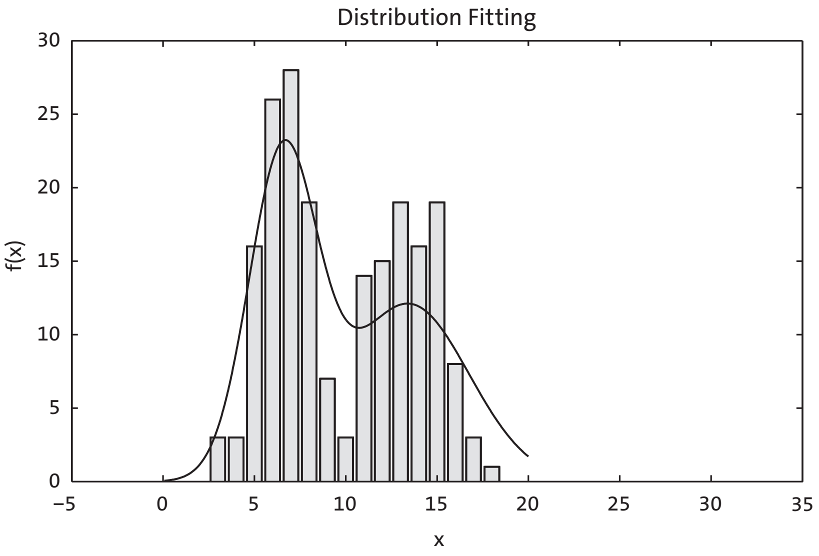


**Fig. 3.13** Fitting Gaussian mixture distributions. As a synthetic example of Gaussian mixture distributions we generate two sets of 100 random numbers with means of 6.4 and 13.3, respectively, and standard deviations of 1.4 and 1.8, respectively. The *Expectation-Maximization (EM) algorithm* is used to fit a Gaussian mixture distribution (solid line) with two components to the data (bars).

```
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));
```

Finally, we can plot the probability density function `y` upon 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. 3.13).

## Recommended Reading

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

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. Phil. Mag. 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

Sachs L, Hedderich J (2009) Angewandte Statistik: Methodensammlung mit R, 13., aktualisierte und erweiterte Auflage. Springer, Berlin Heidelberg New York

Snedecor GW, Cochran WG (1989) Statistical Methods, Eighth Edition. Blackwell Publishers, Oxford

Spiegel MR (2008) Schaum's Outline of Probability and Statistics, 3nd Revised Edition. Schaum's Outlines, McGraw-Hill Professional, New York

Student (1908) On the Probable Error of the Mean. Biometrika 6:1–25

Taylor JR (1996) An Introduction to Error Analysis – The Study of Uncertainties in Physical Measurements, Second Edition. University Science Books, Sausalito, California

The Mathworks (2010) Statistics Toolbox – User s Guide. The MathWorks, Natick, MA

The Mathworks (2010) MATLAB 7. Object-Oriented Programming. The MathWorks, Natick, MA

# 4 Bivariate Statistics

## 4.1 Introduction

Bivariate analysis aims to understand the relationship between two variables $x$ and $y$. Examples are the length and the width of a fossil, the sodium and potassium content of volcanic glass or 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 regression equation can then be used to calculate the second parameter.

This chapter first introduces Pearson's correlation coefficient (Section 4.2), and then explains the widely-used methods of linear and curvilinear 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 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.

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*, 3rd ed.,
DOI 10.1007/978-3-642-12762-5_4, © Springer-Verlag Berlin Heidelberg 2010

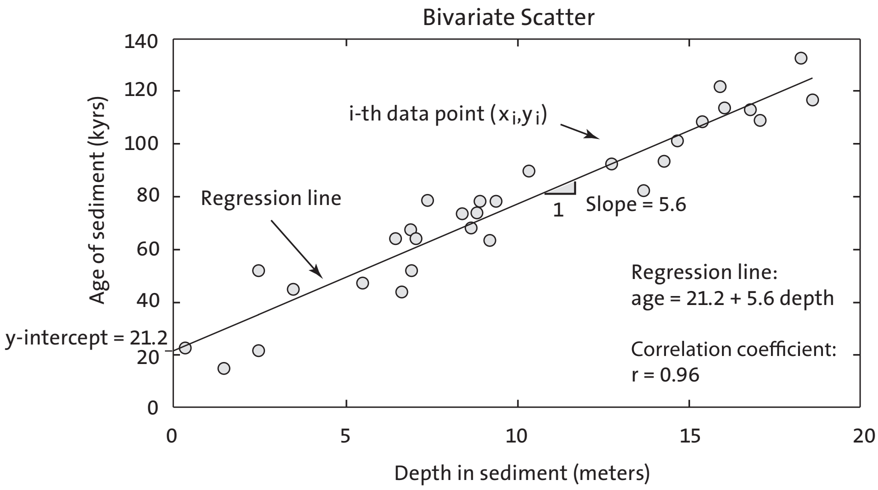


**Fig. 4.1** Display of a bivariate data set. The thirty data points represent the *age* of a sediment (in kiloyears before present) at a certain *depth* (in meters) below the sediment-water interface. The combined distribution of the two variables suggests a linear relationship between *age* and *depth*, i.e., the rate of increase in the sediment age with depth is constant. 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+5.6 *depth*, indicating an increase in sediment age of 5.6 kyrs per meter of sediment depth (the slope of the regression line). The inverse of the slope is the sedimentation rate of ca. 0.2 meters/kyr. Furthermore, the equation defines an age for the sediment surface of 21.2 kyrs (the intercept of the regression line with the $y$-axis). The deviation of the surface age from zero can be attributed either to the statistical uncertainty of regression or to a natural process such as erosion or bioturbation. The assessment of the statistical uncertainty of regression is discussed in this chapter, but a careful evaluation of the possible effects of the various natural processes at the sediment-water interface will be required.

## 4.2 Pearson's Correlation Coefficient

*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.

The most popular correlation coefficient is *Pearson's linear product-moment correlation coefficient* $\rho$ (Fig. 4.2). We estimate the population's correlation coefficient $\rho$ from the sample data, i.e., we compute the sample correlation coefficient $r$, which is defined as

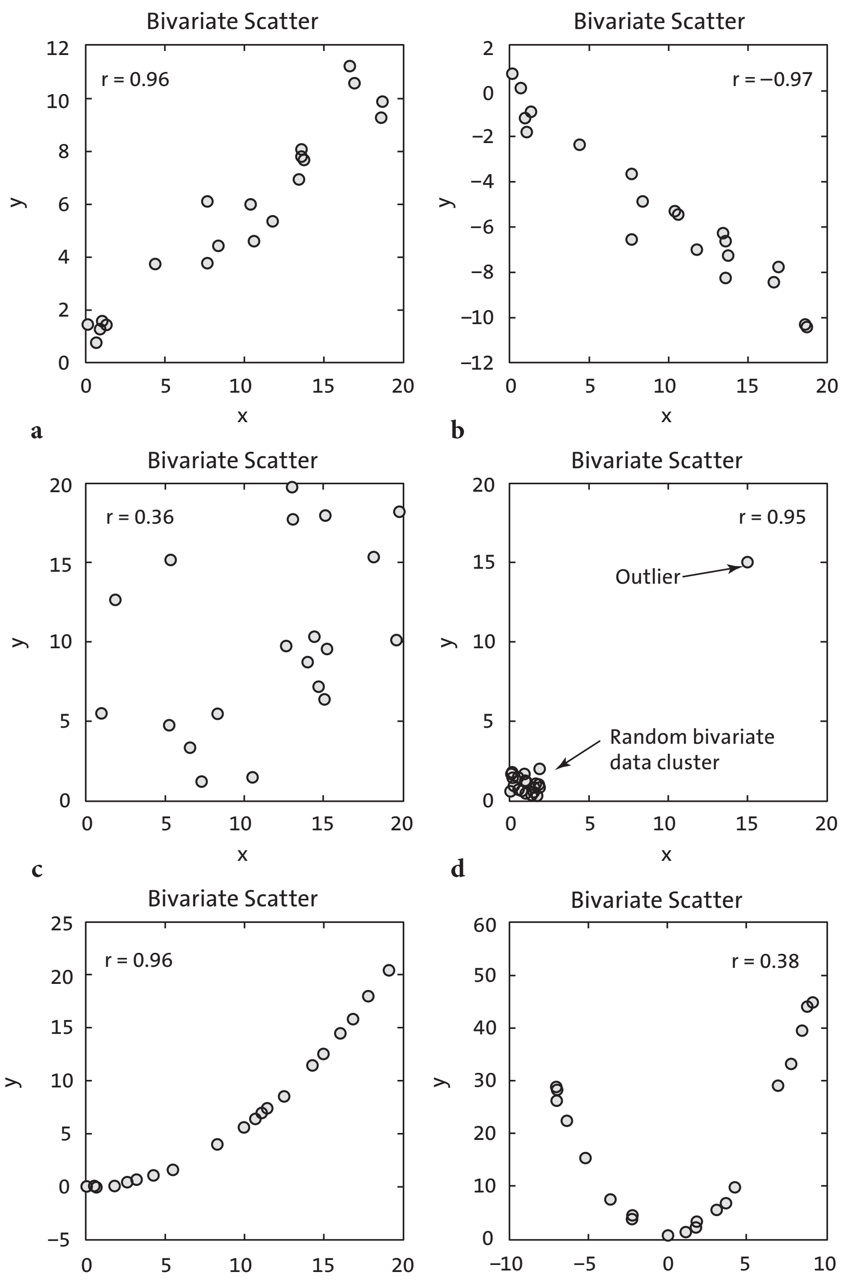


**Fig. 4.2** Pearson's correlation coefficent *r* for various sample data sets. **a–b** Positive and negative linear correlation, **c** random scatter with no linear correlation, **d** an outlier causing a misleading value of *r*, **e** curvilinear relationship causing a high *r* since the curve is close to a straight line, **f** curvilinear relationship clearly not described by *r*.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y}$$

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*

$$cov_{xy} = \frac{1}{(n-1)}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

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 dimension of the data. Dividing the covariance by the univariate standard deviations removes this effect 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 higher than the critical $t$ ($n$–2 degrees of freedom, $\alpha=0.05$). This test, however, is only valid if both variables are Gaussian distributed with respect to both variables.

Pearson's correlation coefficient is very sensitive to various disturbances in the bivariate data set. 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 $\rho$. 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.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 zero mean and a standard deviation of 10 has been added to the *age* data.

```
clear

rand('seed',40), randn('seed',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 dependency between the 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 =
    1.0000    0.9567
    0.9567    1.0000
```

The function `corrcoef` calculates a matrix of correlation coefficients for all possible combinations of the two variables `age` and `meters`. The value of $r = 0.9567$ 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 zero mean and a standard deviation one. To obtain identical data values, we reset the random number generator by using the integer 5 as seed.

```
clear

randn('seed',5);
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    0.1021
    0.1021    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    0.4641
    0.4641    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)
```

```
ans =
    1.0000    0.7636
    0.7636    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    0.9275
    0.9275    1.0000
```

The bivariate data set still does not provide much evidence for a strong interdependency between the variables. As we have seen, however, the combination of the random bivariate data with a single outlier results in a dramatic increase in the correlation coefficient. Although outliers are easy to identify in a bivariate scatter, erroneous values can easily be overlooked in large multivariate data sets.

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.

```
[r,p] = corrcoef(x,y)

r =
    1.0000    0.9275
    0.9275    1.0000

p =
    1.0000    0.0000
    0.0000    1.0000
```

In our example, the $p$-value is 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, $\alpha=0.05$).

```
tcalc = r(2,1) * ((length(x)-2)/(1-r(2,1)^2))^0.5
tcrit = tinv(0.95,length(x)-2)

tcalc =
   13.3594

tcrit =
    1.6991
```

This result indeed indicates that we can reject the null hypothesis and the correlation coefficient is significant. 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)`.

```
rhos1000 = bootstrp(1000,'corrcoef',x,y);
```

This command first resamples the data a thousand times, calculates the correlation coefficient for each new subsample and stores the result in the variable `rhos1000`. Since `corrcoef` delivers a $2\times2$ matrix as mentioned above, `rhos1000` has the dimension $1000\times4$, i.e., 1000 values for each element of the $2\times2$ 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.

```
hist(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 no strong dependence 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.

```
clear

agedepth = load('agedepth_1.txt');

meters = agedepth(:,1);
age = agedepth(:,2);

corrcoef(meters,age)

ans =
    1.0000    0.9567
    0.9567    1.0000

rhos1000 = bootstrp(1000,'corrcoef',meters,age);

hist(rhos1000(:,2),30)
```

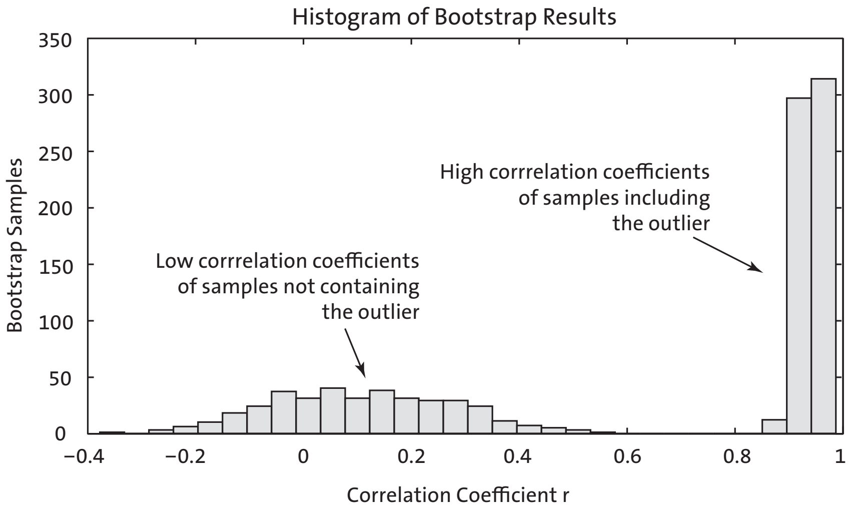


**Fig. 4.3** Bootstrap result for Pearson's correlation coefficient $r$ from 1000 subsamples. The histogram shows a roughly normally-distributed cluster of correlation coefficients at around $r=0.1$ suggesting that these subsamples do not include the outlier. The strong peak close to $r=1$, however, suggests that an outlier with high values of the two variables $x$ and $y$ is present in the corresponding subsamples.

Most of the values for `rhos1000` fall within the interval between 0.92 and 0.98. Since the correlation coefficients for the resampled data sets have an obvious Gaussian distribution, we can use their mean as a good estimate for the true correlation coefficient.

```
mean(rhos1000(:,2))

ans =
    0.9562
```

This value is similar to our first result of $r=0.9567$, 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 and Prediction

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 in the $y$-value (Fig. 4.4). This means that $x$ is then the independent, regressor, or predictor variable. The values of $x$ are defined by the experimenter and are often regarded as being free of errors. An example is the location $x$ within a sediment core of a clay sample 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 $\Delta y$ 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

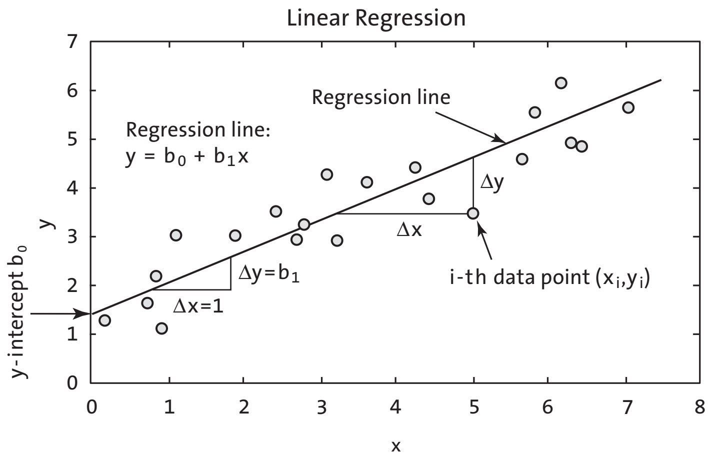


**Fig. 4.4** Linear regression. Whereas classical regression minimizes the *Δy* deviations, reduced major axis regression minimizes the triangular area 0.5*(*ΔxΔy*) between the data points and the regression line, where *Δx* and *Δy* are the distances between the predicted and the true *x* and *y* values. 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.

*Δy* deviations to be minimized is

$$\sum_{i=1}^{n}(\Delta y_i)^2=\sum_{i=1}^{n}\left(y_i-(b_0+b_1x_i)\right)^2$$

Partial differentiation of the right-hand term in the equation and setting it to zero yields a simple equation for the regression coefficient $b_1$:

$$b_1=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$$

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 of more 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 =
    5.5760   21.2480
```

Since we are working with synthetic data, we know the values for the slope and the intercept with the $y$-axis. The estimated slope (5.5760) and the intercept with the $y$-axis (21.2480) 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
```

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
```

Both, 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 =
   116.0405
```

This result suggests that the sediment at 17 meters depth has an age of ca. 116 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

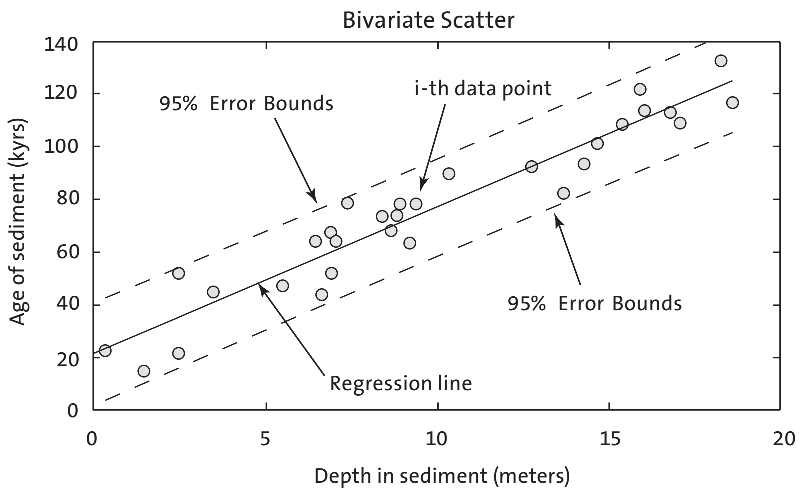


**Fig. 4.5** The result of linear regression. The plot shows the original data points (circles), the regression line (solid line), and the error bounds (dashed lines) of the regression. Note that the error bounds are actually curved although they seem to be almost straight lines in the example.

error in predicting a future observation at `x` by `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.

## 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., Gaussian distributed with zero mean. We therefore test the hypothesis that our residuals are Gaussian distributed by visual inspection of the histogram and by employing a $\chi^2$-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);
```

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 calculate the corresponding frequencies.

```
[n_exp,x] = hist(res,6)

n_exp =
     4     6     5     7     5     3

x =
   -12.7006   -7.3402   -1.9797    3.3807    8.7412   14.1016
```

By selecting bin centers in the locations defined by the function `hist`, a more practical set of classes can be defined.

```
v = -25 : 8 : 20;

n_exp = hist(res,v);
```

The mean and the standard deviation of the residuals are then computed, and used to generate a theoretical distribution that can be compared with the frequency distribution of the residuals. The mean is found to be close to zero, and the standard deviation is 8.7922. The function `normpdf` is used to create the frequency distribution `n_syn` similar to that in our example. The theoretical distribution is scaled according to our original sample data and displayed.

```
n_syn = normpdf(v,mean(res),std(res));

n_syn = n_syn ./ sum(n_syn);
n_syn = sum(n_exp) * n_syn;
```

The first line normalizes `n_syn` to a total of one. The second command scales `n_syn` to the sum of `n_exp`. We can now plot both distributions for comparison.

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

Visual inspection of the bar plots reveals similarities between the data sets. Hence, the $\chi^2$-test can be used to test the hypothesis that the residuals follow a Gaussian distribution.

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

chi2calc =
    3.3747
```

The critical $\chi^2$ can be calculated using `chi2inv`. The $\chi^2$ test requires the degrees of freedom $\Phi$, which is the number of classes reduced by the number of variables and the number of parameters involved. In our example, we defined six classes, we tested the residuals for a Gaussian distribution with two parameters (i.e., the mean and the standard deviation), testing at a 95 % significance level, and the number of variables (i.e., the residuals) is one. The degrees of freedom are therefore $\Phi$=6–(2+1)=3.

```
chi2crit = chi2inv(0.95,3)

chi2crit =
    7.8147
```

Since the critical $\chi^2$ of 7.8147 is well above the measured $\chi^2$ of 3.3747, it is not possible to reject the null hypothesis. Hence, we can conclude that our residuals follow a Gaussian distribution and that the bivariate data set is therefore well described by the linear model.

## 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);

p_bootstrp = bootstrp(1000,'polyfit',meters,age,1);
```

The statistic of the first coefficient, i.e., the slope of the regression line is

```
hist(p_bootstrp(:,1),15)

mean(p_bootstrp(:,1))
std(p_bootstrp(:,1))

ans =
    5.5644

ans =
    0.3378
```

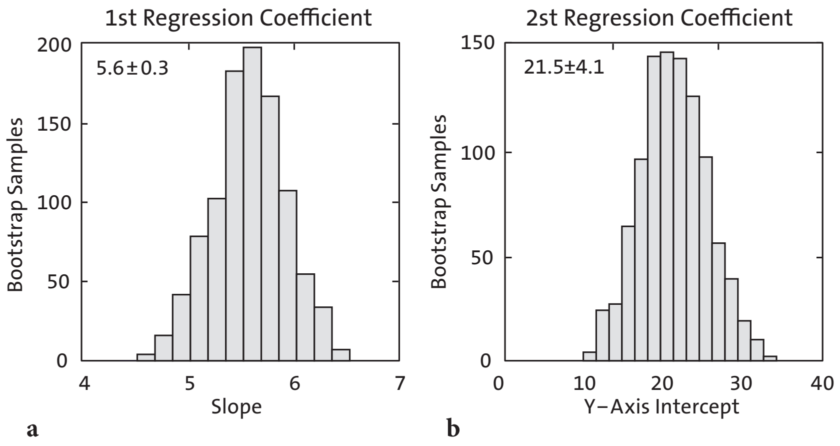


**Fig. 4.6** Histogram of **a**, the first ($y$-axis intercept of the regression line) and **b**, the second (slope of the line) regression coefficient as estimated from bootstrap resampling. Whereas the first coefficient is well constrained, the second coefficient shows a broad scatter.

Because of variations in the random number generators used by `bootstrp` results might vary slightly. The small standard deviation indicates that we have an accurate estimate. In contrast, the statistic of the second parameter shows a significant dispersion.

```
hist(p_bootstrp(:,2),15)

mean(p_bootstrp(:,2))
std(p_bootstrp(:,2))

ans =
    21.5378

ans =
    4.0745
```

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 4.0745 and 21.5378.

## 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 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:

```
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. Within each iteration, the *i*th data point 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,

```
mean(p(:,1))

ans =
    5.5757
```

compared to 5.56±0.34 calculated by the bootstrap method and

```
mean(p(:,2))

ans =
    21.2528
```

compared to 21.54±4.07 from the bootstrap method. 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), hist(p(:,1)), axis square
subplot(1,2,2), hist(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);

mean(p(:,1))
mean(p(:,2))

ans =
    5.5757

ans =
   21.2528

subplot(1,2,1), hist(p(:,1)), axis square
subplot(1,2,2), hist(p(:,2)), axis square
```

The results are identical to the ones 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.

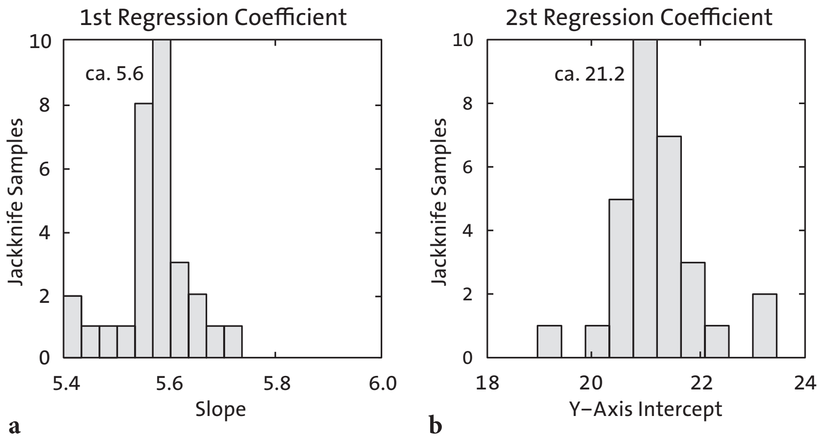


**Fig. 4.7** Histogram of **a**, the first ($y$-axis intercept of the regression line) and **b**, the second (slope of the line) regression coefficient as estimated from jackknife resampling. Note that the parameters are not as well defined as those from bootstrapping.

## 4.7 Cross Validation

A third method to test the goodness-of-fit of a regression is *cross validation*. The regression line is computed by using $n-1$ data points. The $n$th 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. 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 best case – Gaussian distributed with zero mean.

```
mean(p_error)

ans =
    0.0550
```

The standard deviation is an unbiased mean of the deviations of the true data points from the predicted straight line.

```
std(p_error)

ans =
    9.6801
```

Cross validation gives valuable information on the goodness-of-fit of the regression result, and can also be used also 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 cases, neither variable is manipulated and both can therefore be considered to be independent. In these 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 area 0.5*($\Delta x \Delta y$) between the data points and the regression line, where $\Delta x$ and $\Delta y$ are the distances between predicted and the true $x$ and $y$ values (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 =
    5.8286
```

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 =
    5.8286   18.7686
```

The regression line can be plotted by

```
plot(meters,age,'o'), hold on
plot(meters,polyval(p,meters),'r'), hold off
```

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 =
   -0.0544    6.6600   17.3246
```

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 ± 2*s*, 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. 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

rand('seed',40), randn('seed',40)
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 =
    1.6471   -4.2200   80.0314
```

As shown above, the true values for the three coefficients are + 1.6, –1.1 and + 50, 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
plot(meters,polyval(p,meters),'r'), hold off
```

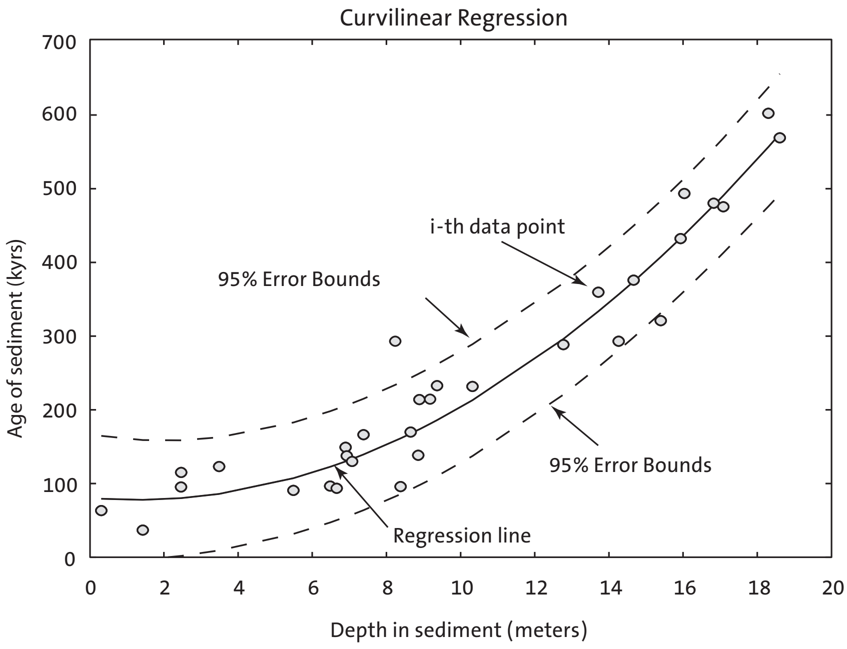


**Fig. 4.8** Curvilinear regression from measurements of barium contents. The plot shows the original data points (circles), the regression line for a polynomial of degree $n$=2 (solid line), and the error bounds (dashed lines) of the regression.

```
[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 0 700]), 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 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 0.1 and 3 in 0.1 intervals and add some Gaussian noise with a standard deviation of 0.2 to make the data unevenly spaced. The resulting 30 data points are stored in the first column of the variable `data`.

```
clear

randn('seed',0)
data(:,1) = 0.1 : 0.1 : 3;
data(:,1) = data(:,1) + 0.2*randn(size(data(:,1)));
```

Next, we can 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 `beta=nlinfit(data(:,1),data(:,2),fun,beta0)` returns a vector `beta` of coefficient estimates for a nonlinear regression of the responses in `data(:,2)` on 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. Constructing a function handle uses the *at* sign, @, before the function name. The variable `beta0` is a vector containing initial values for the coefficients, and is the same length as `beta`. We can design a function handle model representing an exponential function with variables an input variable `t` and coefficients `phi`. The initial values of `beta` are `[0 0]`. We can then use `nlinfit` to estimate the coefficients `beta` using the data, the model and the initial values.

```
model = @(phi,t)(phi(1)*exp(t) + phi(2));
beta0 = [0 0];
beta = nlinfit(data(:,1),data(:,2),model,beta0)

beta =
    0.2006    3.0107
```

We can now use the resulting coefficients `beta(1)` and `beta(2)` to calculate the function values `fittedcurve` using the model and compare the results with the original data.

```
fittedcurve = beta(1)*exp(data(:,1)) + beta(2);
plot(data(:,1),data(:,2),'o')
hold on
plot(data(:,1),fittedcurve,'r')
xlabel('x-Axis'), ylabel('y-Axis')
title('Unweighted Fit')
hold off
```

As we can see from the output of `beta` and the graph, the fitted red curve describes the data fairly 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) = data(:,3)/sum(data(:,3));
```

To make a weighted fit, we first define *weighted* versions of the data `data(:,5)` and the model function `model`, and then use nonlinear least squares to make the fit.

```
data(:,5) = data(:,4).*data(:,2);
model = @(phi,t)(data(:,4).*(phi(1)*exp(t) + phi(2)));
beta0 = [0 0];
beta = nlinfit(data(:,1),data(:,5),model,beta0)

beta =
    0.2045    2.9875
```

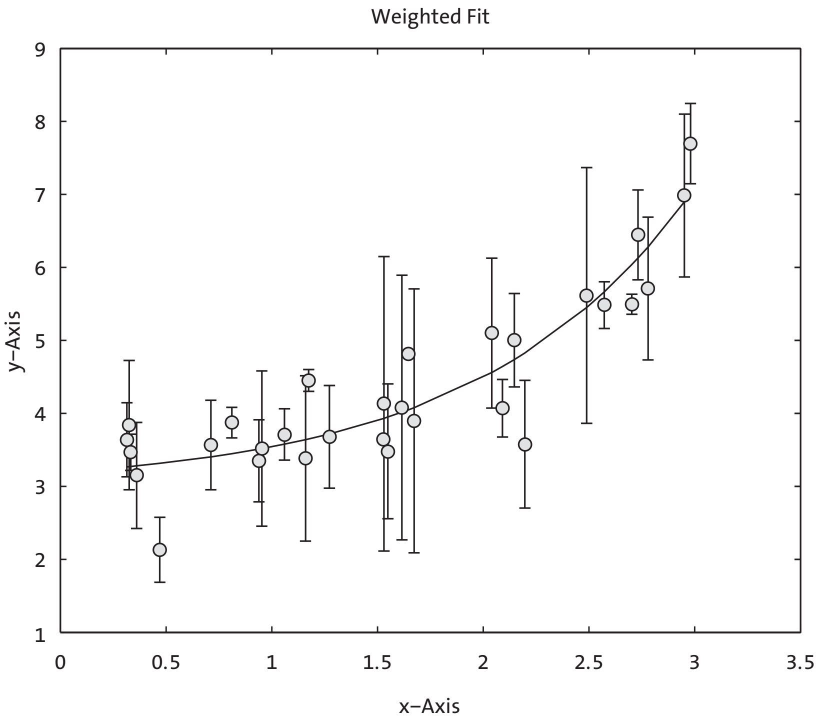


**Fig. 4.9** Weighted regression from synthetic data. The plot shows the original data points (circles), the error bars of all data points, and the regression line for an exponential model function (solid line).

As before, `nlinfit` will compute weighted parameter estimates `beta`. We again use the resulting coefficents `beta(1)` and `beta(2)` to calculate the function values `fittedcurve` using the model and compare the results with the original data.

```
fittedcurve = beta(1)*exp(data(:,1)) + beta(2);
errorbar(data(:,1),data(:,2),data(:,3),'o')
hold on
plot(data(:,1),fittedcurve,'r')
xlabel('x-Axis'), ylabel('y-Axis')
title('Weighted Fit')
hold off
```

Comparing the coefficients `beta` and the red curves from the weighted regression with the previous results from the unweighted regression reveals slightly different results (Fig. 4.9).

## Recommended Reading

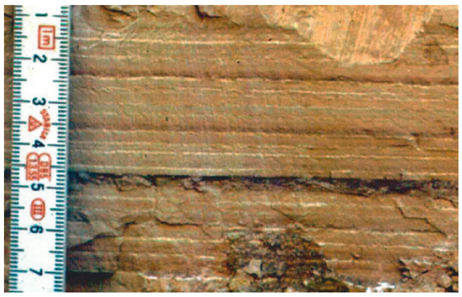
Alberède F (2002) Introduction to Geochemical Modeling. Cambridge University Press, Cambridge
Davis JC (2002) Statistics and Data Analysis in Geology, Third Edition. John Wiley and Sons, New York
Draper NR, Smith, H (1998) Applied Regression Analysis. Wiley Series in Probability and Statistics, John 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
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
Pearson K (1894–98) Mathematical Contributions to the Theory of Evolution, Part I to IV. Philosophical Transactions of the Royal Society 185–191
The Mathworks (2010) Statistics Toolbox 7 – User's Guide. The MathWorks, Natick, MA

# 5 Time-Series Analysis

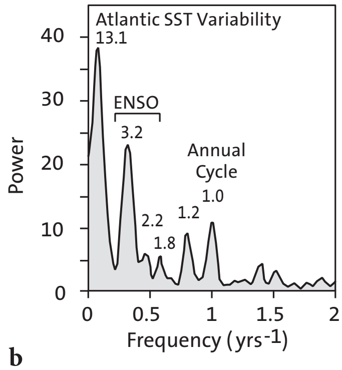
## 5.1 Introduction

Time-series analysis aims to investigate the temporal behavior of one of several variables $x(t)$. 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, millenium-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.

The next section discusses signals in general and contains a technical description of how to generate synthetic signals for time-series analysis (Section 5.2). 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 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 introduced in Section 5.5. Evolutionary power spectra to map changes in cyclicities through time are demonstrated in Section 5.6. An alternative technique for analyzing unevenly-spaced data is explained in Section 5.7. In the subsequent Section 5.8, the very popular wavelet power spectrum is introduced, that has the capability to map temporal variations in the spectra, in a similar way to the method demonstrated in Section 5.6. The chapter closes with an overview of nonlinear techniques, in particular the method of recurrence plots (Section 5.9).



a



b

**Fig. 5.1** **a** Photograph of ca. 30 kyr-old varved sediments from a landslide-dammed lake in the Andes of Northwest Argentina. The mixed clastic-biogenic varves consist of reddish-brown and green to buff-colored clays sourced from Cretaceous redbeds (red-brown) and Precambrian-Early Paleozoic greenshists (green-buff colored). The clastic varves are capped by thin white diatomite layers documenting the bloom of silica algae after the austral-summer rainy season. The distribution of the two source rocks and the interannual precipitation pattern in the area suggests that the reddish-brown layers reflect cyclic recurrence of enhanced precipitation, erosion and sediment input into the landslide-dammed 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$^{-1}$ corresponding to periods of 13.1, 3.2, 2.2, and around 1.0 years. These cyclicities suggest a strong influence of the tropical Atlantic sea-surface temperature (SST) variability (characterized by 10 to 15 year cycles), the El Niño/Southern Oscillation (ENSO) (cycles between 2 and 7 years) and the annual cycle at 30 kyrs ago, similar to today (Trauth et al. 2003).

## 5.2 Generating Signals

A time series is an ordered sequence of values of a variable $x(t)$ at certain times $t_k$.

$$x(t_k) = x(t_1), x(t_2), \ldots, x(t_N)$$

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 $\Delta t$. In most cases, we try to sample at regular time intervals or constant sampling frequencies. However, in some cases equally-spaced data are not available. 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 therefore 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 $\psi_i$.

$$x_p(t_k) = \sum_i A_i \cdot \sin(2\pi f_i t_k - \psi_i)$$

The phase angle $\psi$ 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 $\psi_x$ and $\psi_y$ is equal to zero (Fig. 5.2).

The frequency $f$ of a periodic signal is the inverse of the period $\tau$. 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 disks (CDs) are sampled at frequencies of 44,100 Hz (Hertz, where 1 Hz =1 cycle per second). 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 reduces the frequency band and causes 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

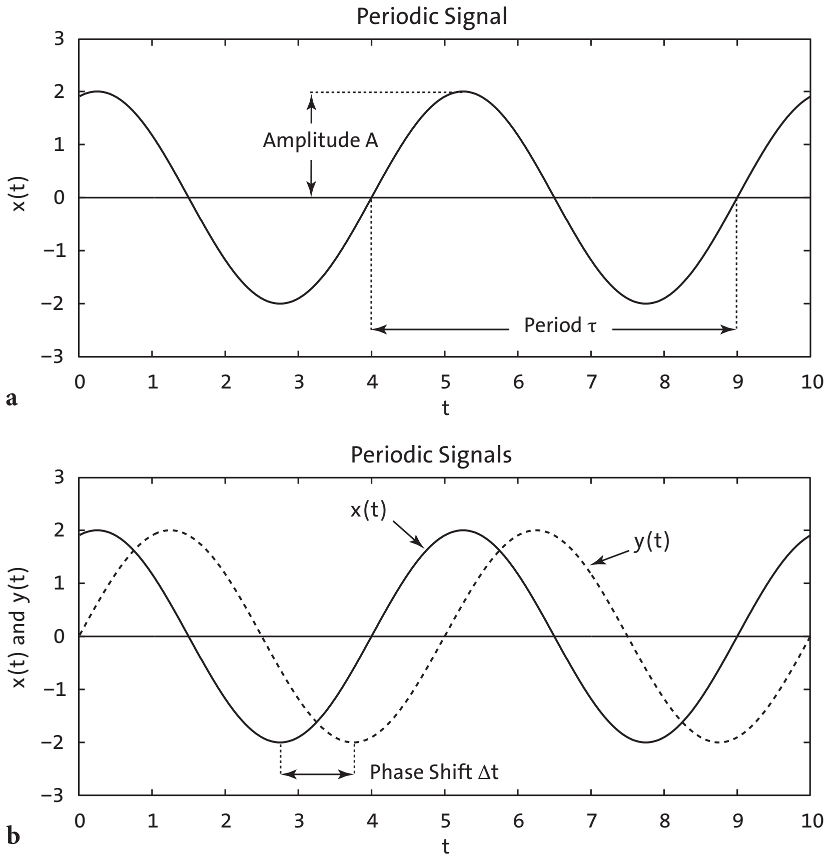


**Fig. 5.2** **a** Periodic signal *x* a function of time *t* defined by the amplitude *A*, and the period $\tau$ which is the inverse of the frequency *f*. **b** Two signals *x* and *y* of the same period are out of phase if the difference between $\psi_x$ and $\psi_y$ is not equal to zero.

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 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(t)`: 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 $\tau$=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 $\tau_1$=50 ($f_1$=0.02), $\tau_2$=15 ($f_2\approx$0.07) and $\tau_3$=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. 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 zero mean and standard deviation one. The seed of the algorithm should be set to zero. One thousand random numbers are then generated using the function `randn`.

```
randn('seed',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). 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 powerspectra.

```
xn = x + n;

plot(t,x,'b-',t,xn,'r-'), axis([0 200 -4 4])
```

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 powerspectra.

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). 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.

## 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 $\Delta t$ 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 $\sigma^2$ of $x(t)$ yields the autocorrelation sequence. Autocorrelation involves correlating a series of data with itself as a function of a time lag $k$.

$$corr_{xx}(k) = \frac{cov_{xx}(k)}{cov_{xx}(0)} = \frac{cov_{xx}(k)}{\sigma_x^{\ 2}}$$

The most popular method used to compute powerspectra in earth sciences is the method introduced by Blackman and Tukey (1958). The *Blackman-*

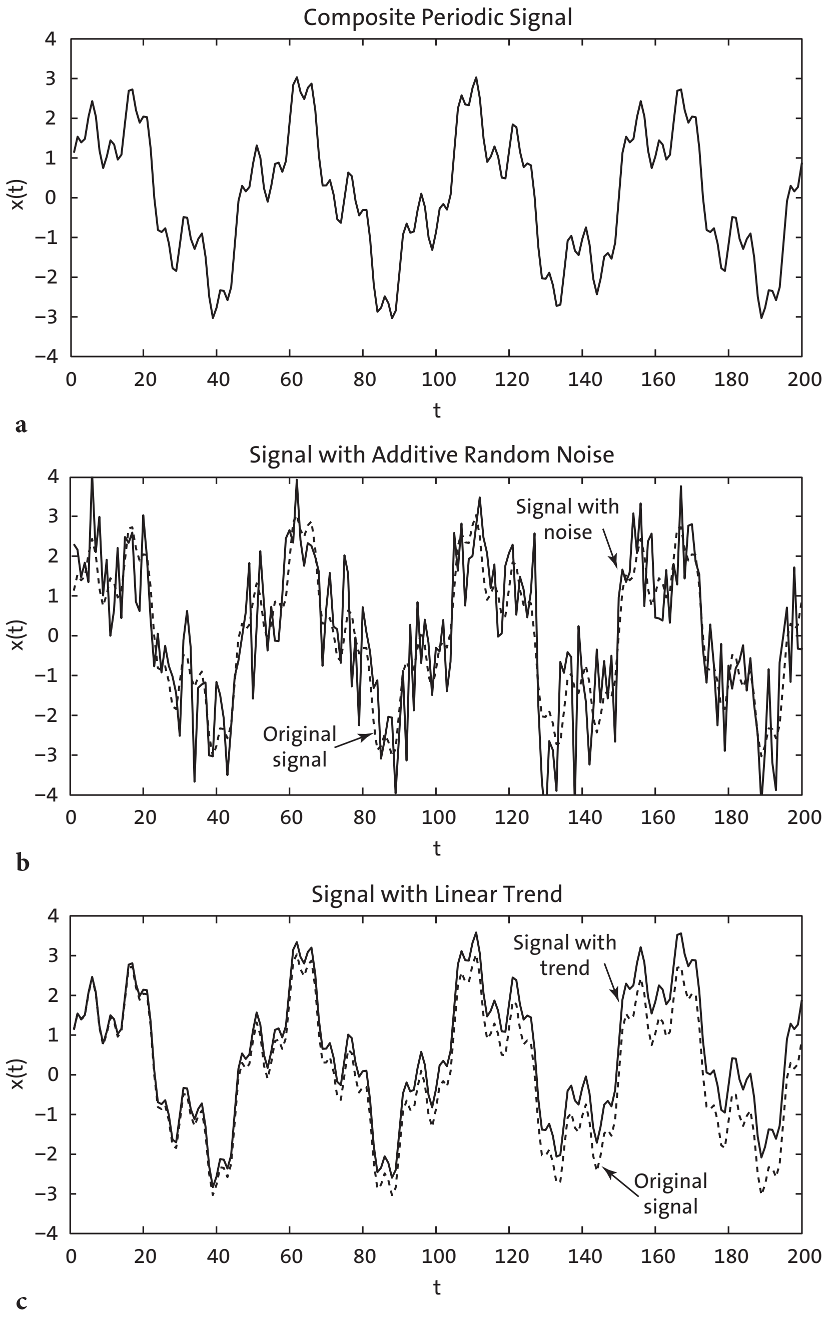


**Fig. 5.3** **a** Synthetic signal with the periodicities $\tau_1$=50, $\tau_2$=15 and $\tau_3$=5, with different amplitudes, and **b** the same signal overprinted with Gaussian noise. **c** In addition, the time series shows a significant linear trend.

*Tukey method* uses the complex Fourier transform $X_{xx}(f)$ of the autocorrelation sequence $corr_{xx}(k)$,

$$X_{xx}(f)=\sum_{k=0}^{M} corr_{xx}(k)\, e^{i2\pi fk/f_s}$$

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 frequency points 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).

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

$$X_{xx}(f) = \sum_{k=0}^{M} corr_{xx}(k) w(k)\, e^{i2\pi fk/f_s}$$

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 do-

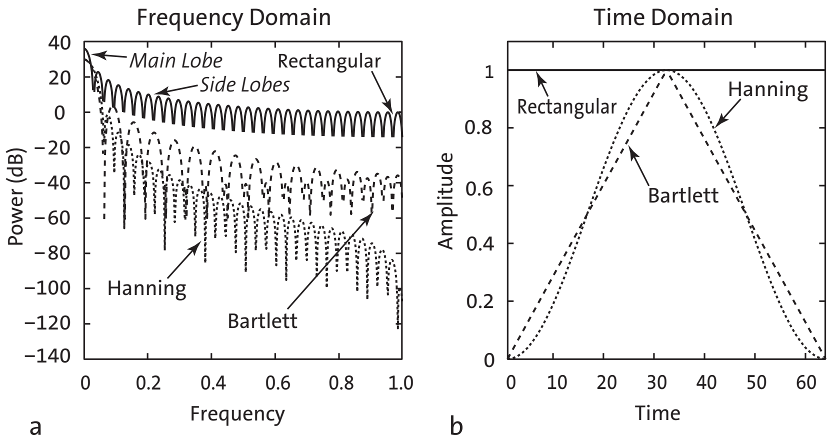


**Fig. 5.4** Spectral leakage. **a** The relative amplitude of the side lobes compared to the main lobe is reduced by multiplying the corresponding time series by **b** a smooth bell-shaped window function. A number of different windows with advantages and disadvantages are available for use instead of the default rectangular window, including *Bartlett* (triangular) and *Hanning* (cosinusoidal) windows. Graph generated using the function `wvtool`.

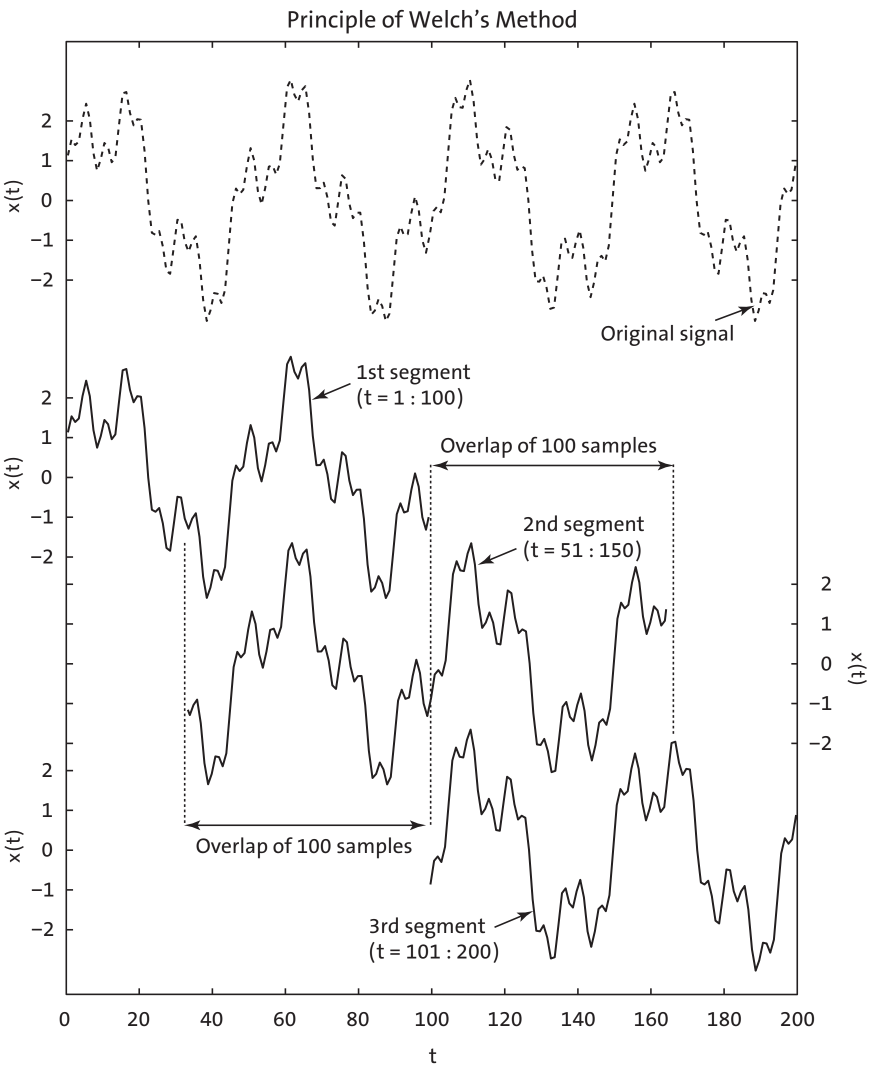


**Fig. 5.5** Principle of Welch's power spectral analysis. The time series is first divided into overlapping segments; the power spectrum for each segment is then computed and all spectra are averaged to improve the signal-to-noise ratio of the power spectrum.

main. The *cross-covariance* is a measure for the variance in 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 $\Delta t$ is

$$cov_{xy}(k) = \frac{1}{N-k-1} \sum_{t=1}^{N-k} (x_i - \bar{x})(y_{i+k} - \bar{y})$$

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 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 lag parameter $k$ 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 Turkey (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;
```

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.

```
plot(f,Pxx), grid
```

The graphical output shows that there are three significant peaks at the positions of the original frequencies 1/50, 1/15 and 1/5 of the three sine waves. 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)])
```

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 of the function `periodogram(x,window,nfft,fs)`. This function allows the windowing of the signals with various window shapes to overcome spectral leakage. We use, however, 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 a 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.

```
[Pxx,f] = periodogram(x,[],1024,1);

plot(f,Pxx), grid
xlabel('Frequency')
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 position of the original frequencies 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 using Gaussian noise with a standard deviation of five and zero mean.

```
randn('seed',0);
n = 5 * randn(size(x));
xn = x + n;

[Pxx,f] = periodogram(xn,[],1024,1);

plot(f,Pxx), grid
xlabel('Frequency')
ylabel('Power')
title('Auto-Spectrum')
```

This spectrum resembles a real data spectrum in the earth sciences. The spectral peaks now sit on 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 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')
```

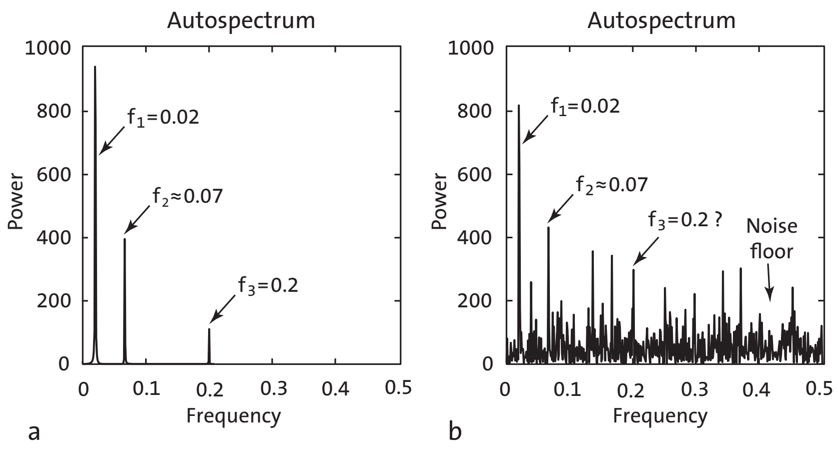


**Fig. 5.6** Comparison of the auto-spectra for **a** the noise-free and **b** the noisy synthetic signals with the periods $\tau_1$=50 ($f_1$=0.02), $\tau_2$=15 ($f_2\approx$0.07) and $\tau_3$=5 ($f_3$=0.2). In particular, the highest frequency peak disappears into the background noise and cannot be distinguished from peaks attributed to the Gaussian noise.

```
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`.

```
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])
```

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')
```

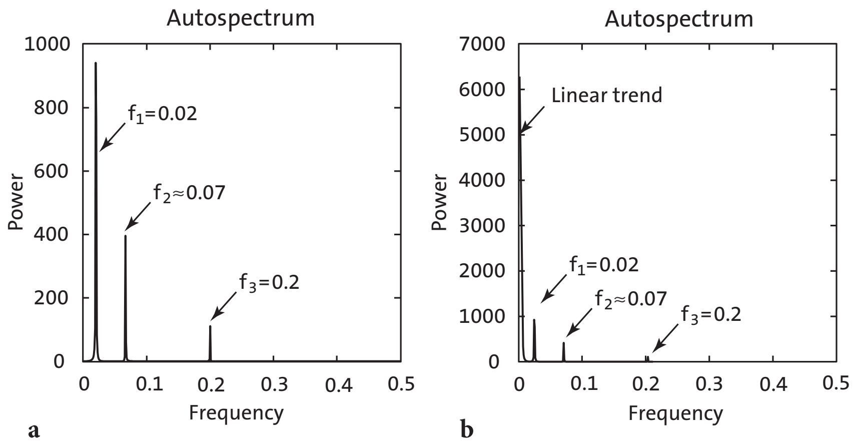


**Fig. 5.7** Comparison of the auto-spectra for **a** the original noise-free signal with the periods $\tau_1$=50 ($f_1$=0.02), $\tau_2$=15 ($f_2\approx$0.07) and $\tau_3$=5 ($f_3$=0.2) and **b** the same signal overprinted on a linear trend. The linear trend is misinterpreted by the FFT as a very long period with a high amplitude.

```
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 by subtracting the corresponding $x(t)$ values.

We now use two sine waves with identical periodicities $\tau$=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 $\tau$=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

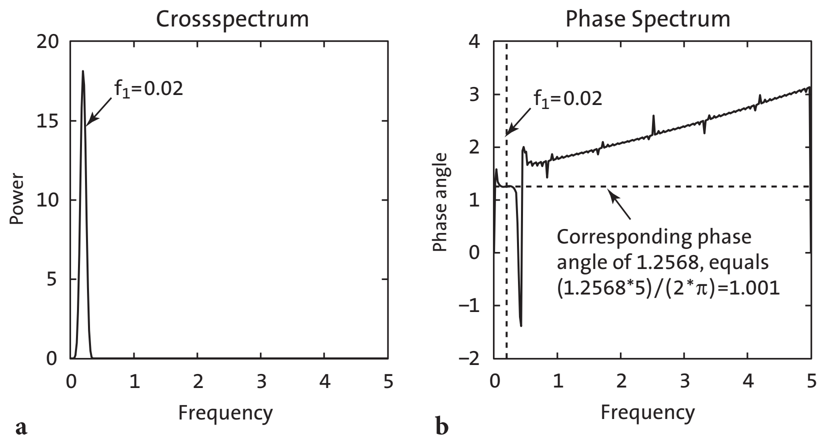


**Fig. 5.8** Cross-spectrum of two sine waves with identical periodicities $\tau$=5 (equivalent to $f$=0.2) and amplitudes of 2. The sine waves show a relative phase shift of $t$=1. In the argument of the second sine wave this corresponds to 2$\pi$/5, which is one fifth of the full wavelength of $\tau$=5. **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*5)/(2*$\pi$)=1.0000, which is the phase shift of 1 that we introduced at the beginning.

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 therefore the default rectangular window is 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 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')
```

The function `mscohere(x,y,window,noverlap,nfft,fs)` specifies the number of FFT points `nfft`=1024, the default rectangular window, and the overlap of zero data points. 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 $\tau$=5) can be interpolated from the phase spectrum

```
interp1(f,phase,0.2)
```

which produces the output

```
ans =
  -1.2566
```

The phase spectrum is normalized to one full period $\tau$=2$\pi$, therefore the phase shift of –1.2566 equals (–1.2566*5)/(2*$\pi$) = –1.0000, which is the phase shift of one that we introduced at the beginning.

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 with 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)
```

which produces the output of

```
ans =
  -1.2572
```

The phase spectrum is normalized to one full period $\tau=2\pi$, therefore the phase shift of –1.2572 equals (–1.2572*5)/(2*$\pi$) = –1.0004, which is again the phase shift of one that we introduced at the beginning.

## 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 (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 algae (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 kyrs. 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;
```

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');
```

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. 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 kyrs$^{-1}$.

```
[Pxx,f] = periodogram(series1L,[],256,1/3);

plot(f,Pxx)
xlabel('Frequency')
```

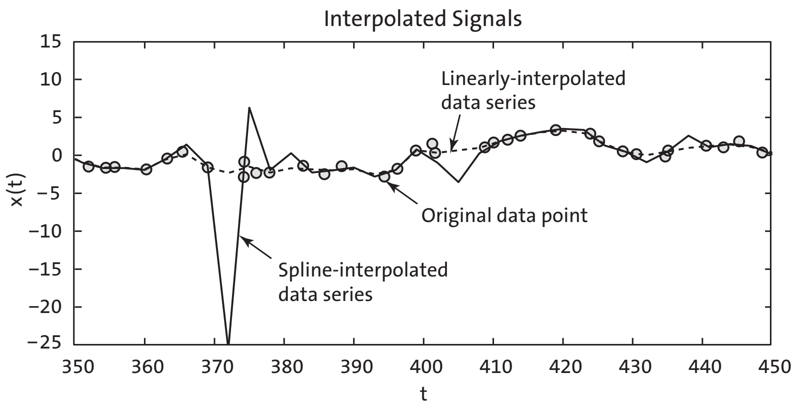


**Fig. 5.9** Interpolation artifacts. Whereas the linearly interpolated points are always within the range of the original data, the spline interpolation method causes unrealistic high and low values.

```
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). Now we 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);

plot(f,phase)
xlabel('Frequency')
ylabel('Phase Angle')
title('Phase spectrum')
```

The phase shift at a frequency of $f$=0.01 is calculated by

```
interp1(f,phase,0.01)
```

which produces the output of

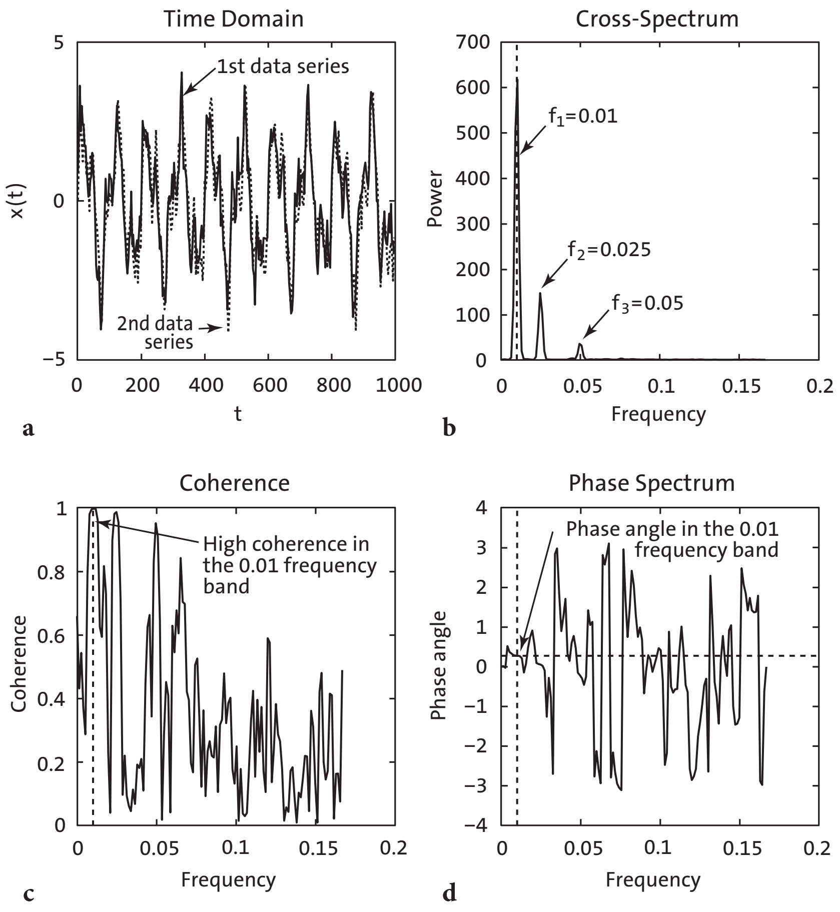


**Fig. 5.10** Result from cross-spectral analysis of the two linearly-interpolated signals: **a** signals in the time domain, **b** cross-spectrum of both signals, **c** coherence of the signals in the frequency domain and **d** phase spectrum in radians.

```
ans =
  -0.2796
```

The phase spectrum is normalized to a full period $\tau=2\pi$, and therefore the phase shift of $-0.2796$ equals $(-0.2796*100$ kyrs$)/(2*\pi)=-4.45$ kyrs. This corresponds roughly to the phase shift of 5 kyrs introduced to the second data series with respect to the first series.

As a more convenient tool for spectral analysis, the Signal Processing Toolbox also contains a GUI function named `sptool`, which stands for *Signal Processing Tool*.

## 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 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 powerspectra 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 powerspectra. The evolutionary power spectrum method therefore uses the Short-Time Fourier Transform (STFT) instead of the Fast Fourier Transformation (FFT). The output of 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 generate 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. We first create a time vector `t`.

```
clear

t = 0 : 3 : 1000;
```

We then introduce some Gaussian noise to the time vector `t` to make the data unevenly spaced.

```
randn('seed',0);
t = t + randn(size(t));
```

Next, we compute the signal with the three periodicities and varying amplitudes. The 40 kyr cycle appears after ca. 450 kyrs, whereas the 100 and 20 kyr cycles are present throughout the time series.

```
x1 = 0.5*sin(2*pi*t/100) + ...
     1.0*sin(2*pi*t/40)  + ...
     0.5*sin(2*pi*t/20);
x2 = 0.5*sin(2*pi*t/100) + ...
     0.5*sin(2*pi*t/20);
```

```
x = x1; x(1,150:end) = x2(1,150:end);
```

We then add Gaussian noise to the signal.

```
x = x + 0.5*randn(size(x));
```

Finally, we save the synthetic data series to the file *series3.txt* on the hard disk and clear the workspace.

```
series3(:,1) = t;
series3(:,2) = x;
series3(1,1) = 0;
series3(end,1) = 1000;
series3 = sortrows(series3,1);
save series3.txt series3 -ascii
```

The above series of commands illustrates how to generate synthetic time series that show the same characteristics as oxygen-isotope data from calcareous algae (foraminifera) in deep-sea sediments. This synthetic data set is suitable for use in demonstrating the application of methods for spectral analysis. The following sequence of commands assumes that real data are contained in a file named *series3.txt*. We first load 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')
```

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')
```

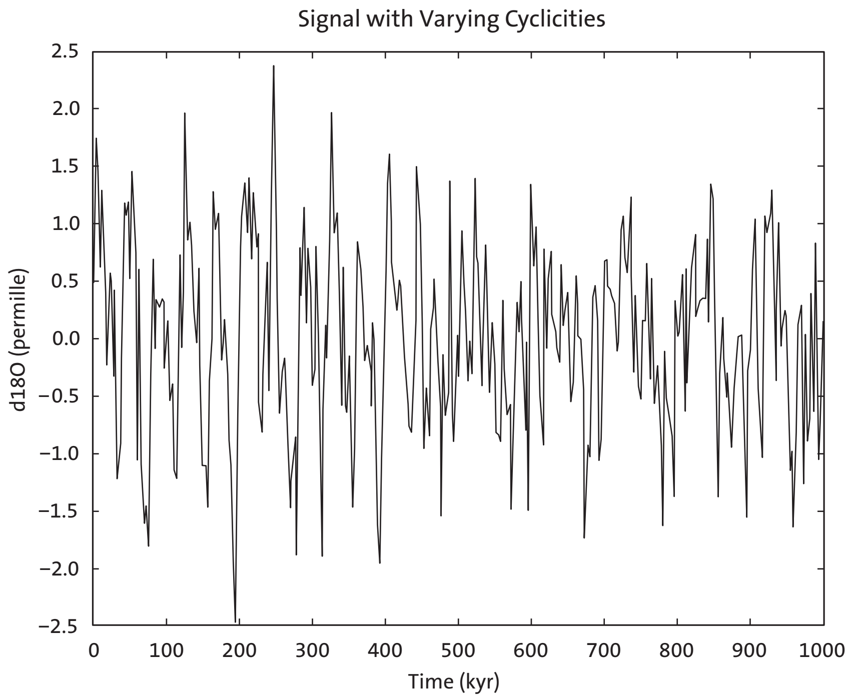


**Fig. 5.11** Synthetic data set containing three main periodicities of 100, 40, and 20 kyrs and additive Gaussian noise. Whereas the 100 and 20 kyr cycles are present throughout the time series, the 40 kyr cycle only appears at around 450 kyrs before present.

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 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,

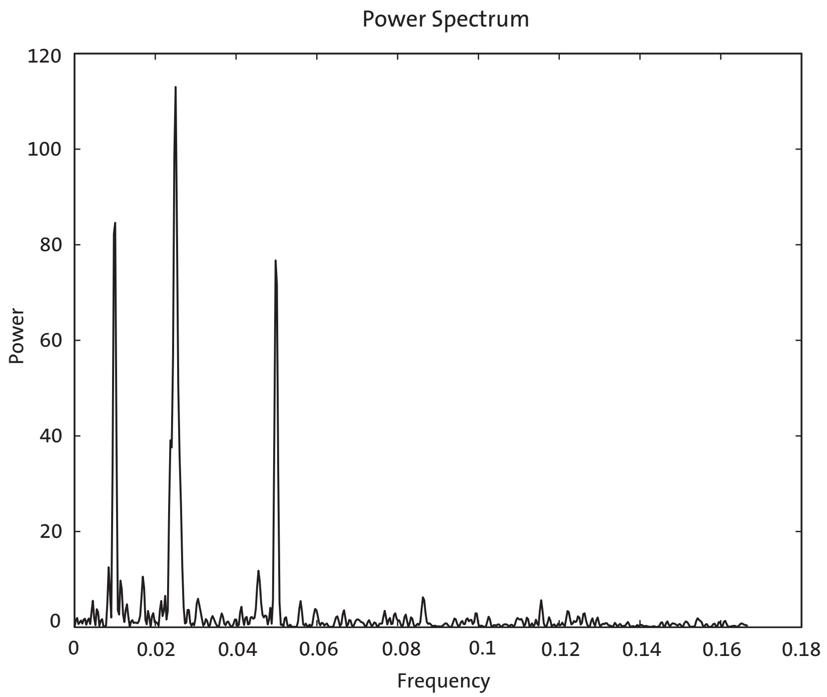


**Fig. 5.12** Power spectrum for the full time series showing significant peaks at 100, 40 and 20 kyrs. The plot, however, does not provide any information on the temporal behavior of the cyclicities.

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)')
```

The output of `spectrogram` is a color plot (Fig. 5.13) that displays vertical stripes in red representing significant maxima at frequencies of 0.01 and 0.05 kyr$^{-1}$, or 100 and 20 kyr cyclicities. The 40 kyr cycle (corresponding to a frequency of 0.025 kyr$^{-1}$), however, 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 coloration of the graph can be modified using the colormap editor.

```
colormapeditor
```

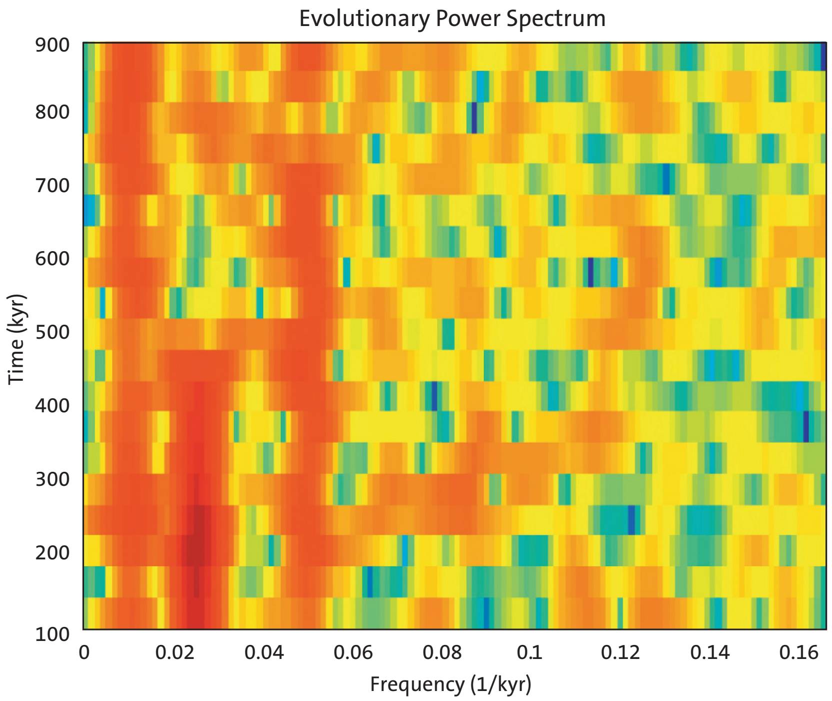


**Fig. 5.13** Evolutionary power spectrum using `spectrogram`, which computes the short-time Fourier transform STFT of overlapping segments of the time series. We use a Hamming window of 64 data points and 50 data points overlap. The STFT is computed for a `nfft`=256. Since the spacing of the interpolated 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.

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 vertical peak areas at 100, 40 and 20 kyrs, and the background.

## 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 to 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 *Lomb-Scargle algorithm* (e. g., Scargle 1981, 1982, 1989, 1990, Press et al. 1992, Schulz et al. 1998).

The Lomb-Scargle algorithm evaluates the data of the time series only 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

$$P_x(\omega) = \frac{1}{2\sigma^2}\left\{\frac{\left[\sum_j (y_j - \overline{y})\cos\omega(t_j - \tau)\right]^2}{\sum_j \cos^2\omega(t_j - \tau)} + \frac{\left[\sum_j (y_j - \overline{y})\sin\omega(t_j - \tau)\right]^2}{\sum_j \sin^2\omega(t_j - \tau)}\right\}$$

where

$$\overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

and

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

are the arithmetic mean and the variance of the data (Section 3.2). The constant $\tau$ is an offset that makes $\mathrm{P}_x(\omega)$ independent of shifting the $t_i$'s by any constant amount. Scargle (1982) showed that this particular choice of the offset $\tau$ 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 unit mean. 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 larger value than $z$ is $(1-\exp(-z))M$. We can therefore compute the false-alarm probability of the null hypothesis, e. g., the probability that a given peak in the periodogram is not significant, by

$$P(>z) \equiv 1-(1-e^{-z})^M$$

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.com/matlabcentral/fileexchange/
```

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. 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 zero-frequency portion, i. e., with infinite periods, we start at a frequency value that is equivalent to the spacing of the frequency vector. `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: a common way to choose `fhi` is to take the Nyquist frequency `fnyq` that would be obtained if the `N` data points were evenly spaced over the same time interval. 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
```

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)).^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 95 % significant level suggesting that they represent significant cyclicities. We have therefore obtained similar results to those obtained from the periodogram method. However, the Lomb-Scargle method does not require any interpolation of unevenly-spaced data. Furthermore, it allows for 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. In principle, a similar modification could be applied to the Lomb-Scargle method, which

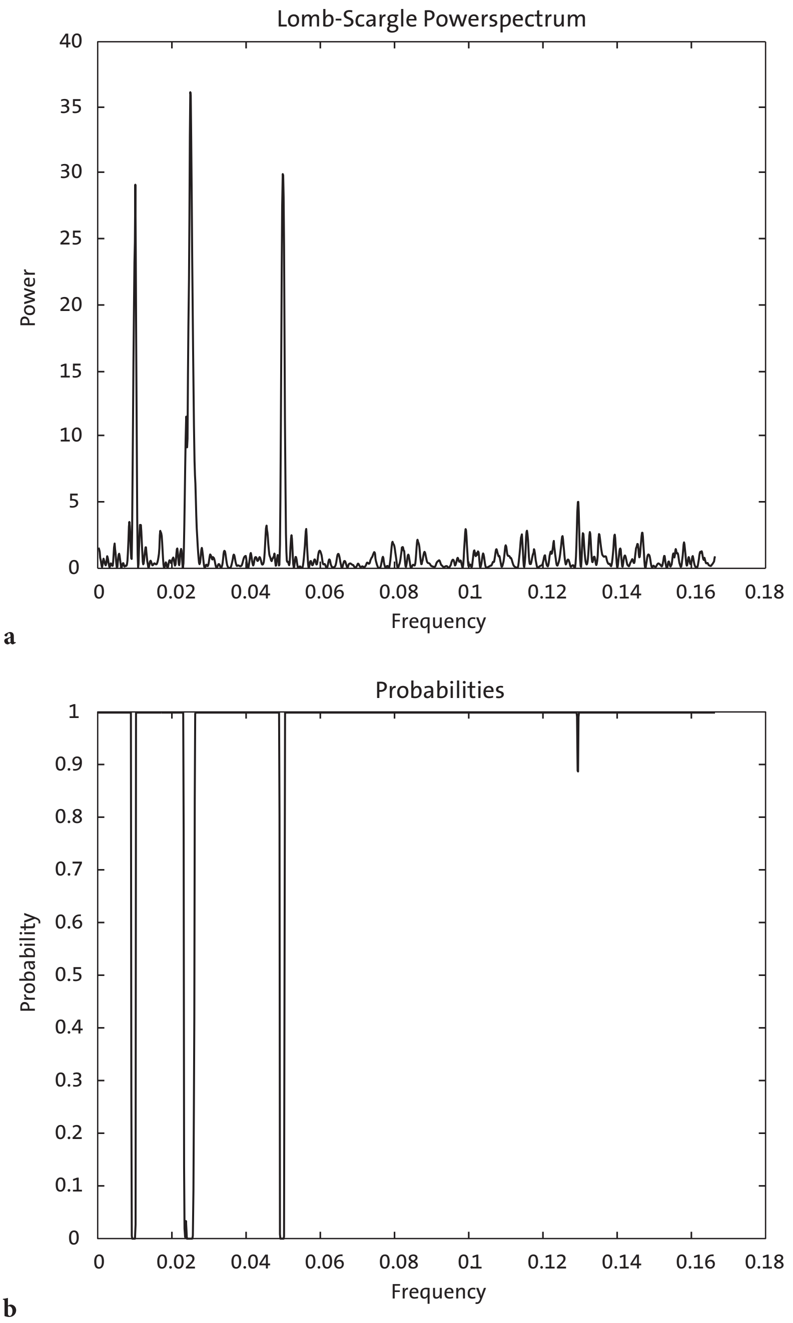


**Fig. 5.14** **a** Lomb-Scargle power spectrum and **b** the false-alarm probability of the null hypothesis. The plot suggests that the 100, 40 and 20 kyr cycles are highly significant.

would have the advantage that it could then be applied to unevenly-spaced data. Both methods, however, assume that the data are a composite 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. Mathematically, a wavelet transformation decomposes a signal $y(t)$ into elementary functions $\psi_{a,b}(t)$ derived from a *mother wavelet* $\psi(t)$ by dilation and translation,

$$\psi_{a,b} = \frac{1}{(a)^{1/2}} \psi\left(\frac{t-b}{a}\right)$$

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 $\psi^*$ is the complex conjugate of $\psi$ defined on the open time and scale real $(b,a)$ half plane.

There are many mother wavelets available in the literature, such as the classic *Haar* wavelet, the *Morlet* wavelet and the *Daubechies* wavelet. The most popular wavelet in geosciences is the Morlet wavelet, which is given by

$$\psi_0(\eta) = \pi^{-1/4} \exp(i\omega_0\eta)\exp(-\eta^2/2)$$

where $\eta$ is the time and $\omega_0$ is the wavenumber (Torrence and Compo 1998). The wavenumber 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 wavenumber. 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 powerspectra, we use a pure

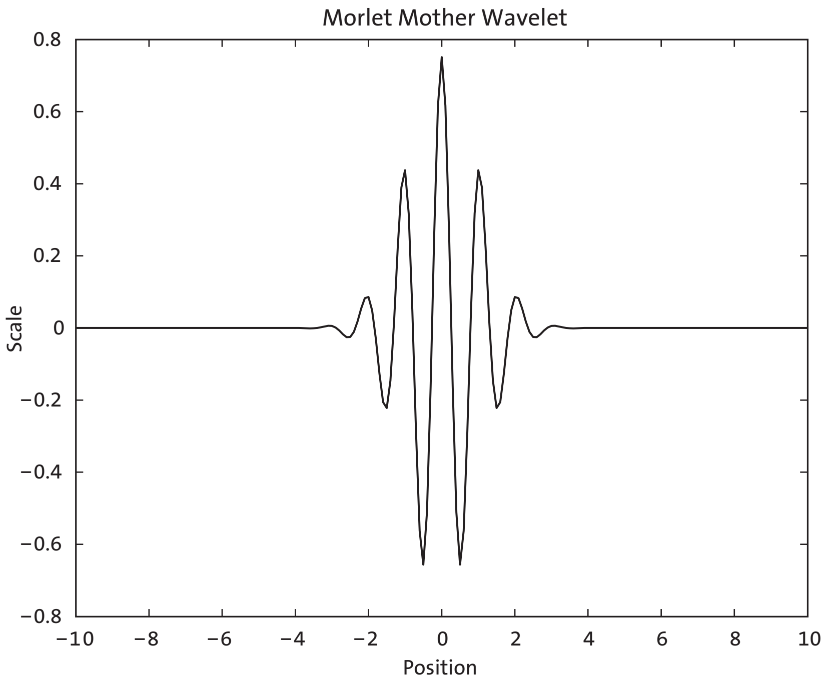


**Fig. 5.15** Morlet mother wavelet with wavenumber 6.

sine wave with a period five and additive Gaussian noise.

```
clear

t = 0 : 0.5 : 50;
x = sin(2*pi*t/5) + randn(size(t));
```

As a first step, we need to define the number of scales for which the wavelet transform will be computed. The scales define how much a wavelet is stretched or compressed to map the variability of the time series at different wavelengths. Lower scales correspond to higher frequencies and therefore map rapidly-changing details, whereas higher scales map the long-term variations. As an example, we can run the wavelet analysis for 120 different `scales` between 1 and 120.

```
scales = 1 : 120;
```

In the next step, we compute the real or complex continuous Morlet wavelet coefficients, using the function `cwt` contained in the Wavelet Toolbox.

```
coefs = cwt(x,scales,'morl');
```

The function `scal2frq` converts scales to pseudo-frequencies, using the Morley mother wavelet and a sampling period of 0.5.

```
f = scal2frq(scales,'morl',0.5);
```

We use a filled contour plot to portray the power spectrum, i. e., the absolute value of the wavelet coefficients (Fig. 5.16).

```
contour(t,f,abs(coefs),'LineStyle','none','LineColor', ...
   [0 0 0],'Fill','on')
xlabel('Time')
ylabel('Frequency')
title('Wavelet Power Spectrum')
```

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, 20 kyr as well as additive Gaussian noise, and 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');
```

We then compute the wavelet transform for 120 scales using the function `cwt` and a Morley mother wavelet, as we did for the previous example.

```
scales = 1 : 120;
coefs = cwt(series3L,scales,'morl');
```

We use `scal2freq` to convert scales to pseudo-frequencies, using the Morley mother wavelet and the sampling period of three.

```
f = scal2frq(scales,'morl',3);
```

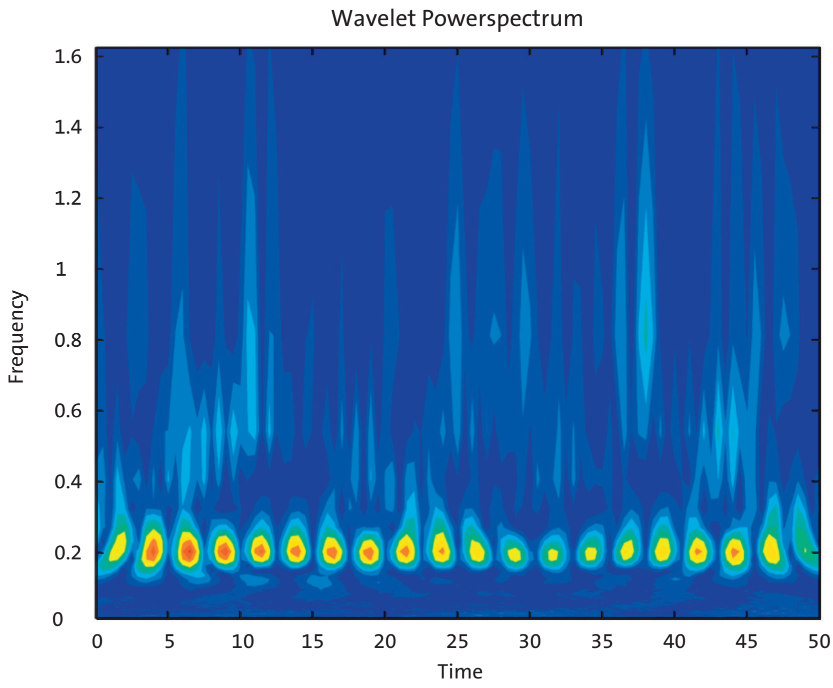


**Fig. 5.16** Wavelet power spectrum showing a significant period at 5 cycles that persists throughout the full length of the time vector.

We can now use a filled contour plot to portray the wavelet power spectrum, i. e., the absolute value of the wavelet coefficients (Fig. 5.17).

```
contour(t,f,abs(coefs),'LineStyle', 'none', ...
   'LineColor',[0 0 0],'Fill','on')
xlabel('Time'),ylabel('Frequency')
title('Wavelet Power Spectrum')
```

The graph shows horizontal clusters of peaks at 0.01 and 0.05 kyr$^{-1}$ corresponding to 100 and 20 kyr cycles, although the 20 kyr cycle is not very clear. The wavelet power spectrum also reveals a significant 40 kyr cycle or a frequency of 0.025 kyr$^{-1}$ that appears at ca. 450 kyrs before present. Compared to the windowed power spectrum method, the wavelet power spectrum clearly shows a much higher resolution on both the time and the frequency axes. Instead of dividing the time series into overlapping seg-

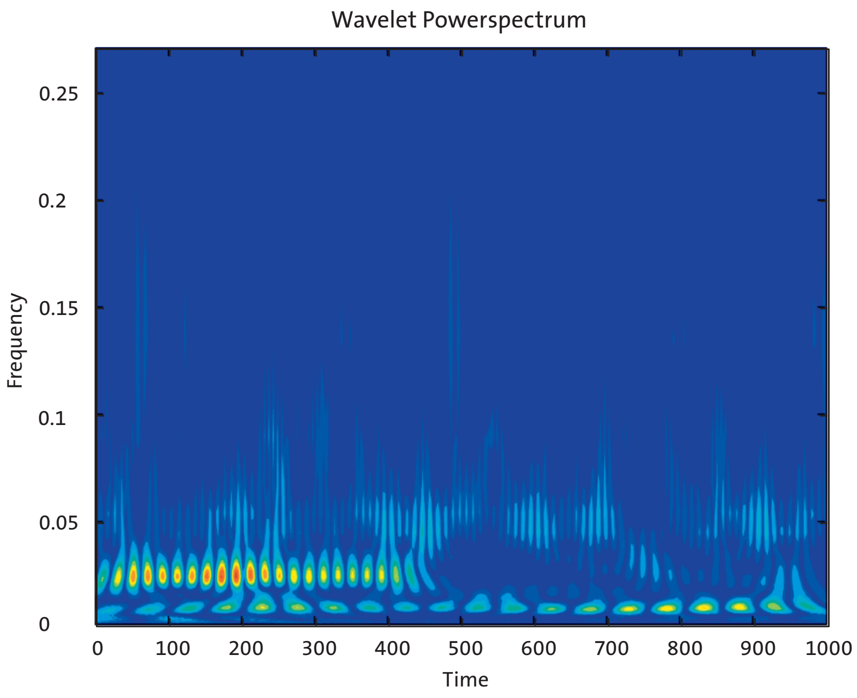


**Fig. 5.17** Wavelet power spectrum for the synthetic data series contained in *series_3.txt*. The plot clearly shows significant periodicities at frequencies of 0.01, 0.25, and 0.5 kyr$^{-1}$ corresponding to the 100, 40, and 20 kyr cycles. The 100 kyr cycle is present throughout the entire time series, whereas the 40 kyr cycle only appears at around 450 kyrs before present. The 20 kyr cycle is relatively weak but probably present throughout the entire time series.

ments 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 spectral analyses, however, is the requirement for evenly-spaced data. The Lomb-Scargle method overcomes this problem but as for the power spectrum method, has limitations in its ability to map temporal changes in the frequency domain.

## 5.9 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 therefore yield unsatisfactory results. In recent decades, new techniques for nonlinear data analysis derived from chaos theory have become increasingly popular. Such methods have, for example, been employed to describe nonlinear behavior by defining, e. g., 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.

In the last decade, *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 arbitrarily close to a former state and pass through a similar evolution. Recurrence plots allow such recurrent behavior of dynamical systems to be visually portrayed. The method is now a widely accepted 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 $x_1(t), x_2(t), \ldots, x_d(t)$. 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 – the so-called 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 course of 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 $\tau$ is the time delay (index based; the real time delay is $\tau=\Delta t$). This reconstruction of the phase space is called *time delay embedding*. The phase space reconstruction is not exactly the same as the original phase space, but its topological properties are preserved, provided the embedding dimension is large enough. In practice, the embedding dimension must be more than twice the the dimension of the attractor, or exactly $m>2d+1$. 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. First, we 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. 5.18). Using the time delay embedding, we can reconstruct this phase space portrait using only a single observation, e.g., the velocity vector, and a delay of 5, which corresponds to a quarter of the period of our pendulum.

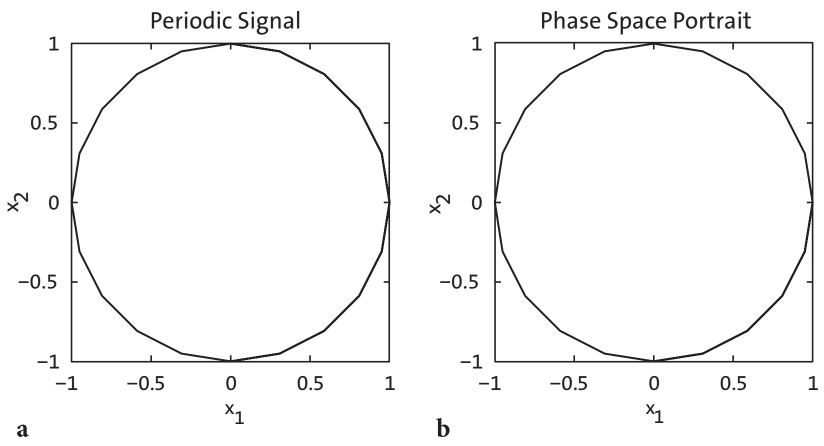


**Fig. 5.18** **a** Original and **b** reconstructed phase space portrait for a periodic system. The reconstructed phase space is almost the same as the original phase space.

```
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 the one for a typical nonlinear system, the Lorenz system (Lorenz 1963). When studying weather patterns, it is clear that weather often does not change as predicted. 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 system is 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
```

Typical traces of a variable, such as the first variable can be viewed by plotting `x(:,1)` over time in seconds (Fig. 5.19).

```
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. 5.20).

```
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 of such a system cause exponential divergences of its state, the system returns approximately to a previous state through a similar course.

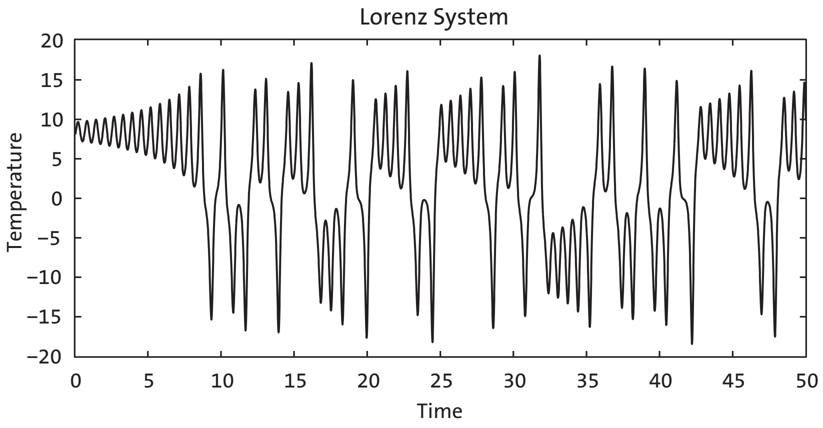


**Fig. 5.19** The Lorenz system. As system parameters we use `s=10`, `r=28` and `b=8/3`; the time delay is `dt=0.01`.

The reconstruction of the phase space portrait using only the first state and a delay of six

```
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')
```

reveals a similar phase portrait with the two typical *ears* (Fig. 5.20). The characteristic properties of chaotic systems can also be observed in this reconstruction.

The time delay and embedding dimension need to be chosen by a preceding 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
```

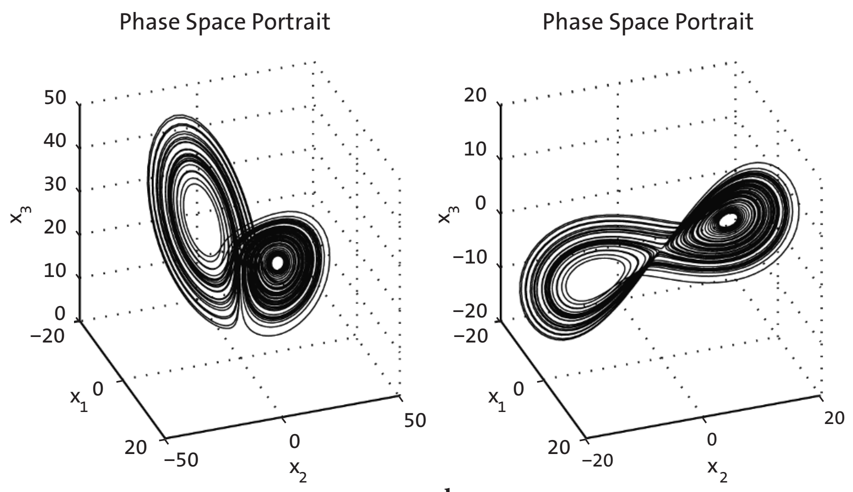


**Fig. 5.20** **a** The phase space portrait for the Lorenz system. In contrast to the simple periodic system, the trajectories of the Lorenz system obviously do not follow precisely the previous course, but recur very close to it. **b** The reconstruction of the phase space portrait using only the first state and a delay of 6 seconds reveals a topologically similar phase portrait to a, with the two typical ears.

```
plot(C)
xlabel('Delay'), ylabel('Autocorrelation')
grid on
```

We now choose a delay such that the autocorrelation function equals zero for the first time. In our case this is 5, which is the value that we have already used in our example of phase space reconstruction. The appropriate embedding dimension can be estimated by 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 interrelations or synchronisation between several 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.

$$R_{i,j} = \begin{cases} 0 & \left\| x_i - x_j \right\| > \varepsilon \\ 1 & \left\| x_i - x_j \right\| \le \varepsilon \end{cases}$$

If the distance between two states *i* and *j* on the trajectory is smaller than a given threshold $\varepsilon$, 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 interrelations within a system (cf. Fig. 5.21).

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 have 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. The calculation of the distances between all points of the phase space trajectory produces the distance matrix `S`.

```
N = length(series1L);
S = zeros(N, N);
for i = 1 : N,
    S(:,i) = abs(repmat(series1L(i), N, 1 ) - series1L(:));
end
```

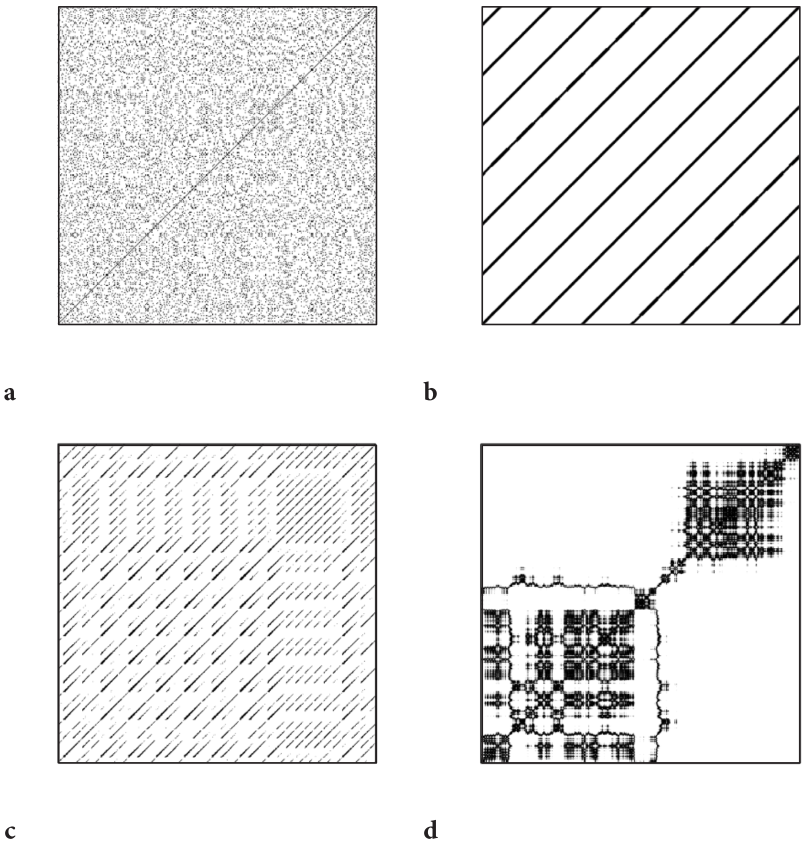


**Fig. 5.21** Recurrence plots representing typical dynamical behaviors: **a** stationary uncorrelated data (white noise), **b** periodic oscillation, **c** chaotic data (Roessler system) and **d** non-stationary data with abrupt changes.

We can now plot the distance matrix

```
imagesc(t,t,S)
colorbar
xlabel('Time'), ylabel('Time')
```

for the data set, where a colorbar provides a quantitative measure of the distances between states (Fig. 5.22). We now apply a threshold $\varepsilon$ to the distance matrix to generate the black/white recurrence plot (Fig. 5.23).

```
imagesc(t,t,S<1)
colormap([1 1 1;0 0 0])
xlabel('Time'), ylabel('Time')
```

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 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, which 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;
```

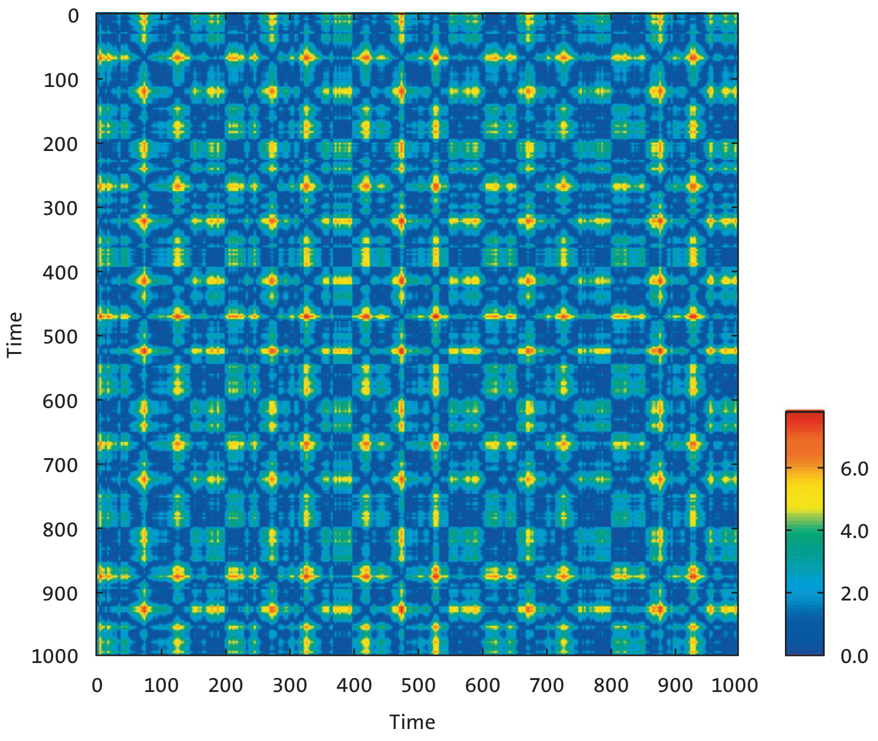


**Fig. 5.22** Display of the distance matrix from the synthetic data providing a quantitative measure for the distances between states at certain times; blue colors indicate small distances, red colors represent large distances.

```
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 $\tau$=2, which corresponds to a delay of 0.17 seconds.

```
t = 0.01 : 0.05 : 50;
y = x(1:5:5000,1);
m = 3; tau = 2;

N = length(y);
N2 = N - tau*(m - 1);
```

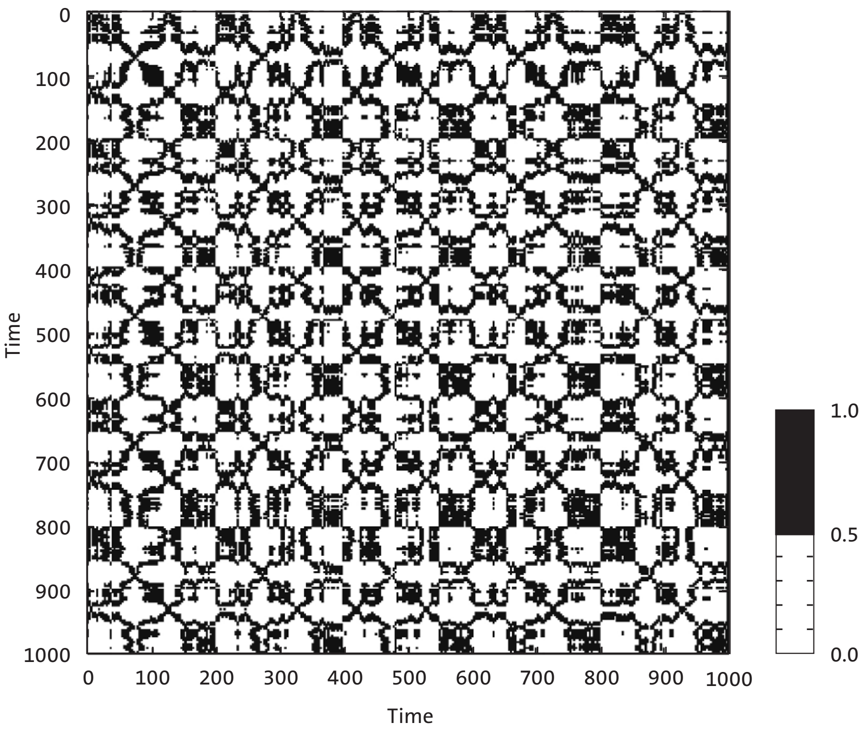


**Fig. 5.23** The recurrence plot for the synthetic data derived from the distance matrix as shown in Fig. 5.22 after applying a threshold of $\varepsilon$=1.

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 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);
```

Using 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')
```

This recurrence plot reveals many short diagonal lines (Fig. 5.24). These lines represent epochs, where the phase space trajectory runs parallel to earlier or later sequences in this trajectory, i. e., at the times when 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. 5.21).

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. These parameters can be used to trace hidden transitions in a process. Bivariate and multivariate extensions of recurrence plots furthermore permit nonlinear correlation tests and synchronization analyses to be carried out. A detailed introduction to recurrence plot based methods can be found at the 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 precipita-

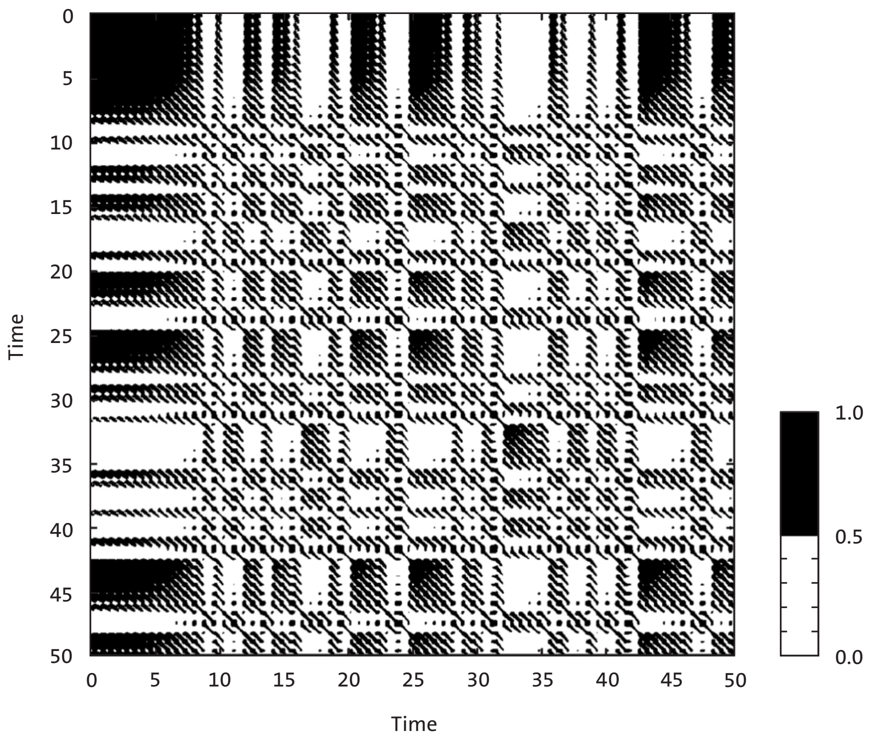


**Fig. 5.24** The recurrence plot for the Lorenz system using a threshold of $\varepsilon=2$. The regions with regular diagonal lines reveal unstable periodic orbits, typical of chaotic systems.

tion 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 example, the method of recurrence plots was applied to red-color intensity transects across ca. 30 kyr-old varved lake sediments (Section 8.7). Comparing the recurrence plots from the sediments with the ones from modern precipitation data revealed that the reddish layers document more intense rainy seasons that occurred during the La Niña years. The application of linear techniques was not able to link the increased flux of reddish clays and enhanced precipitation to 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

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.

Eckmann JP, Kamphorst SO, Ruelle D (1987) Recurrence Plots of Dynamical Systems. Europhysics Letters 5:973–977

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

Lomb NR (1972) Least-Squared Frequency Analysis of Unequally Spaced Data. Astrophysics 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

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

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, Flannery BP (2007) Numerical Recipes: The Art of Scientific Computing – 3rd 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 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

The Mathworks (2010) Signal Processing Toolbox – User's Guide. The MathWorks, Natick, MA

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
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 Powerspectra: A Method Based on Time Averaging over Short, Modified Periodograms. IEEE Trans. Audio Electroacoustics AU–15:70–73

# 6 Signal Processing

## 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 to deep-sea sediment records. Aside from such natural filters, the field collection and sampling of geological data alters and

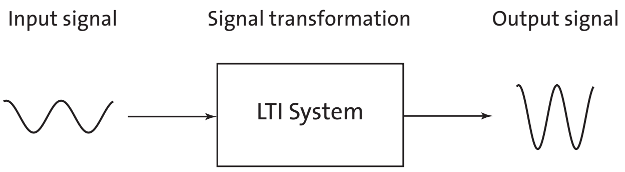


**Fig. 6.1** Schematic of a linear time-invariant (LTI) system. The input signal is transformed into an output signal.

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*, 3rd ed.,
DOI 10.1007/978-3-642-12762-5_6, © Springer-Verlag Berlin Heidelberg 2010

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, such 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 on generating periodic signals, trends and noise, similar to 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 in telecommunication. Adaptive filters automatically extract noisefree 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.

## 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 $\tau = 10$, 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 for a single data point, at which it equals one. The function `stem` plots the data sequence `x` as stems from the *t*-axis with circles for the data values.

```
t = (1:100)';
x = [zeros(49,1);1;zeros(50,1)];

stem(t,x)
axis([0 100 -4 4])
```

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 $x_i(t)$

  $$x(t) = k_1 x_1(t) + k_2 x_2(t)$$

  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 means that input and output can 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;

  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 a such 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. As an example, a core logger measuring magnetic susceptibility with a loop sensor integrates over a certain core interval with highest sensitivity at the location of the loop and decreasing sensitivity down- 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 system 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

```
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 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

$$y(t) = \frac{1}{5} \sum_{k=-2}^{2} x(t-k)$$

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 also 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 may 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 energy 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

$$y(t) = \sum_{k=0}^{N} b_k \cdot x(t-k)$$

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.

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)';
randn('seed',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(b1,x1);
```

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 the length $n+m-1$, 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
```

Here, we see that the actual input series `x1` has a length of 100 data points, whereas the output `y1` has two additional elements. Generally, convolution introduces $(m-1)/2$ data points at each end of the data series. To compare input and output signal, we 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. 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 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.

```
b2 = [1 1 1 1 1]/5;
m2 = length(b2);

y2 = conv(b2,x1);
y2 = y2(1+(m2-1)/2:end-(m2-1)/2,1);

plot(t,x1,'b-',t,y1,'r-',t,y2,'g-')
legend('x1(t)','y1(t)','y2(t)')
```

The next section provides a more general description 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 more general 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)';
randn('seed',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`. As in the examples demonstrated in the previous section, the phase shift of `conv` needs to be corrected.

```
y3 = conv(b3,x3);
y3 = y3(1+(m3-1)/2:end-(m3-1)/2,1);
```

Next, we follow a similar procedure with the function `filter` and compare the result with that obtained using the function `conv`. In contrast to the function `conv`, the function `filter` yields an output vector with the same length as the input vector; we also have to correct this output. Here, 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, in most applications 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 then the phase correction is 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 =
    0.3734
    0.4437
    0.3044
```

```
        0.4106
        0.2971

    ans =
        0.3734
        0.4437
        0.3044
        0.4106
        0.2971
```

This was the lower end of the output. We can see that both vectors `y3` and `y4` contain the same elements. Now we explorer the upper end of the data vector, where

```
    y3(end-5:end,1)
    y4(end-5:end,1)
```

causes the output

```
    ans =
        0.2268
        0.1592
        0.3292
        0.2110
        0.3683
        0.2414

    ans =
        0.2268
        0.1592
             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,'g-')
    subplot(2,1,2), plot(t,x3,'b-',t,y4,'g-')
```

or in one single plot,

```
    plot(t,x3,'b-',t,y3,'g-',t,y4,'r-')
```

shows that the results of `conv` and `filter` are identical except for 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,

$$y(t) = \sum_{k=0}^{N} b_k \cdot x(t-k) - \sum_{k=1}^{M} a_k \cdot y(t-k)$$

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 as in the previous example to illustrate the performance of a recursive filter.

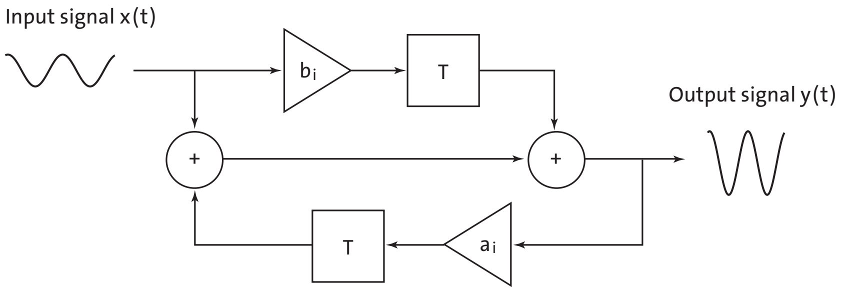


**Fig. 6.2** Schematic of a linear time-invariant filter with an input $x(t)$ and an output $y(t)$. The filter is characterized by its weights $a_i$ and $b_i$. and the delay elements $T$. Nonrecursive filters have only nonrecursive weights $b_i$, whereas the recursive filter also requires the recursive filter weights $a_i$.

```
clear

t = (1:100)';
randn('seed',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 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);
```

Now we can plot the results.

```
plot(t,x5,'b-',t,y5,'r-')
```

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 $\tau=10$ are 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 other periodicities remain relatively unaffected. The next section introduces tools that are used to characterize a filter in the time and frequency domains, and that help 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 $y(t)$.

$$y(t) = \sum_{k=-N_1}^{N_2} h_k \cdot x(t-k)$$

It can be shown that the impulse response $h$ is identical to the filter weights in nonrecursive filters, but not for 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 $Y(f)$. The 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 do-

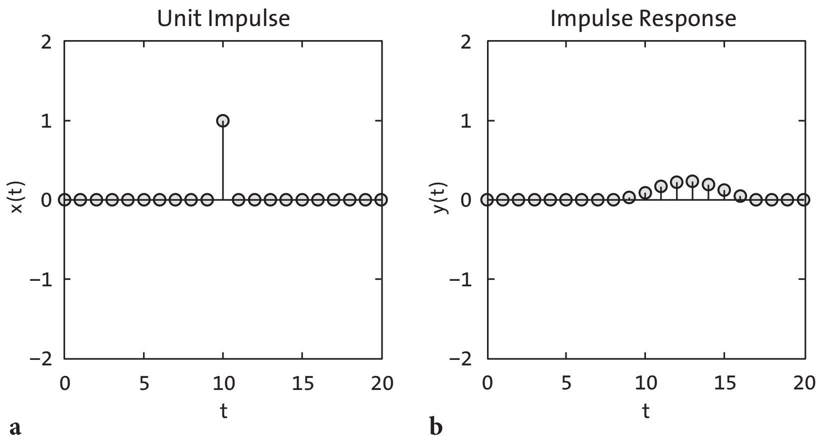


**Fig. 6.3** Transformation of **a** a unit impulse to compute **b** the impulse response of a system. The impulse response is often used to describe and predict the performance of a filter.

main 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 might 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. 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);
```

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 equal the filter weights of `a6`, in our example 0.2 for all elements of `a6` and `y6`.

For a recursive filter, however, the output `y6` does not match with the filter weights. Once again, an impulse is generated first of all.

```
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.

## 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 $h(f)$. The absolute value of the complex frequency response $H(f)$ is the magnitude response of the filter $A(f)$.

$$A(f) = \left|H(f)\right|$$

The argument of the complex frequency response $H(f)$ is the phase response of the filter.

$$\Phi(f) = \arg\left(H(f)\right)$$

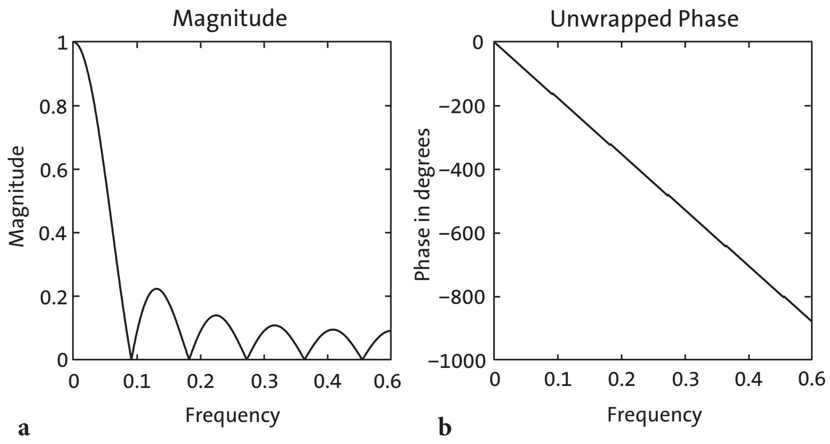


**Fig. 6.4 a** Magnitude and **b** phase response of a running mean over eleven elements.

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);
```

A running mean over eleven elements is designed and this filter applied to the input signal.

```
b8 = ones(1,11)/11;
m8 = length(b8);

y8 = filter(b8,1,x8);
```

The phase 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 has an amplitude of 2, the output 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 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 for the phase shift of the function `filter` yields

```
b9 = ones(1,11)/11;
m9 = length(b9);

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 is again in phase with the input, 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
```

The running mean over eleven elements reduces the amplitude of this signal by 67 %. 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 to the true frequency values. The true frequency values 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 of a signal's amplitude for a specific frequency. As an example, the interpolation of `magnitude` for a frequency of 1/20

```
1-interp1(f,magnitude,1/20)

ans =
    0.4260
```

results in the expected ca. 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., around 68 % similar to the value observed previously. 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 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 is plotted in degrees. We observe frequent 180° jumps in this plot that are an artifact of the function `arctangent` within the function `angle`. We can unwrap the phase response to eliminate the 180° jumps using the function `unwrap`.

```
plot(f,unwrap(phase))
xlabel('Frequency'), ylabel('Phase in degrees')
title('Phase')
```

Since the filter has a linear phase response, 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 can be computed using

```
interp1(f,unwrap(phase),1/20) * 20/360

ans =
   -5.0000
```

and

```
interp1(f,unwrap(phase),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 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;

f = 1*w/(2*pi);

plot(f,unwrap(phase))
xlabel('Frequency'), ylabel('Phase in degrees')
title('Magnitude Response')

interp1(f,unwrap(phase),1/8) * 8/360
```

```
ans =
    -5.0144
```

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 –5.0144. This suggests a corrected phase shift of –3.0144, 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 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 frequency, which in our example is 0.5, i. e., half of the sampling frequency.

```
[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.

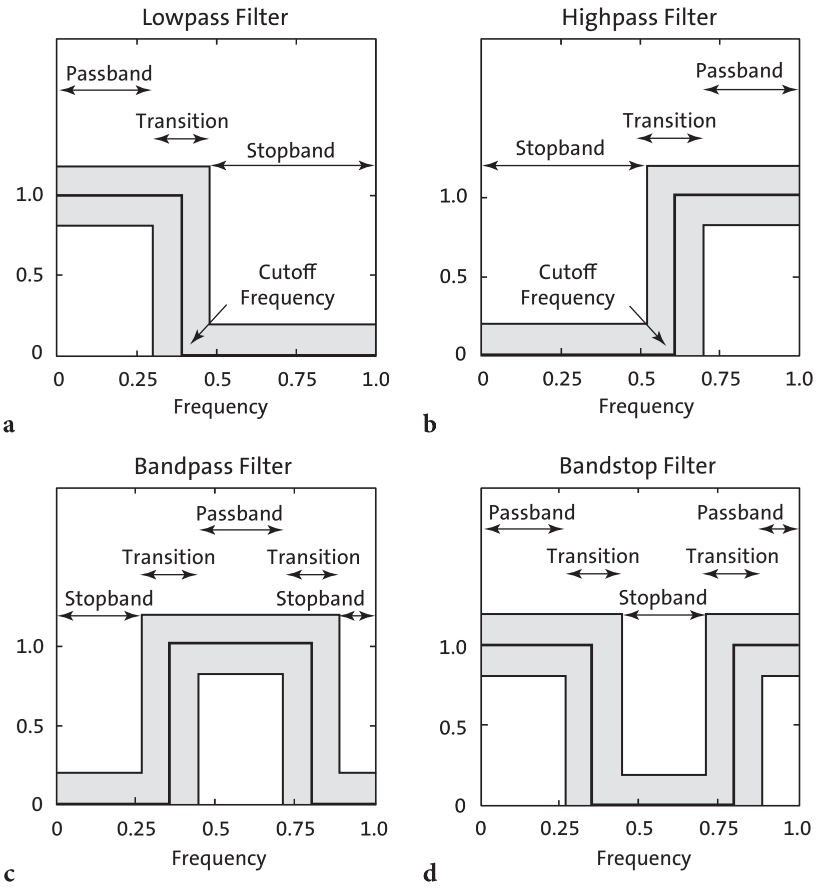


**Fig. 6.5** Frequency responses for the fundamental types of frequency-selective filters. **a** Lowpass filter to suppress the high-frequency component of a signal. In earth sciences, such filters are often used to suppress high-frequency noise in a low-frequency signal. **b** Highpass filters are employed to remove all low frequencies and trends in natural data. **c–d** Bandpass and bandstop filters extract or suppress a certain frequency band. The solid line in all graphs 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.

```
[h,w] = freqz(b12,a12,1024);
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`.

However, 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 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])
```

One 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. 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;
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.

```
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])
```

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. In most applications, however, an *a priori* knowledge of the signal and noise statistical characteristics is not usually available. 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 the 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 example for 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).

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 to 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 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 required 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 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 relation 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 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.

In the following 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 containing the same sine wave, but different Gaussian noise.

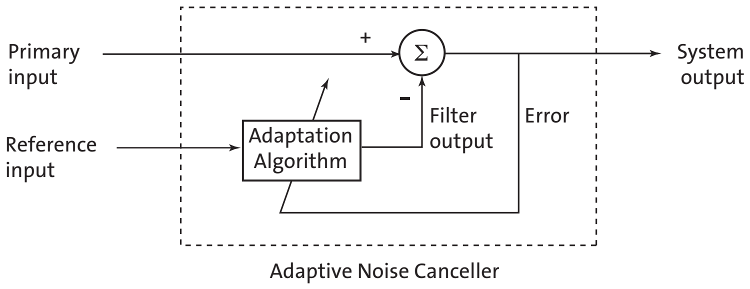


**Fig. 6.6** Schematic of an adaptive filter. Each iteration involves a new estimate of the filter weights $W_{i+1}$ based on the previous set of filter weights $W_i$ plus a term which is the product of a bounded step size $u$, a function of the filter input $X_i$, and a function of the error $e_i$. 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).

```
clear

x = 0 : 0.1 : 100;
y = sin(x);
yn1 = y + 0.2*randn(size(y));
yn2 = y + 0.2*randn(size(y));

plot(x,yn1,x,yn2)
```

We then save the following code in a text file *canc.m* and include it into the search path. 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.

```
function [zz,ee,mer] = canc(x,s,u,l,iter)
% CANC Correlated Adaptive Noise Canceling
[n1,n2] = size(s); n = n2; index = 0;       % Formatting
if n1 > n2
    s = s'; x = x'; n = n1; index = 1;
end
w(1:l) = zeros(1,l); e(1:n) = zeros(1,n);   % Initialization
xx(1:l) = zeros(1,l); ss(1:l) = zeros(1,l);
z(1:n) = zeros(1,n); y(1:n) = zeros(1,n);
ors = s; ms(1:n) = mean(s) .* ones(size(1:n));
s = s - ms; x = x - ms; ors = ors - ms;
for it = 1 : iter                           % Iterations
    for I = (l+1) : (n+1)                   % Filter loop
        for k = 1 : l
            xx(k) = x(I-k); ss(k) = s(I-k);
        end
        for J = 1 : l
```

```
                y(I-1) = y(I-1) + w(J) .* xx(J);
                z(I-1) = z(I-1) + w(J) .* ss(J);
            end
                e(I-1) = ors(I-1-(fix(l/2)))-y(I-1);
            for J = 1 : l
                w(J) = w(J) + 2.*u.*e(I-1).*xx(J);
            end
        end                                   % End filter loop
        for I = 1 : n                         % Phase correction
            if I <= fix(l/2)
                yy(I) = 0; zz(I) = 0; ee(I) = 0;
            elseif I > n-fix(l/2)
                yy(I) = 0; zz(I) = 0; ee(I) = 0;
            else
                yy(I) = y(I+fix(l/2));
                zz(I) = z(I+fix(l/2));
                ee(I) = abs(e(I+fix(l/2)));
            end
                yy(I) = yy(I) + ms(I);
                zz(I) = zz(I) + ms(I);
        end                                   % End phase
    correction
        y(1:n) = zeros(size(1:n));
        z(1:n) = zeros(size(1:n));
        mer(it) = mean(ee((fix(l/2)):(n-fix(l/2))).^2);
    end                                       % End iterations
    if index == 1                             % Reformatting
        zz = zz'; yy = yy'; ee = ee';
    end
```

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 $0 < u < 1/\lambda_{max}$ where $\lambda_{max}$ 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 by

```
k = kron(yn1,yn1');
u = 1/max(eig(k))
```

which yields

```
u =
    0.0018
```

We now run the adaptive filter `canc` for 20 iterations and use the above value of `u`.

```
[z,e,mer] = canc(yn1,yn2,0.0019,5,20);
```

The plot of the mean-squared error `mer` versus the number of performed

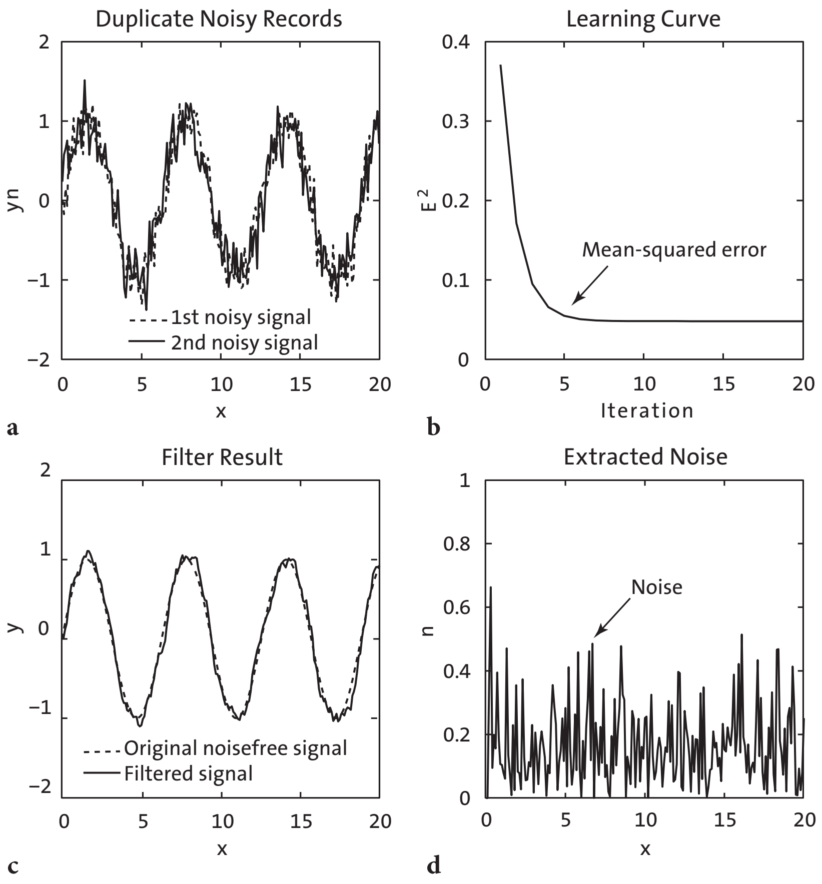


**Fig. 6.7** Output of the adaptive filter. **a** The duplicate records corrupted by uncorrelated noise are fed into the adaptive filter with 5 weights with a convergence factor of 0.0019. After 20 iterations, the filter yields the **b** learning curve, **c** the noisefree record and **d** the noise extracted from the duplicate records.

iterations `it` with stepsize `u`

```
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 increases the quality of the final result. We therefore reduce `u` by one order of magnitude and run the filter again with further iterations.

```
[z,e,mer] = canc(yn1,yn2,0.0001,5,20);
```

The plot of the mean-squared error against the iterations

```
plot(mer)
```

now converges after about six iterations. 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. Finally, the plot

```
plot(x,e,'r')
```

shows the noise extracted from the signal. In practice, the user should vary the parameters `u` and `l` to obtain the optimum result.

The application of this algorithm has been demonstrated on duplicate oxygen-isotope records from ocean sediments (Trauth 1998). This work by M. 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 (Haykin 1991) in various environments.

## 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
Sibul LH (1987) Adaptive Signal Processing. IEEE Press
The Mathworks (2010) Signal Processing Toolbox – User's Guide. The MathWorks, Natick, MA
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)

# 7 Spatial Data

## 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 (ca. 8 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 (ca. 4 km), and just recently the ETOPO1 became available, which has a resolution of 1 arc minutes (ca. 2 km). In addition, there is a data set for topography called GTOPO30 completed in 1996 that has a horizontal grid spacing of 30 arc seconds (ca. 1 km). Most recently, the 30 and 90 m resolution data from the Shuttle Radar Topography Mission (SRTM) have replaced the older data sets in most scientific studies.

The second part of this chapter deals with the estimation of continuous surfaces from unevenly-spaced data, and statistics of spatial data (Sections 7.6 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 7.10).

This chapter requires the Mapping Toolbox although most graphics routines used in our examples can be easily replaced by standard MATLAB functions. 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 also requires a computing system with at least 2 GB physical memory.

## 7.2 The GSHHS Shoreline Data Set

The Global Self-consistent, Hierarchical, High-resolution Shoreline (GSHHS) 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). The shoreline vector data can be downloaded as MATLAB vector data from the web page of the US National Geophysical Data Center (NGDC):

```
http://rimmer.ngdc.noaa.gov/mgg/coast/getcoast.html
```

We first need to define the geographic range of interest in decimal degrees, with west and south denoted by a negative sign. For example, the north-east coast of African would be displayed between the latitudes of 0 and +15 degrees and between the longitudes of +35 and +55 degrees. We then need to choose the coastline data base from which the data is to be extracted. As an example, the *World Data Bank II* provides maps at the scale 1:2,000,000. Finally, the compression method is set to *None* for the ASCII data and the coast format option is set to MATLAB. The resulting Generic Mapping Tool (GMT) map and a link to the raw text data can be displayed by pressing the *Submit-Extract* button at the end of the web page. By opening the 350 KB text file on a browser, the data can be saved onto a new file called *coastline.txt*. 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),'k'), axis equal
```

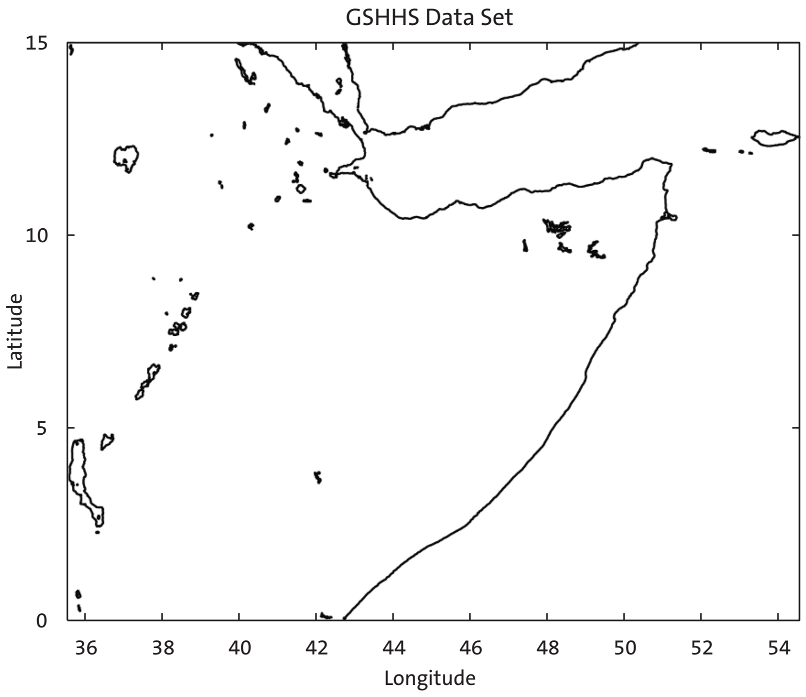


**Fig. 7.1** Display of the GSHHS shoreline data set. The map shows an area between latitudes 0° and 15° north, and longitudes 40° and 50° east. This simple map is made using the function `plot` with equal axis aspect ratios.

```
xlabel('Longitude'), ylabel('Latitude')
```

More advanced plotting functions are contained in the Mapping Toolbox, which allows 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),'k');
```

Note that the input for `plotm` is given in the order *longitude*, followed by the *latitude*, i.e., the second column of the data matrix is entered first. In contrast, the function `plot` requires an *xy* input, i.e., the first column is entered first. The function `axesm` defines the map axis and sets various map properties such as the map projection, the map limits and the axis labels.

## 7.3 The 2-Minute Gridded Global Relief Data ETOPO2

ETOPO2 is a global data base of topography and bathymetry on a regular 2-minute grid (approximately 4 km). It 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/fliers/01mgg04.html
```

From the menu bar *Free On-line* we select *custom grids* which is linked to the *GEODAS Grid Translator*. First, we choose a Grid ID (e.g., *grid01*), the Grid Database (e.g., *ETOPO2 2-minute Global Relief Ver 2*), our computer system (e.g., *Macintosh*) and the Grid Format (e.g., *ASCII*) for both the data and the header. Next we define the *latitude* and *longitude* bounds; for example, the latitude (*lat*) from 20 S to 20 N and a longitude (*lon*) between 30 E and 60 E corresponds to the East African coast. The selected area can be transformed into a digital elevation matrix by pressing *Submit*. This matrix may be downloaded from the web page by pressing *Compress and Retrieve Your Grid* followed by *Retrieve compressed file* in the subsequent windows. Decompressing the file *grid01.tgz* creates a directory *grid01_data*, which contains various data and help files. The subdirectory *grid01* contains the ASCII raster grid file *grid01.asc* that has the following content:

```
NCOLS   901
NROWS  1201
```

```
XLLCORNER   30.00000
YLLCORNER  -20.00000
CELLSIZE 0.03333333
NODATA_VALUE  -32768
299      286      285      273      267      260 ...
298      279      282      273      263      254 ...
284      272      275      274      266      260 ...
267      267      269      270      272      269 ...
254      267      268      268      264      258 ...
(cont'd)
```

The headers document the size of the data matrix (e. g., 901 columns and

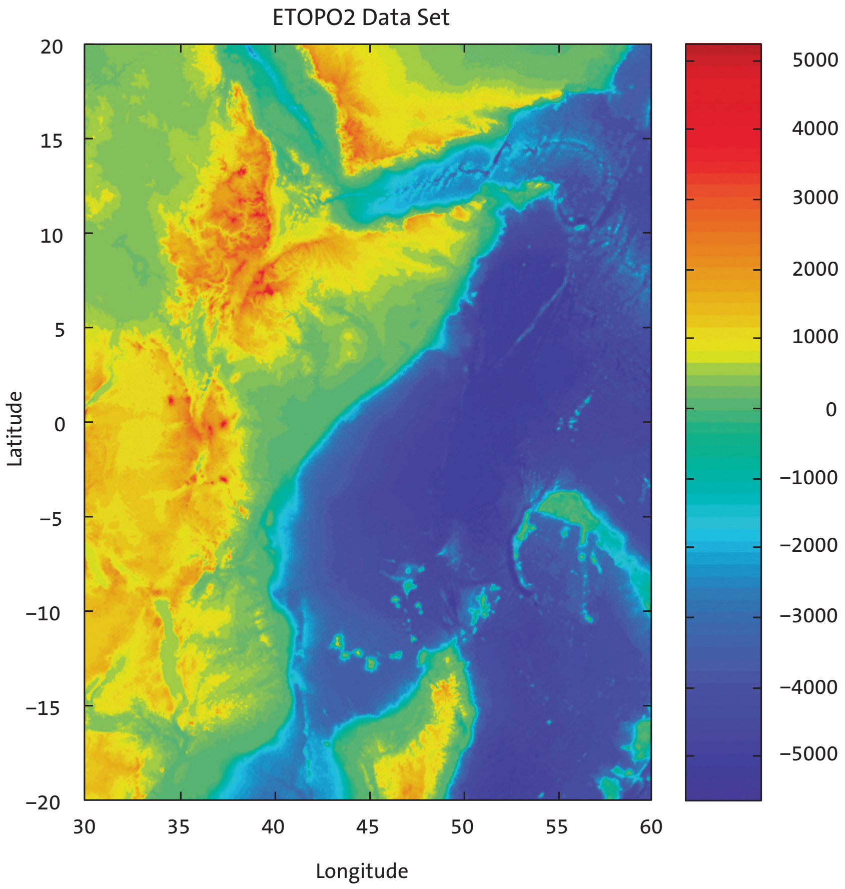


**Fig. 7.2** Display of the ETOPO2 elevation data set. The map uses the function `surf` to generate a colored surface. The colorbar provides information on the colormap used to portray the topographic and bathymetric variations.

1201 rows in our example), the coordinates of the lower-left corner (e. g., $x$=30 and $y$=–20), the cell size (e. g., 0.033333 = 1/30 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    901
%NROWS   1201
%XLLCORNER  30.00000
%YLLCORNER -20.00000
%CELLSIZE 0.03333333
%NODATA_VALUE  -32768
299    286    285    273    267    260 ...
298    279    282    273    263    254 ...
284    272    275    274    266    260 ...
267    267    269    270    272    269 ...
254    267    268    268    264    258 ...
(cont'd)
```

and load the data into the workspace.

```
clear

ETOPO2 = load('grid01.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`.

```
ETOPO2 = flipud(ETOPO2);
ETOPO2(find(ETOPO2 == -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(ETOPO2(:))
min(ETOPO2(:))
```

In this example, the maximum elevation for the area is 5,149 m and the minimum elevation is –5,656 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 2 arc minutes corresponding to 1/30 degree.

```
[LON,LAT] = meshgrid(30:1/30:60,-20:1/30:20);
```

We now generate a colored surface from the elevation data using the function `surf`.

```
surf(LON,LAT,ETOPO2)
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 in an overhead view (Fig. 7.2). Additional display methods will be described in the section on SRTM elevation data.

## 7.4 The 30-Arc Seconds Elevation Model GTOPO30

The 30 arc second (approximately 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 web page

```
http://eros.usgs.gov/#/Find_Data/Products_and_Data_Available/
   gtopo30_info
```

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 dimension of 1,200 × 1,200 cells. The coordinate system is defined by using the *lon/lat* limits as listed above. The resolution is 30 arc seconds corresponding to 1/120 degrees.

```
[LON,LAT] = meshgrid(30:1/120:40-1/120,-5:1/120:5-1/120);
```

We need to reduce the limits by 1/120 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 used to darken the map at higher levels of elevation, and the colormap is then 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
```

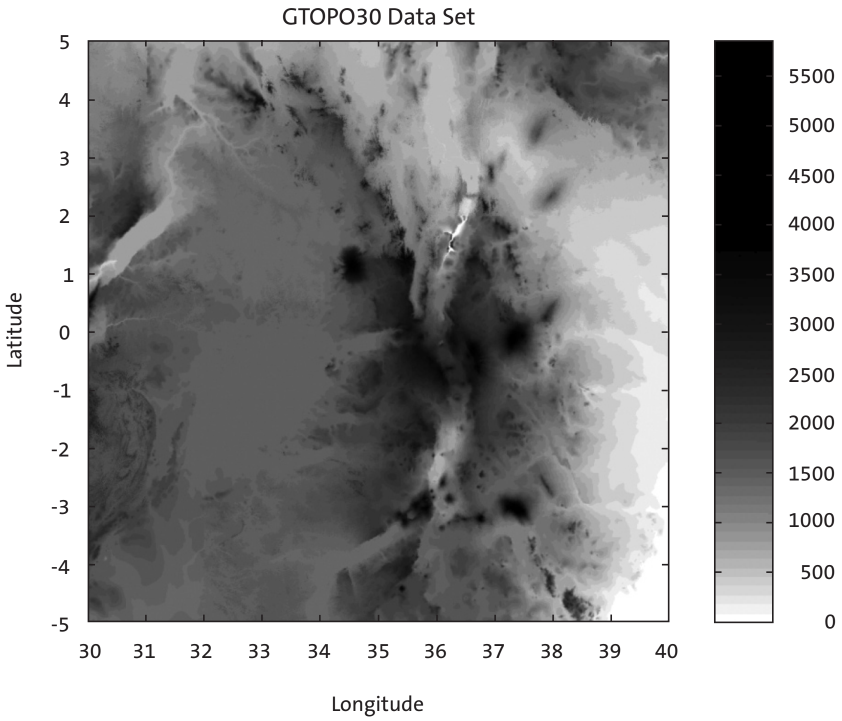


**Fig. 7.3** Display of the GTOPO30 elevation data set. The map uses the function `surf` to generate a gray surface. We use the colormap `gray` to the power of four in order to darken the colormap with respect to the higher elevation. In addition, we flip the colormap in up/down directions using `flipud` to obtain dark colors for high altitudes and light colors for low elevations.

This script opens a new figure window and generates the gray surface using interpolated shading in an overhead view (Fig. 7.3).

## 7.5 The Shuttle Radar Topography Mission SRTM

The Shuttle Radar Topography Mission (SRTM) consists of a radar system that flew onboard the Space Shuttle *Endeavour* during an 11-day mission in February 2000. 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) EROS Data Center using the Seamless Data Distribution System.

```
http://seamless.usgs.gov/
```

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 large 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. Bit order is Motorola (*big-endian*) standard with the most significant bit first. The data are imported into the workspace using

```
clear

fid = fopen('S01E036.hgt','r');
SRTM = fread(fid,[1201,inf],'int16','b');
fclose(fid);
```

This script opens the file *s01e036.hgt* for read access using `fopen` and defines the file identifier `fid`, which is then used for reading the binaries from the file using `fread`, and writing it into the matrix `SRTM`. Function `fclose` closes the file defined by `fid`. First, the matrix 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 filling gaps with the arithmetic means of adjacent elements. We use the function `nanmean` since it treats NaNs as missing values and returns the mean of the remaining elements that are not NaNs. 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
```

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
```

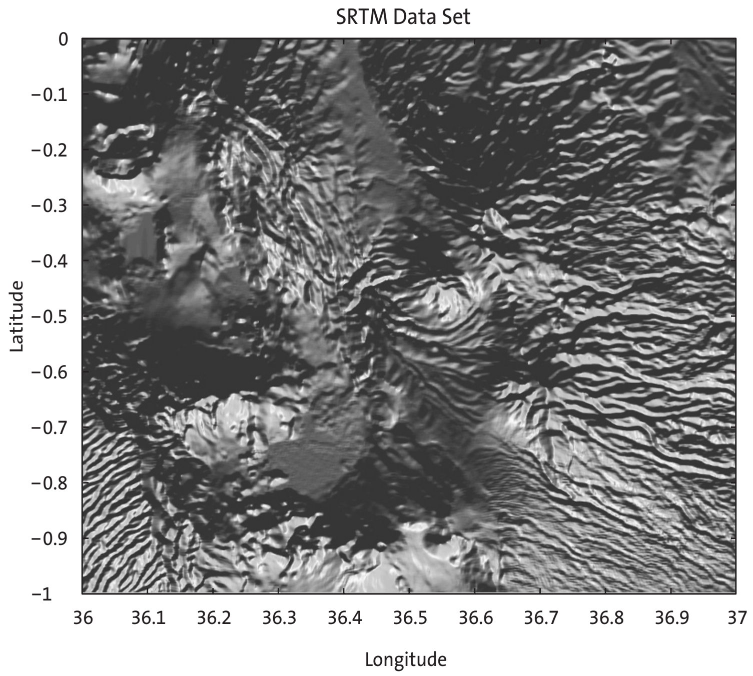


**Fig. 7.4** Display of the filtered SRTM elevation data set. The map uses the function `surfl` to generate a shaded-relief map with simulated lighting using interpolated shading and a gray colormap in an overhead view. Note that the SRTM data set contains a lot of gaps, in particular in the lake areas.

```
min(SRTM(:))

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 by using the information that the lower-left corner is *s01e036*. The resolution is 3 arc seconds corresponding to 1/1,200 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 in an overhead view. Since SRTM data contain a large amount of noise, we first smooth the data using an arbitrary $9 \times 9$ 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.

```
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.

## 7.6 Gridding and Contouring Background

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. Firstly, the number of *control points* needs to be selected, and secondly, the value of the variable of interest 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. As an 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 (Fig. 7.5). 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. 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, too small radii 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.

It is perhaps due to these difficulties that *triangulation* is often used as an alternative method for selecting the control points (Fig. 7.5b). In this

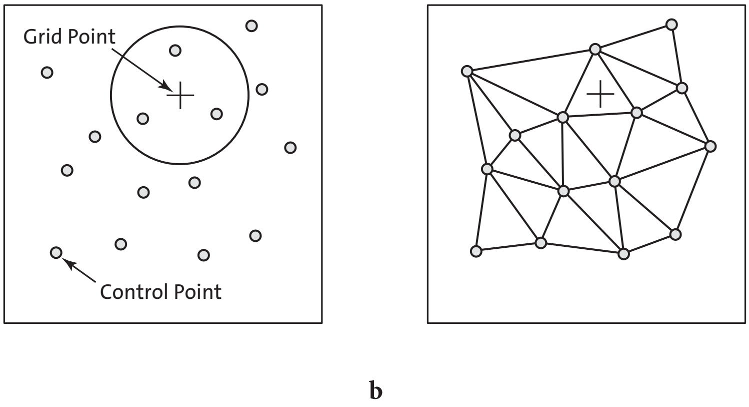


**Fig. 7.5** Methods of selecting the control points to use for estimating the values at the grid points. **a** Construction of a circle around the grid point (plus sign) with a radius defined by spatial autocorrelation of the *z*-values at the control points (small circles). **b** Triangulation: the 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.

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 selecting control points. It is often regarded as *the* 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

$$\bar{z} = \frac{1}{N}\sum_{i=1}^{N} z_i$$

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 by 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.

$$\bar{z} = \frac{\sum_{i=1}^{N}(z_i / d_i)}{\sum_{i=1}^{N}(1/d_i)}$$

Depending on the spatial scaling relationship of the parameter $z$, 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, which 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 least six adjacent control points. The final surface consists of a composite of portions of these splines. MATLAB also provides interpolation with biharmonic splines generating very smooth surfaces (Sandwell, 1987).

## 7.7 Gridding Example

MATLAB has, from the start, provided a biharmonic spline interpolation method that 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 by using the `plot` command. Secondly, the data are labeled by 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
```

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 grid intervals.

```
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);
```

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 by 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. Herein, `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*

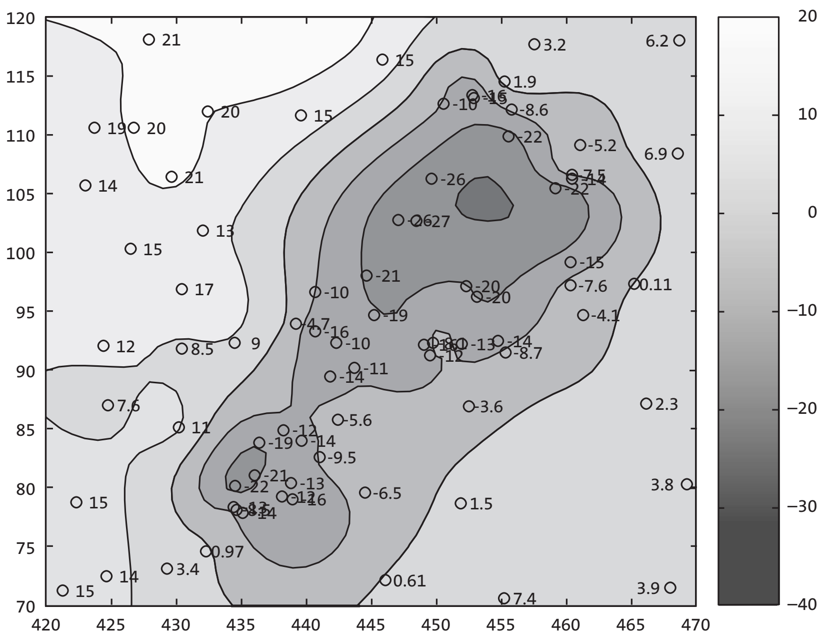


**Fig. 7.6** Contour plot with the locations (small circles) and $z$-values (contour labels) of the true data points.

menu. We also introduce the function `colormap`, which uses predefined color look-up tables for 3D graphs. 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-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).

```
[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
```

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

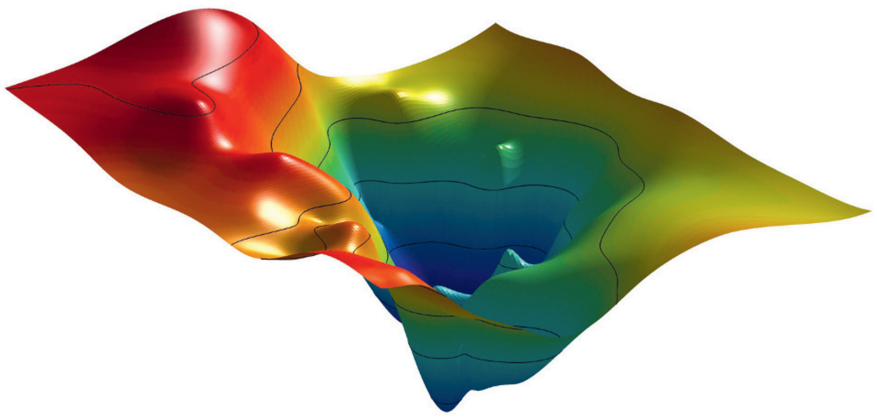


**Fig. 7.7** Three-dimensional colored surface with interpolated shading and simulated lighting. The axis labeling, ticks and background are turned off. The plot also contains 3D contours, in black.

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.

```
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. In many applications, this solution is very useful, but the method also produces a number of artifacts. As we can see from the next plot, the estimated values at the grid 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 employ 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
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.

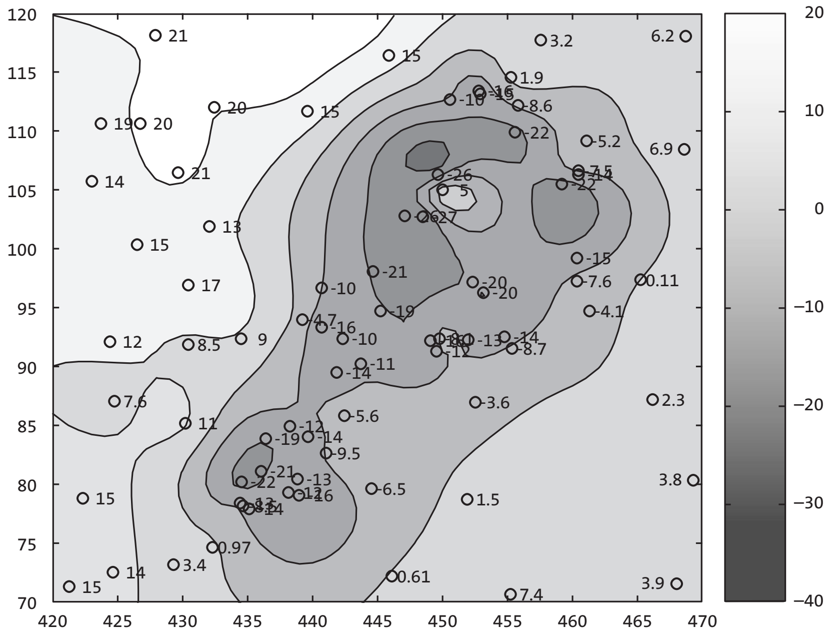


**Fig. 7.8** Contour plot of a data set gridded using a biharmonic spline interpolation. At the location (450,105), very close control points with different $z$ values have been introduced. Interpolation causes a paired low and high, which is a common artefact in spline interpolation of noisy data.

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')
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
```

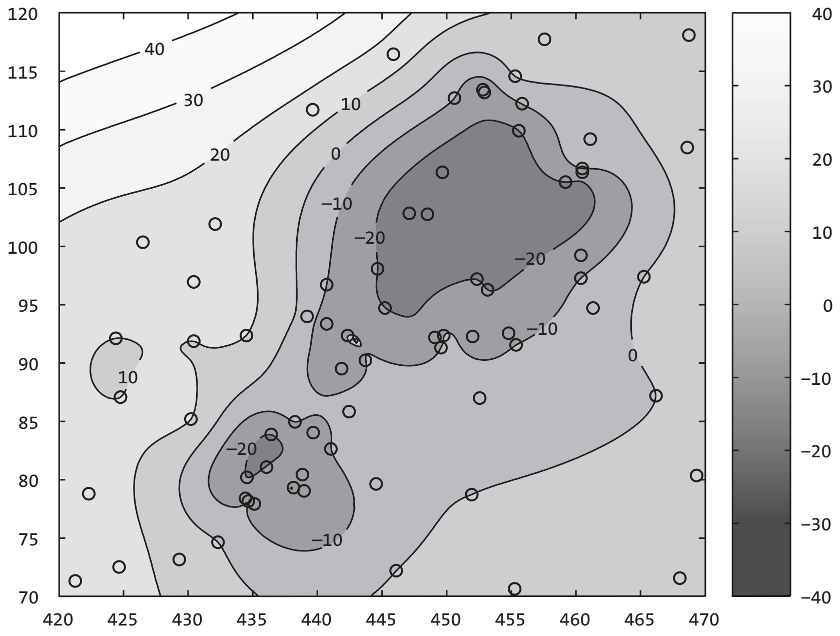


**Fig. 7.9** Contour plot of a data set gridded using a biharmonic spline interpolation. No control points are available in the upper left corner. The spline interpolation then beyond the area with control points using gradients at the map edges resulting in unrealistic $z$ estimates at the grid points.

```
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 on this part of the map.

Another very useful MATLAB gridding method is *splines with tension* by Wessel and Bercovici (1998). 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.

## 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.

The rest of the section introduces methods for the statistical analysis of point distributions. We first consider a test for a uniform spatial distribution of objects, followed by a test for a random spatial distribution. Finally, a simple test for clustered distributions of objects is presented.

### 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 unit interval. We compute *xy* data using `rand` and multiply the data by ten to obtain data within the interval [0,10].

```
clear

rand('seed',0)
data = 10 * rand(100,2);
```

We can use the $\chi^2$-test introduced in Section 3.8 to test the hypothesis that the data have a uniform distribution. The *xy* data are now organized in 25 classes that are square subareas of the dimension 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 empty classes, which should be avoided when applying the $\chi^2$-test. Furthermore, 25 classes produce integer values for the expected number of observations that are more easy 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
hold off
```

A three-dimensional version of a histogram `hist3` is used to display the

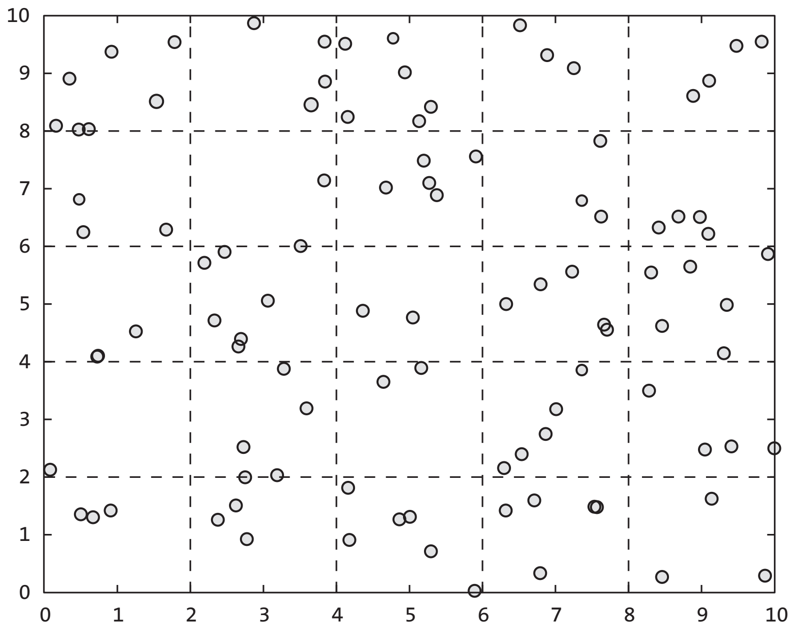


**Fig. 7.10** Two-dimensional plot of a point distribution. The distribution of objects in the field is tested for uniform distribution using the χ2–test. The *xy* data are now organized in 25 classes that are subareas of the size 2-by-2.

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 $10 \times 10 = 100$ divided by the 25 subareas or classes, which comes to be 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 four objects.

```
n_exp = 4 * ones(25,1);
```

The $\chi^2$-test explores the squared differences between the observed and ex-

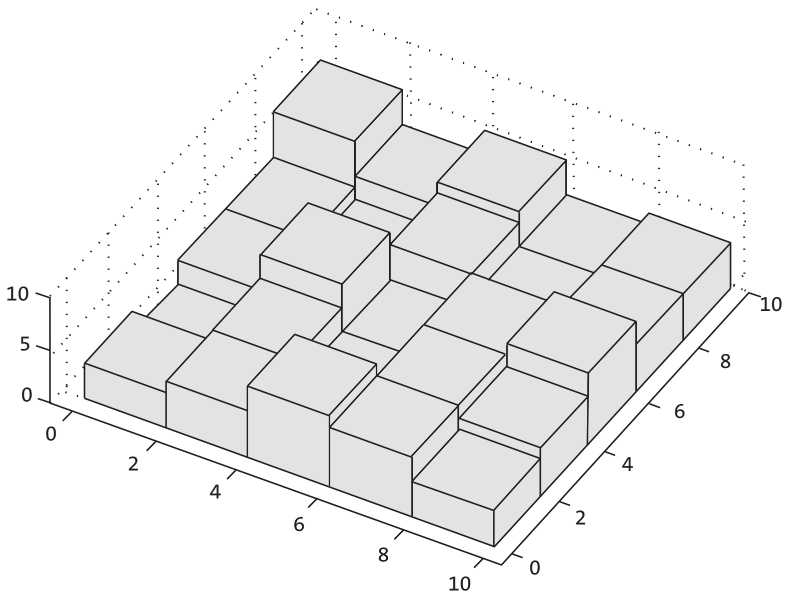


**Fig. 7.11** Three-dimensional histogram displaying the numbers of objects for each subarea. The histogram was created using `hist3`.

pected frequencies (Section 3.8). The quantity $\chi^2$ 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 =
    14
```

The critical $\chi^2$ can be calculated by using `chi2inv`. The $\chi^2$-test requires the degrees of freedom $\Phi$. In our example, we test the hypothesis that the data are uniformly distributed, i.e., we estimate only one parameter (Section 3.4). The number of degrees of freedom is therefore $\Phi=25-(1+1)=23$. We test the hypothesis at a $p=95\,\%$ significance level. The function `chi2inv` computes the inverse of the $\chi^2$ CDF with parameters specified by $\Phi$ for the corresponding probabilities in $p$.

```
chi2_theo = chi2inv(0.95,25-1-1)

ans =
    35.1725
```

Since the critical $\chi^2$ of 35.1725 is well above the measured $\chi^2$ of 14, we cannot reject the null hypothesis and conclude that our data follow a uniform distribution.

### 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

rand('seed',0)
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 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 usually 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 3.4) 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 `hist` (Fig. 7.12).

```
N = 0 : 5;

[n_obs,v] = hist(counts,N);

hist(counts,N)
title('Histogram')
xlabel('Number of observations N')
ylabel('Subareas with N observations')
```

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

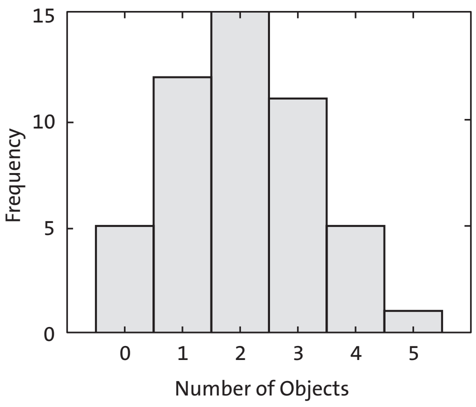


**Fig. 7.12** Frequency distribution of subareas with *N* objects. In our example, the subareas with 0, …, 5 objects are counted. The histogram of the frequency distribution is displayed as a two-dimensional histogram using `hist`.

$$E_j = Te^{-n/T} \frac{(n/T)^j}{j!}$$

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 theoretical frequency distribution 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 $\chi^2$-test is again employed to compare the empirical and theoretical distributions. The test is performed at a $p$=95% significance level. Since the Poisson distribution is defined by only one parameter (Section 3.4), the number of degrees of freedom is $\Phi$=6–(1+1)=4. The measured $\chi^2$ of

```
chi2 = sum((n_obs - n_exp).^2 ./n_exp)

chi2 =
    1.4357
```

is well below the critical $\chi^2$, 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) 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 3.4), which is

$$Z = \frac{\bar{d} - \bar{\delta}}{s_e}$$

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
```

```
rand('seed',0)
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 9.4). 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)

observednearest =
    0.5171
```

In our example, the mean nearest distance `observednearest` comes to 0.5471. Next, we 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 =
    0.4940
```

In our example, the expected mean nearest-neighbor distance `expected-nearest` is 0.4940. Finally, we compute the standard error of the mean nearest-neighbor distance `se`

```
se = 0.26136/sqrt(length(data).^2/maparea)

se =
    0.0258
```

and the test statistic `Z`.

```
Z = (observednearest - expectednearest)/se
```

```
Z =
    0.8960
```

In our example, `Z` is 0.8960. 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 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

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 3×3 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. 7.15a).

The SRTM subset is now analyzed for slope and aspect. While we are working with DEMs on a regular grid, slope and aspect can be estimated using centered finite differences in a local 3×3 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:

$$z_x = \frac{z_{r,c-1} - z_{r,c+1}}{2h}$$

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 quantity slope is then calculated by

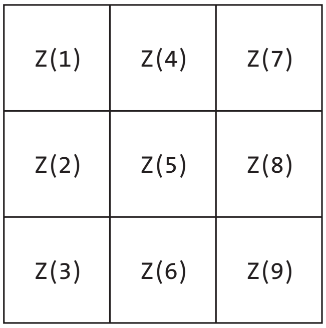


**Fig. 7.13** Local neighborhood showing the MATLAB cell number convention.

$$SLP_{DF} = \sqrt{{z_x}^2 + {z_y}^2}$$

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. 7.15b).

```
h = pcolor(slp);
colormap(jet), colorbar
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. 7.15c).

```
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 colors, whereas green areas depict east-facing slopes.

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 . 7.15d), according to its numerical label.

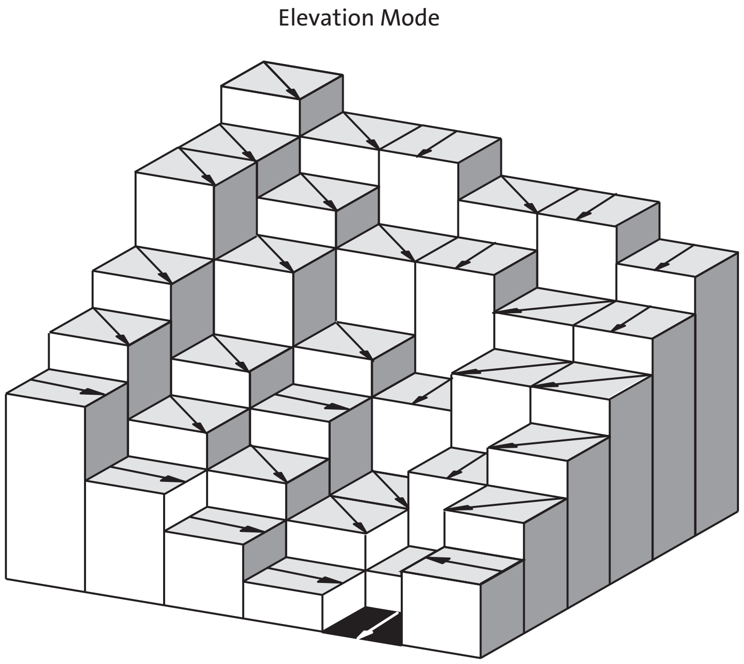
```
h = pcolor(watersh);
colormap(hsv), colorbar
set(h,'LineStyle','none')
axis equal
title('Watershed')
[r c] = size(watersh);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

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 three catchments areas (represented by green colors) with no outlets. As in this example, `watershed` often generates unrealistic results as 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 that has the value 1 corresponding to the elements of the DEM that belong to local minima and the value of 0 otherwise.

```
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 born in mind during any subsequent compu-



a

Flow Direction

| | | | | | |
|---|---|---|---|---|---|
| ↘ | ↘ | ↓ | ↘ | ↓ | ↓ |
| ↘ | ↘ | ↘ | ↓ | ↙ | ↓ |
| ↘ | ↘ | ↘ | ↓ | ↙ | ↙ |
| ↘ | ↘ | → | ↘ | ↓ | ↙ |
| ↘ | ↘ | ↘ | ↘ | ↓ | ↙ |
| → | → | → | → | ↓ | ← |

Flow Accumulation

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 3 | 1 | 3 | 2 |
| 1 | 2 | 3 | 8 | 1 | 3 |
| 1 | 2 | 3 | 16 | 4 | 1 |
| 1 | 2 | 3 | 1 | 22 | 1 |
| 1 | 3 | 6 | 10 | 36 | 1 |

b c

**Fig. 7.14** Schematic of calculation of flow accumulation by the D8 method.

tation 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, that contribute runoff to a particular cell (Fig. 7.14). In contrast to the local parameters slope and aspect, flow accumulation can only be determined from the global neighborhood. The principal operation is to add cell inflows from 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 an amount of rows and columns given by `r` and `c`, respectively. 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 (the so-called *rook's case*) and by

$$grad = \frac{z_{r+y,c+x} - z_{r,c}}{\sqrt{2h}}$$

for the diagonal neighbors (*bishop's case*). In these formulae, $h$ is the cell size, $z_{r,c}$ is the elevation of the central cell and $z_{r+y,c+x}$ the elevation of a neighbor. 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 gradients towards the neighboring cells to the east, `grads(:,:,2)` contains the gradients 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 by the inverse tangent of `grads` divided by $0.5\pi$.

```
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 (know as the D8 method, Fig. 7.14), which allows flow to only one of the cell's eight neighboring cells. This method cannot model divergence in ridge areas and tends to produce parallel flow lines in some situations. Here, we illustrate the use of a multiple-flow-direction method, which allows flow from a cell to multiple neighboring cells. The proportion of the total flow that is assigned to a neighboring cell is dependent on the gradient between the central cell and that particular neighboring cell, and is therefore a fraction of the total outflow. 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.

$$flow_i = \frac{grad_i^w}{\sum_{i=1}^{8} grad_i^w} \quad \text{for} \quad grad_i^w < 0$$

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 an 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 division 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

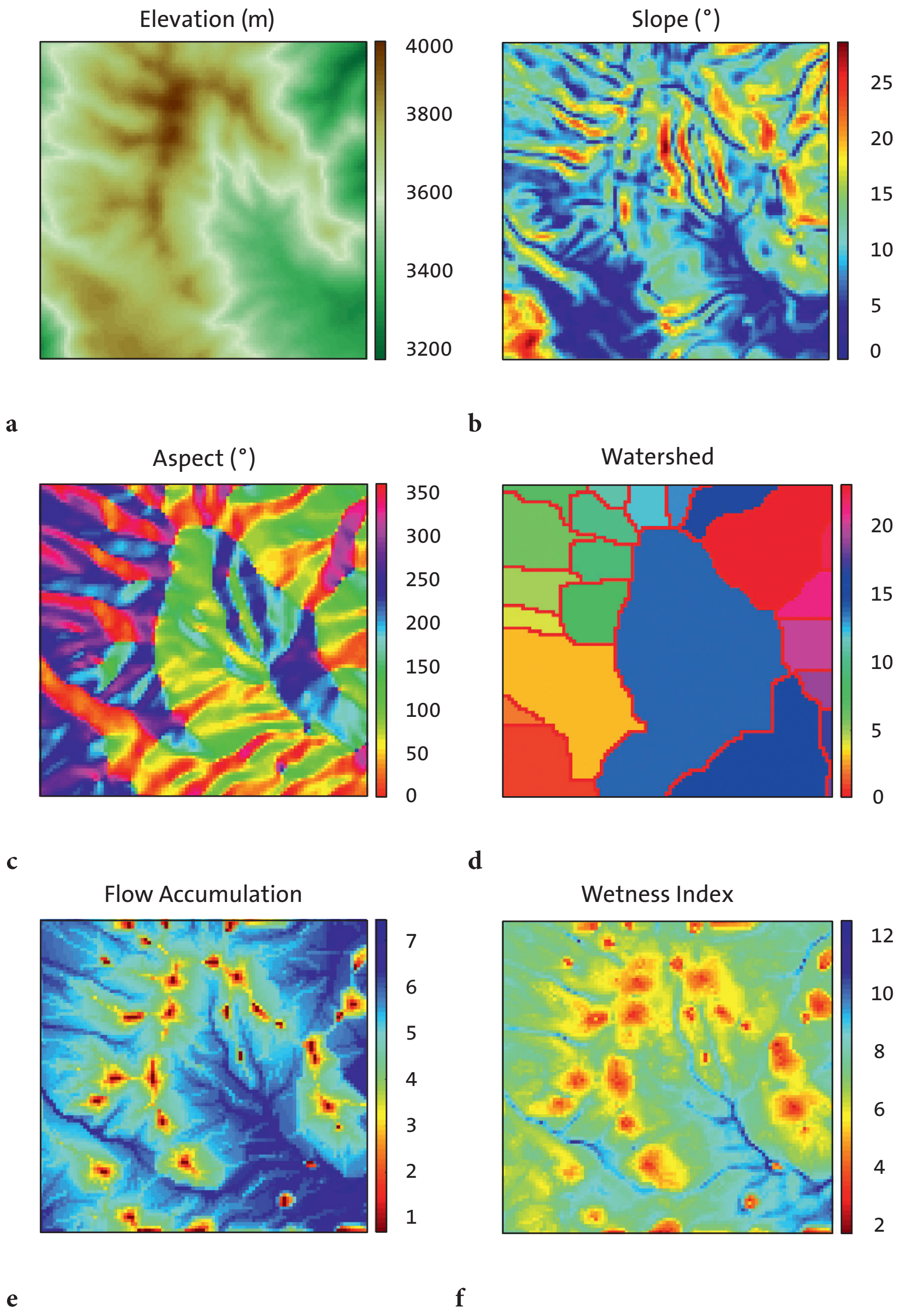


**Fig. 7.15** Display of a subset of the SRTM data set used in Section 7.5 and primary and secondary attributes of the digital elevation model; **a** elevation, **b** slope, **c** aspect, **d** watershed, **e** flow accumulation and **f** wetness index.

total of one. This is achieved separately for each layer of the 3D `flow` matrix by a `for` loop:

```
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 translated 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 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 summation 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. 7.15e).

```
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');
```

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. 7.15f). In our example, soils in the southeast are the most likely to have a high water content due to the runoff from the large central valley and the flatness of the terrain.

The *stream power index* is another important secondary relief attribute which 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 area of catchment 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, and 3D space, or even in 4D space-time, to make predictions for unobserved locations, to give information about the accuracy of the predictions and to reproduce spatial variability and uncertainty. The shape, range, and direction of the spatial autocorrelation are described by 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 as well 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 is composed of a purely random part and an autocorrelated part. The autocorrelated component implies that on average, closer observations are more similar to each other than more widely separated observations. This behavior is described by the *variogram* where squared differences between observations are plotted against their separation distances. Krige's fundamental idea was to use the variogram for interpolation, as 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 here 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

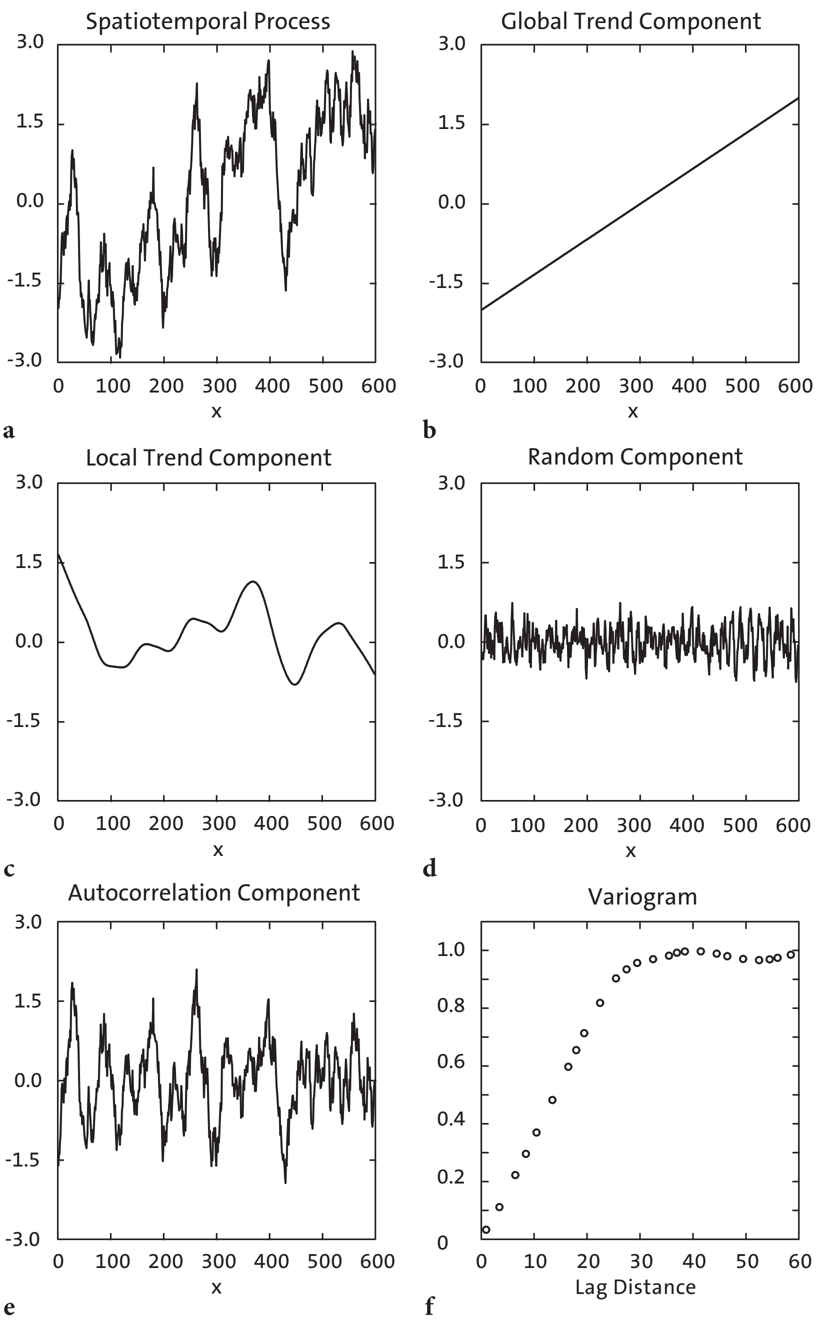


**Fig. 7.16** Components of a spatiotemporal process and the variogram. The variogram (**f**) should only be derived from the autocorrelated component.

*geost_dat.mat* containing *xyz* data triplets, and then plot the sampling locations. By doing this, we can check the point distribution and detect any major errors in the data coordinates, *x* and *y*.

```
clear

load geost_dat.mat

plot(x,y,'.')
```

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 ca. 100. One can also calculate the skewness and kurtosis of the data.

```
hist(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 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:

$$z^* = \begin{cases} \dfrac{(z+m)^k - 1}{k} & \text{for } k \neq 0 \\ \log(z+m) & \text{for } k = 0 \end{cases}$$

for min($z$)+$m$>0. This is the so 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 uni- 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 the 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 $h$. 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 of the difference between $z_x$ and $z_{x+h}$. It is, nevertheless, the variance per point when points are considered as in pairs (Webster and Oliver, 2001). Conventionally, $\gamma(h)$ is termed *variogram* instead of 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

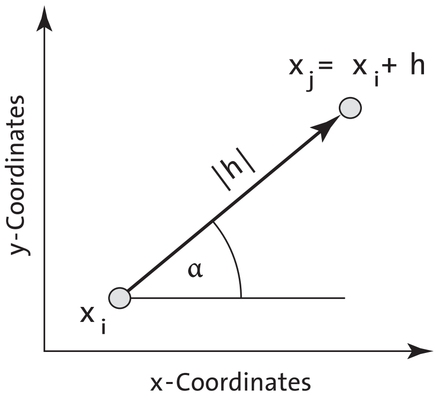


**Fig. 7.17** Separation vector $h$ between two points.

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);
```

We then get the experimental variogram `G` as half the squared differences between the observed values:

```
G = 0.5*(Z1 - Z2).^2;
```

In order to 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 $n^2$ 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 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.

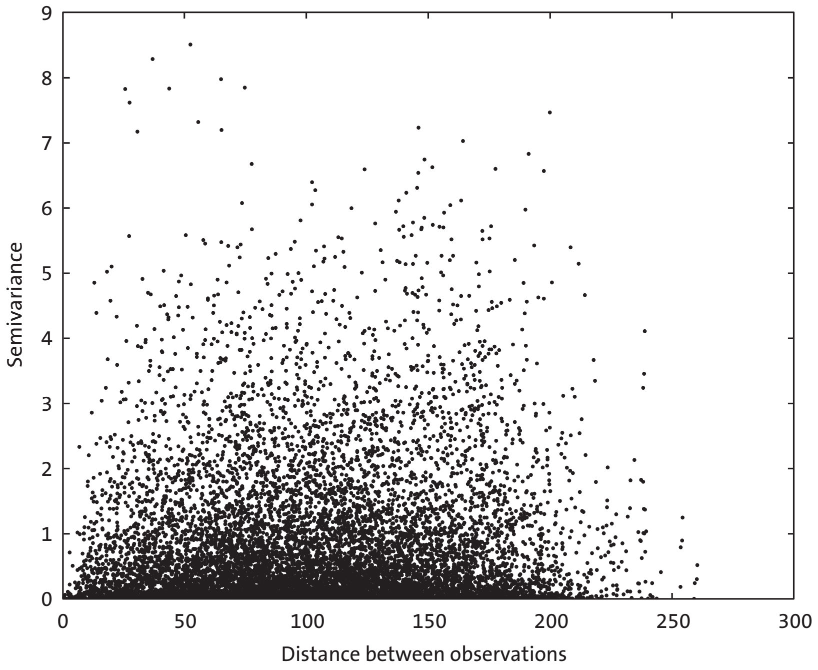


**Fig. 7.18** Variogram cloud: Plot of the experimental variogram (half the squared difference between pairs of observations) versus the lag distance (separation distance between the two components of a pair).

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:

```
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:

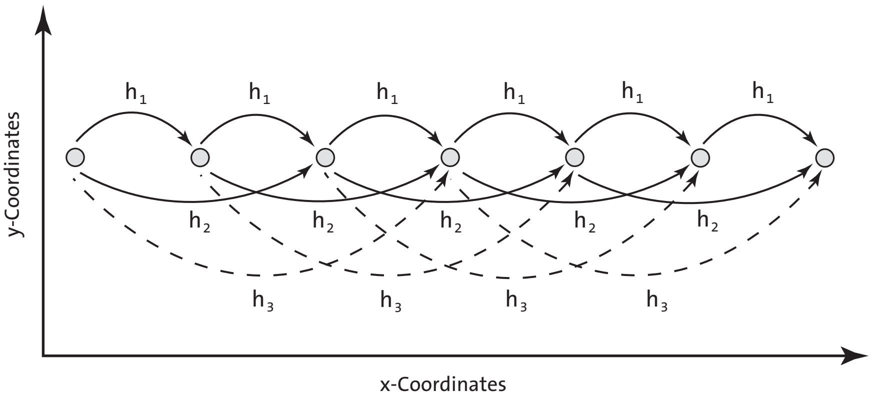


**Fig. 7.19** Classification of separation distances for observations that are equally spaced. The lag interval is $h_1$, and $h_2$, $h_3$ etc. are multiples of the lag interval.

```
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 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 distances until reaching a plateau (*sill*) which is close to the population variance. This indicates that the spatial process is correlated over short distances while 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),

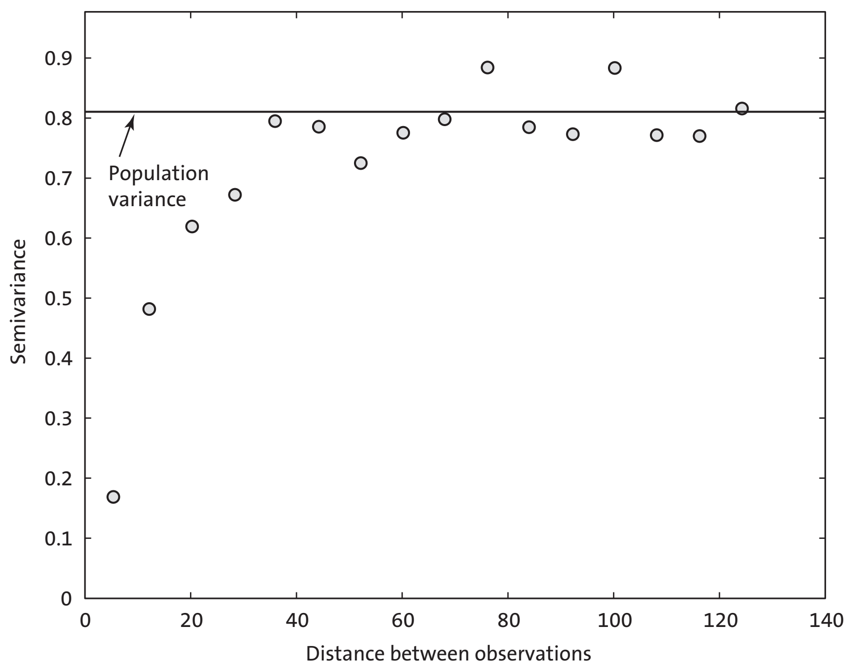


**Fig. 7.20** The classical variogram estimator (dots) and the population variance (dashed line).

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 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:

$$\gamma_{\exp}(h) = c \cdot \left(1 - \exp\left(-3 \cdot \frac{h}{a}\right)\right)$$

Linear model:

$$\gamma_{\mathrm{lin}}(h) = b \cdot h$$

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 so called *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*), which 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 previous knowledge. More details about 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:

$$\gamma_{\exp+\mathrm{nug}}(h) = c_0 + c \cdot \left(1 - \exp\left(-3 \cdot \frac{h}{a}\right)\right)$$

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 which should be interpreted as 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
```

The techniques of variogram modeling are 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, either procedure will yield similar results.

Another important question in variogram modeling is the intended use of the model. In our case, the linear model at first does not 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). Thus, different variogram models with similar fits close to the origin will yield similar kriging results when the sampling points are regularly distributed. If the objective is to describe the spatial

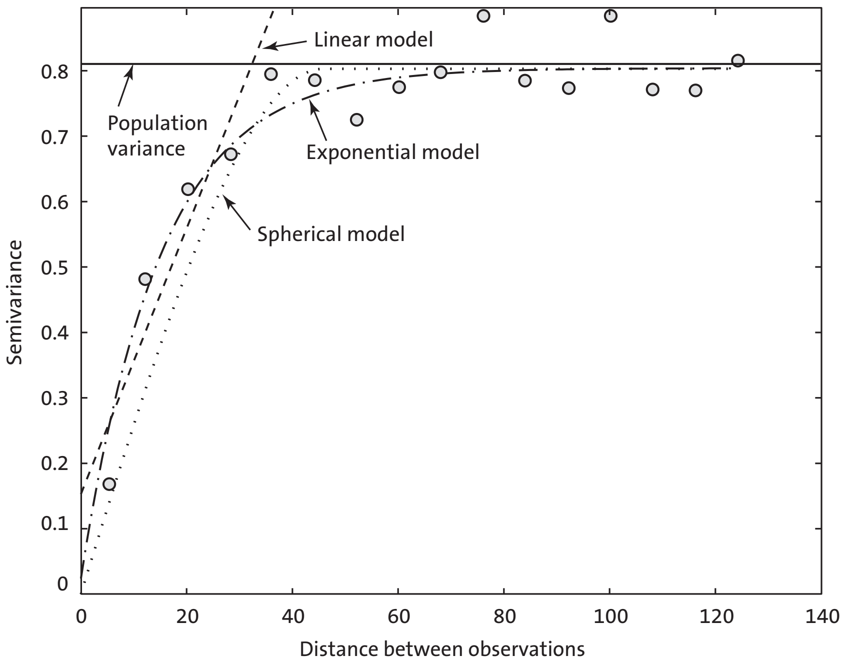


**Fig. 7.21** Variogram estimator (gray circles), population variance (solid line), and spherical, exponential, and linear models (dotted and dashed lines).

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 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 sometimes can 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 software VarioWin 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 smoothing, with a resulting loss of detail. The separation distance and the lag width therefore must 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 $\lambda_i$ 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 $\lambda_i$ and $\lambda_j$ are the weights of the $i$th and $j$th data point.

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 $v$ resulting in a *linear kriging system* of $N$+1 equations and $N$+1 unknowns:

$$\sum_{i=1}^{N} \lambda_i \gamma(x_i, x_j) + v = \gamma(x_i, x_0)$$

After obtaining the weights $\lambda_i$, the kriging variance is given by

$$\sigma^2(x_0) = \sum_{i=1}^{N} \lambda_i \gamma(x_i, x_0) + v(x_0)$$

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 $\gamma$ 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.

$$G\_R = \begin{bmatrix} \gamma(x_1,x_0) \\ \gamma(x_2,x_0) \\ \vdots \\ \gamma(x_N,x_0) \\ 1 \end{bmatrix}$$

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 with 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 ones 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;
```

Now the `G_mod` matrix 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* direction. 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);
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)`. Then we construct the right-hand-side vector of the kriging system by 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. First, we 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')
colorbar
```

The left hand subplot presents the kriging variance:

```
subplot(1,2,2)
h = pcolor(Xg1,Xg2,SK);
set(h,'LineStyle','none')
axis equal
ylim([0 200])
title('Kriging Variance')
xlabel('x-Coordinates')
ylabel('y-Coordinates')
colorbar
hold on
```

and we overlay overlaying the sampling positions:

```
plot(x,y,'ok')
hold off
```

The kriged values are shown in Figure 7.22a. The kriging variance depends only on the distance from the observations and not on the observed values (Fig. 7.22b). 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

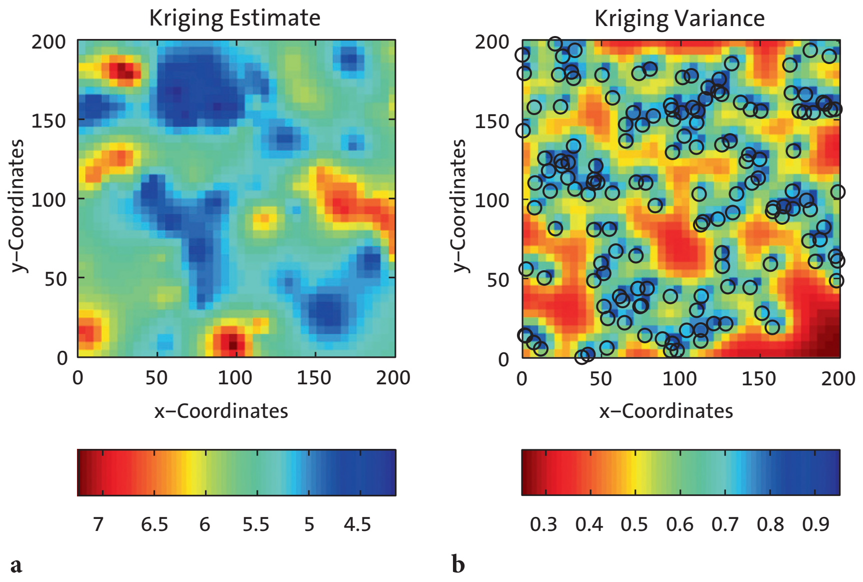


**Fig. 7.22** Interpolated values on a regular grid by ordinary point kriging using **a** an exponential variogram model; **b** kriging variance as a function of the distance from the observations (empty circles).

for lognormal kriging is achieved by:

$$y(x_0) = \exp(z(x_0) + 0.5 \cdot \sigma^2(x_0) - \nu)$$

## 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 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, Deutsch and Journel (1992) is essential reading. The best internet source is the homepage for AI-GEOSTATISTICS:

```
http://www.ai-geostats.org
```

## Recommended Reading

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

Freeman TG (1991) Calculating Catchment Area with Divergent Flow Based on a Regular Grid. Computers and Geosciences 17:413–422

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

Gringarten E, Deutsch CV (2001) Teacher's Aide Variogram Interpretation and Modeling. Mathematical Geology 33:507–534

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

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

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

The Mathworks (2010) Mapping Toolbox – User's Guide. The MathWorks, Natick, MA

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

Wilson JP, Gallant JC (2000) Terrain Analysis, Principles and Applications. John Wiley & Sons, New York

# 8 Image Processing

## 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 for the analysis of 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 use of a great variety of image processing methods.

This chapter is concerned with the analysis and display of image data. Firstly, the various ways that raster data can be stored on the computer are explained (Section 8.2). The main tools for importing, manipulating and exporting image data are presented in Section 8.3. This knowledge is then used to process and to georeference satellite images (Sections 8.4 and 8.5). On-screen digitization techniques are discussed in Section 8.6. Finally, sections 8.7 to 8.9 are concerned with color-intensity transects of laminated sediments, and with automated grain size analysis and charcoal quantification in microscope images. The Image Processing Toolbox is used for the specific examples throughout this chapter. While the MATLAB User's Guide to the Image Processing Toolbox provides an excellent general in-

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*, 3rd ed.,
DOI 10.1007/978-3-642-12762-5_8, © Springer-Verlag Berlin Heidelberg 2010

troduction 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
189 190 190 188 186 183
```

Raster data can be visualized as 3D plot. The $x$ and $y$ 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). Alternatively, 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, blue (RGB) color. An example is the above array that 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, one and 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. Hence, each element of the 2D array or pixel 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. Therefore, 8 bits are enough 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 give four numbers, three bits show 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:

```
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 one-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
```

Compared to the 1-bit and 8-bit representations of raster data, the 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 (first bit), the exponent (bits 2 to 12) and roughly 16 significant decimal digits in the range of approximately $10^{-308}$ to $10^{+308}$ (bits 13 to 64). However, MATLAB also works with other data types, such as 1-bit, 8-bit and 24-bit raster data, to save memory.

The memory required for storing a raster image depends on the data type and the image's dimension. The dimension of an image can be described by the numbers of pixels, which is the number of rows multiplied by the number of columns of the 2D array. Let us assume an image of $729 \times 713$ 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 $729 \times 713 \times 8 = 4{,}158{,}216$ 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) $\approx 1{,}523$ KB $\approx 1{,}559{,}331/1{,}024^2 = 1.487$ megabytes (MB).

However, the dimension of an image is often given, not by the total number of pixels, but by the length and height of the picture 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 $1{,}024 \times 768$ 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 pixels is therefore $72 \times 9.92 \approx 713$ in a horizontal direction, and $72 \times 10.12 \approx 729$ in a vertical direction.

Numerous formats are available for saving vector and raster data into a file. All of these formats have 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 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/).

- *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 compressions 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 in different resolutions and with different colormaps. The main advantage of TIFF files was originally their portability. A TIFF should perform on all computer platforms, but unfortunately, numerous modifications of the TIFF evolved in subsequent years, resulting in incompatibilities. The TIFF is therefore now often called the *Thousands of Incompatible File Formats*.

- *PostScript* (PS) and *Encapsulated PostScript* (EPS) – The PS format has been 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 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 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 consists of grouping pixel values into 8 × 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/).

- The PICT format was developed by Apple Computers in 1984 as the default format for Macintosh graphics. The PICT format can be used for raster images and for vector illustrations. PICT uses various methods for compressing data. The PICT 1 format only supports monochrome graphics, but PICT 2 supports a color depth of up to 32 bits. The PICT format is not supported on other platforms although some PC software tools can work with PICT files (http://www.apple.com).

## 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

unconform1 = imread('unconform.jpg');
```

reads and decompresses the JPEG file, imports the data as 24-bit RGB image array and stores the data in a variable `unconform1`. The command

```
whos
```

shows how the RGB array is stored in the workspace:

```
Name                  Size                    Bytes  Class    Attributes
unconform1          729x713x3               1559331  uint8
```

The details indicate that the image is stored as a 729 × 713 × 3 array representing a 729 × 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:

```
unconform1(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
```

Next, we can view the image using the command

```
imshow(unconform1)
```

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):

```
unconform2 = rgb2gray (unconform1);
```

The new workspace listing now reads

```
Name                Size                  Bytes  Class    Attributes
ans                 6x6                      36  uint8
unconform1        729x713x3             1559331  uint8
unconform2        729x713                519777  uint8
```

in which the difference between the 24-bit RGB and the 8-bit grayscale arrays can be observed. The commands

```
imshow(unconform2)
```

display the result. It is easy to see the difference between the two images in separate Figure Windows (Figs. 8.1 and 8.2). Let us now process the grayscale image. First, we compute a histogram of the distribution of intensity values.

```
imhist(unconform2)
```

A simple technique to enhance the contrast of such an image is to transform this histogram to obtain an equal distribution of grayscales.

```
unconform3 = histeq(unconform2);
```

We can view the difference again using

```
imshow(unconform3)
```

and save the results in a new file.

```
imwrite(unconform3,'unconform3.jpg')
```

We can read the header of the new file by typing

```
imfinfo('unconform3.jpg')
```

which yields

```
Filename: 'unconform3.jpg'
FileModDate: '20-Jan-2010 16:11:12'
FileSize: 138419
Format: 'jpg'
FormatVersion: ''
Width: 713
Height: 729
BitDepth: 8
ColorType: 'grayscale'
FormatSignature: ''
NumberOfSamples: 1
CodingMethod: 'Huffman'
CodingProcess: 'Sequential'
Comment: {}
```

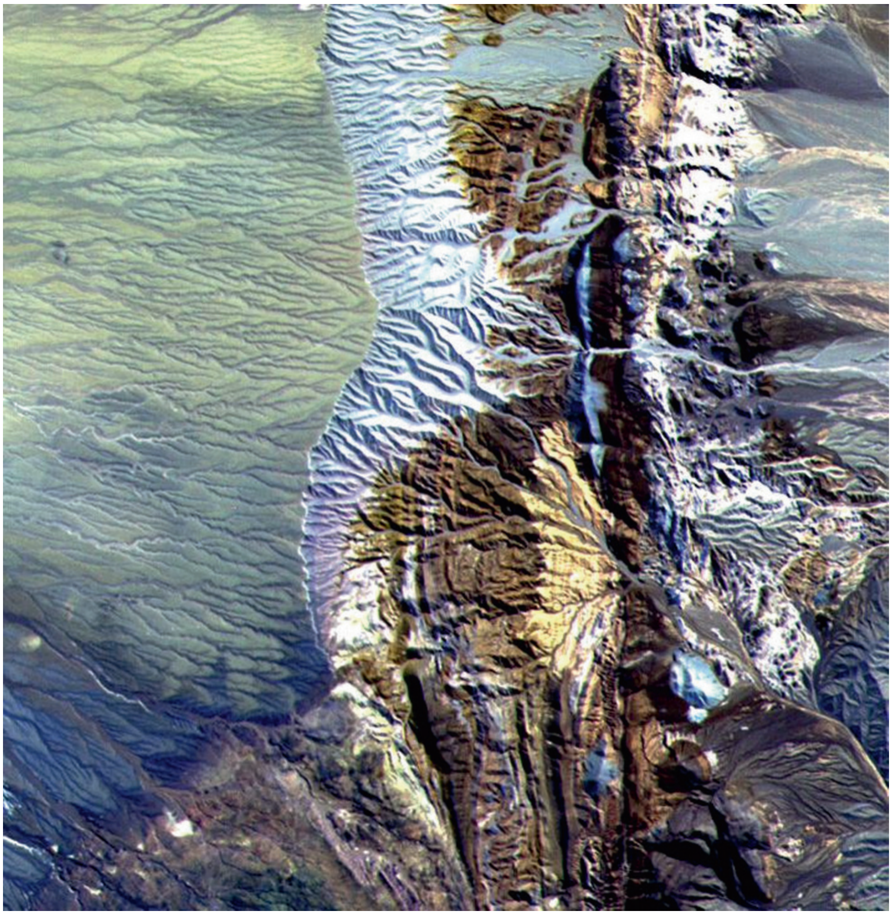


**Fig. 8.1** RGB true color image contained in the file *unconform.jpg*. After decompressing and reading the JPEG file into a 729×713×3 array, MATLAB interprets and displays the RGB composite using the function `imshow`. See detailed description of the image on the NASA TERRA-ASTER webpage http://asterweb.jpl.nasa.gov. Original image courtesy of NASA/GSFC/METI/ERSDAC/JAROS and U.S./Japan ASTER Science Team.

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 ele-

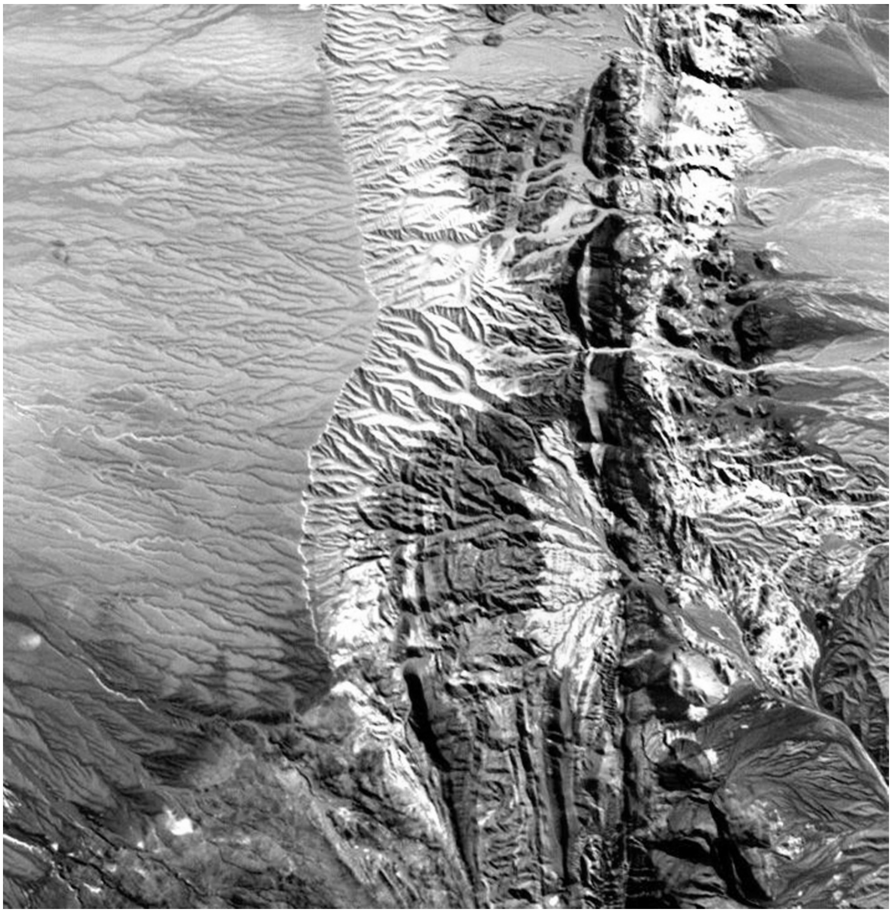


**Fig. 8.2** Grayscale image. After converting the RGB image stored in a 729×713×3 array into a grayscale image stored in a 729×713 array, the result is displayed using `imshow`. Original image courtesy of NASA/GSFC/METI/ERSDAC/JAROS and U.S./Japan ASTER Science Team.

ments 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

```
[x,map] = rgb2ind(unconform1,16);
imshow(unconform1), figure, imshow(x,map)
```

clearly shows the difference between the original 24-bit RGB image ($256^3$ or

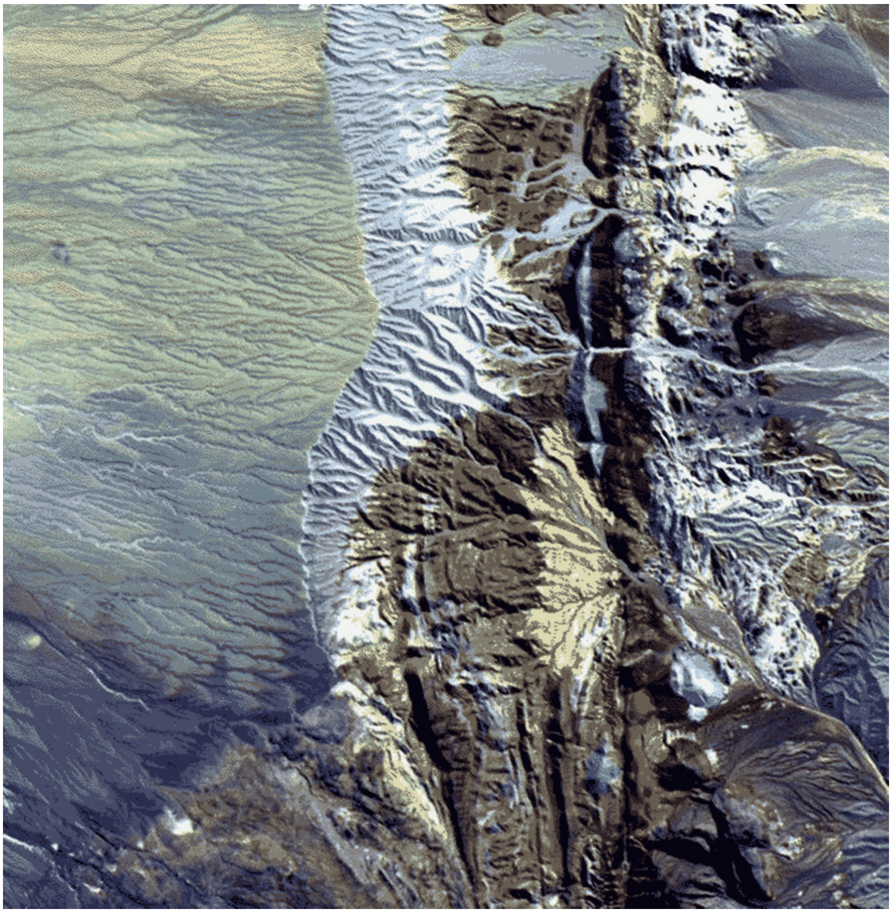


**Fig. 8.3** Indexed color image using a colormap containing 16 different colors. The result is displayed using `imshow`. Original image courtesy of NASA/GSFC/METI/ERSDAC/JAROS and U.S./Japan ASTER Science Team.

ca. 16.7 million different colors) and a color image of only 16 different colors (Figs. 8.1 and 8.3).

## 8.4 Importing, Processing and Exporting Satellite Images

In the previous section, we used a processed ASTER image that we downloaded from the ASTER web page. The original ASTER raw data contain a lot more information and higher resolution than the free-of-charge image stored in *unconform.jpg*. The ASTER instrument produces two types of data, Level-1A and Level-1B. Whereas the L1A data are reconstructed, unprocessed instrument data, L1B data are radiometrically and geometrically corrected. Each ASTER data set contains 15 data arrays representing the intensity values from 15 spectral bands (see the ASTER-web page for more detailed information) and various other items of information such as location, date and time. The raw satellite data can be purchased from the USGS online store:

```
https://wist.echo.nasa.gov/wist-bin/api/ims.cgi
```

On this webpage we can select a discipline/topic (e. g., *Land: ASTER*), and then choose from the list of data sets (e. g., *DEM, Level 1A* or *1B* data), define the search area, and click *Start Search*. The system now needs a few minutes to list all relevant data sets. A list of data sets, including various types of additional information (cloud coverage, exposure date, latitude and longitude), can be obtained by clicking on *List Data Granules*. Furthermore, a low resolution preview can be accessed by selecting *Image*. Having purchased a particular data set, the raw image can be downloaded using a temporary FTP-access. As an example, we process an image from an area in Kenya showing Lake Naivasha. The data are stored in two files

```
naivasha.hdf
naivasha.hdf.met
```

The first file (111 MB large) contains the actual raw data, whereas the second file (100 KB) contains the header, together with all sorts of information about the data. We save both files in our working directory. 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('naivasha.hdf')
```

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. An image such as that used in the previous section is typically achieved by computing an RGB composite from the *vnir_Band3n*, *2* and *1* in the data file. First, we read the data

```
clear

I1 = hdfread('naivasha.hdf','VNIR_Band3N','Fields','ImageData');
```

```
I2 = hdfread('naivasha.hdf','VNIR_Band2','Fields','ImageData');
I3 = hdfread('naivasha.hdf','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 4200×4100 pixel image. The *vnir_Band3n*, *2* and *1* typically contain much information about lithology (including soils), vegetation and water on the Earth's surface. These bands are therefore usually combined into 24-bit RGB images

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

As with the previous examples, the 4200×4100×3 array can now be displayed using

```
imshow(naivasha_rgb);
```

MATLAB scales the 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 (Fig. 8.4), we use the command

```
imwrite(naivasha_rgb,'naivasha.tif','tif')
```

This command saves the RGB composite as a TIFF-file *naivasha.tif* (ca. 50 MB) in the working directory, which can then be processed with other software such as Adobe Photoshop.

## 8.5 Georeferencing Satellite Images

The processed ASTER image does not yet have a coordinate system, and the image therefore needs to be tied to a geographical reference frame (*georeferencing*). The raw data can be loaded and transformed into a RGB composite by typing

```
clear

I1 = hdfread('naivasha.hdf','VNIR_Band3N','Fields','ImageData');
I2 = hdfread('naivasha.hdf','VNIR_Band2','Fields','ImageData');
I3 = hdfread('naivasha.hdf','VNIR_Band1','Fields','ImageData');

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

The HDF browser

```
hdftool('naivasha.hdf')
```

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* that contains the following information:

```
upperleft  = [-0.319922, 36.214332];
upperright = [-0.400443, 36.770406];
lowerleft  = [-0.878267, 36.096003];
lowerright = [-0.958743, 36.652213];
```

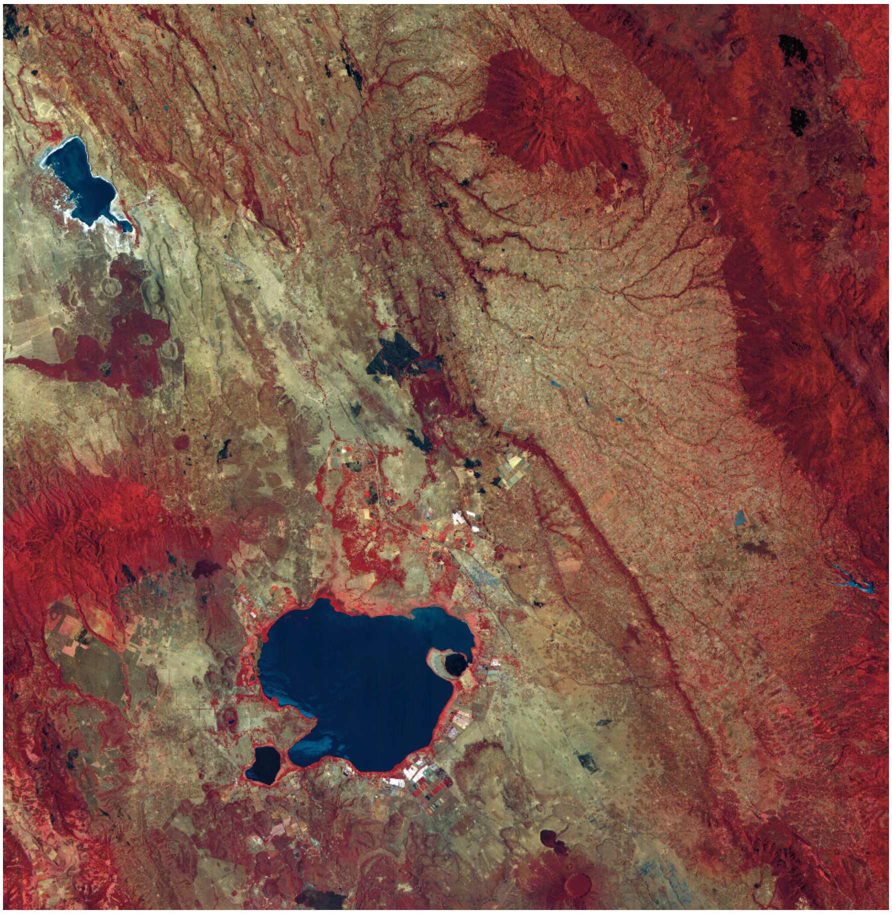


**Fig. 8.4** RGB composite of a TERRA-ASTER image using the spectral infrared bands *vnir_Band3n*, *2* and *1*. The result is displayed using `imshow`. Original image courtesy of NASA/GSFC/METI/ERSDAC/JAROS and U.S./Japan ASTER Science Team.

These two-element vectors can be collected into a single array `inputpoints`. Subsequently, the left and right columns can be flipped in order to have *x=longitudes* and *y=latitudes*.

```
inputpoints(1,:) = upperleft;
inputpoints(2,:) = lowerleft;
inputpoints(3,:) = upperright;
inputpoints(4,:) = lowerright;
inputpoints = fliplr(inputpoints);
```

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,:) = [1,4200];
basepoints(2,:) = [1,1];
basepoints(3,:) = [4100,4200];
basepoints(4,:) = [4100,1];
```

The function `cp2tform` now takes the pairs of control points, `inputpoints` and `basepoints`, and uses them to infer a spatial transformation matrix `tform`.

```
tform = cp2tform(inputpoints,basepoints,'affine');
```

This transformation can be applied to the original RGB composite `naivasha_rgb` in order to obtain a georeferenced version of the satellite image `newnaivasha_rgb`.

```
[newnaivasha_rgb,x,y] = imtransform(naivasha_rgb,tform);
```

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 longitude and latitude range by the array dimension and by subtracting one from the result. Note the difference between the MATLAB numbering convention and the common coding of maps used in the literature. The north/south suffix is generally replaced by a negative sign for south, whereas MATLAB coding conventions require negative signs for north.

```
X = 36.096003 : (36.770406-36.096003)/8569 : 36.770406;
Y =  0.319922 : ( 0.958743- 0.319922)/8400 :  0.958743;
```

The georeferenced image is displayed with coordinates on the axes and a superimposed grid (Fig. 8.5).

```
imshow(newnaivasha_rgb,'XData',X,'YData',Y), axis on, grid on
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 %, with a resolution of 600 dpi.

## 8.6 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

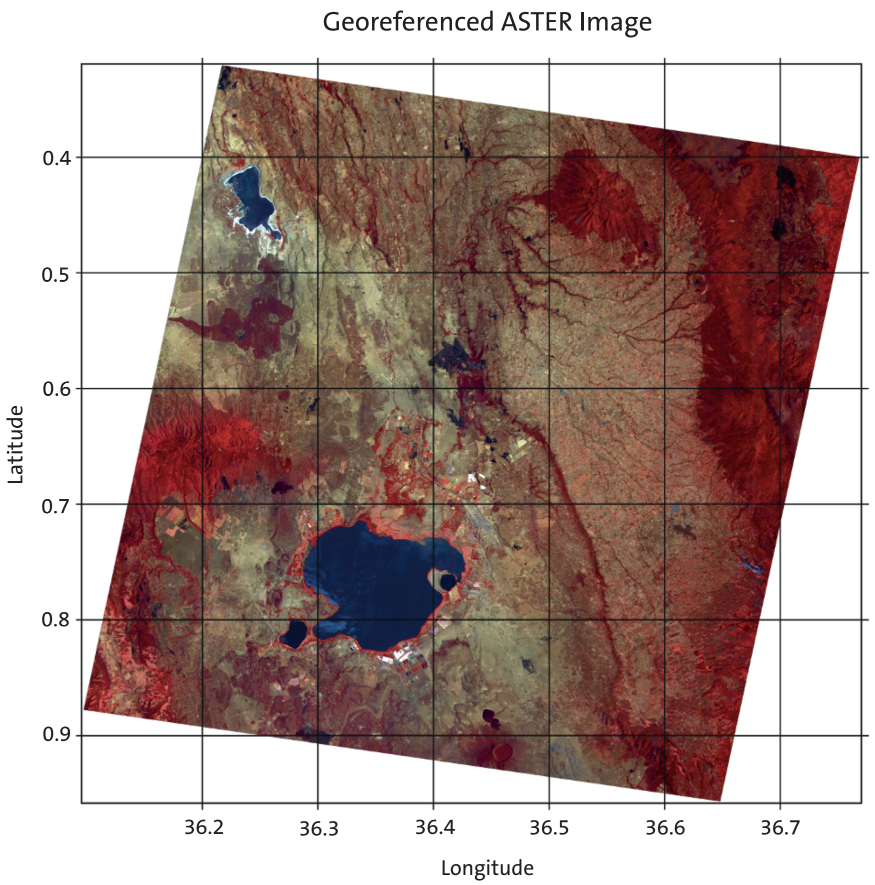


**Fig. 8.5** Geoferenced RGB composite of a TERRA-ASTER image using the infrared bands *vnir_Band3n, 2* and *1*. The result is displayed using `imshow`. Original image courtesy of NASA/GSFC/METI/ERSDAC/JAROS and U.S./Japan ASTER Science Team.

digitizing vector data from the screen. Examples for this type of application include the digitizing of river networks and catchment areas on topographic maps, the outlines of lithologic units on geological maps, the distribution of landslides on satellite images, or the distribution of mineral grains in a microscope image. The digitizing procedure consists of the following steps. Firstly, the image is 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 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 using a mouse on the screen. 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! ');
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. Next, the image is 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 for the limits of the *x*- and *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

```
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 dimension 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.

## 8.7 Color-Intensity Transects of 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 coloration of the sediments has 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 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. 8.6). 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,

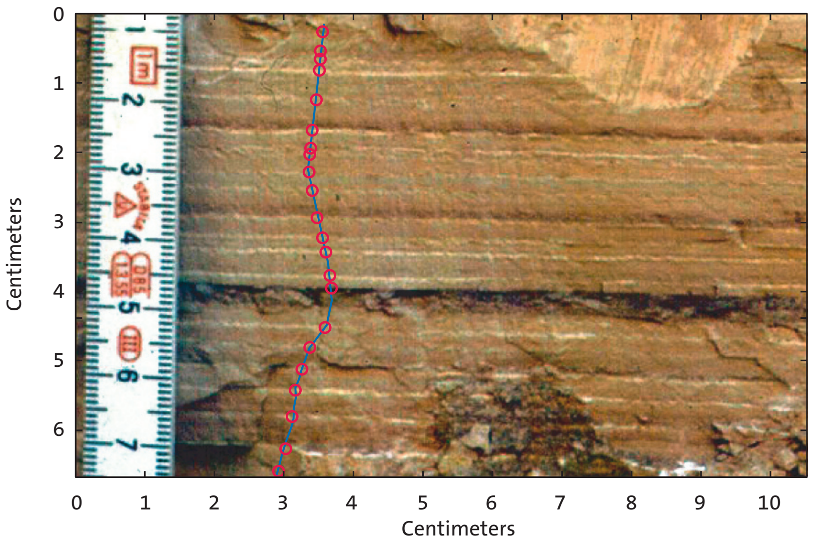


**Fig. 8.6** Photograph of varved lake sediments from the Quebrada de Cafayate in the Santa Maria Basin with cyclic occurrence of intense dark-red coloration reflecting enhanced precipitation and sediment input during ENSO- and TAD-type periodicities (350 cm above the base of the El Paso section). The 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.

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 watershed source areas. 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 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 in the field during an 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.tif* as a 24-bit RGB image array and store the data in a variable `I`.

```
clear

I = imread('varves.tif');
```

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 numbers of pixels per centimeter. The function `ginput` provides a crosshair to gather an unlimited number of points until the *return* key is pressed. 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` corresponds 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 in centimeters.

```
imshow(I,'XData',[0 ix],'YData',[0 iy]), axis on
xlabel('Centimeters'), ylabel('Centimeters')
```

We now digitize the color-intensity transect from top to bottom of the image. 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 use ginput again 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
```

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). 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. 8.7).

```
subplot(2,1,1), plot(CY,C(:,1),CY,C(:,2),CY,C(:,3))
xlabel('Pixel ID'), 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 above revealed significant peaks at 13.1, 3.2, 2.2, and around 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).

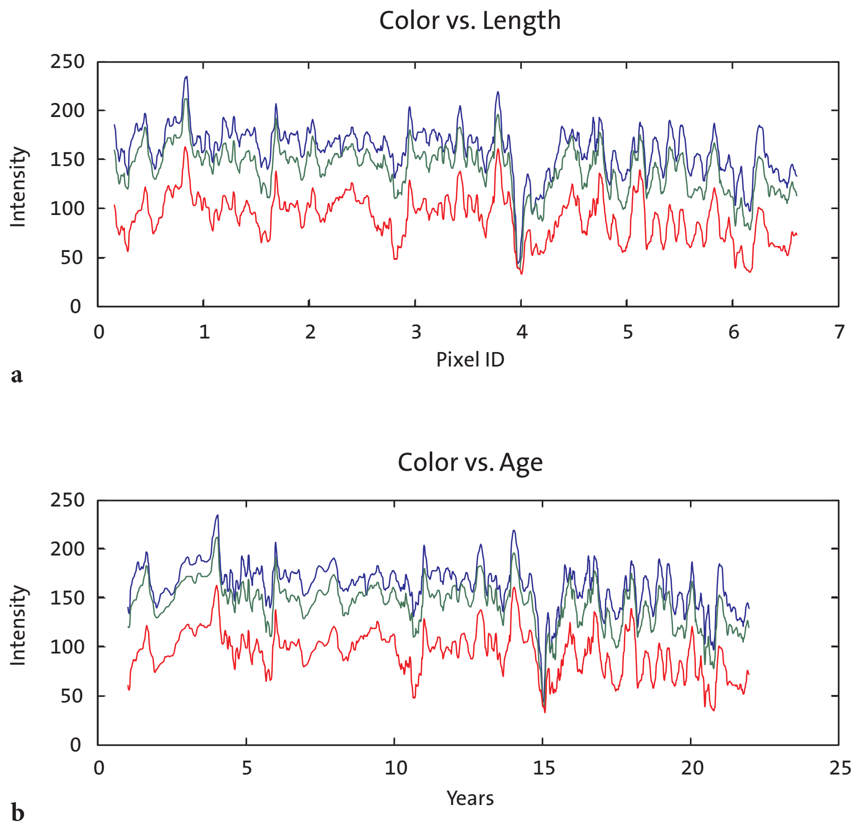


**Fig. 8.7** Color-intensity curves red, green and blue plotted against **a** depth and **b** age.

## 8.8 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 size 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 Transformation 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. Here, we will describe the application of image processing in identifying, measuring and counting particles by means of two examples. 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 also implemented in the MATLAB-based RADIUS software 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 identification of sedimentation processes within undisturbed samples. The MATLAB code for RADIUS can be downloaded from

```
http://www.particle-analysis.info/
```

Readers interested in the use of the Hough Transformation to detect objects such as pollen grains are referred to the excellent algorithms by Tao Peng (University of Maryland), for the detection of circles and lines in images.

The MATLAB routines `circularhough_grd.m` and `hough_grd.m` can be downloaded from File Exchange at

```
http://www.mathworks.com/matlabcentral/fileexchange/
```

The file `circularhough_grd.m` was selected as the MATLAB Central *pick of the week* on May 23rd, 2008. In this blog, the use of the Hough Transformation to detect circles is demonstrated for counting red blood cells in an image.

```
http://blogs.mathworks.com/pick/2008/05/23/
   detecting-circles-in-an-image/
```

The blog describes all steps in great detail and the algorithm described can be easily modified for the application to other objects such as sand grains or pollen grains. The sister algorithm `hough_grd.m` for detection of lines and line segments in an image can be used for fracture-trace and lineament analysis in photogeologic applications.

The following example for object segmentation illustrates the segmentation, measuring and counting of objects using the watershed segmentation algorithm (Fig. 8.8). 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 $284 \times 367$ pixels, and since the width is 3 cm, the height is 3 cm$\times$284 pixels/367 pixels=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 therefore will be ignored in the following experiment. 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` is relatively low in 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

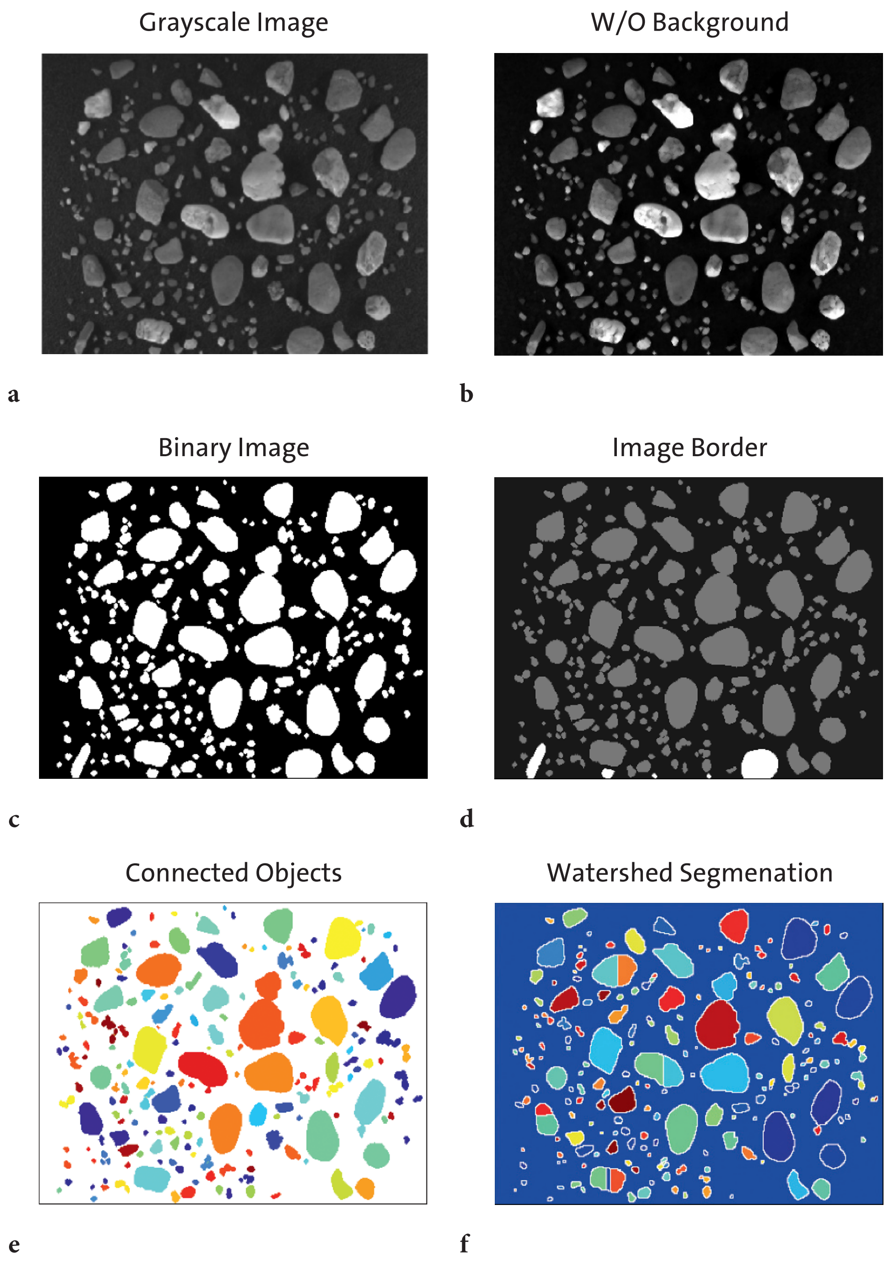


**Fig. 8.8** Display of results from automated grain size analysis of a microscope image; **a** original grayscale image, **b** image after removal of background, **c** image after conversion to binary image, **d** image after eliminating objects overlapping the image border, **e** image with objects detected by tracing the boundaries of connected pixels, and **f** image with objects detected using a watershed segmentation algorithm.

that 1 % of the data is saturated at low and high intensities. This increases the contrast of 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 image I3, i. e., basically the texture of the black foil upon 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 3 pixels generated 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 substract 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])
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, equivalent 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 whereas zero pixels are background. By default, the function also traces the boundaries of holes in the image `I7`. 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` that simply shuffles the colors of `jet` pseudorandomly.

```
[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);
numObjects
```

In our example, the method identified 192 grains, which is significantly lower 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 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
v = 0.0005 : 0.0005 : 0.15;
hist(objectareas,v)
xlabel('Grain Size in Millimeters^2')
ylabel('Number of Grains')
axis([0 0.1 0 30])
```

Several methods exist that partly overcome the artifact from 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 commonly used criterion to identify pixels that belong to one object is the nearest-neighbor distance. We use the distance transform performed 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 increasing 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 zero values. We therefore have to apply `bwdist` to the complement of the binary image `I7` instead of 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';
objectsizes = sortrows(objectsizes,1);
objectsizes = objectsizes(objectsizes~=0);
```

We have now recognized 205 objects, i. e., more objects than were identified in the previous experiment without watershed segmentation. Visual inspection of the result, however, reveals some oversegmentation due to noise or other irregularities in the image, i. e., 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 $(3\,\mathrm{cm}/367)^2$.

```
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 $\mathrm{cm}^2$.

```
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 $0.09\,\mathrm{cm}^2$, which represents the maximum size of all grains in the image. Finally, we plot the histogram of all the grain areas.

```
clf
```

```
v = 0.0005 : 0.0005 : 0.15;
hist(objectareas,v)
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. Then, we display the polygon upon 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))

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.

## 8.9 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 colora tion 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. 8.9). 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 × 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`.

```
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 therefore 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 of the new image 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, i. e., basically the texture of the black foil upon 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.

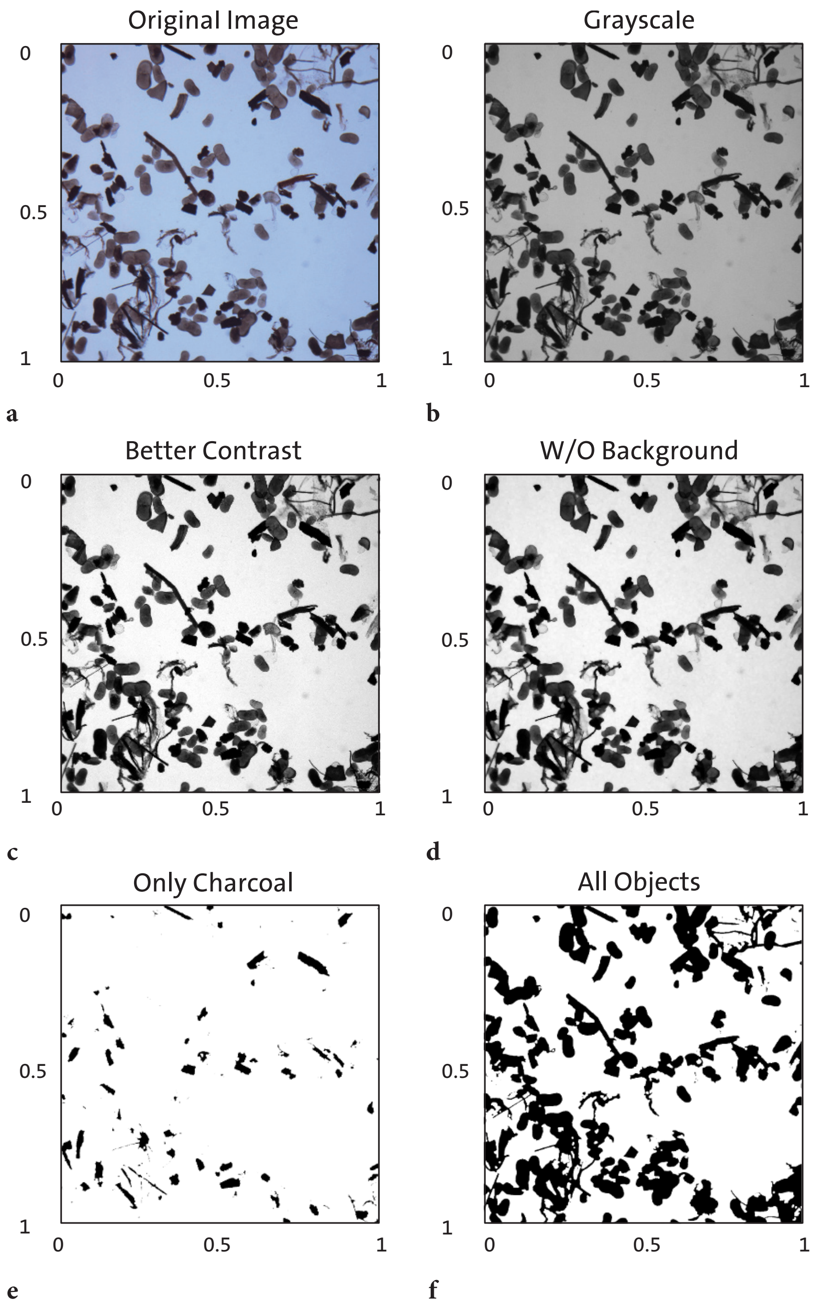


**Fig. 8.9** Display of results from automatic quantification of charcoal in a microscope image; **a** original color image, **b** grayscale image, **c** image after enhancement of contrast, **d** image after removal of background, **e** image after thresholding to separate charcoal particles, **f** image after thresholding to separate all objects.

```
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 substract 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 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, equivalent 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')
```

We can now simply count the number of pixels to estimate the total amount of charcoal in the image knowing the size of a pixel. 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.

## Recommended Reading

Abrams M, Hook S (2002) ASTER User Handbook – Version 2. Jet Propulsion Laboratory and EROS Data Center, Sioux Falls

Campbell JB (2002) Introduction to Remote Sensing. Taylor & Francis, London

Francus P (2005) Image Analysis, Sediments and Paleoenvironments – Developments in Paleoenvironmental Research. Springer, Berlin Heidelberg New York

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

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

Seelos K, Sirocko F (2005) RADIUS – Rapid Particle Analysis of digital images by ultra-high-resolution scanning of thin sections. Sedimentology 52: 669–681.

The Mathworks (2010) Image Processing Toolbox – User's Guide. The MathWorks, Natick, MA

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.

# 9 Multivariate Statistics

## 9.1 Introduction

Multivariate analysis aims to understand and describe the relationship 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 clay mineral contents of sedimentary sequences. If there are complex relationships between the different parameters, univariate statistics ignores the information content of the data. There are, however, a number of methods available for investigating the scaling properties of multivariate data.

A multivariate data set consists of measurements of $p$ variables on $n$ objects. Such data sets 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 columns of the array represent the $p$ variables, and the rows represent the $n$ objects. The characteristics of the 2nd object in the suite of samples are described by the vector in the second row of the data array:

$$X_2 = (\, x_{21} \quad x_{22} \quad \ldots \quad x_{2p} \,)$$

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 of the multi-parameter methods simply try to overcome the main

M.H. Trauth, *MATLAB® Recipes for Earth Sciences*, 3rd ed.,
DOI 10.1007/978-3-642-12762-5_9, © Springer-Verlag Berlin Heidelberg 2010

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 into 2D. 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 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 by *principal component analysis* (PCA), and by the *factor analysis* (FA). These methods seek the directions of maximum variance in the 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 often help in the interpretation of the multivariate data set since they often contain valuable information on the process itself that generated 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 ion sizes that are observed in similar locations within the lattices of certain minerals. Examples of such behavior are $Si^{4+}$ 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 applied to correlate volcanic ash layers as described in the above example. Tephrochronology attempts to correlate tephra by means of their geochemical fingerprints. In combination with a few radiometric age determinations from the key ash layers, this method allows correlation of the sedimentary sequences that contain these ash layers (e. g., Westgate 1998, Hermanns et al. 2000). Additional examples for the application of cluster analysis come from the field of micropaleontology. In this context, multivariate methods are employed, for example, to compare the pollen, foraminifera or diatoms contents of microfossil assemblages (e. g., Birks and Gordon 1985).

The following sections introduce the most important techniques of multivariate statistics: principal component analysis (PCA) and cluster analysis (CA) in sections 9.2 and 9.4, and *independent component analysis* (ICA), which is a nonlinear extension of PCA, in section 9.3. These sections first provide an introduction to the theory behind the techniques, followed by illustration of the use of these methods in analyzing earth sciences data is illustrated with MATLAB functions.

## 9.2 Principal Component Analysis

Principal component analysis (PCA) detects linear dependencies between variables and replaces groups of correlated variables by new uncorrelated variables, the *principal components* (PC). The method was introduced by Karl Pearson (1901) and further developed by Harold Hotelling (1931). The performance of PCA is better illustrated with help of 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 ad-

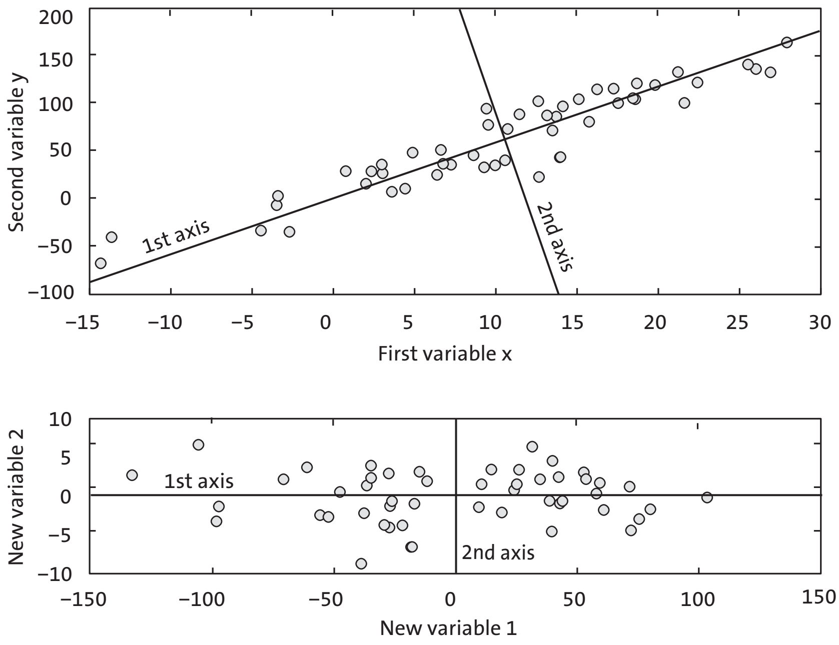


**Fig. 9.1** Principal component analysis (PCA) illustrated for a bivariate scatter. The original $xy$ coordinate system is replaced by a new orthogonal system, where the first axis passes through the long axis of the data scatter and the new origin is the bivariate mean. We can now reduce the number of dimensions by dropping the second axis without losing much information.

vantage that the first axis can be used to describe most of the variance, while the second axis contributes only a little additional information. Prior to the transformation two axes were required to describe the data set, but it is now possible to reduce the data dimension 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 samples. Assume a data set comprised of measurements of $p$ parameters 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

$$\begin{aligned} Y_1 &= a_{11}X_1 + a_{12}X_2 + \ldots + a_{1p}X_p \\ Y_2 &= a_{21}X_1 + a_{22}X_2 + \ldots + a_{2p}X_p \\ \\ Y_p &= a_{p1}X_1 + a_{p2}X_2 + \ldots + a_{pp}X_p \end{aligned}$$

The first principle component PC$_1$, denoted by $Y_1$, contains the greatest variance, PC$_2$ contains the second highest variance, and so forth. All the PCs together contain the full variance of the data set. The variance is largely concentrated in the first few PCs, which include most of the information content of the data set. The last PCs are generally ignored to reduce the data dimensions. The factors $a_{ij}$ in the above equations are the *principal component loads*. The values of these factors represent the relative contribution 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, the influence of this variable is low. A high positive or negative $a_{ij}$ suggests a strong contribution of the variable $X_j$. 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*.

In the following example, a synthetic data set is used to illustrate the use of the function `princomp` included in the Statistics Toolbox. Thirty samples were taken from different levels in a sedimentary sequence containing varying quantities of certain minerals. Our data set contains proportions of various minerals identified in these sediment samples. The sediments derived from two distinct rock types. PCA is used to verify the influence of the two different source rocks and to estimate their relative contributions. First, the data stored in *sediment_1.txt* are loaded by typing

```
clear

data = load('sediments_1.txt');
```

We display the quantities of minerals in the thirty samples as a time series, in two separate plots.

```
subplot(2,1,1)
plot(data(:,1:3)), grid
legend('MinA1','MinA2','MinA3')
xlabel('Sample ID'), ylabel('Quantity')

subplot(2,1,2)
plot(data(:,4:6)), grid
legend('MinB1','MinB2','MinB3')
xlabel('Sample ID'), ylabel('Quantity')
```

The first source rock contains minerals A1 and A2. The quantities of A1 and A2 are therefore highly correlated within the sample series. Mineral A3 is a weathering product of one of these minerals and has therefore a negative correlation with A1 and A2. The second source rock contains mineral B1, while B2 is the weathering product of B1 and has therefore a negative correlation with the main constituent of the second rock type. Mineral B3 occurs in both source rocks and therefore shows a complex correlation with the other minerals. We now define labels for the various graphs created by the PCA. We number the samples 1 to 30, with the minerals being identified by five-character abbreviations.

```
for i = 1 : 30
    sample(i,:) = [sprintf('%02.0f',i)];
end
minerals = ['MinA1';'MinA2';'MinA3';'MinB1';'MinB2';'MinB3'];
```

A successful PCA requires linear correlations between variables. The *correlation matrix* provides a technique for exploring such dependencies in the data set (Chapter 4). The elements of the correlation matrix are Pearson's correlation coefficients for each pair of variables, as shown in Figure 9.2. Here, the variables are minerals.

```
corrmatrix = corrcoef(data);
corrmatrix = flipud(corrmatrix);

imagesc(corrmatrix), colormap(hot), caxis([-1 1])
```

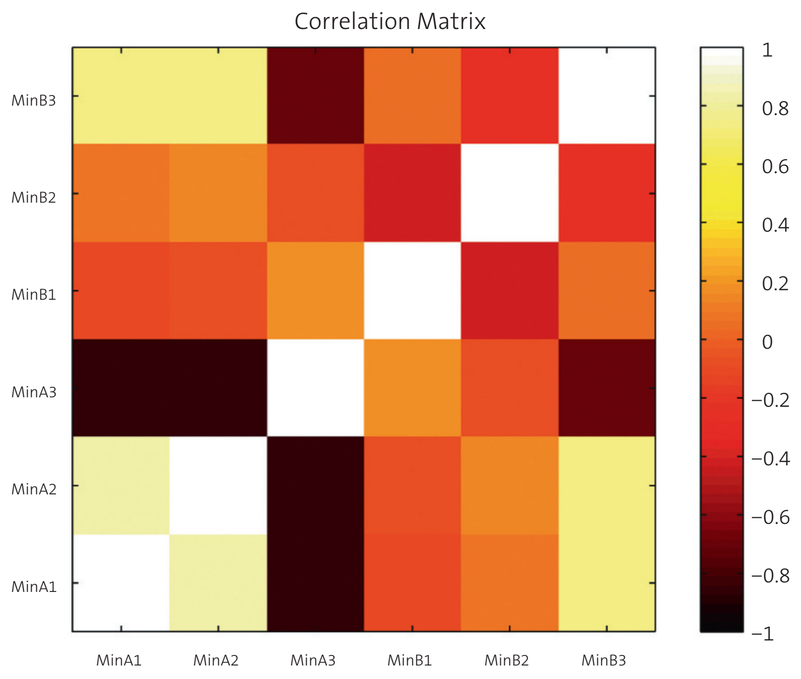


**Fig. 9.2** Correlation matrix containing Pearson's correlation coefficients for each pair of variables, such as minerals in a sediment sample. Light colors represent strong positive linear correlations, whereas dark colors document negative correlations. Orange suggests no correlation.

```
    title('Correlation Matrix')
    axis square, colorbar, hold
    set(gca,'XTickLabel',minerals,'YTickLabel',flipud(minerals))
```

This pseudocolor plot of the correlation coefficients shows strong positive correlations between the minerals A1 and A2 and a strong negative correlation between these minerals and A3, as would be expected from the composition of the first source rock. Minerals B1 and B2 are significantly anticorrelated, whereas B3 shows positive and negative correlations with minerals from both source rocks. We also observe no dependency between some of the variables, for instance between B2 and the two minerals A1 and A2, and between B1 and A3. From the observed dependencies, we would expect interesting results from the application of a PCA.

Various methods exist for scaling the original data before applying the PCA, such as *mean centering* (a mean equal to zero) or *autoscaling* (a mean equal to zero and a standard deviation equal to one). However, we use the original data for computing the PCA. The output of the function `princomp` includes the principal component loads `pcs`, the scores `newdata` and the variances `variances`.

```
[pcs,newdata,variances] = princomp(data);
```

The loads of the first five principal components $PC_1$ to $PC_5$ can be shown by typing

```
pcs(:,1:5)

ans =
    0.5377   -0.1595   -0.7201    0.0675   -0.3224
    0.5514   -0.2354    0.5115   -0.4833    0.0730
   -0.3889    0.1213    0.1727    0.1442   -0.5364
   -0.0179    0.2249    0.0322   -0.5469   -0.6666
    0.0040   -0.7541    0.2708    0.3820   -0.3602
    0.5052    0.5340    0.3401    0.5442   -0.1702
```

We observe that $PC_1$ (first column) has high positive loads in the first two variables A1 and A2 (first and second row), a high negative load in the third variable A3 (third row) and a high positive load in the sixth variable B3 (sixth row), whereas the other loads are close to zero. $PC_2$ (second column) has a high negative load in variable B2 (fifth column) and a positive load in B3 (sixth column), whereas the load of variable B2 is only slightly higher in $PC_2$ than in $PC_1$. We create a number of plots to visualize the PCs.

```
subplot(2,2,1), plot(1:6,pcs(:,1),'o'), axis([1 6 -1 1])
text((1:6)+0.2,pcs(:,1),minerals,'FontSize',8), hold
plot(1:6,zeros(6,1),'r'), title('PC 1')

subplot(2,2,2), plot(1:6,pcs(:,2),'o'), axis([1 6 -1 1])
text((1:6)+0.2,pcs(:,2),minerals,'FontSize',8), hold
plot(1:6,zeros(6,1),'r'), title('PC 2')

subplot(2,2,3), plot(1:6,pcs(:,3),'o'), axis([1 6 -1 1])
text((1:6)+0.2,pcs(:,3),minerals,'FontSize',8), hold
plot(1:6,zeros(6,1),'r'), title('PC 3')

subplot(2,2,4), plot(1:6,pcs(:,4),'o'), axis([1 6 -1 1])
text((1:6)+0.2,pcs(:,4),minerals,'FontSize',8), hold
plot(1:6,zeros(6,1),'r'), title('PC 4')
```

The loads of the index minerals and their relationships to the PCs can be used to interpret the relative influence of the different source rocks. $PC_1$ is characterized by strong contributions of A1 and A2, reflecting a relatively strong influence of the first rock type as a source of the sediments. A contribution of A3 with an opposite sign appears to support the interpretation of A3 as a weathering product of A1 and A2. The second principal component $PC_2$ is clearly dominated by the second source rock, as indicated by the high load of B1. Again, the high influence of B2 with an opposite sign reflects the weathering of B1 to produce B2. The occurrence of B3 in both source rocks results in high loads of B3 in both $PC_1$ and $PC_2$. Principle components $PC_3$ and $PC_4$ show mixed and contradictory patterns of loads and are therefore not easy to interpret, most probably as a result of noise in the data set.

An alternative way to plot of the loads is as a bivariate plot of two principal components. We therefore ignore $PC_3$ and $PC_4$ 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([-0.8 0.8],[0 0],'r')
plot([0 0],[-0.8 0.8],'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 were previously shown on several graphs (Fig. 9.3). It is also possible to plot the data set as functions of the new variables. This requires the second output of `princomp`, containing the principal component scores.

```
plot(newdata(:,1),newdata(:,2),'+'), hold on
text(newdata(:,1)+0.01,newdata(:,2),sample)
plot([-80 100],[0 0],'r')
plot([0 0],[-60 80],'r')
xlabel('First Principal Component Scores')
ylabel('Second Principal Component Scores')
hold off
```

This plot clearly defines groups of samples with similar influences. Samples 5, 8, 20, 27 and 30, dominated by influences of the first source rock, all cluster in the right half of the diagram, while samples 7, 9, 24 and once again 30, strongly influenced by the second rock type, all fall in the upper half of the graph. Next, we use the third output of the function `princomp` to compute the variances of the PCs.

```
percent_explained = 100*variances/sum(variances)

percent_explained =
   72.7390
   14.6658
    4.3129
    4.1775
    2.7791
    1.3257
```

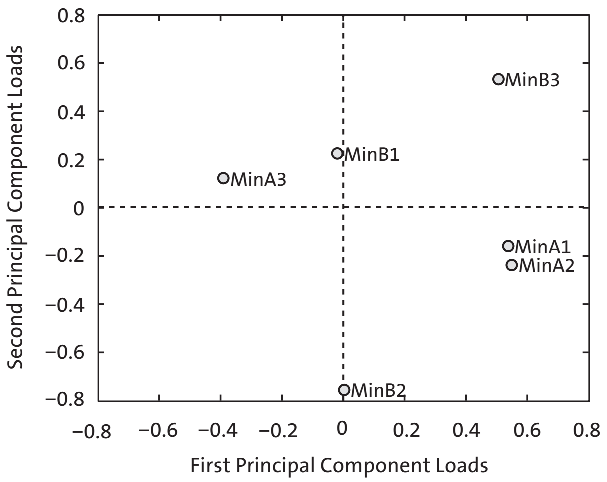


**Fig. 9.3** Principal components suggesting that the PCs are influenced by different minerals. See text for detailed interpretation of the PCs.

We see that almost 73 % of the total variance is contained in $PC_1$, and around 15 % is contained in $PC_2$, while none of the other PCs contribute very much to the total variance of the data set. 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 `newdata(:,1)` and `newdata(:,2)` that is close to zero.

```
corrcoef(newdata(:,1),newdata(:,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,newdata(:,1),1:30,newdata(:,2)), grid,
legend('PC1','PC2')
xlabel('Sample ID'), ylabel('Value')
```

This plot displays ca. 73 %+15 %=88 % of the variance contained in the multivariate data set. According to our interpretation of $PC_1$ and $PC_2$ this plot shows the variability in the relative contributions from the two sources to the sedimentary column under investigation.

In summary, the approach described above has been used to study the provenance of varved lake sediments 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 watershed source areas. 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 (see also Section 8.7).

## 9.3 Independent Component Analysis (by N. Marwan)

Principal component analysis (PCA) is the standard method for separating mixed signals. Such analyses produce signals that are linearly uncorrelated. This method is also called *whitening* since this property is characteristic of white noise. Although the separated signals are uncorrelated, they can still be interdependent, i.e., nonlinear correlation may still remain. The *independent component analysis* (ICA) was developed to investigate such data. It separates mixed signals into independent signals, which are then nonlinearly uncorrelated. *Fast ICA algorithms* use a criterion that estimates how Gaussian the combined distribution of the independent components is. The less Gaussian this distribution is, the more independent the individual components are.

According to the model, $n$ independent signals $x(t)$ are linearly mixed in $m$ measurements,

$$x(t) = As(t)$$

in which we are interested in the source signals $s_i$ and the mixing matrix $A$. 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

$$s(t) = W^T x(t)$$

where $W^T$ is the separation matrix required to reverse the mixing and obtain the original signals. Let us consider a mixing of three signals, $s_1$, $s_2$ and $s_3$, and their separation using PCA and ICA. First, we create three periodic signals

```
clear

i = (1:0.01:10 * pi)';
[dummy index] = sort(sin(i));

s1(index,1) = i/31; s1 = s1 - mean(s1);
s2 = abs(cos(1.89*i)); s2 = s2 - mean(s2);
s3 = sin(3.43*i);

subplot(3,2,1), plot(s1), ylabel('s_1'), title('Raw signals')
subplot(3,2,3), plot(s2), ylabel('s_2')
subplot(3,2,5), plot(s3), ylabel('s_3')
```

Now we mix these signals and add some observational noise. We obtain a three-column vector `x` which corresponds to our measurements (Fig. 9.4).

```
randn('state',1);

x = [.1*s1 + .8*s2 + .01*randn(length(i),1), ...
     .4*s1 + .3*s2 + .01*randn(length(i),1), ...
     .1*s1 +   s3  + .02*randn(length(i),1)];

subplot(3,2,2), plot(x(:,1)), ylabel('x_1'), title('Mixed
signals')
subplot(3,2,4), plot(x(:,2)), ylabel('x_2')
subplot(3,2,6), plot(x(:,3)), ylabel('x_3')
```

We begin with the separation of the signals using PCA. We calculate the principal components and the whitening matrix `W_PCA` with

```
[E sPCA D] = princomp(x);
sPCA = sPCA./repmat(std(sPCA),length(sPCA),1);
```

The PC scores `sPCA` are the linearly separated components of the mixed signals $x$ (Fig. 9.5).

```
subplot(3,2,1), plot(sPCA(:,1))
ylabel('s_{PCA1}'), title('Separated signals - PCA')
subplot(3,2,3), plot(sPCA(:,2)), ylabel('s_{PCA2}')
subplot(3,2,5), plot(sPCA(:,3)), ylabel('s_{PCA3}')
```

The mixing matrix $A$ can be found with

```
A_PCA = E * sqrt(diag(D));
W_PCA = inv(sqrt(diag(D))) * E';
```

Next, we separate the signals into independent components. We will do this by using a FastICA algorithm, which is based on a fixed-point iteration scheme, to find the least Gaussian distributed of the independent components $W^T x$. For the nonlinearity function we use a power of three function, as an example,

```
rand('state',1);

div = 0;
B = orth(rand(3, 3) - .5);
BOld = zeros(size(B));

while (1 - div) > eps
   B = B * real(inv(B' * B)^(1/2));
   div = min(abs(diag(B' * BOld)));
   BOld  = B;
```

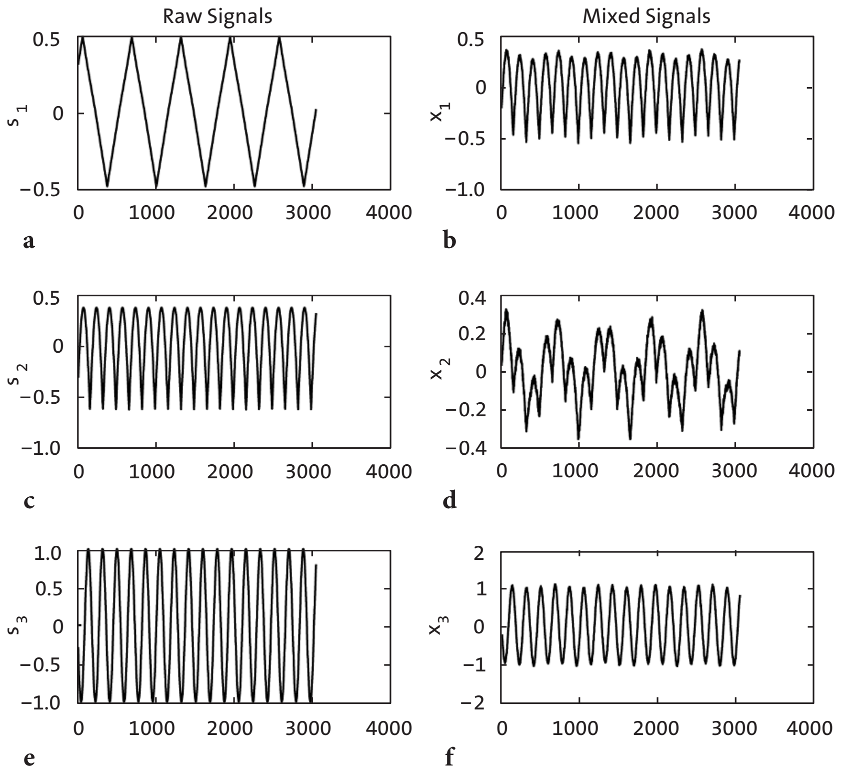


**Fig. 9.4** Sample input for the independent component analysis. We first generate three period signals (**a**, **c**, **e**), mix the signals and add some Gausssian noise (**b**, **d**, **f**).

```
      B = (sPCA' * (sPCA * B) .^ 3) / length(sPCA) - 3 * B;
      sICA = sPCA * B;
   end
```

We plot the separated components (Fig. 9.5) with

```
subplot(3,2,2), plot(sICA(:,1)), ylabel('s_{ICA1}'),
   title('Separated signals - ICA')
subplot(3,2,4), plot(sICA(:,2)), ylabel('s_{ICA2}')
subplot(3,2,6), plot(sICA(:,3)), ylabel('s_{ICA3}')
```

We can now see that the PCA algorithm has not provided a satisfactory separation of the mixed signals. The saw-tooth signal, in particular, was not correctly identified. In contrast, the ICA has identified the source signals almost perfectly. The only noticeable differences are the noise level, which

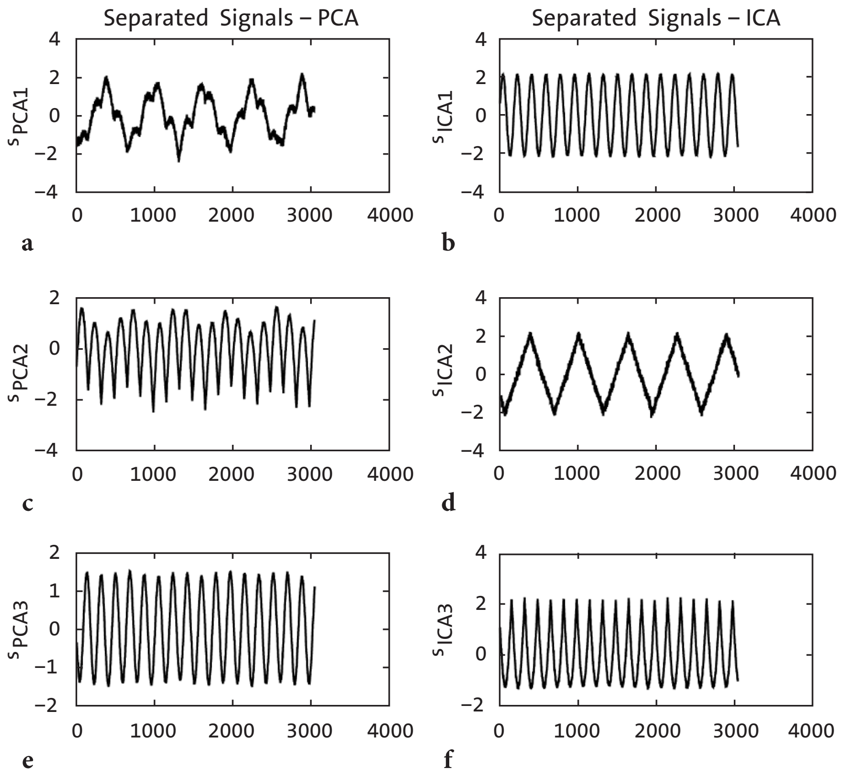


**Fig. 9.5** Output of the principal component analysis (**a**, **c**, **e**) compared with the output of the independent component analysis (**b**, **d**, **f**). The PCA has not reliably separated the mixed signals, whereas the ICA identified the source signals almost perfectly.

derives from the observations, the incorrect sign, and the incorrect order of the signals. However, the sign and the order of the signals are not really important, since we generally do not have any knowledge of the real sources or of their order. With

```
A_ICA = A_PCA * B;
W_ICA = B' * W_PCA;
```

we compute the mixing matrix $A$ and the separation matrix $W$. The mixing matrix $A$ can be used to estimate the proportions of the separated signals in our measurements The components $a_{ij}$ of the mixing matrix A correspond to the principal component loads, as introduced in Section 9.2. A FastICA package is available for MATLAB and can be found at

```
http://www.cis.hut.fi/projects/ica/fastica/
```

## 9.4 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 between all pairs of objects, and then ranks the groups according to their similarity, finally creating a hierarchical tree visualized as a dendrogram. Examples for grouping objects in earth sciences are correlations within volcanic ash layers (Hermanns et al. 2000) and comparisons between microfossil assemblages (Birks and Gordon 1985).

There are numerous methods for calculating the similarity 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.

$$X_1 = (x_{11} \quad x_{12} \quad \dots \quad x_{1p})$$
$$X_2 = (x_{21} \quad x_{22} \quad \dots \quad x_{2p})$$

The most popular measures of similarity 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 distance* – In the city of Manhattan, one must walk along perpendicular avenues rather than crossing blocks diagonally. The Manhattan distance is therefore the sum of all differences:

$$\Delta_{12} = |x_{11} - x_{21}| + |x_{12} - x_{22}| + \ldots + |x_{1p} - x_{2p}|$$

Other alternative measures of similarity include the

- *Correlation similarity coefficient* – This uses Pearson's linear product-moment correlation coefficient to compute the similarity of two objects:

$$r_{x1x2} = \frac{\sum_{i=1}^{n} (x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)}{(n-1) s_{x1} s_{x2}}$$

  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).

- *Inner-product similarity index* – Normalizing the length of the data vectors to a value of one and computing the inner product of these 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 to reconstruct the environmental conditions in the past.

$$s_{12} = \frac{1}{|X_1|} \frac{1}{|X_2|} \left( x_{11} \; x_{12} \ldots x_{1p} \right) \begin{pmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2p} \end{pmatrix}$$

  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 build a hierarchical tree visualized as a dendrogram. Most clustering algorithms simply link the two objects with the highest level of similarity. 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* – Here, the Euclidean distance between the multivariate means of a number of *K* clusters is used 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.

- *K-nearest-neighbors clustering* – Alternatively, the Euclidean distance of the nearest neighbors is used 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.

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. Here, the data need to be transformed to zero means (*mean centering*). Differences in both the variances and the means are corrected by *autoscaling*, i.e., the data are standardized to zero means 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 example for performing a cluster analysis, the sediment data stored in *sediment_2.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 occurrence of fluorite (*flu*), sphalerite (*sph*) and galena (*gal*), as well as 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 containing varying amounts of these minerals. First, the distances between pairs of samples can be computed. The function `pdist` provides many ways for computing this distance, such as the Euclidian or Manhattan *city block* distance. We use the default setting which is the Euclidian distance.

```
clear

data = load('sediments_2.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)
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. Next, we 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

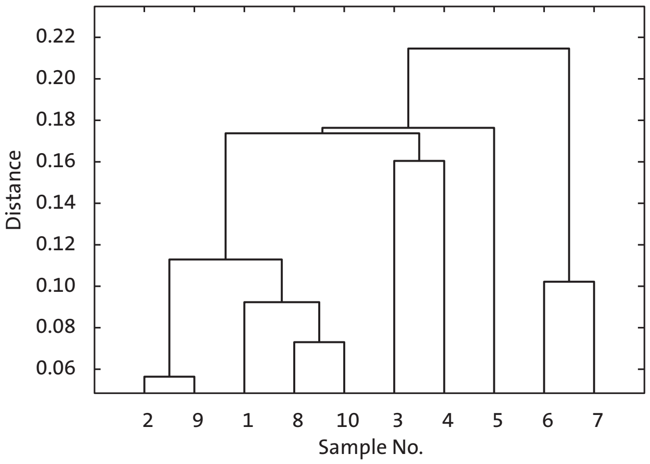


**Fig. 9.6** Output of the cluster analysis. The dendrogram shows clear groups consisting of samples 1, 2, 8, 9 and 10 (the magmatic source rocks), samples 3, 4 and 5 (the magmatic dyke containing ore minerals), and samples 6 and 7 (the sandstone unit).

row (link) between objects (or samples) 1 and 2 has the smallest distance, corresponding to the highest 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 lowest difference of 0.0730. The next row shows that the new group 12 is then grouped with sample 1, which have a difference of 0.0923, and so forth. Finally, we visualize the hierarchical clusters as a dendrogram which is shown in Figure 9.6.

```
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)

ans =
    0.7579
```

The result is convincing since the closer this coefficient is to one, the better is the cluster solution.

## 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

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

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

The Mathworks (2010) Statistics Toolbox – User's Guide. The MathWorks, Natick, MA

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

# 10 Statistics on Directional Data

## 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). In principle, two types of directional data exist in earth sciences: directional data *sensu stricto*, and oriented data. Directional data have a true polarity, 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. Oriented data describe axial data and lines without any sense of direction, such as the orientation of joints.

MATLAB is not the first choice for analyzing directional data since it does not provide the relevant functions, such as an algorithm 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, available for Mac OS 9 and OS X as well as for Microsoft Windows:

```
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 3 on univariate statistics, the next sections are on 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

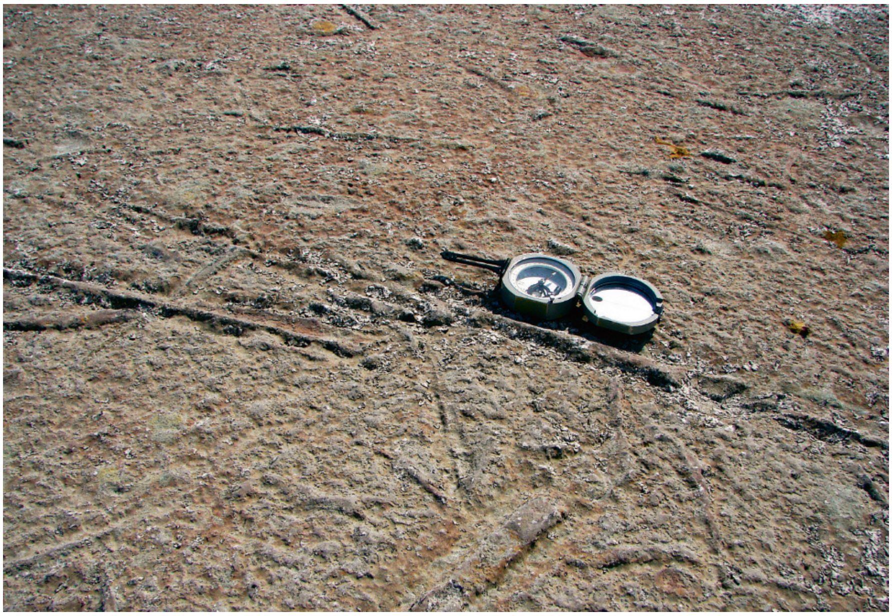


**Fig. 10.1** *Orthoceras* fossils from an outcrop Neptuni Acrar near Byxelkrok on Öland, Sweden. *Orthoceras* is a cephalopod with a straight shell and that lived in the Ordovician era, about 450 million years ago. Such elongated, asymmetric objects tend to orient themselves in the hydrodynamically most stable position. The fossils can 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.

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 diagrams 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');
```

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 in which zero degrees lies along the *x*-axis of the coordinate graph. In geosciences, however, 0° points due north, 90° points due east and the angles increase clockwise. 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 frequency. In a final 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 modified as follows. After the histogram of the angles is computed in line 58 by 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.

```
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.

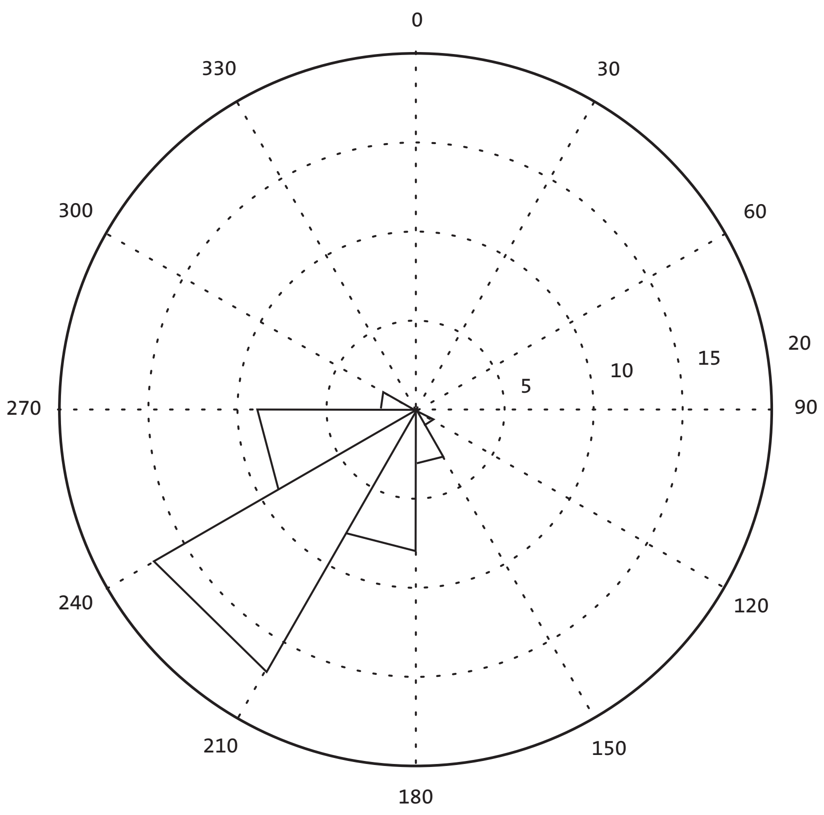


**Fig. 10.2** Rose diagram to display directional data using the function `rose`. The radii of the area segments are proportional to the frequencies for each class.

The collection of data can be written as

$$\theta_1, \theta_2, \theta_3, \ldots, \theta_N$$

containing N observations $\theta_i$. Sine and cosine values are computed for each direction $\theta_i$ to compute the *resultant* or *mean direction* for the set of angular data.

$$x_r = \sum \sin \theta_i$$
$$y_r = \sum \cos \theta_i$$

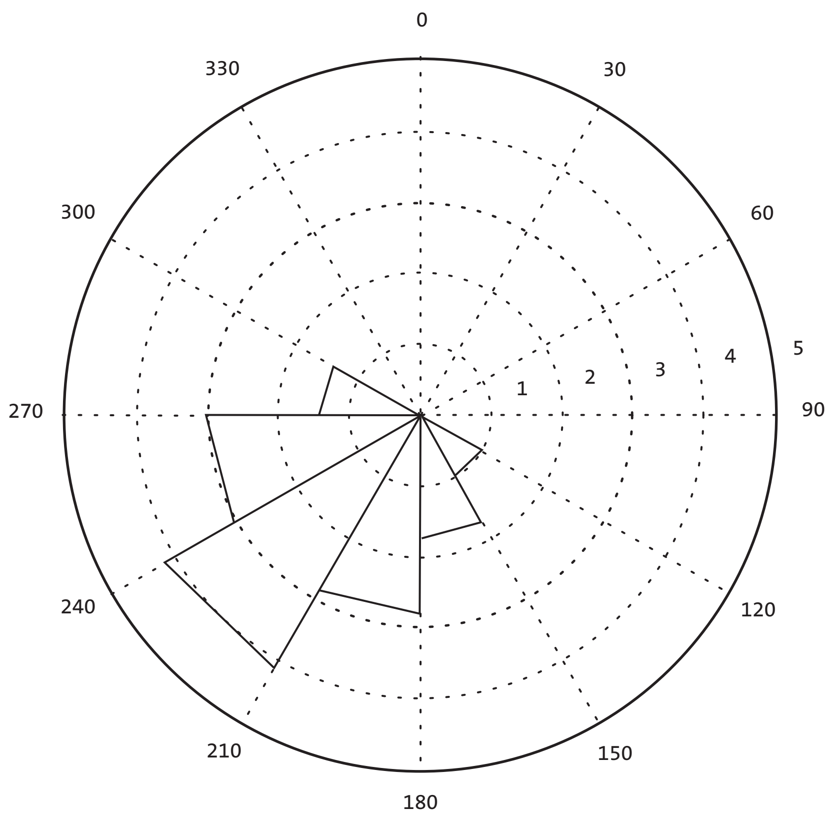


**Fig. 10.3** Modified rose diagram to display directional data using the function `rose_sqrt`. In this version of `rose`, 0° points due north, 90° points due east and the angles increase clockwise. The plot scales the rose diagram to the square root of the class frequency. The area of the arc segments now increases with frequency.

The resultant direction of the data set is

$$\bar{\theta} = \tan^{-1}(x_r / y_r)$$

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.

$$\bar{R} = R / N$$

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,

$$\sigma_0 = 1 - \bar{R}$$

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))
```

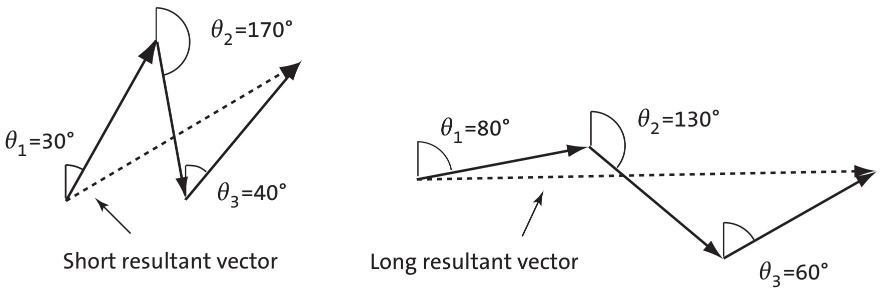


**Fig. 10.4** Resultant direction of directional data. The resultant length R of a sample decreases with increasing dispersion of the data θi..

```
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

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 of

```
Rm_1 = R_1 / (length(data_radians_1))

Rm_1 =
    0.8901
```

Higher `Rm` 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 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

$$f(\theta)=\frac{1}{2\pi I_0(\kappa)}e^{\kappa\cos(\theta-\mu)}$$

where $\mu$ is the mean direction and $\kappa$ is the concentration parameter (Fig. 10.4). $I_0(\kappa)$ is the modified Bessel function of the first kind and order zero of $\kappa$. 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

mu = 0; kappa = [0 1 2 3 4]';
```

In a first step, an angle scale for a plot that runs from –180 to 180 degrees is defined in 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;
```

In a second step, we compute the von Mises distribution for these values. The formula uses the modified Bessel function of the first kind and order zero that can be calculated by 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 parameter of such theoretical distributions are easily modified to compare them with empirical distributions.

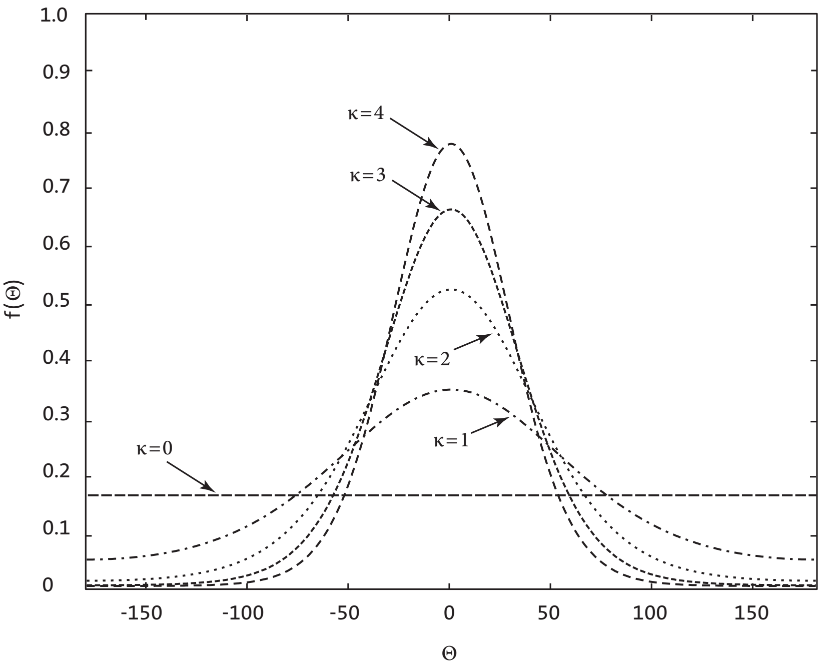


**Fig. 10.5** Probability density function $f(x)$ of a von Mises distribution with $\mu=0$ and five different values for $\kappa$.

## 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 $\chi^2$-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 `hist` to count the number of observations within 12 classes, each with a width of 30 degrees.

```
counts = hist(data_degrees_1,15:30:345);
```

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 $\chi^2$-test explores the squared differences between the observed and expected frequencies. The quantity $\chi^2$ is defined as the sum of these squared differences divided by the expected frequencies.

```
chi2 = sum((counts - expect).^2 ./expect)

chi2 =
    89.6000
```

The critical $\chi^2$ can be calculated by using `chi2inv`. The $\chi^2$-test requires the degrees of freedom $\Phi$. 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 $\Phi=12-(1+1)=10$. We test our hypothesis on a $p=95\%$ significance level. The function `chi2inv` computes the inverse of the cumulative distribution function (CDF) of the $\chi^2$ distribution with parameters specified by $\Phi$ for the corresponding probabilities in $p$.

```
chi2inv(0.95,12-1-1)

ans =
    18.3070
```

Since the critical $\chi^2$ of 18.3070 is well below the measured $\chi^2$ of 89.600, 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. This test uses the mean resultant length introduced in Section 10.3, which increases with a more significant preferred direction.

$$\bar{R} = \frac{1}{n}\sqrt{(\sum \sin\theta_i)^2 + (\sum \cos\theta_i)^2}$$

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`.

```
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` ($\alpha$=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. The test statistic for testing the similarity between two mean directions is the F-statistic (Section 3.7)

$$F = \left(1 + \frac{3}{8\kappa}\right) \frac{(n-2)(R_A + R_B - R_T)}{n - R_A - R_B}$$

where $\kappa$ is the concentration parameter, $R_A$ and $R_B$ are the resultants of samples A and B, respectively, and $R_T$ is the resultant of the combined samples. 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. The two mean directions are not significantly different if the measured F-value is lower than the critical F-value, which depends on the degrees of freedom $\Phi_a=1$ and $\Phi_b=n-2$, and also on the significance level $\alpha$. Both samples must follow a von 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;
```

**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, $\alpha$ | | | | |
|---|---|---|---|---|---|
| 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 |

```
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 ca. 223° and 201°. We also need the resultant length for both samples combined, so we combine both data sets and compute the resultant length again.

```
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))

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 $\alpha=0.05$, the critical value is

```
finv(0.95,1,78)

ans =
    3.9635
```

which is well below the observed value of `F`=12.5844. We therefore reject the null hypothesis and conclude that the two samples could have not been drawn from populations with the same mean direction.

**Table 10.2** Maximum likelihood estimates of concentration parameter κ for calculated mean resultant length (adapted from Batschelet, 1965 and Gumbel et al., 1953).

| R | $\kappa$ | R | $\kappa$ | R | $\kappa$ | R | $\kappa$ |
|---|---|---|---|---|---|---|---|
| 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 | | |

## 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, D.C.

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

Gumbel EJ, Greenwood JA, Durand D (1953) The Circular Normal Distribution: Tables and Theory. Journal of the American Statistical Association 48:131–152

Mardia KV (1972) Statistics of Directional Data. Academic Press, London

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

# General Index

## A

accessible population 2
adaptive filtering 185
adaptive process 185
addition 17
Aitchison's log-ratio transformation 306
alternative hypothesis 61
amplitude 176, 109
analog filters 161
analysis of residuals 92
angles 312
angular data 313, 314
anisotropy 247
ans 15, 27
answer 15
arithmetic mean 39, 206
array 14, 18
artifacts 211
artificial filters 162
ASCII 19
aspect 224, 226
ASTER 267
asterisk 17
autocorrelation 112
autocovariance 112
autoscaling 296, 306
auto-spectral analysis 112
available population 2
axes 34
axesm 196
axis 32, 84
azimuth 313

## B

background 282
bandpass filter 183, 184
bandstop filter 182, 184
bar plot 30
bars 30
Bartlett 115
bathymetry 196
Bernoulli distribution 53
Bessel function 318
besseli 318
bilinear interpolation 211
bimodal 40
binary digits 18
binomial distribution 53
bits 18, 257
bivariate analysis 79
bivariate data set 80
Blackman-Tukey method 112, 118
blank space 14
blind source separation 300
block kriging 252
BMP 260

DOI 10.1007/978-3-642-12762-5, © Springer-Verlag Berlin Heidelberg 2010

bootstrap 86, 94
bootstrp 86
box and whisker plot 47
boxplot 47
butter 182
Butterworth filter 182
bwboundaries 282
bwdist 284
bytes 15, 257

## C

canc 187
capital letters 15
case sensitive 15
cat 23
causal 167
causal indexing 172
causality 166
ceil 23
cell arrays 25
Cell Mode 35
central tendency 38
chaotic systems 149
char 24
χ2 distribution 59
χ2-test 70, 221, 320
chi2inv 72, 94
chi-squared distribution 59
chi-squared test 70
circshift 229
circular and spherical data 311
circular normal distribution 318
circular statistics 311
circular variance 316
clabel 209
class 21
classes 15, 38
classical regression 88
classical variogram 238
clear ans 15
closed data 6
cluster analysis 292, 304
clustered sampling 4
clustering 222
coastline data 194
cocktail party problem 300
coherence 117
colon operator 16
colorbar 199
colormap 200, 210, 264
column 14
comma 14
Command History 12, 13
Command Window 12, 13
comment 19, 28
complex 23
concentration parameter 318
confidence interval 101
confidence testing 92
continuity 164
contourf 209
contouring 204
Control+C 17
control points 204
conv 168, 169
convolution 167
cophenet correlation coefficient 308
corrcoef 83
corrected sum of products 82
correlation coefficient 80, 83, 308
correlation matrix 295
correlation similarity coefficient 305
counting particles 279
covariance 82
cp2tform 270
cpsd 123
createfigure 34
cross-correlation 117

cross-covariance 116
cross-spectral analysis 115
cross-spectrum 117
cross validation 98
Ctrl+C 17
cubic polynomial splines 206
cubic-spline interpolation 126
cumulative distribution function 51, 60
Current Folder 12, 13
curvilinear regression 100
cutoff frequency 182
cwt 143

## D

date 25
Daubechies 141
degrees of freedom 43
Delauney triangulation 206
DEM 224
demcmap 224
dendrogram 308
dependent variable 79, 88
descriptive statistics 37
difference equation 172
digital elevation models 23, 199, 224
digital filters 161
digital terrain elevation model 199
digitizing 193, 271
dilation 141
dimension 15
directional data 6, 311
directional data sensu stricto 311
directional variograms 247
dispersion 38, 42
display 29
distribution fitting 73
disttool 61
divergent flow 230
dots per inch 259
double 21
double percent signs 35
double precision 21
dpi 259
drifts 235
DTEM 199

## E

edge effects 213
edit 13
Editor 12, 13, 19, 20
Edit Plot 31, 32
element-by-element 18
elevation 313
ellipsis 92
embedding dimension 150
empirical distribution 37, 51
Enable Cell Mode 35
Encapsulated PostScript 260
end 26
EPS 260
error bounds 91
ETOPO2 196
evolutionary power spectrum 131
expectation-maximization (EM) algorithm 75
expected frequencies 71
experimental variogram 238
export data 18
exposure 224

## F

factor analysis 292
fast Fourier transformation 114
fast Fourier transform (FFT) 118

F distribution 58
fft 118
figure 34
Figure 30
Figure Toolbar 33
Figure Window 30, 31
File 13
File menu 13, 31, 34
filter 161, 168, 169, 183
filter2 224
filter design 182
filter weights 167
filtfilt 167, 184
find 47, 202
finite differences 225
finv 67
flow accumulation 229
for 26
format 22
Fourier transform 118, 174
frequencies 109, 112, 184
frequency characteristics 182
frequency distribution 51
frequency domain 174
frequency response 176, 183
frequency-selective filter 163, 183
freqz 179
F-statistic 322
F-test 66
function 26, 28

## G

gamma function 59
gaps 19
gaussian distribution 55
Gaussian mixture distributions 74
gaussian noise 188
general shape 38
Generate M-File 31, 33
geometric anisotropy 247
geometric mean 40
georeferencing 268
geostatistics 206, 235
ginput 272
global trends 235
gmfit 75
goodness-of-fit 91, 98
gradient 224
gradientm 226
grain size analysis 279
graph3d 210
graphical user interface 60
graphics functions 29
grayscale image 258
grid 32
griddata 208, 212
gridding 193, 204
grid points 204
GSHHS 194
GTOPO30 199
GUI 60

## H

Haar 141
Hamming 115
Hanning 115
harmonic mean 41
HDF 268
help 28
highest frequency 138
highpass filter 182, 183
hist 44
hist3 218
histogram 38
History 12
hold 30

Hough transformation 279
hypothesis 61
hypothesis testing 61
hypothetical population 2

## I

if 26
im2bw 282, 289
imadjust 280, 287
image processing 255
images 23, 255
imagesc 307
imclearborder 282
imfinfo 263
imhist 263
imopen 282, 287
import data 18
impulse response 173, 174
imshow 24, 262
imwrite 263
independent component analysis 292, 300
independent frequencies 137
independent variable 79, 88
indexed color image 264
indexing 16
inner product 17
inner-product similarity index 305
input 28
input signal 161
Insert Legend 32
int8 24
integer 23
intensity image 258
intensity values 258
interp1 127
interpolating 126
interval data 6
invertibility 165
iterations 189

## J

jackknife 86, 95
Joint Photographic Experts Group 261
JPEG 261

## K

K-means clustering 306
K-nearest-neighbors clustering 306
kriging 206, 235
kriging variance 248
kurtosis 43, 48

## L

label2grb 283
lag distance 238
lag tolerance 247
lag width 247
least-mean-squares algorithm 186
length 66
linear interpolation 126
linear kriging system 248
linear regression 88, 91
linear system 164
linear time-invariant filter 172
linear time-invariant systems 166
linear transformation 17
linear trend 83, 90
linkage 307
LMS algorithm 187
load 20
loads 294

local neighborhood 225
local trends 235
logarithmic normal distribution 56
log-normal distribution 56
lognormal kriging 238
log-ratio transformation 306
Lomb-Scargle algorithm 136
Lorenz system 148
lower-case letters 15
lowpass 182
LTI systems 166

## M

magnitude response 176
Manhattan distance 305
MATLAB 11
MATLAB Editor 19
matrix 14
matrix division 17
matrix element 15
matrix indexing 15
matrix multiplication 17
max 45
mean 39, 46, 55
mean centering 296, 306
mean direction 314, 316, 317
mean resultant length 316
mean-squared error 187
median 39, 46
mesh 209
meshgrid 200, 203
M-files 26
mgdistribution.fit 75
Microsoft Windows Bitmap Format 260
min 45
minput 272
missing data 19
mixed distributions 74
mixing matrix 301
mle 74
mode 40
modelfun 104
monitor 259
Morlet 141
mother wavelet 141
mscohere 124
multi-dimensional arrays 23
multimodal 40
multi-parameter methods 291
multiplication 17
multiplying element-by-element 18
multivariate data set 291

## N

NaN 19, 198
nanmean 48
natural filters 161
nearest-neighbor criterion 205, 221
nearest-neighbor distance 284
nested models 244
nextpow2 119
nlinfit 103, 104
noise 161, 185
nominal data 3
non-causal filters 167
nonlinear regression 103
nonlinear system 164
nonlinear time-series analysis 146
nonlinear trends 103
nonrecursive filters 172
normal distribution 55
normalizing 71
normfit 73

Not-a-Number 19, 198
nugget effect 244
null hypothesis 61
Nyquist frequency 109, 138, 182

## O

objective variogram modeling 245
observed frequencies 71
observed values 92
omnidirectional variograms 247
optimization problem 187
order of the filter 167
ordinal data 3
ordinary point kriging 247
oriented data 311
outlier 85
output 28
output signal 161
oversampling parameter 138
oversegmentation 285

## P

paired low and high 213
paleocurrent directions 312
passband 182
path 13
pathdef 13
pcolor 209, 224
PDF 261
pdist 223, 307
Pearson's correlation coefficients 80, 296
percentiles 40
percent sign 19
periodic component 109
periodogram 118, 173
phase 176
phase angles 109
phase shift 175
phase space portrait 146
phase space trajectory 147
picture elements 257
pixels 257
pixels per inch 259
plan 226
plot 29, 34
point distributions 216
point kriging 252
Poisson distribution 54
polyarea 286
polyfit 90
polytool 90
polyval 91
population 1, 37
Portable Document Format 261
position 141
PostScript 260
powers of matrices 17
ppi 259
predicted values 92
prediction error 98
predictor variable 88
primary input 187
principal component analysis 292, 293
principal component loads 294
principal components 293
principal component scores 294
princomp 294, 297
print 271
probability density function 51, 60
profile 226
Property Editor 32
PS 260
Publishing M-Files 35

## Q

quantifying charcoal 286
quantile 40, 46
quartiles 40
quintiles 40

## R

rand 21
randn 84
randomness 222
random noise 109
random number generators 21
random numbers 21, 59
random sampling 4
randtool 59
range 39, 41, 45, 242
raster data 193, 255, 256
ratio data 6
realization 161
rectangular distribution 51
recurrence plots 146
recursive filters 172
reduced major axis regression 89, 99
reduce the number of dimensions 292
reference input 187
regionalized variables 235
regression coefficient 89
regressor variable 88
regularity 222
regular sampling 4
resampling schemes 86
residuals 92
resolution 259
resultant direction 314, 315
resultant length 315
return 14
RGB 23, 258, 262
rgb2gray 280, 287
RGB composite 269
RMA regression 99
rolling die 52
rose 313
rose diagram 312
Rotate 3D 32
row 14
running mean 177

## S

sample 1, 37
sample size 2, 246
sampling design 246
sampling frequencies 108
sampling interval 108
satellite images 266
saturation 233
save 20
Save and Publish 35
Save as 31, 34
scal2frq 143
scalar 14
scale 141
scales 143
scaling 71
scatter plot 90
scores 294
scripts 26
seed 21
semicolons 14
semivariance 238
semivariogram 238
separated components 301
separation distance 247
separation vector 238
Set Path 13
shading 198

shape 38, 42
shoreline data 194
short-time Fourier transform 131
Shuttle Radar Topography Mission 201
signal 185
signal processing 161
Signal Processing Tool 130
significance 86
significance level 62
sill 242
similarity coefficient 305
similarity index 305
single 22
single precision 22
size 26
skewness 43, 48
slope 224
sortrows 103
spatial data 6
spatially distributed data 193
spatial sampling scheme 2
spectral leakage 114
splines 206
splines with tension 216
square brackets 14
squareform 223, 307
SRTM 201, 224
standard deviation 39, 41, 55
standard normal distribution 56
statistical significance 86
std 47
stem 175
step function 163
stopband 182
store data 18
stream power index 234
strel 282, 285, 287
structure arrays 24
Student's t distribution 57
subplot 30, 31
subtraction 17
sum 14
superposition 164
surf 200, 210
surface estimation 204
surfc 210
surrogates 86
system theory 161

## T

Tagged Image File Format 260
tangential curvature 226
t distribution 57
TERRA-ASTER satellite image 261
tform 270
theoretical distribution 37, 51
theory of regionalized variables, 235
thresholding 282, 286
TIFF 260
time delay 150
time delay embedding 147
time domain 174
time invariance 165
time series 14, 108
time-series analysis 107
title 32
Tools menu 31
topography 196
translation 141
transpose 17
trapz 76
trend 109
triangulation 205
trimodal 40
true color image 259
tsplines 216
t-test 61
ttest2 63

U

uint8 23, 262
unevenly-spaced data 126
unevenly spaced time series 135
uniform distribution 51
uniformity 222
uniform sampling 4
unimodal 40
unit impulse 163, 174
univariate analysis 37
unwrap 180

V

var 47
variables 15
variance 43
variogram 235
variogram cloud 239
variogram estimator 238, 240
variogram model 242
variography 238
vector data 193, 255, 256
vectors 14
vertcat 74
visualization 29
von Mises distribution 311, 318

W

watershed 227
watershed segmentation 279, 280, 284
watershed transform 285
wavelength 112
wavelet powerspectra 142
wavelet power spectrum 139
wavelets 141
wavelet transform 141
weighted mean 206
weighted regression 104
Welch's power spectral analysis 116
wetness index 233
whitening 300
whos 15, 16
windowed power spectrum 131
workspace 14
Workspace 12, 13

X

xlabel 32

Y

ylabel 32

Z

z distribution 56
zonal anisotropy 247
Zoom 32