Interactive Linear Algebra with R and Python

SAMUEL S.P. SHEN

Department of Mathematics and Statistics, San Diego State University,
and
Scripps Institution of Oceanography, University of California, San Diego

August 2024 Version for SDSU Course Math 524

Contents

Give the pupils something to do, not something to learn; and doing is of such a nature as to demand thinking; learning naturally results.

— John Dewey

Preface

According to `Encyclopedia.com`, "linear algebra originated as the study of linear equations." Linear algebra deals with vectors, matrices and vector spaces. Before the 1950s, it was part of Abstract Algebra (Tucker 1993). For example, the book "A Survey of Modern Algebra" by Garrett Birkhoff and Saunders Mac Lane (3rd edition, Macmillan Company, New York, 1965) contains three chapters of linear algebra materials out of its 15 chapters: Chapter 7 (Vectors and Vector Spaces), Chapter 8 (The Algebra of Matrices), and Chapter 10 (Determinants and Canonical Forms).

In 1965 the Committee on the Undergraduate Program in Mathematics, Mathematical Association of America (MAA), outlined the following topics for a stand-alone linear algebra course: Linear systems, matrices, vectors, linear transformations, unitary geometry with characteristic values.

The first stand-alone book that was named "Linear Algebra" in the United States might be that by Charles W. Curtis of the University of Wisconsin-Madison, published in 1963 by Allyn and Bacon, Inc., Boston. The book title is "Linear Algebra: An Introductory Approach." This book contains all the materials defined by MAA in 1965. Most linear algebra books today would also cover these materials, plus some modern materials on singular value decomposition (SVD) (Strang 2016). Modern advanced materials may include spectral theorem, Jordan forms, polynomials, and Cayley-Hamilton theorem (Garcia and Horn 2022).

Paul R. Halmos' book "Finite-Dimensional Vector Space" was mostly about linear algebra although did not name so. This book was first published in 1942 and had its second edition in 1958. However, the book missed an important part of linear algebra: The solution of a set of linear equations, which became extremely important in the 1960s when computer applications for solving many linear equations in engineering were popular due to the invention of the finite element method.

Although Jean Dieudonne's book "Linear Algebra and Geometry" in 1964 bore the name of Linear Algebra, focused mainly on linear transforms, and had no solutions of linear equations either. Ross A. Beaumont's 1965 book also had its name "Linear Algebra", but again contained no materials on linear equations.

Modern linear algebra books have more materials on rectangular matrices, SVD, data science applications, graphic visualization, and computer codes. Our book is in this category in terms of materials. In the methodology aspect of learning and teaching, we use progressive education pedagogy and emphasize intuition, graphics, storytelling, computing, and rigorous proof.

What are the distinguished features of this book?

Every chapter starts with elementary materials and common sense, and can be understood without reading previous chapters. Almost all the chapters include advanced materials which are prepared for the second-semester linear algebra course, or called advanced linear algebra. These advanced materials can be used as a research handbook or as advanced examples to support a research project. Different from the traditional linear algebra book, chapters are entitled according to the nature of the relevant theory, e.g., linear transform and vector space, our chapters are entitled according to the usage or functions of a method, e.g., Data matrices, operations, and visualizations for Chapter 1, and Space-time decomposition: SVD for Chapter 2. We follow the RUM principle when selecting materials. Here R is for Relevant to both courses and research in the science; U is for Useful to not only the courses and research, but also to future jobs; and M is for Modern, incorporating recent progress and tools, especially progress in computing and data science. In this book, our modern materials are reflected in SVD applications, matrix visualization, machine learning, and the use ChatGPT, Python, and R.

Meeting these RUM criteria in courses, including linear algebra, will require time and commitment to changing the curricula of relevant study programs to better support students learning. Progress will be incremental. Mathematics courses and instructors can both evolve to help meet the objective of well-educated students prepared for successful careers.

In terms of the book style, we follow the principle of interaction. We make sure that each method has a computer code to allow a reader to interact with the book. Traditional linear algebra books, like most other math books, often stack with definitions of mathematical objects, such as subspace, and isomorphism, dense formulas, and awkwardly-worded theorems. The theorem statements are only comprehensible by professional mathematicians in the name of "mathematical rigor," with which mathematicians are extremely proud of.

Bearing in mind that most students of linear algebra are not going to become professional mathematicians, this book tries to do it differently. We start with common sense, data, visualization, examples, and interpretations that lead to names, definitions, and theorems. Most math objects in our book are named after the objects have appeared in a computational process or an algorithm, so that students have an image in their mind before we name them. This is like that the 10-year-old Mike learns to name Uncle Tom after seeing him, in contrast to the legendary Uncle Tom who suddenly appears in front of Mike.

Despite our emphasis on the details of modern and useful materials, the book still includes almost all the contents of a conventional linear algebra book, such as vector spaces, inner product space, linear equations, orthogonality, eigenvectors, SVD, linear transform, polynomials, operators, spectral theory, Jordan forms, Cayley-Hamilton theorem, and determinants. Of course, some materials are very brief, such as determinants.

Another feature of this book is to minimize the use of mathematical symbols and notations. We understand that professional mathematicians do not mind fancy

notations, but non-mathematicians may be scared away by them. Instead, we use some repeated notations for matrices. When we introduce a name and a notation, we attempt to justify its use from the perspectives of history, interpretation, and applications. This way may help a reader to develop a picture in her mind when reading a method, or a definition. We use visualization as much as we can, such as the visualization of a data matrix, and that of left and right eigenvectors from SVD.

What is the philosophy of our pedagogy?

We follow the education theory of *learning-by-doing*, which in our case means using your computer and our R or Python code to interact with our book and modifying our computer codes and websites to analyze your own data and generate corresponding figures. Learning-by-doing is the core methodology of the progressive education of John Dewey (1859-1952), an American philosopher and educator. Dewey felt that the experience of students and teachers together yields extra value for both. Instructors are not just to lecture and project authority, instead they are to collaborate with students and guide to students to gain experience of solving problems of their interest. Although Dewey's education theory was established initially for school children, we feel that the same is applicable to undergraduate and graduate students. Our way of learning-by-doing is to enable students to use R or Python code and other resources in the book and its website to reproduce the figures and numerical results in the book. Further, students can modify the computer code and solve their own problems, such as visualizing the health data or weather data in a similar format and of their own interest, or analyzing their own data using a similar or a modified method. Thus, the audience interaction with the book is the main innovative feature of our book, which allows the audience to gain experience of practicing, thinking, applying, and consequently understanding. The ancient Chinese educator Confucius (551-479 BC) said that "*I hear, and I forget; I see, and I remember; and I do, and I understand.*" Although John Dewey and Confucius were approximately 2,500 years apart, they shared a similar philosophy of learning-by-doing.

As illustrated in Fig. 0.1, our pedagogy has three stages: Do, reflect, and apply. The author has systematically practiced this pedagogy since 2015 when he created and taught the course *Climate Mathematics* at the Scripps Institution of Oceanography. Usually, he presents a question or a problem at the beginning of a class. Then he asks students to orally ask the same question, or describe the same problem, or answer his question in their own words. For example, why the tiny amount of carbon dioxide in the atmosphere is important for climate change? Next, he and students search and re-search for data and literature, work on the observed carbon dioxide data at Mauna Loa using computer data visualization to see the 30% increase of carbon dioxide in the atmosphere since 1958, and discuss the structure of greenhouse gasses whose molecules have three or more atoms. Next, he encourages his students to share their experiences with their grandparents, other family

Fig. 0.1 John Dewey's pedagogy of learning-by-doing: A cycle from a problem to skill training via solving the problem, to reflection, to success and applications, and to reward and new problems. It is a cycle of understanding and progress. The size of each piece of the "Do, Reflect, Apply" pie approximates the proportion of the learning effort. "Do" is emphasized here, indicated by the largest piece of the pie.

members, or friends. Finally, students apply the skills gained to solve their own problems with their own data, by doing homework, working on projects, finding employment, or making innovations. In this cycle, students have gathered experience and skills to improve their life and to engage with the community. In the short term, students trained in this progressive cycle of learning-by-doing have a better chance to become a good problem solver, a smooth story narrator, and an active leader in a research project in a lab or an internship company, and consequently become competitive in the job market. In the long term, the students trained in this cycle are likely to become to a life-time learner and educator. John Dewey said that "*Education is not preparation for life but life itself.*" We would like to modify this to that "Education is not only preparation for a better life, but also is life itself."

Dewey's progressive education theory is in a sharp contrast to the traditional

learning process based on the logic method, which aims at cultivating highly achieved scholars. The commonly used pedagogy of lecture-read-homework-exam often uses the logic-based approach. The climax of the logic-based education is that the instructor has the pleasure to enjoy presenting her method and theory, while students are so creative that they will produce a new or a better theory. Many outstanding scholars went through their education this way, and many excellent textbooks were written in this approach. However, our book does not follow this approach, since our book is written for the general population of students, not just for the elite class or even scholars-to-be. If you wish to be such an ambitious scholar, then you may use our book very differently: you can read the book quickly and critically for gaining knowledge instead of skills, and skip our reader-book interaction functions.

Our pedagogy is result-oriented. Our philosophy is to minimize the mathematical challenge, but to deepen the understanding of the ideas using visual tools and using story-telling experience. Our audience will be able to make an accurate problem statement, ask pointed questions, set up linear algebra models, solve the models or establish theorems, prove the theorems in both intuitive and rigorous ways, and interpret solutions and theorems using both daily life language and rigorous mathematical language. Therefore, instead of training the few "mechanics" with a high risk of failure, we wish to train a large number of good "drivers" with a large probability of success, although this book also tries to inspire a few students to become "mechanics."

Samuel S. P. Shen, San Diego, California, August 2024

References:

1. Beaumont, R. A., 1965. Linear Algebra. Harcourt, Brace & World, Inc., New York.
2. Birkhoff, G. and Mac Lane, S., 2017. A survey of modern algebra. 3rd edition, Macmillan Company, New York, 1965.
3. Curtis, C.W., 1963. Linear algebra: an introductory approach. Allyn and Bacon, Inc., Boston.
4. Garcia, S.R. and Horn, R.A., 2023. Matrix Mathematics: A Second Course in Linear Algebra. 2nd edition, Cambridge University Press.
5. Grothendieck, A., Dieudonn, J. and Dieudonn, J., 1964. Elments de gomtrie algbrique. Hermann, Paris.
6. Halmos, P.R., 1958. Finite-dimensional vector spaces. 2nd edition, D. Van Nostrand Company Inc., Princeton.
7. Strang, G., 2016. Introduction to linear algebra. 5th edition, Wellesley-Cambridge Press.
8. Tucker, A., 1993. The growing importance of linear algebra in undergraduate mathematics. The college mathematics journal, 24(1), pp.3-9.

Acknowledgements

We thank the friendly editorial team of Cambridge University Press (CUP) for their professional and high quality work.

1 Basic Matrix Operations

Data matrices appear everywhere in our daily life, such as the daily body weight data of 100 clients of your diet clinic in the last 21 days. If you put each client's data on a row, then the row has 21 data entries. This row may be called a data sequence, or a row vector. You stack the data sequence of all the 100 clients together one above another ordered according to their last name. You thus have created a rectangular 2-dimensional array of numbers, a 100-by-21 matrix. This data matrix has 100 rows and 21 columns.

You may add client names for the row names and date for the column names to the data matrix and form a dataframe. Dataframe is a commonly used concept in R and Python data analysis. Dataframe has a nice presentation of data and also allows you to name the data in rows and columns, such as John Smith's body weight data, in contrast to referring to the 83rd row.

This chapter is limited to (i) Describing the basic matrix operations commonly used in your coursework or career, e.g., eigenvector decomposition of a matrix and solution of linear equations; (ii) presenting matrix application examples of data science, e.g., sub-matrix generation, array-matrix conversion, and data statistics; and (iii) solutions of linear equations.

1.1 Matrix as a data array

In general, a matrix is a rectangular array of numbers or symbols or expressions, which are called elements, arranged in rows or columns. A table such as that shown in Fig. 1.1 is a matrix, consisting of N rows and Y columns of numbers. Figure 1.1 shows the 10 ten-year annual precipitation anomalies from the year 1900 to 1909 for the 15 five-degree latitude-longitude boxes centered at $2.5°E$ longitude for different latitudes over the Northern Hemisphere ranging from latitudes $2.5°N$ to $72.5°N$. The matrix shown in Fig. 1.1 thus has 15 rows ($N = 15$), and 10 columns ($Y = 10$). An anomaly of a climate parameter is defined as its actual value minus its normal value that is an average of 30 or more years.

Many other types of climate data can also be represented as matrices (which is the plural of matrix). Precipitation data [units: mm/day] at multiple stations and multiple days can also form a matrix, normally with stations [each marked by a station identifier, or station ID] represented in rows, and time [units: day] represented in columns. The daily minimum surface air temperature (Tmin) data

Lat	Lon	1900	1901	1902	1903	1904	1905	1906	1907	1908	1909
2.5	2.5	0.283240	-0.131860	-0.190500	0.160040	-0.878110	0.080356	0.059193	-0.136900	0.200420	0.822600
7.5	2.5	0.172670	0.830550	-0.180350	-0.203630	-0.238590	0.425310	0.002805	0.102780	0.254050	0.516200
12.5	2.5	0.024392	0.152030	-0.034115	-0.062696	-0.192070	0.074360	0.201970	-0.011311	0.035259	0.272010
17.5	2.5	0.006780	0.066783	-0.084581	-0.008636	-0.038109	-0.001092	0.088250	0.011047	0.029358	0.082329
22.5	2.5	0.021162	0.079977	0.020016	-0.022142	-0.027032	0.065704	0.012937	-0.003823	0.032545	0.028636
27.5	2.5	0.049846	0.057413	0.026621	0.019914	-0.002651	0.071242	0.012837	0.001567	0.051857	0.099650
32.5	2.5	0.107740	0.143510	0.061613	0.076137	0.147760	0.137890	-0.074612	0.110300	0.087752	0.126920
37.5	2.5	0.128250	0.211940	0.113010	0.027472	0.183710	0.125550	-0.267500	0.215980	0.007609	0.055573
42.5	2.5	0.158490	0.800950	0.292690	0.172930	0.272010	0.126370	-0.017183	0.184880	0.118980	0.200520
47.5	2.5	-0.112800	0.243130	-0.121630	-0.076247	-0.047231	0.110160	0.080978	-0.091371	0.016172	-0.060487
52.5	2.5	-0.199840	-0.381070	-0.217570	-0.107760	-0.124700	-0.117470	-0.062448	-0.171070	-0.277650	-0.132690
57.5	2.5	-0.076619	-0.515070	0.005342	0.016647	0.137820	0.038041	0.131370	-0.196490	-0.132480	0.014887
62.5	2.5	-0.261760	-0.402600	0.137200	-0.214960	0.249210	0.147550	0.866120	-0.453910	-0.026134	0.053409
67.5	2.5	0.034079	0.223610	0.314090	-0.044832	0.130470	0.201260	0.554170	-0.054434	0.185870	0.308950
72.5	2.5	-0.119680	0.022949	0.004324	-0.050248	0.251330	-0.233080	-1.043800	0.363850	-0.315400	-0.113080

Fig. 1.1 Annual precipitation anomalies data of the Northern Hemisphere at longitude $2.5°E$ [units: mm/day]. The annual total of the anomalies should be multiplied by 365.

for the same stations and the same days form another matrix. In general, a space-time climate data table always forms a matrix. Conventionally, the spatial locations correspond to the rows, and the time coordinate corresponds to the columns.

Another matrix example, taken from everyday life, is that the ages of members of an audience, sitting in a movie theater in seats arranged in rows and columns, also form a matrix. The weights of these audience members form another matrix. Their bank account balances form still another matrix, and so on. Thus, a matrix is a data table, and extensive mathematical methods have been developed in the 20th century to study matrices. Computer software systems, such as R, have also been developed in recent years that greatly facilitate working with matrices.

This chapter will discuss following topics:

(i) Matrix algebra of addition, subtraction, multiplication, and division (i.e., inverse matrix));

(ii) Linear equations;

(iii) Space-time decomposition, eigenvalues, eigenvectors, and the climate dynamics interpretation of a space-time climate data matrix;

(iv) A matrix application example in balancing the mass in a chemical reaction equation by solving a set of linear equations; and

(v) A matrix application in multivariate linear regression.

1.2 Matrix algebra

Matrix algebra is quite different from the algebra for a few scalars of x, y, z as we learned in high school. For example, matrix multiplication does not have the commutative property, i.e., matrix A times matrix B is not always the same as matrix B times matrix A. This section describes a set of rules for doing matrix algebra.

1.2.1 Matrix equality, addition and subtraction

An $m \times n$ matrix A has m rows and n columns and can be written as

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \tag{1.1}$$

or

$$A_{m\times n} = [a_{ij}], \tag{1.2}$$

or simply

$$A = [a_{ij}], \tag{1.3}$$

where $a_{ij}, i = 1, \cdots, m, j = 1, \cdots n$ are the mn elements of the rectangular matrix A, and $m \times n$ is often called the size or order of a matrix. We say that A is an m times n matrix, or an m-by-n matrix. The elements of the matrix shown in Fig. 1.1 are the precipitation anomaly data, $m = 15$, and $n = 10$. Therefore, Fig. 1.1 is a 15 times 10 matrix.

Matrix $A = [a_{ij}]$ is equal to matrix $B = [b_{ij}]$ if and only if $a_{ij} = b_{ij}$ for all the elements. That is, A is identical to B. We can understand this by considering the two identical photos. A black and white photo is a matrix of pixel brightness values. Two photos are identical only when the corresponding brightness values of the two photos are the same. Thus, when considering that a matrix is equal to another, we may regard the equality as two same-size photos, maps, or climate charts being identical to one another.

Matrix addition is simply the addition of the corresponding elements:

$$A + B = [a_{ij} + b_{ij}]. \tag{1.4}$$

Of course, two matrices can be added together only when they have the same size $m \times n$. If the two matrices represent dimensional data, such as precipitation, then their units must be the same in order to add corresponding elements. However, each element in a matrix can represent a different climate parameter. For example, a climate data matrix for San Diego for 24 hours may have its first row representing temperature, the second precipitation, the third atmospheric pressure, the fourth

relative humidity, etc, while the first columns represent time from 1:00 (i.e., 1:00 AM) to 24:00 (i.e., 12:00 AM).

Matrix subtraction is defined in a similar way:

$$A - B = [a_{ij} - b_{ij}]. \tag{1.5}$$

Example 1.1 Matrix subtraction:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} - \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -2 & -1 \end{bmatrix} \tag{1.6}$$

The R code for the above matrix subtraction can be written as follows:

```
matrix(c(1,1,1,-1), nrow=2) - matrix(c(1,3,2,0), nrow=2)
#      [,1] [,2]
#[1,]    0   -1
#[2,]   -2   -1
```

One can use matrix subtraction to quantify the differences of climate between two space-time domains when the climate data for each domain are in a matrix form.

1.2.2 Matrix multiplication

While the matrix addition and subtraction are similar to those of scalars, the matrix multiplication has several distinct properties.

1.2.2.1 A row vector times a column vector

The single-column n-row matrix is often called an n-dimensional vector, or an n-dimensional column vector. Similarly, one can define an n-dimensional row vector as a single-row n-column matrix.

A row vector of n elements times a column vector of the same number of elements is equal to a scalar, which is the sum of the products of each pair of corresponding elements:

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = a_1 b_1 + a_2 b_2 \cdots + a_n b_n \tag{1.7}$$

This is also called the dot product of two vectors $\mathbf{a} = (a_1, a_2, \cdots, a_n)$ and $\mathbf{b} = (b_1, b_2, \cdots, b_n)$, denoted by

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 \cdots + a_n b_n. \tag{1.8}$$

In 2- or 3-dimensional space, the dot product of two vectors has a simple geometric interpretation:

$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}||\ ||\mathbf{b}|| \cos\gamma, \tag{1.9}$$

where $||\mathbf{a}||$ stands for the length of a vector $\mathbf{a}$, and γ is the angle between $\mathbf{a}$ and $\mathbf{b}$. The proof of the equivalence of the above two expressions (1.7) and (1.9) is given in Appendix A.

When $\mathbf{a}$ is force and $\mathbf{b}$ is the displacement of an object moved by the force, then $\mathbf{a} \cdot \mathbf{b}$ is the work done by the force on the subject.

Example 1.2 Dot product of two vectors in a 2-dimensional space:

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1 \tag{1.10}$$

The same result can be obtained from the geometric formula (1.9). The length of the first vector is $\sqrt{2}$ since it is the diagonal vector of a unit square in the first quadrant, and the length of the second vector is 1 since it is the unit square's side on the x-axis. The angle between the two vectors is 45°. Thus, the dot product of the two vectors is $\sqrt{2} \times 1 \times \cos(45°) = 1$.

```
#R code for dot product
#install.packages('geometry')
library(geometry)
a = c(1, 1)
b = c(1, 0)
# Calculating dot product using dot()
dot(a, b)

#Another way to compute dot product is to
a%*%b
#      [,1]
#[1,]    1

#Calculate the angle between two vectors
am = norm(a, type="2")
am
#[1] 1.414214
bm = norm(b, type="2")
bm
#[1] 1
angleab = acos(dot(a, b)/(am*bm))*180/pi
angleab
#[1] 45 degrees
```

1.2.2.2 A scalar times a matrix

A scalar c times a matrix $A = [a_{ij}]$ is formed by multiplying the scalar into every element of the matrix.

$$c \times A = [c \times a_{ij}]. \tag{1.11}$$

The product is again a matrix of the same size. A physically meaningful product requires that the dimension of the scalar and the dimensions of the matrix elements are compatible. For example, if the matrix is the price data of many products, and if the scalar is the number of sales of each product, then the scalar times the matrix gives the revenue matrix of all the products.

Example 1.3 A scalar 3 times a 2-by-2 matrix

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

is

$$3 \times A = 3 \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 3 & -3 \end{bmatrix} \tag{1.12}$$

```
#R code for a scaler times a matrix
A = matrix(c(1,1,1,-1), ncol = 2, byrow=T)
A
#      [,1] [,2]
#[1,]    1    1
#[2,]    1   -1
3*A
#      [,1] [,2]
#[1,]    3    3
#[2,]    3   -3
```

1.2.2.3 An m-by-n matrix times an n-by-k matrix

The multiplication of two matrices is defined as a set of dot products between the row vectors of the first matrix and the column vectors of the second matrix. Because of the requirement to form dot products, the column number of the first matrix must be the same as the row number of the second matrix. $A_{m\times n}B_{n\times k}$ is defined as a new matrix $C_{m\times k} = [c_{ij}]$, where the element c_{ij} is the dot product of the ith row vector of $A_{m\times n}$ and jth column vector of $B_{n\times k}$. Therefore, we may write,

$$A_{m\times n}B_{n\times k} = \left[\sum_{l=1}^{n} a_{il}b_{lj}\right]_{m\times k}. \tag{1.13}$$

Thus, $A_{m\times n}B_{n\times k}$ is equal to a matrix of $m \times k$ dot products, and this can be tedious to compute.

Example 1.4 Matrix multiplications:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 7 \\ -1 & -1 \end{bmatrix}, \tag{1.14}$$

and

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 4 & -2 \\ 6 & -2 \end{bmatrix}. \tag{1.15}$$

In the above formula (1.14), the first first element of the right-hand-side matrix is 3, which is the product of the first row vector of A: $(1, 1)$, and the first column vector of B: $(1, 2)$. Their dot product is

$$(1, 1) \cdot (1, 2) = 1 \times 1 + 1 \times 2 = 3. \tag{1.16}$$

In the same way, one can verify every element of the above multiplication results.

These matrix products can be computed by the following computer code.

```
#R code for matrix multiplication by command %*%
A = matrix(c(1,1,1,-1), nrow=2)
B = matrix(c(1,2,3,4), nrow=2)
A%*%B
#      [,1] [,2]
#[1,]    3    7
#[2,]   -1   -1
B%*%A
#      [,1] [,2]
#[1,]    4   -2
#[2,]    6   -2
```

In the this example, the second matrix multiplication (1.15) involves the same matrices as the first one (1.14), but in a different order: If eq. (1.14) is denoted by AB, then eq. (1.15) is BA. Clearly, the results are different. In general,

$$AB \neq BA \tag{1.17}$$

for a matrix multiplication. Thus, matrix multiplication does not have the commutative property which the multiplication of two scalars x and y does have: $xy = yx$.

1.2.2.4 Transpose matrix, and diagonal, identity and zero matrices

Although a space-time climate data matrix often has its rows to mark spatial locations and columns to mark time, the row and column roles may need to exchange for some applications, which use rows to mark time and columns to mark spatial locations. This operation of exchanging rows and columns is called matrix transpose. The transposed matrix of A is denoted by A^t. The columns of A^t are the rows of A.

Example 1.5

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, A^t = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \tag{1.18}$$

The transpose can be computed by the following computer code.

```
#R code for matrix transpose
A = matrix(c(1,2,3,4), ncol =2, byrow=T)
A
#       [,1] [,2]
#[1,]    1    2
#[2,]    3    4
t(A)
#       [,1] [,2]
#[1,]    1    3
#[2,]    2    4
```

It is obvious that

$$(A + B)^t = A^t + B^t. \tag{1.19}$$

However, a true but less obvious formula is the transpose of a matrix multiplication:

$$(AB)^t = B^t A^t. \tag{1.20}$$

When a matrix whose only non-zero elements are on the diagonal of elements, this matrix is call a diagonal matrix:

$$D = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix}, \tag{1.21}$$

An identity matrix is a special type of diagonal matrix, whose diagonal elements are all one and whose off-diagonal elements are all zero, and is denoted by I:

$$I = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}, \tag{1.22}$$

which plays a role in matrix operations similar to the role of the value 1.0 in the familiar real number system.

A zero matrix is a matrix whose elements are all zero. If two matrices are the same, then their difference is a zero matrix.

These matrices may be generated by the following computer code.

```
#Generate a diagonal matrix
D = diag(c(2, 1, -3))
round(D, digits = 0)
#      [,1] [,2] [,3]
#[1,]    2    0    0
#[2,]    0    1    0
#[3,]    0    0   -3

#Generate an 3-dimensional identity matrix
I = diag(3)
I
#      [,1] [,2] [,3]
#[1,]    1    0    0
#[2,]    0    1    0
#[3,]    0    0    1

#Generate a 2-by-3 zero matrix
M = matrix(0, 2, 3)
M
#      [,1] [,2] [,3]
#[1,]    0    0    0
#[2,]    0    0    0
```

1.2.2.5 Matrix division and inverse

The division of a scalar y by another non-zero scalar x can be written as y times the inverse of x:

$$y/x = y \times x^{-1}. \tag{1.23}$$

Thus, the division problem becomes a multiplication problem when the inverse is found. The inverse of x is defined as $x^{-1} \times x = 1$.

Matrix division is defined in the same way:

$$A/B = A \times B^{-1}, \tag{1.24}$$

where B^{-1} is the inverse matrix of B defined as

$$B^{-1}B = I\ , \tag{1.25}$$

where I is the identity matrix.

Example 1.6 The R command to find the inverse of matrix A is `solve(A)` as shown by a numerical example below.

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^{-1} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \tag{1.26}$$

You can compute and verify this "by hand", or by the following computer code

```
#R code to compute the inverse of a matrix
A = matrix(c(1,1,1,-1), nrow=2)
invA = solve(A)
```

```
invA
#      [,1] [,2]
#[1,]  0.5  0.5
#[2,]  0.5 -0.5

#Verify the inverse
invA %*% A
#      [,1] [,2]
#[1,]    1    0
#[2,]    0    1
```

Finding the inverse matrix of a matrix "by hand" is usually very difficult and involves a long procedure for a large matrix, say, a 4×4 matrix. Modern climate models can involve multiple $n \times n$ matrices with n from several hundred to several million, or even billion. For detailed information, see the excellent text *Introduction to Linear Algebra* by Gilbert Strang (2016).

1.3 Some Useful Operations for Matrices in Data Science but Not Covered in Traditional Linear Algebra Books

1.3.1 Delete a Row or Column or Both; a Sub-Matrix

A column of matrix A may contain some unwanted data, or missing data. You wish to delete this column, say the mth row. The resulting matrix is denoted by $A[-m,]$.

Similarly, you may wish to delete a column. The result is denoted by $A[,-n]$.

Or you may wish to delete the mth row and nth column at the same time. The result is denoted by $A[-m,-n]$.

Still, another case is that you may wish to choose a sub-matrix, such as from ith row to the jth row and from kth column to lth column. The result is denoted as $A[i:j,k:l]$.

These operations can be made by the following computer code.

```
#R code to delete rows or/and columns
A = matrix(1:9, nrow = 3)
A
#      [,1] [,2] [,3]
#[1,]    1    4    7
#[2,]    2    5    8
#[3,]    3    6    9

A[-3,]
#      [,1] [,2] [,3]
#[1,]    1    4    7
#[2,]    2    5    8
```

```
A[,-1]
#      [,1] [,2]
#[1,]    4    7
#[2,]    5    8
#[3,]    6    9
A[-3,-1]
#      [,1] [,2]
#[1,]    4    7
#[2,]    5    8

A[1:2, 2:3] #Sub-matrix
#      [,1] [,2]
#[1,]    4    7
#[2,]    5    8

A[1:2, ] #keep all the columns
#      [,1] [,2] [,3]
#[1,]    1    4    7
#[2,]    2    5    8
```

1.3.2 Insert Rows or/and Columns to a Matrix

You may wish to insert an extra row or a column to matrix A. The row may be inserted at any position, on the top, at the bottom, or immediately below the mth row of A. Below are some examples.

```
#Insert a row or column to a matrix
A = matrix(1:4, nrow = 2)
br = 5:6
bc = 7:8
rbind(A, br)
#    [,1] [,2]
#1    3
#2    4
#br    5    6

rbind(br, A)
#    [,1] [,2]
#br    5    6
#      1    3
#      2    4

rbind(rbind(A[1,], br), A[2,])
#    [,1] [,2]
#      1    3
#br    5    6
#      2    4
Abc = cbind(A, bc)
Abc
Abc = cbind(A, bc)
#         bc
#[1,] 1 3  7
#[2,] 2 4  8
```

```
cbind(Abc, A)#stack two matrices
#          bc
#[1,] 1 3  7 1 3
#[2,] 2 4  8 2 4
```

1.3.3 Statistics of the row or column data

For a data matrix, you may wish to calculate the statistical indices of each row or column, such as mean and standard deviation. Below are some examples of these calculations.

```
#Row or column statistics
A = matrix(1:6, nrow = 2)
rowSums(A)
#[1]  9 12
rowMeans(A)
#[1] 3 4
rowCumsums(A) #cumulative sum
#      [,1] [,2] [,3]
#[1,]    1    4    9
#[2,]    2    6   12
colMeans(A)
#[1] 1.5 3.5 5.5
library(matrixStats) #SD needs the library
rowSds(A)
#[1] 2 2
colSds(A)
#[1] 0.7071068 0.7071068 0.7071068
```

1.3.4 Sweep a matrix by a vector

1.3.4.1 Subtraction sweeping

You may wish to subtract a vector of length n from every row of an $m \times n$ matrix. This is an operation of subtraction sweeping.

```
#Sweep a matrix by a vector using subtraction: R code
A = matrix(1:6, nrow=2, byrow = TRUE)
A
#      [,1] [,2] [,3]
#[1,]    1    2    3
#[2,]    4    5    6
u = 1:3
Br = sweep(A, 2, u) #2 means sweep every row
Br
#      [,1] [,2] [,3]
#[1,]    0    0    0
#[2,]    3    3    3
v= 1:2
Bc = sweep(A, 1, v) #1 means sweep every column
Bc
#      [,1] [,2] [,3]
#[1,]    0    1    2
```

```
#[2,]    2    3    4

c = colMeans(A) #means of each column
sweep(A, 2, c) #compute the anomaly data matrix
#[1,] -1.5 -1.5 -1.5
#[2,]  1.5  1.5  1.5

sin(A)#function operation on each matrix element
#           [,1]       [,2]       [,3]
#[1,]  0.8414710  0.9092974  0.1411200
#[2,] -0.7568025 -0.9589243 -0.2794155

A^2 #not equal to A%*%A
#     [,1] [,2] [,3]
#[1,]    1    4    9
#[2,]   16   25   36
```

The subtraction sweeping is useful in data science, such as computing the anomalies of data matrix A by subtracting the mean of each column. In this case, you may use R command `c = colMeans(A)`, and the anomaly data matrix would be the result of row sweeping by subtraction.

This example also shows how to apply a function to each element of a matrix, such as R command `sin(A)`. This yields a matrix of elements $sin(a_{ij})$.

If you wish to sweep by addition, you can just sweep $-u$ or $-v$.

1.3.4.2 Multiplication sweeping

You can also sweep a matrix A by multiplication with a vector v for every column. The R command is $v*A$. This command makes vector $v_{m\times 1}$ time the first n elements of the matrix A, counting from the first column. If $A_{m\times n}$, then $v * A$ makes each column of $A_{m\times n}$ time vector $v_{m\times 1}$ for every pair of corresponding elements. The computer code for an example is as follows.

```
#R sweep a matrix by a vector using multiplication
A = matrix(1:6, nrow=2, byrow = TRUE)
w = 1:2
w*A
#     [,1] [,2] [,3]
#[1,]    1    2    3
#[2,]    8   10   12
A*w # yields the same result as w*A
w3 = 1:3
#sweep each row by transposing A
t(w3*t(A))
#     [,1] [,2] [,3]
#[1,]    1    4    9
#[2,]    4   10   18
w3*A ##multiplication sweep by row-dimensions not matching
#     [,1] [,2] [,3]
#[1,]    1    6    6
#[2,]    8    5   18

A/w #sweeping by division
```

```
#      [,1] [,2] [,3]
#[1,]    1  2.0    3
#[2,]    2  2.5    3
```

When the multiplication sweeping has unmatched dimensions, the R code continues to work, but the sweeping vector goes to the next column to find elements to multiply until its exhaustion, as shown in this computer example `w3*A`.

An application example is to weight the space-time data matrix A by the weight vector w. The weights depend on spatial locations. Then $w*A$ becomes the weighted space-time data matrix, which is often used in climate science, economics, and sociology.

This computer example also shows the sweeping by division, again counting the elements of the matrix from the first column. See R command `A/w` and its output.

1.3.5 Conversions between a Vector and a Matrix

Sometimes data are given in a sequence form, a vector. However, each datum may correspond to specific meanings, such as the month-end body weight data of a group of five people for a year. Each datum corresponds to a specific person for a specific month. You thus may prefer to present the data in a matrix form, with five rows for people and 12 columns for time.

Below are some examples of the conversions between a vector and a matrix.

```
#Conversions between a Vector and a Matrix
v = c(60, 58, 67, 70, 55, 53)
M = matrix(v, nrow = 2) #from vector to matrix
M
#      [,1] [,2] [,3]
#[1,]   60   67   55
#[2,]   58   70   53
c(M) #from matrix to vector by column
#[1] 60 58 67 70 55 53
c(t(M)) #from matrix to vector by row
#[1] 60 67 55 58 70 53
```

1.3.6 Reduce Dimensions of an n-Dimensional Array

Atmospheric temperature data may be naturally regarded as a 4-dimensional array: three dimensions correspond to latitude, longitude, and elevation, and the fourth to time. For some specific space-time data analysis, such as singular value decomposition to be discussed later, we have to convert the 4D array into a 2D matrix, with the rows corresponding to locations and columns to time, i.e., a space-time data matrix. We can do so, because we can assign each spatial location a location index from one to N, according to a certain rule, saying latitude from south to north, longitude from west to east, and elevation from low to high.

The following is a computing example of reducing a 3-dimensional array to a 2-dimensional array, i.e., a matrix.

```
#R code to reduce the dimension of an nD array
x <- array(1:(2*3*4), dim=c(2,3,4))
dim(x)
#[1] 2 3 4
x #a stack of four 2-by-3 matrices
#, , 1 #the first of the 3rd dim index

#[,1] [,2] [,3]
#[1,]    1    3    5
#[2,]    2    4    6
# ...
#install.packages('R.utils')
library(R.utils)
#flat all the other dim except the 3rd one
#flat the 1st and 2nd dim
y <- wrap(x, map=list(3, NA))
dim(y)
#[1] 4 6
y
#     [,1] [,2] [,3] [,4] [,5] [,6]
#[1,]    1    2    3    4    5    6
#[2,]    7    8    9   10   11   12
#[3,]   13   14   15   16   17   18
#[4,]   19   20   21   22   23   24

#back to the original 3D array
array(t(y), dim = c(2,3,4))
```

1.4 A Set of Linear Equations

A somewhat different matrix example is the coefficient matrix of a system of linear equations. Solving a system of linear equations is very common in science and engineering. Finding numerical solutions for a climate model based on partial differential equations will usually involve solving large systems of linear equations.

As an introduction to the coefficient matrix of linear equations, let us look at a simple elementary school mathematics problem: The sum of the ages of two brothers is 20 years and the difference of the ages is 4 years. What are the ages of the two brothers? One can easily guess that the older brother is 12 years old, and the younger one is 8.

If we form a set of equations, this would be

$$\begin{aligned} x_1 + x_2 &= 20 \\ x_1 - x_2 &= 4 \end{aligned} \tag{1.27}$$

where x_1 and x_2 stand for the brothers' ages.

The matrix form of these equations would be

$$Ax = b \tag{1.28}$$

which involves three matrices:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, b = \begin{bmatrix} 20 \\ 4 \end{bmatrix}. \tag{1.29}$$

Here, Ax means a matrix multiplication: $A_{2\times2}x_{2\times1}$.

The matrix notation of a system of two linear equations can be extended to systems of many linear equations, hundreds or millions of equations in climate modeling and climate data analysis. Typical linear algebra textbooks introduce matrices in this way by describing linear equations in a matrix form. However, this approach may be less intuitive for climate science, which emphasizes data. Thus, our book uses data to introduce matrices as shown at the beginning of this chapter.

Although one can easily guess that the solution to the above simple matrix equation (1.28) is $x_1 = 12$ and $x_2 = 8$, a more general method for computing the solution may be using the R code shown below:

```
#Solve linear equations
A = matrix(c(1,1,1,-1), nrow = 2)
b = c(20,4)
solve(A, b)
#[1] 12  8  #This is the result x1=12, and x2=8.
```

This type of R program can solve more complicated systems of linear equations, such as a system with 1,000 unknowns rather than two unknowns, as in this example.

The solution may be represented as

$$x = A^{-1}b, \tag{1.30}$$

where A^{-1} is the inverse matrix of A and was found earlier in eq. (1.26). One can verify that

$$A^{-1}b = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} 20 \\ 4 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} \tag{1.31}$$

is indeed the solution of the system of two linear equations.

1.5 Eigenvalues and eigenvectors of a square space matrix

"Eigenvalue" is a partial translation of the German word "eigenwert," meaning "self-value" or Òintrinsic value." In German, "eigen" can mean "self" or"own", as in "one's own," and"wert" means"value."

A square matrix is a matrix for which the number of rows is equal to the number of columns. The matrix thus has the shape of a square. Similarly, other matrices may be called rectangular matrices, or tall matrices.

1.5.1 Matrices of data anomalies, standardized anomalies, covariance, and correlation

In climate science, one often considers the covariance or correlation among N stations, N grid boxes, or N grid points. The covariance between station i and station j is denoted by c_{ij}, and the corresponding correlation is denoted by r_{ij}. Then $C = [c_{ij}]_{N\times N}$ is the covariance matrix, and $Corr = [r_{ij}]_{N\times N}$ is the correlation matrix. They can be estimated by the observational or model data.

Let us use an $N \times Y$ matrix

$$X = [x_{it}]_{N\times Y} \tag{1.32}$$

to represent the climate data of these N stations with Y time steps. A time step can be a month, or a year, or a week, depending on the application needs and the data availability. The covariance and correlation matrices of the N stations for a climate parameter are square matrices. Both covariance and correlation matrices can be estimated using the anomaly data, which are defined as data minus their temporal mean of each station, i.e.,

$$A_{N\times Y} = [a_{it}]_{N\times Y} = [x_{it} - \bar{x}_i]_{N\times Y}, \quad \text{(Anomaly data matrix)}, \tag{1.33}$$

where

$$\bar{x}_i = \frac{1}{Y}\sum_{t=1}^{Y} x_{it} \tag{1.34}$$

is the climate data mean of station i ($i = 1, 2, \cdots, N$). The mean is also called climatology or climate normal of the climate parameter. Thus, the temporal mean of the anomaly data for each station should be zero. However, in many climate applications, the temporal data are not continuous and were missing in the earlier period or in the middle. It is impossible to compute the temporal mean for the entire period of length Y. Instead, the climatology is computed using a sub-interval of data period, whose length is smaller than Y, for example, using 30 years from 1971 to 2000. The 30-year climatology has almost become a recent community standard because of the period's best space-time coverage of observations. Consequently, the temporal mean of the anomaly data based on the 30-year climatology is often non-zero.

Another case is that some applications require us to compute climatologies using 10 years or shorter because of climate change. In any event, climatology and anomalies are commonly used terms in climate data analysis. Their exact definitions vary depending on practical applications. Chapter 11 of this book has more detailed discussions on climatology, anomalies, statistics, and visualization for the historical climate data which are incomplete with missing values.

When the anomaly matrix is normalized by the standard deviation of each station, then the result is called the standardized anomaly matrix:

$$A_{sd} = \left[\frac{x_{it} - \bar{x}_i}{s_i}\right]_{N\times Y}, \quad \text{(Standardized anomaly matrix)}, \tag{1.35}$$

where

$$s_i = \left[\frac{1}{Y} \sum_{t=1}^{Y} (x_{it} - \bar{x}_i)^2 \right]^{1/2} \tag{1.36}$$

is the estimated standard deviation of station i ($i = 1, 2, \cdots, N$). Strictly speaking from statistical theory, this standard deviation formula should use $Y - 1$ as the denominator, rather than simply Y, according to formula (B.4) in Chapter 3. However, climate scientists often use Y, not $Y - 1$, perhaps because formula (1.36) has a clear meaning as the mean of square errors. In practice, the difference between the two formulas is small since Y is usually greater or equal to 30. Climate data analysis applications often use 30 years of data, say from 1971 to 2000, to compute climatology and standard deviation.

An advantage of using the standardized anomaly matrix is that it is dimensionless. A climate parameter, such as temperature, often has a larger variance over the land than ocean. The standardized anomaly data show better homogeneity between ocean and land.

Then, the covariance and correlation matrices can be estimated by the following formulas

$$C = \frac{1}{Y} A(A^t) \quad \text{(Covariance matrix)}, \tag{1.37}$$

$$Corr = \frac{1}{Y} A_{sd}(A_{sd})^t \quad \text{(Correlation matrix)}. \tag{1.38}$$

Once more, applications often use a 30-year mean, not the mean of the entire time period of length Y, to compute the covariance and correlation matrices.

Example 1.7 This example's data matrix `A` has $N = 2$ and $Y = 3$. The data's covariance matrix is `covm`.

```
#Spatial covariance matrix
dat = matrix(c(0,-1,1,2,3,4), nrow=3)
dat
colMeans(dat)
A = sweep(dat, 2, colMeans(dat))
A
#     [,1] [,2]
#[1,]    0   -1
#[2,]   -1    0
#[3,]    1    1
covm=(1/(dim(A)[2]))*A%*%t(A)
covm #is the covariance matrix.
#     [,1] [,2] [,3]
#[1,]  0.5  0.0 -0.5
#[2,]  0.0  0.5 -0.5
#[3,] -0.5 -0.5  1.0
```

1.5.2 Eigenvectors and their corresponding eigenvalues

The covariance matrix times a vector `u` yields a new vector in a different direction.

```
u = c(1, 1, 0)
v = covm %*% u
v
#      [,1]
#[1,]  0.5
#[2,]  0.5
#[3,] -1.0
#u and v are in different directions
```

In general, when we consider a given square matrix C and a given vector u, the product Cu is usually not in the same direction as u, as shown in the above example. However, there is always a "self-vector" vector w for each covariance matrix C such that Cw is in the same direction as w, i.e., Cw and w are parallel, and this fact is expressed as

$$Cw = \lambda w, \tag{1.39}$$

where λ is a scalar which has the property that it simply scales w so that the above equation holds. This scalar λ is called an eigenvalue (i.e., a "self-value", "own-value", or "characteristic value" of the matrix C), and w is called an eigenvector.

R can calculate the eigenvalues and eigenvectors of a covariance matrix `covm` with a command `eigen(covm)`. The output is in an R data frame, which has two storages: `ew$values` for eigenvalues and `ew$vector` for eigenvectors, as shown below.

```
#Eigenvectors of a covariance matrix
ew = eigen(covm)
ew
#$values
#[1] 1.500000e+00 5.000000e-01 1.332268e-15

#$vectors
#           [,1]          [,2]      [,3]
#[1,] -0.4082483 -7.071068e-01 0.5773503
#[2,] -0.4082483  7.071068e-01 0.5773503
#[3,]  0.8164966  8.881784e-16 0.5773503

#Verify the eigenvectors and eigenvalues
covm%*%ew$vectors[,1]/ew$values[1]
#           [,1]
#[1,] -0.4082483
#[2,] -0.4082483
#[3,]  0.8164966
#This is the first eigenvector

w = ew$vectors[,1] # is an eigenvector
```

A 3×3 covariance matrix has three eigenvalues, and three eigenvectors (λ_1, w_1) (λ_2, w_2), and (λ_3, w_3). These are called eigenpairs.

In this example,

$$\lambda_1 = 1.5, \quad w_1 = \begin{bmatrix} -0.4082483 \\ -0.4082483 \\ 0.8164966 \end{bmatrix}, \tag{1.40}$$

$$\lambda_2 = 0.5, \quad w_2 = \begin{bmatrix} -0.7071068 \\ 0.7071068 \\ 0 \end{bmatrix}, \tag{1.41}$$

$$\lambda_3 = 0.0, \quad w_3 = \begin{bmatrix} 0.5773503 \\ 0.5773503 \\ 0.5773503 \end{bmatrix}. \tag{1.42}$$

The eigenvectors are frequently called modes, or empirical orthogonal functions (EOFs) in climate science. The term EOF was coined by Edward Lorenz (1917-2008) in his 1956 paper on statistical weather prediction (Lorenz 1956).

Each eigenvalue is equal to the variance of the data projection on the corresponding eigenvector, and is thus positive. The sum of all the eigenvalues of such an $N \times N$ matrix represents the total variance of the climate system observed at these N stations. The first eigenvalue λ_1 is the largest, corresponding to the largest spatial variability of the climate field under study. The eigenvalues sizes follow the order $\lambda_1 \geq \lambda_2 \geq \cdots$.

However, in carrying out an analysis of climate data, one can often find the important patterns as eigenvectors more directly from the anomaly data matrix A without computing the covariance matrix C explicitly. This is known as the singular value decomposition (SVD) approach (Golub and Reinsch 1970), which we discuss next. It separates the space-time anomaly data into a space part, a time part, and what we may think of as a variation in energy part. This mathematical method of space-time decomposition is universally applicable to any data that we sample in space and time, and it can often help to develop physical insight and scientific understanding of the phenomena and their properties as contained in the observational data. Efficient computing methods of SVD have been extensively researched and developed since the 1960s. Gene H. Golub (1932-2007) was a leading researcher in this effort and is remembered as "Professor SVD" by his Stanford colleagues and the world mathematics community.

1.6 An SVD representation model for space-time data

1.6.1 Space-Time Data Matrix and Its decomposition

We encounter space-time data every day, a simple example being the air temperature at different locations at different times. If you take a plane to travel from San Diego to New York, you may experience the temperature at San Diego in the morn-

ing when you depart and that at New York in the evening after your arrival. Such data have many important applications. We may need to examine the precipitation conditions around the world at different days in order to monitor agricultural yields. A cellphone company may need to monitor its market share and the temporal variations of that quantity in different countries. A physician may need to monitor a patient's symptoms in different areas of the body at different times. The observed data in all these examples can form a space-time data matrix with the row position corresponding to the spatial location and the column position corresponding to time, as in Table 1.1.

Table 1.1 Space-time data table

	Time 1	Time 2	Time 3	Time 4
Space 1	D11	D12	D13	$D14$
Space 2	D21	D22	D23	$D24$
Space 3	D31	D32	D33	$D34$
Space 4	D41	D42	D43	$D44$
Space 5	D51	D52	D53	$D54$

Graphically, the space-time data may typically be plotted as a time series at each given spatial position, or as a spatial map at each given time. Although these straight-forward graphical representations can sometimes provide very useful information as input for signal detection, the signals are often buried in the data and may need to be detected by different linear combinations in space and time. Sometimes the data matrices are extremely large, with millions of data points in either space or time. Then the question arises as to how can we extract the essential information in such a big data matrix? Can we somehow manage to represent the data in a more simple and yet more useful way? A very useful approach to such a task involves a space-time separation. Singular value decomposition (SVD) is a method designed for this purpose. SVD decomposes a space-time data matrix into a spatial pattern matrix U, a diagonal energy level matrix D, and a temporal matrix V^t, i.e., the data matrix A is decomposed into

$$A_{N\times Y} = U_{N\times m} D_{m\times m} (V^t)_{m\times Y}. \tag{1.43}$$

where N is the spatial dimension, Y is the temporal length, $m = min(N, Y)$, and V^t is the transpose of V. The columns of U are a set of spatial eigenvectors. They are orthonormal vectors, each of which is orthogonal to another and has its length equal to one. Thus, the word "orthonormal" comes from two words "orthogonal" and "normal." The orthonormal property can be expressed by the following formula:

$$\mathbf{u}_l \cdot \mathbf{u}_k = \delta_{lk}, \tag{1.44}$$

where $\mathbf{u}_l$ and $\mathbf{u}_k$ are column vectors of U, and δ_{lk} is the Kronecker delta equal to

zero when $k \neq l$ and one when $k = l$. The columns of V are also a set of orthonormal vectors known as temporal eigenvectors.

Usually, the elements of the U and V matrices are unitless (i.e., dimensionless), and the unit of the D elements is the same as the elements of the data matrix. For example, if A is a space-time precipitation data matrix with a unit [mm/day], then the dimension of the D elements is also [mm/day].

1.6.2 SVD of the Spae-Time Anomaly Data Matrix and Eigenvalue Problem of a Covariance Matrix

Example 1.8 SVD for a 2×3 matrix.

```
#Develop a 2-by-3 space-time data matrix for SVD
A=matrix(c(1,-1,2,0,3,1), nrow=2)
A
#     [,1] [,2] [,3]
#[1,]    1    2    3
#[2,]   -1    0    1
#Perform SVD calculation
msvd=svd(A)
msvd
msvd$d
#[1] 3.784779 1.294390
msvd$u
#           [,1]       [,2]
#[1,] -0.9870875 -0.1601822
#[2,] -0.1601822  0.9870875
msvd$v
#            [,1]       [,2]
#[1,] -0.2184817 -0.8863403
#[2,] -0.5216090 -0.2475023
#[3,] -0.8247362  0.3913356
#One can verify that A=UDV', where V' is transpose of V.
verim=msvd$u%*%diag(msvd$d)%*%t(msvd$v)
verim
#     [,1]          [,2] [,3]
#[1,]    1 2.000000e+00    3
#[2,]   -1 1.665335e-16    1
round(verim)
#     [,1] [,2] [,3]
#[1,]    1    2    3
#[2,]   -1    0    1
#This is the original data matrix A
```

The covariance of the space-time anomaly matrix A is a spatial matrix:

$$C = \frac{1}{Y} A A^t, \tag{1.45}$$

where Y is the number of columns of A and is equal to the length of time being considered.

Strictly speaking, each column of an anomaly matrix A has a zero mean. But in practice, due to different normalization periods, the mean is not zero.

```
covm = (1/(dim(A)[2]))*A%*%t(A)
eigcov = eigen(covm)
eigcov$values
#[1] 4.7748518 0.5584816
eigcov$vectors
#            [,1]        [,2]
#[1,] -0.9870875  0.1601822
#[2,] -0.1601822 -0.9870875
```

Thus, the eigenvectors of a covariance matrix are the same as the SVD eigenvectors of the anomaly data matrix. The eigenvalues of the covariance matrix and the SVD have following relationship

```
((msvd$d)^2)/(dim(A)[2])
#[1] 4.7748518 0.5584816
eigcov$values
#[1] 4.7748518 0.5584816
```

Therefore, the EOFs from a given space-time dataset can be calculated directly by using an SVD command and do not need the step of calculating the covariance matrix. With efficient SVD algorithms, this shortcut can save significant amounts of time for an EOF analysis, also known as the principal component analysis (PCA) in the statistics community, compared to the traditional covariance matrix approach. Therefore, the result is extremely helpful for the EOF analysis, which is an indispensable modern tool of climate data analysis. The result is formally described as a theorem whose proof is also provided as follows.

Theorem 1.1 *The eigenvectors* $\mathbf{u}_k$ *of the covariance matrix* $C = (1/Y)AA^t$

$$\mathbf{C}\mathbf{u}_k = \lambda_k \mathbf{u}_k, \quad (k = 1, 2, \cdots, N), \tag{1.46}$$

are the same as the SVD spatial modes of $A = UDV^t$*. The eigenvalues* λ_k *of* $C_{N\times N}$ *and the SVD eigenvalues* d_k *of* $A_{N\times Y}$ *have the following relationship*

$$\lambda_k = d_k^2/Y \quad (k = 1, 2, \cdots, Y), \quad \textit{when} \quad Y \leq N, \tag{1.47}$$

where Y *is the total time length (i.e., time dimension) of the anomaly data matrix* A*, and* N *is the total number of stations for* A *(i.e., space dimension).*

Proof The SVD of the space-time data matrix is

$$A = UDV^t. \tag{1.48}$$

The data matrix A's corresponding covariance matrix is thus

$$\begin{aligned} C &= \frac{1}{Y}AA^t \\ &= \frac{1}{Y}UDV^t(UDV^t)^t \\ &= \frac{1}{Y}UDV^t(VDU^t) \\ &= \frac{1}{Y}UD(V^tV)DU^t \\ &= \frac{1}{Y}UDIDU^t \\ &= \frac{1}{Y}UD^2U^t. \end{aligned} \tag{1.49}$$

In the above, we have used $V^tV = I_Y$ is a $Y \times Y$ identity matrix according to the SVD definition. The identity matrix's dimension is Y if $Y \leq N$. Otherwise, $V^tV = I_N$.

The expression $C = \frac{1}{Y}UD^2U^t$ is the EOF expansion of the covariance matrix and means that a covariance matrix consists of EOFs and its associated variance, or "energy."

The covariance matrix's eigenvalue problem is

$$CU = \frac{1}{Y}UD^2U^tU = \frac{1}{Y}UD^2 = U\Lambda, \tag{1.50}$$

where

$$\Lambda = \frac{1}{Y}D^2 \tag{1.51}$$

is the diagonal eigenvalue matrix with its elements as

$$\lambda_k = d_k^2/Y \quad (k = 1, 2, \cdots, Y). \tag{1.52}$$

In eq. (1.50), we used $U^tU = I_Y$ based on the SVD definition. Equation (1.50) becomes an eigenvalues problem because Λ is a diagonal matrix: $C\mathbf{u}_k = \lambda_k\mathbf{u}_k$ ($k = 1, 2, \cdots, Y$), where $\mathbf{u}_k$ is the kth column vector of the space matrix U. Thus, eq. (1.50) implies the first part of the theorem: The space eigenvectors U from the SVD of the space-time data matrix A are the same as the eigenvectors of the corresponding covariance matrix C.

Equation (1.52) is exactly the second part of the theorem. The proof is thus complete.

In the above proof, we implicitly assumed that the columns of data matrix A are independent when $Y \leq N$. Independence of a set of vectors means that no one can be expressed in terms of the others, so no vectors can be replaced. Real climate data always satisfy this independence assumption.

1.7 Image analysis using SVD and R

This section shows an example of using SVD to analyze a color photo, The left panel of Fig. 1.2 shows the color photo. The right panel of the figure is the black-and-white photo produced by the grayscale data derived from the color photo. We will also show how to reconstruct a photo from various SVD modes, as wells as how to blend the monotone photos into a color photo.

1.7.1 Data for a color photo and its grayscale figure

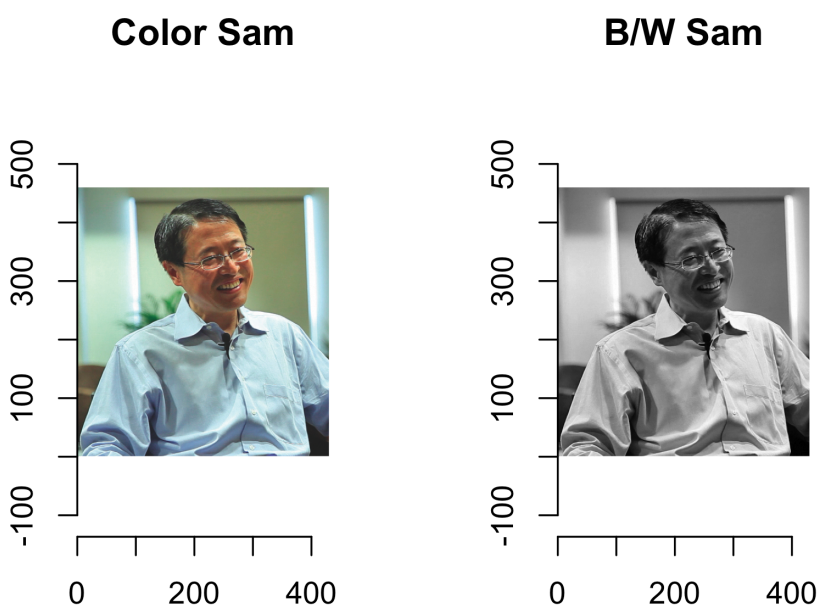


Fig. 1.2 Color and grayscale photos

Figure 1.2 can be generated by the following computer code and data.

```
#R code
#Image analysis using SVD
#Ref: imager: an R package for image processing
#https://dahtah.github.io/imager/imager.html

setwd("~/mathmodel")
#install.packages('imager') #run this if not installed yet
library(imager)
```

```
dat <- load.image('data/SamPhoto.png') #355 KB file size
dim(dat)
#[1] 430 460   1   3
#430 rows and 460 columns, 430*460 = 197,800 pixels
#1 photo frame, 3 RGB colors
#If a video, 1 will become 150 frames or more

dat[1:3, 1:4,1,1]
#Show part of the daat

#plot the color figure
plot(dat,
     xlim = c(0, 430), ylim = c(0, 460),
     main = 'A Color Photo of Sam')

#Make the photo black-and-white
graydat = grayscale(dat)
dim(graydat)
#[1] 430 460   1   1
#430 rows and 460 columns,  1 photo frame, 1 grayscale [0, 1]
#plot the gray b/w photo
plot(graydat,
     xlim = c(0, 430), ylim = c(0, 460),
     main = 'B/W Gray Sam')
#Plot the color and b/w photos together
par(mfrow = c(1, 2))
plot(dat,
     xlim = c(0, 430), ylim = c(0, 460),
     main = 'Color Sam')
plot(graydat,
     xlim = c(0, 430), ylim = c(0, 460),
     main = 'B/W Sam')
dev.off()
```

1.7.2 SVD of the grayscale photo data

1.7.2.1 SVD singular values and their scree plot

Data science and statistics often use the concept of how much variance is explained. Hence, we plot the scree plot (see Fig. 1.3) based on variances, i.e., the eigenvalues of a covariance matrix, which are the squares of SVD's singular values.

The scree plot shows that the first mode is dominant, explaining around 94% of variance. The variance contributions from other modes are almost zero. However, these modes are extremely important in producing the details of the photo, as we will see from the data reconstruction. This will be shown in the next subsection.

Figure 1.3 and its associated SVD can be made using the following computer code.

```
#R code
#SVD analysis of the grayscale data
svdDat = svd(graydat)
SVDd = svdDat$d
percentD = 100*(SVDd^2)/sum(SVDd^2)
```

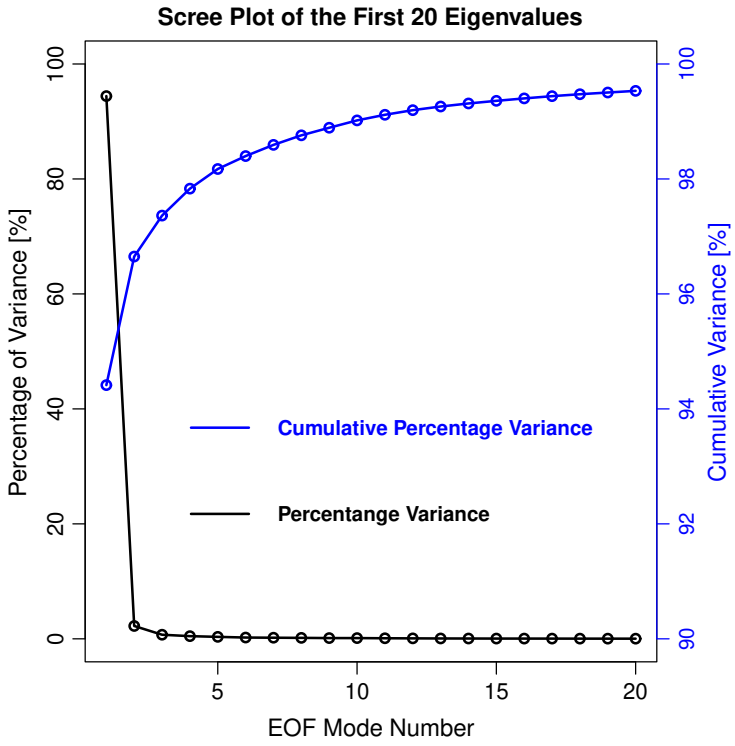


Fig. 1.3 Scree plot based on the grayscale photo's SVD singular values.

```
cumpercentD = cumsum(percentD)
modeK = 1:length(SVDd)
dev.off()
plot(modeK[1:20], percentD[1:20],
     type = 'o', col = 'blue',
     xlab = 'Mode number', pch = 16,
     ylab = 'Percentage of mode variance',
     main = 'Scree Plot of SVD B/W Photo Data')
K = 20
lam = (svdDat$d)^2
lamK=lam[1:K]
lamK
#setEPS() #Plot the figure and save the file
#postscript("fig0608.eps", width = 6, height = 4)
par(mar=c(4,4,2,4), mgp=c(2.2,0.7,0))
plot(1:K, 100*lamK/sum(lam), ylim=c(0,100), type="o",
     ylab="Percentage of Variance [%]",
     xlab="EOF Mode Number",
     cex.lab=1.2, cex.axis = 1.1, lwd=2,
     main="Scree Plot of the First 20 Eigenvalues")
legend(3,30, col=c("black"),lty=1, lwd=2.0,
       legend=c("Percentange Variance"),bty="n",
       text.font=2,cex=1.0, text.col="black")
par(new=TRUE)
```

```
plot(1:K,cumsum(100*lamK/sum(lam)),
     ylim = c(90,100), type="o",
     col="blue",lwd=2, axes=FALSE,
     xlab="",ylab="")
legend(3,80, col=c("blue"),lty=1,lwd=2.0,
       legend=c("Cumulative Percentage Variance"),bty="n",
       text.font=2,cex=1.0, text.col="blue")
axis(4, col="blue", col.axis="blue", mgp=c(3,0.7,0))
mtext("Cumulative Variance [%]",col="blue",
      cex=1.2, side=4,line=2)
```

1.7.2.2 The grayscale photo reconstruction from the SVD modes

Figure 1.4 shows four reconstructions from different SVD modes. When using all the modes, the reconstruction is 100% accurate and completely recovers the original grayscale photo. When using the first 20 modes, the reconstructed photo is fuzzy. Although 99% of variance is explained in the first 20 modes, many details are still contained in the higher modes whose variance contribution is less 1%. When using the three modes, the reconstructed photo is so fuzzy that it is barely recognizable until you look at it carefully in the direct front view. The first three modes explain almost 98% of total variance according to the scree plot. Many fields of natural sciences use only the modes for 90% variance. This practice may need careful scrutiny when more detailed signals of small scales, such as this photo analysis, are important. When using the 80 modes between 21st and the 100th modes, the reconstruction shows the person with glasses without the background layout. These 80 modes account for less than 1% variance but can provide this much information. This seems surprising.

Using the computer code we provide here, you can easily test reconstructions using different modes. You can also use your preferred photos and other figures. You may come out with some surprising discoveries.

Figure 1.4 can be generated by the following computer code.

```
#R code
dev.off()
par(mfrow = c(2, 2))
kR = 430 #Recon from all 413 modes
R430 = as.cimg(U[,1:kR]%*%D[1:kR, 1:kR]%*%t(V[, 1:kR]))
plot(R430, main = "All 430 modes")
kB = 20 #The first 20 modes
R20 = as.cimg(U[,1:kB]%*%D[1:kB, 1:kB]%*%t(V[, 1:kB]))
plot(R20, main = "The first 20 modes")
k = 3 #Recon from the first 3 modes
R3 = as.cimg(U[,1:k]%*%D[1:k, 1:k]%*%t(V[, 1:k]))
plot(R3, main = "The first 3 modes")
k1 = 21; k2 = 100 #Recon from 21st to 100th modes
Rk1_k2 = as.cimg(U[,k1:k2]%*%D[k1:k2, k1:k2]%*%t(V[, k1:k2]))
plot(Rk1_k2, main = "21st to 100th modes")
dev.off()
```

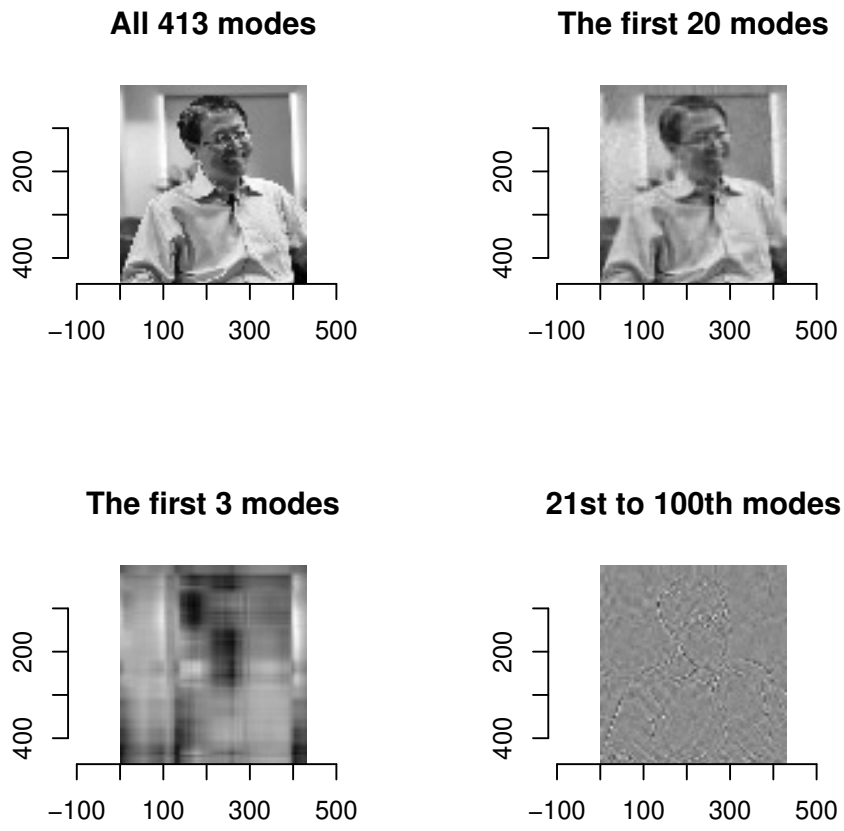


Fig. 1.4 Reconstruction of the grayscale photo using different numbers of SVD modes.

1.7.3 Reconstructing a color photo from the data of monotone photos

We described earlier that the data for a color photo is a 4D array. The first two dimensions are defined on the x-y pixels. The third dimension is the number of photos. The fourth dimension has 3 entries for three monotone colors, such as R (red), G (green), and B (blue). This subsection shows that you can recover the original photo from the data for each of the three monotone figures. Figure 1.5 displays the recovered color photo and the three monotone photos.

Figure 1.5 can be generated by the following computer code.

```
 #Rcode
 #Three monotone photos and their color photo
dim(dat) #4D array of a color photo
```

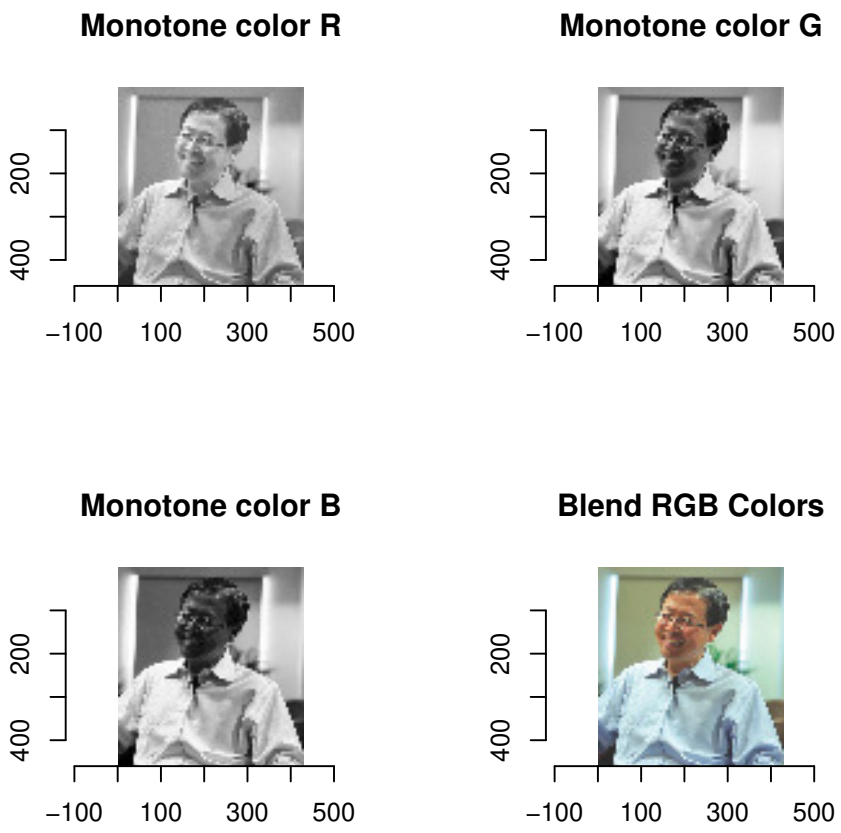


Fig. 1.5 Three monotone photos and the color photo recovered from them.

```
#[1] 430 460   1   3
dev.off()
par(mfrow = c(2, 2))
R = as.cimg(apply(t(dat[,, 1, 1]), 1, rev))
plot(R, main = 'Monotone color R')
G = as.cimg(apply(t(dat[,, 1, 2]), 1, rev))
plot(G, main = 'Monotone color G')
B = as.cimg(apply(t(dat[,, 1, 3]), 1, rev))
plot(B, , main = 'Monotone color B')
#Bind the three monotone photos into one
trippy = imappend(list(R,G,B), "c")
dim(trippy) #color figure data
#[1] 430 460   1   3
plot(trippy, main ='Blend RGB Colors')
dev.off()
```

This example shows a way to color a grayscale photo by blending additional

monotone photos. Then, the difficulty is where to color and what color. AI coloring for old black-and-white films has been made in the market. AI coloring technology still needs improvement. Can SVD help?

1.8 Mass balance for chemical equations in marine chemistry

This section describes another application of linear algebra in climate science. Marine chemistry or atmospheric chemistry involves various kinds of chemical reaction equations which describe the conservation of mass during the chemical reactions. Then, the problem is how to systematically achieve the mass balance, i.e., to determine the numbers of molecules on each side of an equation that depicts a chemical reaction. Here we use photosynthesis as an example to illustrate a linear algebra approach to this problem.

In the process of photosynthesis, plants convert the solar radiant energy carried by photons, plus carbon dioxide (CO_2) and water (H_2O), into glucose ($C_6H_{12}O_6$) and oxygen (O_2). The chemical equation for this conversion could be written schematically as

$$CO_2 + H_2O \longrightarrow C_6H_{12}O_6 + O_2 \tag{1.53}$$

However, the conservation of mass requires that the atomic weights on both sides of the equation be equal. The photons have no mass. Thus, the chemical equation as written above is incorrect. The correct equation should specify precisely how many CO_2 molecules react with how many H_2O molecules to generate how many $C_6H_{12}O_6$ and O_2 molecules. Suppose that these coefficients are x_1, x_2, x_3, x_4. We then have

$$x_1CO_2 + x_2H_2O \longrightarrow x_3C_6H_{12}O_6 + x_4O_2 \tag{1.54}$$

Making the number of atoms of carbon on the left and right sides of the equation equal yields

$$x_1 = 6x_3 \tag{1.55}$$

because water and oxygen contain no carbon. Doing the same for hydrogen atoms leads to

$$2x_2 = 12x_3. \tag{1.56}$$

Similarly, the balance of oxygen atoms is

$$2x_1 + x_2 = 6x_3 + 2x_4. \tag{1.57}$$

We thus have three equations in four variables. Thus, these equations have infinitely many solutions. We can set any variable fixed, and express the other three using

this fixed variable. Since the largest molecule is glucose, we set its coefficient x_3 fixed. Then we have

$$x_1 = 6x_3, \quad x_2 = 6x_3, \quad x_4 = 6x_3. \tag{1.58}$$

Thus, the chemical equation is

$$6x_3CO_2 + 6x_3H_2O \longrightarrow x_3C_6H_{12}O_6 + 6x_3O_2. \tag{1.59}$$

If we want to produce one glucose molecule, i.e., $x_3 = 1$, then we need 6 carbon dioxide and 6 water molecules:

$$6CO_2 + 6H_2O \longrightarrow C_6H_{12}O_6 + 6O_2. \tag{1.60}$$

Similarly, one can write chemical equations for many common reactions, such as iron oxidation

$$3Fe + 4H_2O \longrightarrow 4H_2 + Fe_3O_4, \tag{1.61}$$

and the redox reaction in a human body which consumes glucose and converts the glucose into energy, water and carbon dioxide

$$C_6H_{12}O_6 + 6O_2 \longrightarrow 6CO_2 + 6H_2O. \tag{1.62}$$

1.9 Multivariate linear regression using matrix notations

This section is an application of matrix algebra in statistical data analysis, particularly on a linear regression for more than one variable. The linear regression section discussed the fitting of a straight line on the xy-plane $y = b_1x + b_0$ to a pair of data vectors of length N: $[x_d]_{N\times 1}, [y_d]_{N\times 1}$. It resulted in correlation, trend and other regression quantities. An example is the correlation between the January SOI as x and U.S. temperature as y. The non-trivial correlation can suggest that there may be a physical mechanism to explain how the January SOI influences the U.S. January temperature.

Here, we use $y = b_1x + b_0$ as the representation of a deterministic fitting function. This expression does not involve random variables. Most statistics books would consider random linear models, which distinguishes between random variables and their deterministic estimators by data. Our simple linear mathematical model formulation here is equivalent to using only the deterministic estimators.

The U.S. January temperature can be influenced by multiple factors in addition to the SOI. These factors may include the North Atlantic sea surface temperature (SST), the North Pacic SST, Arctic pressure conditions, etc. Then, the linear model becomes

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n. \tag{1.63}$$

When $n = 1$, this degenerates into the single variable regression.

In the following we will present a few R examples of multivariate regression and its applications. The mathematical theory behind the R code for a multivariate regression deals mostly with the matrix operations, which can be found from any standard textbooks covering multivariate regression.

Example 1.9 This example shows a two-variable regression.

$$y = b_0 + b_1 x_1 + b_2 x_2. \tag{1.64}$$

Geometrically, this is an equation of a plane in a 3D space (x_1, x_2, y). Given three points not on a straight line, a plane can be determined uniquely. This means specifying three x_1 coordinate values, three x_2 coordinate values and three y coordinate values, which can be done by means of the following R code:

```
x1=c(1,2,3) #Given the coordinates of the 3 points
x2=c(2,1,3)
y=c(-1,2,1)
df=data.frame(x1,x2,y) #Put data into the data.frame format
fit <- lm(y ~ x1 + x2, data=df)
fit#Show the regression results
#Call:
#  lm(formula = y ~ x1 + x2, data = df)
#Coefficients:
#  (Intercept)           x1           x2
#-5.128e-16    1.667e+00   -1.333e+00

1.667*x1-1.333*x2  #Verify that 3 points determining a plane
#[1] -0.999  2.001  1.002
```

Example 1.10 This example will show that four arbitrarily specified points cannot all be on a plane. The fitted plane has the shortest distance squares, i.e., the least squares (LS), or minimal mean square error (MMSE). Thus, the residuals are non-zero, in contrast to the zero residuals in the previous example.

```
u=c(1,2,3,1)
v=c(2,4,3,-1)
w=c(1,-2,3,4)
mydata=data.frame(u,v,w)
myfit <- lm(w ~ u + v, data=mydata)
summary(myfit)#Show the result
#Coefficients:
#  Estimate Std. Error t value Pr(>|t|)
#(Intercept)    1.0000     1.8708   0.535    0.687
#u              2.0000     1.2472   1.604    0.355
#v             -1.5000     0.5528  -2.714    0.225
```

Example 1.11 This example will show a general multivariate linear regression using R. It has three independent variables, one dependent variable, and ten data points. For R program simplicity, the data are generated by an R random number generator. Again, R requires that the data be put into data frame format so that

a user can clearly specify which are independent variables, also called explainable variables, and which is the dependent variable, also called response variable.

```
dat=matrix(rnorm(40),nrow=10, dimnames=list(c(letters[1:10]), c(LETTERS[23:26])))
fdat=data.frame(dat)
fit=lm(Z~ W + X + Y, data=fdat)
summary(fit)

#Coefficients\index{linear regression!coefficients}
#             Estimate Std. Error t value Pr(>|t|)
#(Intercept)  0.36680    0.16529   2.219   0.0683
#W            0.11977    0.20782   0.576   0.5853
#X           -0.53277    0.19378  -2.749   0.0333
#Y           -0.04389    0.14601  -0.301   0.7739
```

Thus, the linear model is

$$Z = 0.37 + 0.12W - 0.53X - 0.04Y. \tag{1.65}$$

The 95% confidence interval for W's coefficient is $0.12 \pm 2 \times 0.21$, that for X's coefficient is $-0.53 \pm 2 \times 0.19$,Y's coefficient is $-0.04389 \pm 2 \times 0.15$. Each confidence interval includes zero. Thus, there is no significant non-zero trend for the Z data with respect to W, X, Y. This result is to be expected, because the data are randomly generated and thus should not have a trend.

In practical applications, a user can simply convert the data into the same data frame format as shown here. Then, R command

```
lm(formula = Z   W + X + Y, data = fdat)
```

can do the regression job.

R can also do nonlinear regression with specified functions, such as quadratic functions and exponential functions. See examples from the URLs

```
https://www.zoology.ubc.ca/~schluter/R/fit-model/
https://stat.ethz.ch/R-manual/R-devel/library/stats/html/nls.html
```

1.10 Chapter summary

This chapter introduces matrices and linear algebra from the perspective of space-time climate data, with rows corresponding to spatial locations (e.g., the order of grid boxes, grid point IDs, or climate station IDs) and columns corresponding to time (e.g., the month stamps January 1901, February 1901, $\cdots$, December 2017 for the monthly data, or the day stamps 1 June 2017, 2 June 2017, $\cdots$, 30 June December 2017 for the daily data of June 2017). This way of introducing matrices as organized collections of data is different from that of most textbooks which describe a matrix as originating from a set of linear equations. This particular way to arrange a space-time climate data matrix follows the similar philosophy of the

netCDF data file, which has recently become a very popular format for representing extensive climate datasets.

The most essential methods of linear algebra have to do with matrix properties (e.g., eigenvalues, eigenvectors, and SVD matrices) and matrix operations (e.g., solving linear equations, and making linear transformations). Our chapter has described the following linear algebra methods:

(i) Matrix arithmetic: addition of a matrix plus another matrix, subtraction of a matrix minus another matrix, multiplication of a matrix times another matrix, and multiplication of a matrix by a scalar, and division of the identity matrix by a matrix which is a way to define the inverse matrix of the latter.

(ii) An eigenvector w of a matrix C is such a special vector that Cw is in w's own direction. Thus, there exists a scalar λ such that $Cw = \lambda w$, where λ is called an eigenvalue of the matrix C corresponding to the eigenvector w.

(iii) SVD decomposes the space-time data matrix A into three matrices: a dimensionless spatial orthogonal matrix U as spatial eigenvectors, a dimensionless temporal orthogonal matrix V as temporal eigenvectors, and a diagonal "energy level" matrix D related to the eigenvalues of A's covariance matrix, i.e.,

$$A = UDV^t. \tag{1.66}$$

The column vectors of U may represent the intrinsic spatial patterns of the climate data (e.g., the warming pattern of the eastern tropical Pacific for the El Niño based on the SST data), and those of V may represent the intrinsic temporal patterns (e.g., the El Niño peaks in the time series determined by a column vector of V). See the specific patterns shown in the examples of the section on the SVD analysis for the reanalysis data. The diagonal values of D show the "energy level" (or the strength) of the corresponding spatial and temporal patterns. In the last few years, the SVD method has become more widely used in science and engineering because of its universal power in decomposing any space-time matrix. Mathematicians have now begun to modernize the proofs of the theorems of linear algebra by using the SVD approach, instead of the traditional approach known as echelon reduction through row operations.

(iv) Mathematical models based on linear equations have been developed and employed for many climate science applications, such as the mass balance model for chemical equations.

(v) Multivariate linear regression has been formulated here from the perspective of matrix algebra.

(vi) R codes have been written for all these methods, which can be conveniently used for solving climate data analysis problems in research and practical applications.

Although the basic materials of linear algebra appeared as early as the 17th

century, widespread university classroom education in linear algebra in the United States began only in the 1950s and later (Tucker 1993). At that time, electronic computers began to become available to solve large numbers of linear equations for engineering applications. Today's vastly increased computing power now provides opportunities to modernize educational aspects of this branch of mathematics, such as using SVD to compute various properties of a matrix and to prove numerous theorems of linear algebra. Climate mathematics takes full advantage of this technical progress. Many graduate students in climate science now use EOFs and PCs in their research, whereas the EOF-PC techniques were rarely taught as part of climate science education before the 1990s.

References and Further Readings

[1] Golub, G.H. and C. Reinsch, 1970: Singular value decomposition and least squares solutions. *Numerische mathematik* 14, 403-420.

This seminal paper established an important method, known as the Golub-Reinsch algorithm, to compute the eigenvalues of a covariance matrix from a space-time matrix A without actually first computing the covariance matrix AA^t. This algorithm makes the SVD computation very efficient, which helps scientists consider SVD as a genuine linear algebra method, not a traditionally regarded statistical method based on a covariance matrix.

[2] Lorenz, E.N., 1956: Empirical orthogonal functions and statistical weather prediction. *Scientific Report No. 1, Statistical Forecasting Project.* Air Force Research Laboratories, Office of Aerospace Research, USAF, Bedford, Massachusetts, 49pp.

Edward N. Lorenz (1917-2008) was an American meteorologist and applied mathematician, who has been best known for his theories of chaos and the butterfly effect. Although he was not the first to invent the mathematical method of empirical orthogonal functions (EOFs), Lorenz, together with other scientists, such as Gerald R. North, made EOF a very popular method for climate data analysis.

[3] Strang, G., 2016: *Introduction to Linear Algebra.* 5th edition, Wellesley-Cambridge Press, Wellesley, MA 02482, 574pp.

Gilbert Strang (1934-) is an American mathematician and educator. His textbooks and pedagogy have been internationally influential. This text is one of the very few basic linear algebra books that includes excellent materials on SVD, probability, and statistics.

[4] Tucker, A., 1993: The growing importance of linear algebra in undergraduate mathematics. *College Mathematics Journal* 24, 3-9.

This paper describes the historical development of linear algebra, such as the term "matrix" being coined by J.J. Sylvester in 1848, and pointed out that "tools of linear algebra find use in almost all academic fields and throughout modern society." The use of linear algebra in the big data era is now even more popular.

Exercises

1.1 The following are the SVD results

```
mat
     [,1] [,2]
[1,]    1    1
[2,]    1   -1

svd(mat)
$d
[1] 1.414214 1.414214

$u
           [,1]       [,2]
[1,] -0.7071068 -0.7071068
[2,] -0.7071068  0.7071068

$v
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
```

Use $A = UDV^t$ to recover the first column of

```
mat
     [,1] [,2]
[1,]    1    1
[2,]    1   -1
```

Show detailed calculations of all the relevant matrices and vectors, and use space-time decomposition to describe your results.

For extra credit: Describe the spatial and temporal modes, and their corresponding variances or energies.

1.2 Use R and the updated Darwin and Tahiti standardized SLP data to reproduce the EOFs and PCs and to plot the EOF pattern maps and PC time series.

1.3 Do the same procedures in the previous problem but for original non-standardized data. Comment on the difference of the results from the previous problem.

1.4 (a) Download the monthly precipitation data at five different stations over the United States from the USHCN website:

`http://cdiac.ornl.gov/epubs/ndp/ushcn/ushcn_map_interface.html`

(b) Use R to organize the January data from 1951 to 2010 into the space-time format.
(c) Compute the climatology of each station as the 1971-2000 mean.
(d) Compute the space-time anomaly data matrix A as the original space-time data matrix minus the climatology.
(e) Use R to make the SVD decomposition of the space-time anomaly data matrix $A = UDV^t$.
(f) Output the U and D matrices.

1.5 In the previous problem, use R and the formula UDV^t to reconstruct the original data matrix A. This is a verification of the SVD decomposition, and is also called EOF-PC reconstruction.

1.6 Use R to plot the maps of the first three EOF modes from the U matrix in the previous problem in a way similar to the two El Niño mode maps you can find on the Internet. Try to explain the meaning of climate in the EOF maps.

1.7 Use R to plot the first three PC time series from the V matrix in Problems 4.4 and 4.5. Try to explain the climate meaning of the time series.

1.8 (a) A covariance matrix C can be computed from a space-time observed anomaly data matrix X which has N rows for spatial locations and Y columns for time in years:

$$C = X \cdot X^t / Y \tag{1.67}$$

This is an $N \times N$ matrix. Choose an X data matrix from the USHCN annual total precipitation data at three California stations from north to south [Berkeley, CA (040693); Santa Barbara, CA (047902); Cuyamaca, CA (042239)] and five years from 2001 to 2005. Consider the anomaly data with respect to the 2001-2005 mean and use R to calculate a covariance matrix for $N = 3$ and $Y = 5$.
(b) Use R to find the inverse matrix of the covariance matrix C.
(c) Use R to find the eigenvalues and eigenvectors of C.
(d) Use R to make SVD decomposition of the data matrix $X = UDV^t$. Explicitly write out the three matrices U, D and V.
(e) Use R to explore the relationship between the eigenvalues of C and the matrix D.

(f) Compare the eigenvectors of C and the matrix U.
(g) Plot the PC time series and describe their behavior.

1.9 The burning of methane (CH_4) with oxygen (O_2) produces water (H_2O) and carbon dioxide (CO_2). Balance the chemical reaction equation.

1.10 The burning of propane (C_3H_8) with oxygen (O_2) produces water (H_2O) and carbon dioxide (CO_2). Balance the chemical reaction equation.

1.11 The burning of gasoline (C_8H_{18}) with oxygen (O_2) produces water (H_2O) and carbon dioxide (CO_2). Balance the chemical reaction equation.

2 Matrix Theory and Visualization

This chapter includes a slightly more advanced theory of matrix compared with the last. It includes (i) the concepts of independence, (ii) spanned spaces by multiple vectors, (iii) rank and other properties of a matrix, (iv) more on SVD, and (v) visualization of matrices and their decomposed vectors and singular values.

Many mathematicians regard mathematics as "beautiful," using terms such as "elegant," "deep," and "general." Our book, however, pays more attention to *relevant, useful, and modern (RUM)* from the perspectives of both mathematical sciences and other fields. *Relevant* is reflected by that every matrix and its operations can have an interpretation story that can be easily understood by a layman. *Useful* is reflected by that each theory or method has non-trivial application examples in science or engineering. *Useful* is reflected in the extensive use of matrices in modern data science, machine learning, R or Python programming, which were not the case two or three decades ago. Matrix visualization is an example of modernness.

This chapter continues to feature the space-time data arrangement, which uses rows of a matrix for spatial locations, and columns for temporal steps. This is the universal and fundamental information structure of our world. The singular value decomposition (SVD) helps reveal the spatial and temporal features of climate dynamics as singular vectors and the strength of their variability as singular values.

2.1 Matrix Definitions

A matrix is a rectangular array of numbers (or even expressions), often denoted by an upper case letter in either boldface or plain $\mathbf{A}$ or A:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{np} \end{bmatrix} \tag{2.1}$$

This is an $n \times p$ matrix, in which a_{ij} are called elements or entries of the matrix $\mathbf{A}$, i is the row index from 1 to n and j is the column index from 1 to p. The dimensions of a matrix are denoted by subscript, e.g., $\mathbf{A}_{n\times p}$ indicating that the matrix $\mathbf{A}$ has n rows and p columns. The matrix may also be indicated by square brackets around

a typical element $\mathbf{A} = [a_{ij}]$ or sometimes $\{A\}_{ij}$, maybe even A_{ij}. If $n = p$, then the the array is a square matrix.

Figure 2.1 is an example of a space-time climate data matrix. It is a subset of the $5^\circ \times 5^\circ$ gridded monthly surface air temperature anomalies from the NOAA Merged Land Ocean Global Surface Temperature Analysis (NOAAGlobalTemp) (Version 4.0). The rows are indexed according to the spatial locations prescribed by the latitude and longitude of the centroid of a $5^\circ \times 5^\circ$ grid box (See the entries of the first two columns in boldface). The columns are indexed according to time (See the first row entries in boldface). The other entries are the temperature anomalies with respect to the 1971-2000 monthly climatology. The anomalies are arranged according to the locations by rows and the time by columns. The units for the anomaly data are °C.

For a given month, the spatial temperature data on the Earth's surface is itself a 2-dimensional array. To make a space-time matrix, we assign each grid box a unique index s from 1 to n if the spatial region has n grid boxes. The index assignment is subjective, depending on the application needs. The commonly used way is to fix a latitude and increase the index number as the longitude increases, as indicated by the first two columns of the data matrix shown in Figure 2.1. When the longitude is finished at this latitude band, go to the next latitude band until the completion of the latitude. This can go from south to north, from north to south. Of course, one can fix the longitude first, and increase the index according to the ascending or descending order of latitudes.

Lat	Lon	1934-3	1934-4	1934-5	1934-6	1934-7	1934-8	1934-9	1934-10	1934-11	1934-12	1935-1	1935-2	1935-3	1935-4	1935-5
32.5	**242.5**	1.86	1.14	1.03	-0.65	0.12	-0.27	-0.30	-0.30	0.13	0.34	-0.48	-0.18	-1.43	-0.50	-0.84
32.5	**247.5**	3.14	2.42	2.29	-2.08	0.93	-0.09	-0.49	0.46	0.21	0.79	0.07	-0.18	-2.04	-0.44	-2.32
32.5	**252.5**	1.42	1.94	2.47	-0.24	1.18	1.04	0.11	1.30	0.46	0.73	0.93	-1.07	-0.09	0.18	-2.32
32.5	**257.5**	-0.97	0.96	0.95	1.89	1.44	1.49	0.50	2.68	1.55	0.87	2.65	-0.36	1.95	0.64	-2.18
32.5	**262.5**	-1.89	1.09	0.51	2.64	2.12	2.36	0.07	2.67	1.90	0.47	2.40	0.14	2.50	0.04	-1.66
32.5	**267.5**	-1.36	0.82	-0.06	1.36	0.89	1.46	-0.55	2.28	1.19	-0.17	2.14	0.54	3.27	0.21	-0.35
32.5	**272.5**	-0.98	0.61	0.26	0.82	0.36	0.38	-0.15	1.37	0.93	-0.66	1.31	0.23	2.45	0.76	0.85
32.5	**277.5**	-1.26	0.51	-0.24	0.75	0.65	0.39	0.71	0.91	0.36	-0.85	0.99	-0.25	2.12	0.68	0.90
32.5	**282.5**	0.54	0.88	0.09	0.25	0.32	0.13	0.20	1.08	0.51	0.55	0.81	0.87	1.31	0.94	0.77
32.5	**287.5**	0.72	0.99	0.29	0.39	0.36	0.12	0.50	1.04	0.32	0.38	0.23	0.46	0.62	0.58	0.39
32.5	**292.5**	0.79	0.93	0.27	0.48	0.23	0.18	0.90	1.01	0.48	0.22	-0.23	0.11	0.21	0.30	-0.04
32.5	**297.5**	0.68	0.59	0.26	0.33	0.17	0.17	0.82	0.69	0.50	0.26	-0.42	-0.15	-0.11	0.07	-0.37
32.5	**302.5**	0.63	0.42	0.33	0.35	0.46	0.21	0.65	0.48	0.34	0.27	-0.64	-0.27	-0.40	-0.13	-0.59
32.5	**307.5**	0.69	0.43	0.48	0.54	0.69	0.20	0.46	0.33	0.25	0.26	-0.63	-0.15	-0.37	-0.12	-0.54
32.5	**312.5**	0.80	0.51	0.44	0.44	0.68	0.26	0.45	0.41	0.26	0.28	-0.28	0.08	-0.16	0.05	-0.21
32.5	**317.5**	0.83	0.47	0.16	0.26	0.61	0.36	0.47	0.49	0.14	0.10	-0.01	0.21	0.04	0.23	0.24
32.5	**322.5**	0.62	0.16	-0.19	0.10	0.44	0.39	0.41	0.43	-0.04	-0.09	-0.10	0.10	0.09	0.33	0.45
32.5	**327.5**	0.24	-0.29	-0.54	0.05	0.27	0.29	0.21	0.23	-0.35	-0.19	-0.21	-0.03	0.09	0.40	0.52

Fig. 2.1 A subset of the monthly surface air temperature anomalies from the NOAAGlobalTemp Version 4.0 dataset.

Following this spatial index as the row number, the climate data for a given

month is a column vector. If the dataset has data for p months, then the space-time data matrix has p columns. If the dataset has n grid boxes, then the data forms an $n \times p$ space-time data matrix. You can conveniently use row or column operations of a computer language to calculate statistics of the dataset, such as spatial average, temporal mean, temporal variance, etc.

For more explicit indication of space and time, you may use s for the row index and t for the column index in a space-time data matrix. Thus, $[A_{st}]$ indicates a space-time data matrix.

This space-time indexing can be extended to the data in 3D space and 1D time, as long as we can assign a unique ID s from 1 to n for a 3D grid box and a unique ID t for time. The presently popular netCDF (Network Common Data Form) data format in climate science, denoted by `.nc`, uses this index procedure for a 4D dataset. For example, to express the output of a 3D climate model, you can start your index longitude first for a given altitude and altitude. When longitude exhausts, count the next latitude. When the latitude exhausts, count the next altitude until the last layer of the atmosphere or ocean. Eventually, a space-time data matrix is formed $[A_{st}]$.

To visualize the row data of a space-time data matrix $[A_{st}]$, just plot a line graph of the row data against time. To visualize the column data of of a space-time data matrix $[A_{st}]$, you need to convert the column vector into a 2D pixel format for a 2D domain (e.g., the contiguous United States (CONUS) region), or a 3D data array format for a 3D domain (e.g., the CONUS atmosphere domain from the 1,000 mb surface level to the 10 mb height level). This means that the climate data are be represented in another matrix format, such as the 5° surface air temperature anomaly data for the entire world for December 2015 visualized by Fig. B.8. The data behind the figure is a 36×72 data matrix whose rows are for latitude and columns for longitude. This data matrix is in space-space pixel format like the data for a photo, and each time corresponds to a new space-space pixel data matrix. Thus, the latitude-longitude-time forms a data 3D array. With elevation, then latitude-longitude-altitude-time forms a 4D data array, which is often written in the netCDF file in climate science. You can use the 4DVD data visualization tool `www.4dvd.org`, described in Chapter 1, to visualize the 4D Reanalysis data array as an example to understand the space-time data plot, and netCDF data structure.

The coordinates (32.5, 262.5) in the 6th row of Fig. 2.1 indicates a $5° \times 5°$ grid box centered at (32.5°N, 97.5°W). This box covers part of Texas. The large temperature anomalies for the summer of 1934 (2.64°C for June, 2.12°C for July, and 2.36°C for August) were in the 1930s "Dust Bowl" period. The hot summer of 1934 was a wave of severe drought. The disastrous dust storms in the 1930s over the American and Canadian prairies destroyed many farms and greatly damaged the ecology.

2.2 Fundamental Properties of Matrices

This section provides a concise list to summarize fundamental properties and the commonly used operations of matrices. We limit our materials to the basics that are sufficient for this book.

(i) Zero matrix: A zero matrix has its every entry equal to zero: $\mathbf{0} = [0]$, or $0 = [0]$, or explicitly

$$\mathbf{0} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{2.2}$$

(ii) Identity matrix: An identity matrix is a square matrix whose diagonal entries are all equal to one and whose off-diagonal entries are all equal to zero, and is denoted by I or $\mathbf{I}$. See an expression of an identity matrix below:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.3}$$

People also use the following notation

$$\mathbf{I} = [\delta_{ij}], \tag{2.4}$$

where

$$\delta_{ij} = \begin{cases} 1 & \text{when } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

is called the Kronecker delta.

An identity matrix may be regarded as a special case of a *diagonal matrix*, which refers to any square matrix whose off-diagonal elements are zero. Hence, a diagonal matrix has the following general expression:

$$\mathbf{D} = [d_i \delta_{ij}], \tag{2.6}$$

where $d_1, d_2, \cdots, d_n$ are the n-diagonal elements. If $d_i = 1, i = 1, 2, \cdots, n$, then the diagonal matrix becomes an identity matrix.

(iii) A transpose matrix: The *transpose* of a matrix A is obtained by interchanging the rows and columns. The new matrix is denoted by $\mathbf{A}^t$. Computing the matrix transpose is very easy, simply rotating each horizontal row clockwise to a vertical column, one row at a time. If $\mathbf{A}$ has dimension $n \times p$, its

transpose has dimension $p \times n$. The elements of the transposed matrix are related to the originals by

$$(A^t)_{ij} = A_{ji} \tag{2.7}$$

For example, if

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{2.8}$$

then

$$\mathbf{A}^t = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \tag{2.9}$$

The equation $\mathbf{A}^t = \mathbf{A}$ or $a_{ij} = a_{ji}$ is equivalent to that the matrix $\mathbf{A}$ is symmetric. Of course, a symmetric matrix must be a square matrix.

(iv) Equal matrices: Two matrices $\mathbf{A}$ and $\mathbf{B}$ are equal if every pair of corresponding entries are equal, i.e., the equation $\mathbf{A} = \mathbf{B}$ is equivalent to $a_{ij} = b_{ij}$ for all i and j.

(v) Matrix addition: The sum of two matrices $\mathbf{A}$ and $\mathbf{B}$ is defined by the sum of corresponding entries, i.e., $\mathbf{A} + \mathbf{B} = [a_{ij} + b_{ij}]$.

(vi) Matrix subtraction: The difference of two matrices $\mathbf{A}$ and $\mathbf{B}$ is defined by the difference of corresponding entries, i.e., $\mathbf{A} - \mathbf{B} = [a_{ij} - b_{ij}]$. The equation of $\mathbf{A} = \mathbf{B}$ is equivalent to $\mathbf{A} - \mathbf{B} = 0$.

(vii) Row vector: A row vector is of dimension p is a $1 \times p$ matrix:

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_p \end{bmatrix}. \tag{2.10}$$

Matrix $\mathbf{A}_{n\times p}$ may be regarded as a stack of n row vectors $\mathbf{a}_{i:}, i = 1, 2, \cdots, n$, each of which is a p-dimensional row vector. Hence,

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_{1:} \\ \mathbf{a}_{2:} \\ \cdots \\ \mathbf{a}_{n:} \end{bmatrix}. \tag{2.11}$$

Here, : in the second position of the double-index subscript means the inclusion of all the columns.

(viii) Column vector: A column vector of dimension n is an $n \times 1$ matrix

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_n \end{bmatrix}. \tag{2.12}$$

The transpose of a column vector becomes a row vector, and vice versa.

Matrix $\mathbf{A}_{n\times p}$ may be regarded as an array of p column vectors $\mathbf{a}_{:j}, j = 1, 2, \cdots, p$, each of which is an n-dimensional column vector. Hence,

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_{:1} & \mathbf{a}_{:2} & \cdots & \mathbf{a}_{:p} \end{bmatrix}. \tag{2.13}$$

Here, : in the first position of the double-index subscript means the inclusion of all the rows.

(ix) Dot product of two vectors: Two vectors of the same dimension can form a *dot product* which is equal to the sum of the products of the corresponding entries:

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n. \tag{2.14}$$

The dot product is also called an *inner product*.

As example, if

$$\mathbf{u} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}, \tag{2.15}$$

then

$$\mathbf{u} \cdot \mathbf{v} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 32. \tag{2.16}$$

The *amplitude of vector* $\mathbf{u}$ of dimension n is defined as

$$|\mathbf{u}| = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2}. \tag{2.17}$$

Sometimes, the amplitude is also called length, or Euclidean length, or magnitude. Please do not mix the concept of Euclidean length of a vector with the dimensional length of a vector. The latter means the number of entries of a vector, i.e., n.

If the Euclidean length of $\mathbf{u}$ is equal to one, we say that the $\mathbf{u}$ is a *unit vector*. If every element of $\mathbf{u}$ is zero, then we say that $\mathbf{u}$ is a *zero vector*.

By the definition of dot product, we have

$$|\mathbf{u}|^2 = \mathbf{u} \cdot \mathbf{u}. \tag{2.18}$$

If $\mathbf{u} \cdot \mathbf{v} = 0$, we say that $\mathbf{u}$ and $\mathbf{v}$ are *orthogonal*. Further, if $\mathbf{u} \cdot \mathbf{v} = 0$ and $|\mathbf{u}| = |\mathbf{v}| = 1$, then we say that $\mathbf{u}$ and $\mathbf{v}$ are *orthonormal*.

(x) Matrix multiplication: The product of matrix $\mathbf{A}_{n\times p}$ and matrix $\mathbf{B}_{p\times m}$ is an $n \times m$ matrix $\mathbf{C}_{n\times m}$ whose element c_{ij} is the dot product of the ith row vector of A and jth column vector of $\mathbf{B}$:

$$c_{ij} = \mathbf{a}_{i:} \cdot \mathbf{b}_{:j}. \tag{2.19}$$

We denote

$$\mathbf{C}_{n\times m} = \mathbf{A}_{n\times p}\mathbf{B}_{p\times m}, \tag{2.20}$$

or

$$\mathbf{C} = \mathbf{AB}. \tag{2.21}$$

Note that the number of columns of $\mathbf{A}$ and that of rows of $\mathbf{B}$ must be the same when the multiplication $\mathbf{AB}$ can be made, because the dot product $\mathbf{a}_{i:} \cdot \mathbf{b}_{:j}$ requires this condition. This is referred to as the dimension-matching

condition for matrix multiplication. If this condition is violated, the two matrices cannot be multiplied. For example, for the following two matrices

$$\mathbf{A}_{3\times 2} = \begin{bmatrix} 1 & 0 \\ 0 & 4 \\ 3 & 2 \end{bmatrix} \qquad \mathbf{B}_{2\times 2} = \begin{bmatrix} 0 & -1 \\ 1 & 2 \end{bmatrix} \tag{2.22}$$

We can compute

$$\mathbf{A}_{3\times 2}\mathbf{B}_{2\times 2} = \begin{bmatrix} 0 & -1 \\ 4 & 8 \\ 2 & 1 \end{bmatrix} \tag{2.23}$$

However, the expression

$$\mathbf{B}_{2\times 2}\mathbf{A}_{3\times 2} \tag{2.24}$$

is not defined, because the dimensions do not match. Thus, for matrix multiplication of two matrices, their order is important. The product $\mathbf{BA}$ may not be equal to $\mathbf{AB}$ even when both are defined. That is, the commutative law does not hold for matrix multiplication.

The dot product of two vectors can be written as the product of two matrices. If $\mathbf{u}$ and $\mathbf{v}$ are n-dimensional column vectors, then

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^t\mathbf{v}. \tag{2.25}$$

The right hand side is a $1\times n$ matrix times an $n\times 1$ matrix, and the product is a 1×1 matrix, whose element is the result of the dot product. Computer usually programs dot product in this way of matrix multiplication.

A scaler can always multiply a matrix, which is defined as follows. Given a scalar c and a matrix $\mathbf{A}$, their product is

$$c\mathbf{A} = [ca_{ij}] = \mathbf{A}c. \tag{2.26}$$

The scaler multiplication can be extended to multiple vectors $(\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_p)$ or matrices to form a *linear combination*:

$$\mathbf{u} = c_1\mathbf{u}_1 + c_2\mathbf{u}_2 + \cdots + c_p\mathbf{u}_p. \tag{2.27}$$

where $c_1, c_2, \cdots, c_p$ are coefficients of the linear combination and at least one of the coefficients is non-zero. Multivariate linear regression discussed at the end of last chapter is a linear combination. This is a very useful mathematical expression in data science.

(xi) Matrix inversion: For a given square matrix $\mathbf{A}$, if there is a matrix $\mathbf{B}$ such that

$$\mathbf{BA} = \mathbf{AB} = \mathbf{I}, \tag{2.28}$$

then $\mathbf{B}$ is called the *inverse matrix* of $\mathbf{A}$, denoted by $\mathbf{A}^{-1}$, i.e.,

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}. \tag{2.29}$$

Not all the matrices have an inverse. If a matrix has an inverse, then the matrix is said to be *invertible*. Equivalently, $\mathbf{A}^{-1}$ exists.

As an example of the matrix inversion, given

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix}, \tag{2.30}$$

we have

$$\mathbf{A}^{-1} = \begin{bmatrix} 2/3 & 1/3 \\ -1/3 & 1/3 \end{bmatrix}. \tag{2.31}$$

Hand-calculation for the inverse of a small matrix is already very difficult, and that for the inverse of a large matrix is almost impossible. Computers can do the calculations for us, as shown in examples later.

According to the definition of inverse, we have the following formula for the inverse of the product of two matrices:

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}, \tag{2.32}$$

if both $\mathbf{A}$ and $\mathbf{B}$ are invertible matrices. Please note the order switch of the matrices.

With the definition of an inverse matrix, we can define the *matrix division* by

$$\mathbf{A}/\mathbf{B} = \mathbf{AB}^{-1} \tag{2.33}$$

when $\mathbf{B}^{-1}$ exists. In matrix operations, we usually do not use the concept of matrix division, but always use the matrix inverse and matrix multiplication.

(xii) More properties of the matrix transpose:

$$(\mathbf{A^t})^t = \mathbf{A} \tag{2.34}$$

$$(\mathbf{A} + \mathbf{B})^t = \mathbf{A}^t + \mathbf{B}^t \tag{2.35}$$

$$(\mathbf{AB})^t = \mathbf{B}^t\mathbf{A}^t \tag{2.36}$$

$$(\mathbf{A^{-1}})^t = (\mathbf{A}^t)^{-1}. \tag{2.37}$$

(xiii) Orthogonal matrices: An *orthogonal matrix* [1] is one whose row vectors are orthonormal. In this case, the inverse matrix can be easily found: It is its transpose. That is, if $\mathbf{A}$ is an orthogonal matrix, then

$$\mathbf{A}^{-1} = \mathbf{A}^t. \tag{2.38}$$

The proof of this claim is very simple. The orthonormal property of the row vectors of $\mathbf{A}$ implies that

$$\mathbf{A}\mathbf{A}^t = \mathbf{I}. \tag{2.39}$$

By the definition of matrix inverse, $\mathbf{A}^t$ is the inverse matrix of $\mathbf{A}$.

[1] Although *orthogonal matrix* is a standard mathematical terminology, it is acceptable if you call it *orthonormal matrix*

If $\mathbf{A}$ is an orthogonal matrix, its row vectors are also orthonormal. This can be proved by multiplying both sides of the above by $\mathbf{A}^t$ from the left:

$$\mathbf{A}^t\mathbf{A}\mathbf{A}^t = \mathbf{A}^t\mathbf{I}. \tag{2.40}$$

Then multiply both sides of this equation by $(\mathbf{A}^t)^{-1}$ from the right:

$$\mathbf{A}^t\mathbf{A}(\mathbf{A}^t(\mathbf{A}^t)^{-1}) = \mathbf{A}^t\mathbf{I}(\mathbf{A}^t)^{-1}, \tag{2.41}$$

which yields

$$\mathbf{A}^t\mathbf{A} = \mathbf{I}. \tag{2.42}$$

This implies that the column vectors of $\mathbf{A}$ are orthonormal.

As an example, the following matrix

$$\mathbf{T} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{2.43}$$

is an orthogonal matrix for any given real number θ. You can easily verify this using the trigonometrical identify $\sin^2\theta + \cos^2\theta = 1$. We thus have

$$\mathbf{T}^{-1} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}, \tag{2.44}$$

You can easily verify that $\mathbf{T}^{-1}\mathbf{T} = \mathbf{I}$ by hand-calculation of the product of the two matrices in this equation.

2.3 Some basic concepts and theories of linear algebra

According to `Encyclopedia.com`, "linear algebra originated as the study of linear equations." Linear algebra deals with vectors, matrices and vector spaces. Before the 1950s, it was part of Abstract Algebra (Tucker 1993). In 1965 the Committee on the Undergraduate Program in Mathematics, Mathematical Association of America, outlined the following topics for a stand-alone linear algebra course: Linear systems, matrices, vectors, linear transformations, unitary geometry with characteristic values. The vectors and matrices have been dealt in the previous two sections of this chapter. This section deals with linear systems of equations, and linear transformations, and next with characteristic values.

2.3.1 Linear equations

A meteorologist needs to make a decision on what instruments to order under the following constraint. She is given a budget of $10,000 to purchase 30 instruments for her observational sites. Her supplier has two products for the instrument: The first is $30 per set, and the second $40 per set. She would like to buy the second type of instrument as many as possible under the budget constraint. Then, the question is

how many instruments of the second kind she can buy? This problem leads to the following linear system of two equations:

$$30x_1 + 40x_2 = 1000, \tag{2.45}$$

$$x_1 + x_2 = 30. \tag{2.46}$$

The solution to these linear equations is $x_1 = 20$ and $x_2 = 10$.

This system of linear equations can be expressed in matrix and vectors as follows:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.47}$$

where

$$\mathbf{A} = \begin{bmatrix} 30 & 40 \\ 1 & 1 \end{bmatrix} \tag{2.48}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.49}$$

$$\mathbf{b} = \begin{bmatrix} 1000 \\ 30 \end{bmatrix} \tag{2.50}$$

$$\tag{2.51}$$

Then, the solution of this system may be tightly expressed in the following way:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}. \tag{2.52}$$

This expression is convenient for mathematical proofs, but is rarely used for solving a linear system, because finding an inverse matrix is computationally costly. A way to solve a linear system is to use Gauss elimination. The corresponding computing procedure is called the row operation on a matrix. There are numerous ways of solving a linear system. Some are particularly efficient for a certain system, such as a sparse matrix or a matrix of a diagonal bend of width equal 3 or 5. Efficient algorithms for a linear system, particularly an extremely large system, are forever a research topic. In this book we use a computer to solve a linear system without studying the algorithm details. The R and Python commands are below:

```
solve(A, b) #R code for finding x
numpy.linalg.solve(A, b) #Python code
```

2.3.2 Linear transformations

A *linear transformation* is to convert vector $\mathbf{x}_{n\times 1}$ into $\mathbf{y}_{m\times 1}$ using the multiplication of a matrix $\mathbf{T}_{m\times n}$:

$$\mathbf{y}_{m\times 1} = \mathbf{T}_{m\times n}\mathbf{x}_{n\times 1} \tag{2.53}$$

For example, the matrix

$$\mathbf{T} = \begin{bmatrix} -0.1 & 4 \\ 0.1 & -3 \end{bmatrix} \tag{2.54}$$

transforms the vector

$$\begin{bmatrix}1000\\30\end{bmatrix} \tag{2.55}$$

into

$$\begin{bmatrix}20\\10\end{bmatrix} \tag{2.56}$$

This is the solution of the linear system in the previous subsection.

Usually, the linear transformation $\mathbf{Tx}$ changes both direction and magnitude of $\mathbf{x}$. However, if $\mathbf{T}$ is an orthogonal matrix, then $\mathbf{Tx}$ does not change the magnitude of $\mathbf{x}$, and changes only the direction. This claim can be simply proved by the following formula:

$$|\mathbf{Tx}|^2 = (\mathbf{Tx})^t\mathbf{Tx} = \mathbf{x}^t\mathbf{T}^t\mathbf{Tx} = \mathbf{x}^t(\mathbf{T}^t\mathbf{T})\mathbf{x} = \mathbf{x}^t\mathbf{Ix} = |\mathbf{x}|^2. \tag{2.57}$$

Thus, if $\mathbf{T}$ is an orthogonal matrix, then $\mathbf{Tx}$ is a rotation of the vector $\mathbf{x}$.

2.3.3 Linear independence

The vectors $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_p}$ are *linearly dependent* if no vector can be represented by a linear combination of other $p-1$ vectors in this group. Otherwise, the group of vectors are linearly dependent.

If it is not linearly independent, then there must be a vector which can be represented by the other vectors through a linear combination. Suppose this vector is $\mathbf{x_1}$, then

$$\mathbf{x_1} = d_2\mathbf{x_2} + \cdots + d_p\mathbf{x_p}, \tag{2.58}$$

where at least one of the coefficients $d_2, d_3, \cdots, d_p$ is non-zero. Thus, the linear system of equations for $c_1, c_2, c_3, \cdots, c_p$

$$c_1\mathbf{x_1} + c_2\mathbf{x_2} + \cdots + c_p\mathbf{x_p} = 0 \tag{2.59}$$

has a non-zero solution. This system can be written as a matrix form

$$\mathbf{Xc} = \mathbf{0}, \tag{2.60}$$

where column vectors $\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_p}$ form the matrix $\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix}\mathbf{x_1} & \mathbf{x_2} & \cdots & \mathbf{x_p}\end{bmatrix}, \tag{2.61}$$

the unknown vector is $\mathbf{c}$

$$\mathbf{c} = \begin{bmatrix}c_1\\c_2\\\vdots\\c_p\end{bmatrix}, \tag{2.62}$$

and $\mathbf{0}$ is the p-dimensional zero column vector. The solution of this matrix equation is

$$\mathbf{c} = \mathbf{X}^{-1}\mathbf{0} = \mathbf{0}. \tag{2.63}$$

However, $\mathbf{c}$ must not be zero. This contradiction implies that $\mathbf{X}^{-1}$ does not exist if the the column vectors are linearly dependent. In other words, if $\mathbf{X}^{-1}$ exists, then its column vectors are linearly independent.

Consider vectors in a three dimensional space. Any two column vectors $\mathbf{x}_2$ and $\mathbf{x}_3$ define a plane. If $\mathbf{x}_1$ can be written as a linear combination of $\mathbf{x}_2$ and $\mathbf{x}_3$, then it must lie in the same plane. So, $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ are linearly dependent. The matrix $[\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]_{3\times 3}$ is not invertible.

2.3.4 Determinants

For a square matrix $\mathbf{A}$, a convenient notation and concept is its *determinant*. It is a scaler and is denoted by $\det(\mathbf{A})$ or $|\mathbf{A}|$. For a 2×2 matrix

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \tag{2.64}$$

its determinant is

$$\det[A] = ad - cd. \tag{2.65}$$

For a higher-dimensional matrix, the computation is quite complex and is computationally expensive. We usually do not need to calculate the determinant of a large matrix, say $\mathbf{A}_{172\times 172}$. The computer command for computing the determinant of a small square matrix is as follows:

```
det(A) #R command for determinant
np.linalg.det(a) #Python command for determinant
```

Two 2-dimensional column vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ can span a parallelogram, whose area S is equal to the absolute value of the determinant of the matrix consisting of the two vectors $\mathbf{A} = [\mathbf{x}_1\ \mathbf{x}_2]$:

$$S = \big|\det[\mathbf{x}_1\ \mathbf{x}_2]\big|. \tag{2.66}$$

Three 3-dimensional column vectors $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$ can span a parallelepiped, whose volume V is equal to the absolute value of the determinant of the matrix consisting of the three vectors $\mathbf{A} = [\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]$:

$$V = \big|\det[\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]\big|. \tag{2.67}$$

A few commonly used properties of determinant are listed below

(a) The determinant of a diagonal matrix is the product of its diagonal elements.
(b) If a determinant has a zero row or column, the determinant is zero.
(c) The determinant does not change after a matrix transpose, i.e., $\det[\mathbf{A}^t] = \det[\mathbf{A}]$.
(d) The determinant of the product of two matrices: $\det[\mathbf{AB}] = \det[\mathbf{A}]\det[\mathbf{B}]$.
(e) The determinant of the product of a matrix with a scaler: $\det[c\mathbf{B}] = c^n \det[\mathbf{A}]$, if $\mathbf{A}$ is an $n \times n$ matrix.
(f) The determinant of an orthogonal matrix is equal to 1 or -1.

2.3.5 Rank of a matrix

The *rank* of $\mathbf{A}$ is the greatest number of columns of the matrix that are linearly independent, and is denoted by $\mathrm{r}[\mathbf{A}]$.

For a 3×3 matrix $\mathbf{A}$, if we treat each column as a 3-dimensional vector. If all three lie along a line (i.e., colinear), the rank of $\mathbf{A}$ is one. If all three of the vectors lie in a plane, but are not collinear, the rank is two. If none of the three are collinear or lie in a plane, the rank is three.

If the rank of a square matrix is less than its dimension, then at least one column can be a linear combinations of other columns, which implies that the determinant vanishes. If $\mathbf{A}$ has rank r, it is possible to find r linearly independent columns, and all the other columns are linear combinations of these r independent columns.

Some properties about the matrix rank are listed below:

(a) If $\det[\mathbf{A}_{n\times n}] \neq 0$, then $\mathrm{r}[\mathbf{A}] = n$, and the matrix $\mathbf{A}_{n\times n}$ is invertible and is said to be *nonsingular*.
(b) If $\det[\mathbf{A}] = 0$, then the rank of $\mathbf{A}$ is less than n, and $\mathbf{A}$ is not invertible and is said to be *singular*.
(c) If $\mathbf{B}$ is multiplied by a nonsingular matrix $\mathbf{A}$, the product has the same rank as $\mathbf{B}$.
(d) $0 \leq \mathrm{r}[\mathbf{A}_{n\times p}] \leq \min(n, p)$.
(e) $\mathrm{r}[\mathbf{A}] = \mathrm{r}[\mathbf{A}^t]$.
(f) $\mathrm{r}[\mathbf{AB}] \leq \min(\mathrm{r}[\mathbf{A}], \mathrm{r}[\mathbf{B}])$.
(g) $\mathrm{r}[\mathbf{AA}^t] = \mathrm{r}[\mathbf{A}^t\mathbf{A}] = \mathrm{r}[\mathbf{A}]$.
(h) $\mathrm{r}[\mathbf{A} + \mathbf{B}] \leq \mathrm{r}[\mathbf{A}] + \mathrm{r}[\mathbf{B}]$.

Computers can easily demonstrate the matrix computations following the theories presented in this chapter so far. The computer code is below.

```
#R code: Computational examples of matrices
A = matrix(c(1,0,0,4,3, 2), nrow = 3, byrow = TRUE)
B = matrix(c(0,1,-1,2), nrow = 2) #form a matrix by columns
C = A%*%B #matrix multiplication
C
#[1,]    0   -1
#[2,]    4    8
#[3,]    2    1
t(C) # transpose matrix of C
#[1,]    0    4    2
#[2,]   -1    8    1

A = matrix(c(1, -1, 1, 2), nrow =2, byrow = TRUE)
solve(A) #compute the inverse of A
#[1,]  0.6666667 0.3333333
#[2,] -0.3333333 0.3333333
A%*%solve(A) #verify the inverse of A
#[1,] 1.000000e+00    0
#[2,] 1.110223e-16    1

#Solve linear equations
```

```
A = matrix(c(30, 40, 1, 1), nrow =2, byrow = TRUE)
b = c(1000, 30)
solve(A,b)
#[1] 20 10
solve(A)%*%b #Another way to solve the equations
det(A) #compute the determinant
#[1] -10

library(Matrix)
rankMatrix(A) #Find the rank of a matrix
#[1] 2 #rank(A) = 2

#Orthogonal matrices
p = sqrt(2)/2
Q = matrix(c(p,-p,p,p), nrow=2)
Q #is an orthogonal matrix
#           [,1]      [,2]
#[1,]  0.7071068 0.7071068
#[2,] -0.7071068 0.7071068
Q%*%t(Q) #verify Q as an orthogonal matrix
#     [,1] [,2]
#[1,]    1    0
#[2,]    0    1
det(Q) #The determinant of an orthogonal matrix is 1 or -1
#[1] 1
```

```
# Python matrix multiplication
A = [[1, 0],  [0, 4],[3, 2]]
B = [[0,-1],[1,2]]
C = np.matmul(A,B) #Or C = np.dot(A,B)
print('C=', C)
#C= [[ 0 -1]
# [ 4  8]
# [ 2  1]]
print('Transpose matrix of C =', C.transpose())
#Transpose matrix of C = [[ 0  4  2]
# [-1  8  1]]

#matrix inversion
A = [[1,-1],[1,2]]
np.linalg.inv(A)# compute the inverse of A
#array([[ 0.66666667,  0.33333333],
#       [-0.33333333,  0.33333333]])

#Solve a system of linear equations
A = [[30, 40],[1, 1]]
b = [[1000],[30]]
x = np.linalg.solve(A,b)
print('x=',x)
#x= [[20.]
# [10.]]

#Compute determinant of the previous matrix A
print('Determinant det(A)=', np.linalg.det(A))
#Determinant det(A)= -10.000000000000002

#An orthogonal matrix
p = np.sqrt(2)/2
Q = [[p,p],[-p,p]]
print('Orthogonal matrix Q=', np.round(Q,2))
T = np.transpose(Q)
print('Q times transpose of Q = ', np.matmul(Q,T))
print('Determinant of Q =', np.linalg.det(Q))
#Orthogonal matrix Q= [[ 0.71  0.71]
# [-0.71  0.71]]
#Q times transpose of Q =  [[1. 0.]
# [0. 1.]]
#Determinant of Q = 1.0
```

2.4 Eigenvectors and eigenvalues

2.4.1 Definition of eigenvectors and eigenvalues

The linear transform $\mathbf{y} = \mathbf{Au}$ usually results in $\mathbf{y}$ not parallel to $\mathbf{u}$. For example,

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{2.68}$$

The vectors $\mathbf{u} = (1, 0)$ and $\mathbf{Au} = (1, 2)$ are not parallel in the 2-dimensional space. See the blue vectors in Fig. 2.2.

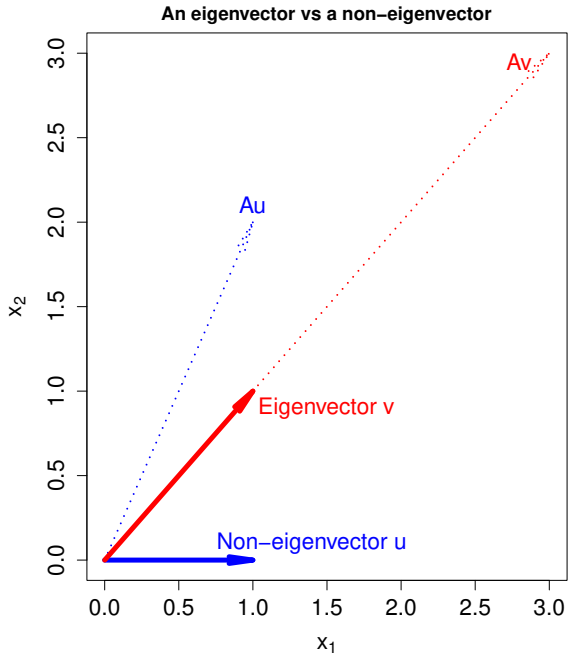


Fig. 2.2 An eigenvector $\mathbf{v}$, a non-eigenvector $\mathbf{u}$, and their linear transforms by matrix $\mathbf{A}$: $\mathbf{Av}$ and $\mathbf{Au}$. Here, $\mathbf{Av}$ and $\mathbf{v}$ are parallel, and $\mathbf{Au}$ and $\mathbf{u}$ are not parallel.

However, there exist some special vectors $\mathbf{v}$ such that such as $\mathbf{Av}$ is parallel to $\mathbf{v}$. For example, $\mathbf{v} = (1, 1)$ is such a vector, since $\mathbf{Av} = (3, 3)$ is in the same direction as $\mathbf{v} = (1, 1)$. See the red vectors in Fig. 2.2. If two vectors are parallel, then one vector is a scalar multiplication of the other, e.g., $(3, 3) = 3(1, 1)$. We denote this scalar by λ. Thus,

$$\mathbf{Av} = \lambda\mathbf{v}. \tag{2.69}$$

These vectors v are special to A, maintain their own orientation when multiplied by A, and are called *eigenvectors*. Here, "eigen" is from German, meaning "self",

"own", "particular", or "special." [2] The corresponding scalars λ are called *eigenvalues*. The formula (2.69) is a mathematical definition of the eigenvalue problem for matrix $\mathbf{A}$.

If $\mathbf{v}$ is an eigenvector, then its multiplication to a scalar c is also an eigenvector, since

$$\mathbf{A}(c\mathbf{v}) = c\mathbf{A}\mathbf{v} = c\lambda\mathbf{v} = \lambda(c\mathbf{v}).$$

Namely, all the vectors in the same direction $\mathbf{v}$ are also eigenvectors. The eigenvectors of length equal one are called unit eigenvectors, or unitary eigenvectors, and are unique up to a positive or negative sign. Most computer programs output unit eigenvectors. Thus, eigenvector describes a direction or an oriantation. Each square matrix has its own special orientations.

The aforementioned vector

$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.70}$$

is en eigenvector that maintains its own direction after multiplied by $\mathbf{A}$:

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{2.71}$$

Thus,

$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

is an eigenvector of A and $\lambda = 3$ is an eigenvalue of A. The corresponding unit eigenvector is

$$\mathbf{e} = \mathbf{v}/|\mathbf{v}| = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

Another eigenvector for the above matrix $\mathbf{A}$ is $\mathbf{v}_2 = (1, -1)$:

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -1 \times \begin{bmatrix} 1 \\ -1 \end{bmatrix}. \tag{2.72}$$

The second eigenvalue is $\lambda_2 = -1$.

The computer code for the eigenvalues and eigenvectors of the above matrix $\mathbf{A}$ is below.

```
#R code for eigenvectors and eigenvalues
A = matrix(c(1, 2, 2, 1), nrow=2)
eigen(A)
```

[2] The "eigen" part in the word "eigenvector" is from German or Dutch and means "self" or "own", as in "one's own." Thus, an eigenvector v is A's "own" vector. In English books, the word "eigenvector" is the standard translation of the German word "eigenvektor." The word "eigenvalue" is translated from the German word "eigenwert", as "wert" means"value." Instead of eigenvector, some English publications use "characteristic vector", which indicates "characteristics of a matrix", or "its own property of a matrix." German mathematician David Hilbert (1862-1943) was the first to use "eigenvektor", and "eigenwert" in his 1904 article about a general theory of linear integral equations.

```
#$values
#[1]  3 -1
#$vectors
#[,1]       [,2]
#[1,] 0.7071068 -0.7071068
#[2,] 0.7071068  0.7071068
```

```
#Python code for eigenvectors and eigenvalues
A = [[1,2], [2,1]]
np.linalg.eig(A)
#(array([ 3., -1.]), array([[ 0.70710678, -0.70710678],
#        [ 0.70710678,  0.70710678]]))
```

If $\mathbf{A}$ is an $N \times N$ matrix, then it has N eigenvalues and eigenvectors $(\lambda_n, v_n), n = 1, 2, \cdots, N$ for the following reason. The eigenvector $\mathbf{v}$ satisfies

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{v} = 0. \tag{2.73}$$

The non-zero solution $\mathbf{v}$ of this equation requires that

$$\det(\mathbf{A} - \lambda \mathbf{I}) = 0. \tag{2.74}$$

Expanding the determinant out leads to an Nth degree polynomial in λ, which has exactly N roots. Some roots may be repeated and hence counted multiple times toward N. Some roots may be complex numbers, which are not discussed in this book. Each root is an eigenvalue and corresponds to an eigenvector.

The above concise determinant expression is tidy and useful for mathematical proofs, but is not used as a computer algorithm for calculating eigenvalues or eigenvectors, because it is computationally costly or even impossible for a large matrix. Nonetheless, some traditional textbooks of linear algebra defined eigenvalues using Eq. (2.74). This traditional definition of eigenvalues appears to be abstract. The eigenvector equation (2.73) seems not providing an image of the eigenvector. Thus, many students forget the definition of eigenvalues and eigenvectors shortly after the final exams of a linear algebra course.

Figure 2.2 may be generated by the following computer code.

```
#R plot eigenvector v vs a non-eigenvector u
#Create your working directory named LinAlg and go there
setwd('/Users/sshen/LinAlg')
getwd() #Verify that you are in the directory/folder
#[1] "/Users/sshen/LinAlg"

setEPS() #Plot the figure and save the file
postscript("fig0502.eps", width = 6)
par(mar=c(4.5,4.5,2.0,0.5))
plot(9,9,
     main = 'An␣eigenvector␣vs␣a␣non-eigenvector',
     cex.axis = 1.4, cex.lab = 1.4,
     xlim = c(0,3), ylim=c(0,3),
     xlab = bquote(x[1]), ylab = bquote(x[2]))
```

```
arrows(0,0, 1,0, length = 0.25,
       angle = 8, lwd = 5, col = 'blue')
arrows(0,0, 1,2, length = 0.3,
       angle = 8, lwd = 2, col = 'blue',  lty = 3)
arrows(0,0, 1,1, length = 0.25,
       angle = 8, lwd = 5, col='red')
arrows(0,0, 3,3, length = 0.3,
       angle = 8, lwd = 2, col='red', lty = 3)
text(1.4,0.1, 'Non-eigenvector u', cex =1.4, col = 'blue')
text(1.0,2.1, 'Au', cex =1.4, col = 'blue')
text(1.5,0.9, 'Eigenvector v', cex =1.4, col = 'red')
text(2.8, 2.95, 'Av', cex =1.4, col = 'red')
dev.off()
```

```
#Python plot eigenvector vs a non-eigenvector
import matplotlib.patches as patches
fig = plt.figure(figsize = (12,12))

plt.axes().set_xlim(-0.1,3.1)
plt.axes().set_ylim(-0.1,3.1)
plt.axes().set_aspect(1)
style = "Simple, tail_width=0.5,head_width=8,head_length=18"
kw1 = dict(arrowstyle=style, color="blue")
kw2 = dict(arrowstyle=style, color="red")

a1 = patches.FancyArrowPatch((0,0), (1,0), **kw1,
                             linewidth =5)
a2 = patches.FancyArrowPatch((0, 0), (1,2),**kw1)
a3 = patches.FancyArrowPatch((0, 0), (1,1), **kw2,
                             linewidth = 5)
a4 = patches.FancyArrowPatch((0, 0), (3,3),**kw2)
for a in [a1, a2, a3, a4]:
    plt.gca().add_patch(a)
plt.title('An eigenvector v vs a non-eigenvector u')
plt.xlabel(r'$x_1$', fontsize = 25)
plt.ylabel(r'$x_2$', fontsize = 25)
plt.text(0.6, 0.1, 'Non-eigenvector u',
          color = 'blue', fontsize =25)
plt.text(0.8, 2.0, 'Au',
          color = 'blue', fontsize =25)
plt.text(1.03,0.85, 'Eigenvector v',
          color = 'red', fontsize =25)
plt.text(2.7, 2.9, 'Av',
          color = 'red', fontsize =25)
plt.show()
```

2.4.2 Properties of eigenvectors and eigenvalues for a symmetric matrix

A covariance matrix or a correlation matrix is a symmetric matrix and is often used in climate science. For a symmetric matrix $\mathbf{A}$, its eigenvalues and eigenvectors have the following properties:

(a) Eigenvalues of a symmetric matrix are real numbers.

(b) The n different unit eigenvectors of a symmetric matrix $\mathbf{A}_{n\times n}$ are independent and form an orthonormal set, i.e., $\mathbf{e}^{(\ell')} \cdot \mathbf{e}^{(\ell)} = \delta_{\ell\ell'}$, where $\mathbf{e}^{(\ell')}$ and $\mathbf{e}^{(\ell)}$ are any two different unit eigenvectors. We can use these unit vectors to express any n-dimensional vector using a linear combination:

$$\mathbf{x} = c_1\mathbf{e}^{(1)} + c_2\mathbf{e}^{(2)} + \cdots + c_n\mathbf{e}^{(n)}, \tag{2.75}$$

where $c_1, c_2, \cdots, c_n$ are scaler coefficients of the linear combination.

(c) If all the eigenvalues of $\mathbf{A}_{n\times n}$ are positive, then the *quadratic form*

$$Q(\mathbf{x}) = \mathbf{x}^t\mathbf{A}\mathbf{x} = \sum_{i,j=1}^{n} a_{ij}x_ix_j \tag{2.76}$$

is also a positive scaler for any non-zero vector $\mathbf{x}$. A quadratic form may be used to express kinetic energy or total variance in climate science. The kinetic energy in climate science is always positive. For all the unit vectors $\mathbf{x}$, the maximum of the quadratic form is equal to the largest eigenvalue of $\mathbf{A}$. The maximum is achieved when $\mathbf{x}$ is the corresponding eigenvector of $\mathbf{A}$.

(d) The rank of matrix $\mathbf{A}$ is equal to the number of nonzero eigenvalues, where the multiplicity of repeated eigenvalues is counted.

(e) Eigenvalues of a diagonal matrix are equal to the diagonal elements.

(f) For a symmetric matrix $\mathbf{A}_{n\times n}$, its n unit column eigenvectors form an orthogonal matrix $\mathbf{Q}_{n\times n}$ such that $\mathbf{A}_{n\times n}$ can be diagonalized by $\mathbf{Q}_{n\times n}$ in the following way

$$\mathbf{Q}^t_{n\times n}\mathbf{A}_{n\times n}\mathbf{Q}_{n\times n} = \mathbf{D}, \tag{2.77}$$

where $\mathbf{D}$ is a diagonal matrix whose diagonal elements are eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$ of $\mathbf{A}$. Further, the column vectors of $\mathbf{Q}$ are the unit eigenvectors of $\mathbf{A}$. This is a matrix diagonalization process.

For the symmetric matrix $\mathbf{A}$ in Eq. (4.51)

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \tag{2.78}$$

its eigenvalues and eigenvectors are

$$\lambda_1 = 3, \qquad \lambda_2 = -1, \tag{2.79}$$

$$\mathbf{v}_1 = \begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix}, \qquad \mathbf{v}_2 = \begin{bmatrix} -\sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix} \tag{2.80}$$

Thus, the orthogonal matrix $\mathbf{A}$ and the diagonal matrix $\mathbf{d}$ are as follows

$$\mathbf{Q} = \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & -\sqrt{2}/2 \end{bmatrix}, \qquad \mathbf{D} = \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.81}$$

With $\mathbf{Q}$, $\mathbf{A}$, and $\mathbf{D}$, you can easily verify Eq. (2.77) by hand calculation or by computer coding.

Equation (2.77) can be written in the following way

$$\mathbf{A}_{n\times n} = \mathbf{Q}_{n\times n}\mathbf{D}_{n\times n}\mathbf{Q}^t_{n\times n} \tag{2.82}$$

or

$$\mathbf{A}_{n\times n} = \sum_{k=1}^{n} \lambda_k \left(\mathbf{q}^{(k)}\right)_{n\times 1} \left((\mathbf{q}^{(k)})^t\right)_{1\times n} \tag{2.83}$$

This is a process of matrix decomposition by orthogonal matrices or by eigenvectors. Further, if all the eigenvalues are non-negative, then the matrix is said to be *positive semi-definite*, and the above formula can be written as

$$\mathbf{A}_{n\times n} = \sum_{k=1}^{n} \left(\mathbf{v}^{(k)}\right)_{n\times 1} \left((\mathbf{v}^{(k)})^t\right)_{1\times n} \tag{2.84}$$

where $\mathbf{v}^{(k)} = \sqrt{\lambda_k}\mathbf{q}^{(k)} (k = 1, 2, \cdots, n)$. The sample covariance matrix in climate science satisfies the positive eigenvalue assumption, and will be discussed in more details in the next chapter. The last equation means that a positive semi-definite symmetric matrix $\mathbf{A}_{n\times n}$ can be decomposed into a sum of n outer products of eigenvectors.

Matrix $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \tag{2.85}$$

is not positive semi-definite since its second eigenvalue is -1, but matrix $\mathbf{C}$

$$\mathbf{C} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{2.86}$$

is positive semi-definite since its eigenvalues are 3 and 1, which are positive. The matrix $\mathbf{C}$ is actually positive definite. The eigenvectors are the same those of $\mathbf{A}$. Thus,

$$\mathbf{C} = \mathbf{Q}\mathbf{D}_c\mathbf{Q}^t, \tag{2.87}$$

where $\mathbf{Q}$ is given by Eq. (2.81), and $\mathbf{D}_c$ is

$$\mathbf{D}_c = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}. \tag{2.88}$$

The above can be verified by the following computer code.

```
# Verify diagonalization and decomposition: R code
C = matrix(c(2,1,1,2), nrow = 2)
eigen(C)
```

```
#$values
#[1] 3 1
#$vectors
#   [,1]       [,2]
#[1,] 0.7071068 -0.7071068
#[2,] 0.7071068  0.7071068
Q = eigen(C)$vectors
D = t(Q)%*%C%*%Q #Matrix diagonalization
D
#[1,]    3    0
#[2,]    0    1
Q%*%D%*%t(Q) #Matrix decomposition
#[1,]    2    1
#[2,]    1    2
D[1,1]*Q[,1]%*%t(Q[,1]) + D[2,2]*Q[,2]%*%t(Q[,2])
#[1,]    2    1
#[2,]    1    2
```

```
# Verify diagonalization and decomposition: Python code
C = [[2,1],[1,2]]
valC, Q = np.linalg.eig(C)
print('eigenvalues of C =', valC)
#eigenvalues of C = [3. 1.]
print('eigenvectors of C =', Q)
#eigenvectors of C = [[ 0.70710678 -0.70710678]
# [ 0.70710678  0.70710678]]
D = Q.transpose(1,0).dot(C).dot(Q)
print('D =', D)
#D =  [[3. 0.]
# [0. 1.]]
# Matrix C is decomposed into three matrices: C = Q D Q'
Q.dot(D).dot(Q.transpose(1,0))
#array([[2., 1.],
#       [1., 2.]])

D[0][0]*np.outer(Q[:][0],Q.transpose()[:][0]) + \
D[1][1]*np.outer(Q[:][1],Q.transpose()[:][1])
#array([[ 1.,  2.],
#       [-2., -1.]]) #matrix C is recovered from vectors
```

2.5 Hadammard and Other Matrix Multiplications

In addition to the regular matrix products and sweeping by multiplication described in Chapter 1, there still other matrix products, such as Hadamard product of two matrices. This section presents a few more matrix operations.

2.5.1 Hadamard Product of Two Matrices of the Same Dimensions

Hadamard product of two matrices is the multiplication of the corresponding pair of elements of the two matrices of the same dimension:

$$A \circ B = [a_i j b_{ij}] \tag{2.89}$$

The following is a computer code for an example of Hadamard product.

```
#R Hadamard product of two matrices
#install.packages('matrixcalc')
library(matrixcalc)
A = matrix( c( 1, 2, 3, 4 ), nrow=2, byrow=TRUE )
B = matrix( c( 2, 4, 6, 8 ), nrow=2, byrow=TRUE )
hadamard(A, B)
#      [,1] [,2]
#[1,]    2    8
#[2,]   18   32
```

An application of the Hadamard product is to calculate the value of each item of a store, when the numbers of items for each good are stored as a matrix, and the corresponding unit prices as another. The Hadamard product of the two matrices yields a matrix of values for each kind of good.

2.5.2 Jordan Product of Two Matrices of the Same Dimensions

Jordan product of two square matrices of the same dimensions is defined as follows:

$$A \bullet B = \frac{1}{2}(AB + BA). \tag{2.90}$$

A numerical example is as follows.

```
#R Jordan product of A and B
A = matrix( c( 1, 2, 3, 4 ), nrow=2, byrow=TRUE )
B = matrix( c( 2, 1, 2, 1 ), nrow=2, byrow=TRUE )
(A%*%B + B%*%A)/2
#      [,1] [,2]
#[1,]  5.5  5.5
#[2,]  9.5  7.5
```

Since matrix multiplication is not commutative, $AB \neq BA$, Jordan product is the mean of AB and BA.

2.5.3 Commutator of Two Matrices of the Same Dimensions

The commutator of two matrices of the same dimensions is defined as follows:

$$[A, B] = AB - BA. \tag{2.91}$$

A numerical example is below.

```
#R commutator of A and B
#R commutator of A and B
A = matrix( c( 1, 2, 3, 4 ), nrow=2, byrow=TRUE )
B = matrix( c( 2, 1, 2, 1 ), nrow=2, byrow=TRUE )
A%*%B - B%*%A
#      [,1] [,2]
#[1,]    1   -5
#[2,]    9   -1
#install.packages('psych')
library(psych)
tr(A%*%B - B%*%A) #tr for trace
#[1] 0
```

As matrix multiplication is not commutative, $AB \neq BA$, the commutator measures the difference of AB minus BA. This operation has many applications in theoretical physics.

The trace of $[A, B]$ is zero. This can be easily proved as follows:

$$\mathrm{tr}([A, B]) = \mathrm{tr}(AB - BA) = \mathrm{tr}(AB) - \mathrm{tr}(BA) = \mathrm{tr}(AB) - \mathrm{tr}(AB) = 0. \quad (2.92)$$

Theorem 2.1 *Shoda theorem: If $tr(C)$ is zero, then C is a commutator matrix.*

The proof of this theorem can made through constructing matrices A and B such that

$$C = AB - BA. \quad (2.93)$$

Given C, it is similar to another trace zero matrix D, i.e., there exists an orthogonal matrix S such as

$$D = SCS^{-1} = [d_{ij}]_{n\times n} \quad (2.94)$$

Then, it turns out that

$$A = SXS^{-1}, \quad B = SYS^{-1}, \quad (2.95)$$

where

$$X = \mathrm{diag}(1, 2, \cdots, n), \quad Y = [y_{ij}]_{n\times n} \quad (2.96)$$

with

$$y_{ij} = \begin{cases} (i-j)^{-1} d_{ij} \\ 1 \qquad\qquad \text{else.} \end{cases} \quad (2.97)$$

In fact,

$$D = [X, Y]. \quad (2.98)$$

2.5.4 Outer Product of Two Vectors and Two Matrices

2.5.4.1 Outer Product of Two Vectors

The outer product of two vectors a_m and $b_{p\times 1}$ is defined as

$$a \otimes b = a_{m\times 1} t(b)_{1\times p} \quad (2.99)$$

and is an $m \times p$ matrix.

```
#Outer product of two vectors
a = 1:2
b = 1:4
a%o%b #outer product a_2-by-1 times t(b)_1-by-4
#     [,1] [,2] [,3] [,4]
#[1,]    1    2    3    4
#[2,]    2    4    6    8

#Outer product of A_mn and B_nq
A = matrix(1:4, ncol = 2)
B = matrix(1:6, ncol = 3)
A%o%B
dim(A%o%B)
#[1] 2 2 2 3

#Outer product of A_mn and B_pq
A = matrix(1:4, ncol = 2)
B = matrix(1:9, ncol = 3)
A%o%B
dim(A%o%B)
#[1] 2 2 2 3
```

This concept can be extended to the outer product of two matrices A and B, as shown in this numerical example.

2.5.4.2 Cross Product of Two Vectors in 3D

The cross product of two vectors $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$ in 3-dimensional space is also a 3-dimensional vector and defined as follows

$$xy = (x_2y_3 - x_3y_2, x_3y_1 - x_1y_3, x_1y_2 - x_2y_1) \tag{2.100}$$

A numerical example is below.

```
#R cross product and dot product
library(pracma)
x = 1:3
y = 4:6
cross(x, y)
#[1] -3  6 -3
dot(x, y)
#[1] 32
```

This example also shows a dot product of two vectors. The dot product can be extended to n-dimensional space, but the cross product is limited to the 3-dimensional space. The cross product is used often in physics and engineering, particularly in mechanics and electromagnetism. The cross product of the force vector with an arm vector is a torque vector, which points to the direction of a screw when you drive it into a wall.

The dot product in 3D space also has many physics and engineering applications, such as the force's dot product with displacement vector is work.

2.5.4.3 Anohter Matrix Product Based on Outer Product of Vectors

There is another type of product of two matrices using the outer product of column and row vectors. We regard $A_{mn} = [a[,i]], i = 1, 2, \cdots, n$ consisting of n column vectors, and $B_{n\times q} = [b[j,], j = 1, 2, \cdots, n$ consisting of n row vectors. The outer product is an mq matrix. defined as the

$$(AB)_{m\times q} = \left[\sum_{i=1}^{n} [a[,i]_{m\times 1} t(b[i,])_{1\times q}\right]_{m\times q}. \tag{2.101}$$

2.5.5 Kronecker Product of Two Matrices of the Same Dimensions

Kronecker product of two matrices A_{mn} and $B_{p\times q}$ is defined as the

$$A \diamond B = [a_{ij}B]_{mp\times nq}. \tag{2.102}$$

Two numerical examples are as follows:

```
#R Kronecker product of  two matrices
#Kronecker product
library(fastmatrix)
A <- diag(1:2)
B <- matrix(1:4, ncol = 2)
kronecker.prod(A, B)
#      [,1] [,2] [,3] [,4]
#[1,]    1    3    0    0
#[2,]    2    4    0    0
#[3,]    0    0    2    6
#[4,]    0    0    4    8

# an example with vectors
ones <- rep(1, 2)
y <- 1:4
kronecker.prod(ones, t(y)) # 2-by-4 matrix
#      [,1] [,2] [,3] [,4]
#[1,]    1    2    3    4
#[2,]    1    2    3    4
```

Theorem 2.2 $(A \diamond B)^{-1} = A^{-1} \diamond B^{-1}$. **Proof:**

$$(A)(A^{-1} \diamond B^{-1}) = (AA^{-1}) \diamond (BB^{-1}) = I_n \diamond I_p = I_{np}. \tag{2.103}$$

Here we have used the following equality

$$(A \diamond B)(C \diamond D) = (AC) \diamond (BD). \tag{2.104}$$

This can be proved through direct matrix product computing.

2.6 Direct Sum of Two Matrices

The direct sum of matrices A and B is to stack the two matrices block-diagonally as illustrated in the following numerical example. The mathematical notation of the direct sum operation is often $\oplus$:

$$A \oplus B \tag{2.105}$$

```
#R direct sum of two matrices
A = matrix(1:4, ncol = 2)
B = matrix(1:6, ncol = 3)
A1 = rbind(A, matrix(rep(0, 4), ncol = 2))
B1 = rbind(matrix(rep(0, 6), ncol = 3), B)
C = cbind(A1, B1) #= direct sum of A and B
C
#      [,1] [,2] [,3] [,4] [,5]
#[1,]    1    3    0    0    0
#[2,]    2    4    0    0    0
#[3,]    0    0    1    3    5
#[4,]    0    0    2    4    6

#Express the direct sum by Kronecker products
kronecker.prod(diag(1,0), A) + kronecker.prod(diag(0,1), B)
#      [,1] [,2] [,3] [,4] [,5]
#[1,]    1    3    0    0    0
#[2,]    2    4    0    0    0
#[3,]    0    0    1    3    5
#[4,]    0    0    2    4    6
```

Theorem 2.3 *From Kronecker product to direct sum:*

$$A \oplus B = diag(1,0) \diamond A + diag(0,1) \diamond B. \tag{2.106}$$

The proof follows the definition of Kronecker product and has been verified in the numerical example.

2.7 Visualization of Eigenvalues and Eigenvectors for a Covariance Matrix

If $A_{n\times p}$ is a space-time data matrix, then

$$C_{n\times n} = \frac{1}{p} AA^t \tag{2.107}$$

is a covariance matrix. Here, n is the number of spatial locations, e.g., n clients of your health clinic; and p is apparently the number of time steps, e.g., the number of days under your supervision or treatment.

We wish to ask the following questions:

(i) What are the typical differences between your clients?
(ii) What are the temporal variation patterns that show the beginning of the effectiveness of your treatment?

The eigenvalues and eigenvectors of C may help answer Question (i). The SVD in the next section may help answer Question (ii).

We use the following numerical example to illustrate the procedure of visualization and interpretation of eigenvalues and eigenvectors for C.

2.8 Singular Value Decomposition

Previous section shows an eigenvector-eigenvalue decomposition of a symmetric square matrix. A similar decomposition can be made for a rectangular matrix. The decomposition using unit eigenvectors and eigenvalues for a general rectangular matrix is called the *singular value decomposition* (SVD). Singular value is another name of eigenvalue. Although the basic mathematical theory of SVD was developed almost 200 years ago (Stewart 1993), the modern algorithm of efficient SVD computing was only developed by Gene H. Golub (1932-2007) and his colleagues in the 1970s. Now, SVD has become an important data analysis tool for every field. Climate science is not an exception. A space-time climate data matrix is often a rectangular matrix. since the number of sites is not likely to be equal to the number of temporal observations at those sites. We may denote a space-time climate data matrix by $\mathbf{X}_{n\times m}$, where n denotes the number of sites, and m is the total number of temporal observations. For $\mathbf{X}_{n\times m}$, we can also interpret n as the number of grid boxes of a climate model output, and m as the number of time steps.

2.8.1 SVD formula and a simple SVD example

If $m \leq n$, then matrix $\mathbf{A}_{n\times m}$ has the following SVD decomposition

$$\mathbf{A}_{n\times m} = \mathbf{U}_{n\times m}\mathbf{D}_{m\times m}(\mathbf{V}^t)_{m\times m}. \tag{2.108}$$

Here, $\mathbf{U}$ may be interpreted as a spatial matrix, consisting of m spatial orthonormal column vectors which are unit eigenvectors of $\mathbf{A}\mathbf{A}^t$; $\mathbf{V}$ may be interpreted as a temporal matrix, consisting of m temporal orthonormal column vectors which are unit eigenvectors of $\mathbf{A}^t\mathbf{A}$; and $\mathbf{D}$ is a diagonal matrix whose elements are the square root of the eigenvalues of $\mathbf{A}\mathbf{A}^t$ and may be interpreted as standard deviations. Figure 2.3 may help you understand this formula. The diagonal elements of $\mathbf{D}$ are called singular values, and the column vectors of $\mathbf{U}$ and $\mathbf{U}$ are called singular vectors. Here, the word "singular" may be understood as special, or opposite to "general", or distinguished, or being out of ordinary.

If $m \geq n$, then matrix $\mathbf{A}_{n\times m}$ has the following SVD decomposition

$$\mathbf{A}_{n\times m} = \mathbf{U}_{n\times n}\mathbf{D}_{n\times n}(\mathbf{V}^t)_{n\times m}. \tag{2.109}$$

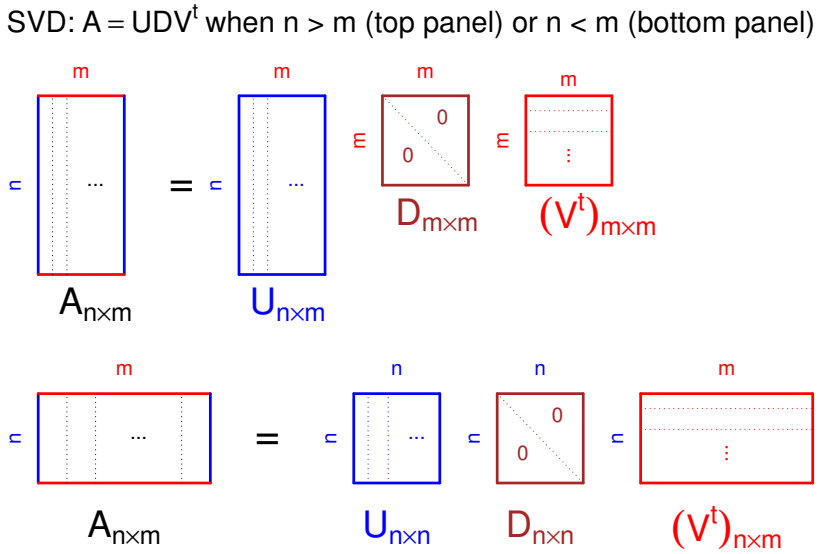


Fig. 2.3 Schematic diagrams of singular value decomposition (SVD): $A = UDV^t$.

The following R code shows a simple SVD example of a 2×3 matrix.

```
#SVD example for a 2-by-3 matrix: R code
A=matrix(c(-1,1,0,2,-2,3),nrow=2)
A #Show the 2-by-3 matrix
#      [,1] [,2] [,3]
#[1,]   -1    0   -2
#[2,]    1    2    3
svdA=svd(A) #Compute the SVD of A and put the results in svdA
svdA #Show SVD results: d, U, and V
round(svdA$d, digits=2) #Show only the singular values
#[1] 4.22 1.09
round(svdA$u, digits=2) #Show only matrix U
#      [,1] [,2]
#[1,] -0.48 0.88
#[2,]  0.88 0.48
round(svdA$v, digits=2)#Show only matrix V
#      [,1]  [,2]
#[1,] 0.32 -0.37
#[2,] 0.42  0.88
#[3,] 0.85 -0.29
sqrt(eigen(A%*%t(A))$values)
#[1] 4.221571 1.085514
```

```
#SVD example for a 2-by-3 matrix: Python code
A = [[-1,0, -2],[1,2,3]]
UsvdA, DsvdA, VsvdA = np.linalg.svd(A)
print('Singular values=', np.round(DsvdA,2))
#Singular values=  [4.22 1.09]
print('Spatial singular vectors=', np.round(UsvdA,2))
#Spatial singular vectors=  [[-0.48  0.88]
# [ 0.88  0.48]]
print('Temporal singular vectors=', np.round(VsvdA,2))
#Temporal singular vectors=  [[ 0.32  0.42  0.85]
# [-0.37  0.88 -0.29]
# [-0.87 -0.22  0.44]]

B = np.array(A)
C = np.matmul(B, B.T) #B times B transpose
valC, vecC = np.linalg.eig(C)
np.sqrt(valC)
#array([1.0855144 , 4.22157062])
```

The above computer code shows the following SVD results expressed in mathematical formulas below:

(a) The vector form:

$$
\begin{aligned}
A &= \begin{bmatrix} -1 & 0 & -2 \\ 1 & 2 & 3 \end{bmatrix} \\
&= 4.22 \begin{bmatrix} -0.48 \\ 0.88 \end{bmatrix} \times \begin{bmatrix} 0.32 & 0.42 & 0.85 \end{bmatrix} + \\
&\quad 1.09 \begin{bmatrix} 0.88 \\ -0.48 \end{bmatrix} \times \begin{bmatrix} -0.37 & 0.88 & -0.29 \end{bmatrix}
\end{aligned} \tag{2.110}
$$

and

(b) The matrix form:

$$
A = \begin{bmatrix} -1 & 0 & -2 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} -0.48 & 0.88 \\ 0.88 & 0.48 \end{bmatrix} \begin{bmatrix} 4.22 & 0 \\ 0 & 1.09 \end{bmatrix} \begin{bmatrix} 0.32 & 0.42 & 0.85 \\ -0.37 & 0.88 & -0.29 \end{bmatrix} \tag{2.111}
$$

If we use only the first singular vectors to approximate A from the two triplets of singular vectors and singular values, the result is below.

```
#Data reconstruction by singular vectors: R code
round(svdA$d[1]*svdA$u[,1]%*%t(svdA$v[,1]),
      digits=1)
#     [,1] [,2] [,3]
#[1,] -0.7 -0.8 -1.7
#[2,]  1.2  1.5  3.2
```

It is quite close to A.

```
#Data reconstruction by singular vectors: Python code
np.round(DsvdA[0]*np.outer(UsvdA[:][0], VsvdA[:][0]),1)
#array([[-0.7, -0.8, -1.7],
#       [ 1.2,  1.5,  3.2]])
```

If we use both singular vectors to reconstruct A, then the reconstruction is exact without errors as expected:

```
round(svdA$d[1]*svdA$u[,1]%*%t(svdA$v[,1]) +
  svdA$d[2]*svdA$u[,2]%*%t(svdA$v[,2]),
  digits =2)
#     [,1] [,2] [,3]
#[1,]   -1    0   -2
#[2,]    1    2    3
```

```
A1 = DsvdA[0]*np.outer(UsvdA[:][0], VsvdA[:][0])
A2 = DsvdA[1]*np.outer(UsvdA[:][1], VsvdA[:][1])
np.round(A1 + A2, 2)
#array([[-1.,  0., -2.],
#       [ 1.,  2.,  3.]])
```

Figure 2.3 for the schematic diagram of SVD may be plotted by the following computer code.

```
#R plot schematic diagram of SVD
setwd('/Users/sshen/LinAlg')
setEPS() #Plot the figure and save the file
postscript("fig0503.eps", width = 11)
par(mar=c(0,0,0,0))
plot(200, axes = FALSE,
     xlab = "", ylab = "",
     xlim = c(-3,28), ylim = c(-3,16))
text(13,15.5, cex=2.2,
     bquote("SVD:" ~ A==UDV^t~ "when n > m or n < m"))
#Space-time data matrix A when n>m
segments(x0 = c(0,0,3,3),
         y0 = c(6,12,12,6) +1,
         x1 = c(0,3,3,0),
         y1 = c(12,12,6,6) +1,
         col = c('blue','red','blue','red'),lwd =3)
segments(x0 = c(0.5,1.0),
         y0 = c(6,6)+1,
         x1 = c(0.5,1.0),
         y1 = c(12,12)+1,
         lwd =1.3, lty = 3)
text(-.8, 9+1, 'n', srt=90, col ='blue', cex = 1.4)
text(1.5, 12.8+1, 'm', col = 'red',  cex = 1.4)
text(2.0, 9+1, '...',  cex = 1.4)
text(2, 5+1, bquote(A[n%*%m]),  cex = 2.5)
text(5, 9+1, '=',  cex = 3)
#Spatial matrix U
```

```
segments(x0 = c(7,7,10,10),
         y0 = c(6,12,12,6)+1,
         x1 = c(7,10,10,7),
         y1 = c(12,12,6,6)+1,
         col = c('blue','blue','blue','blue'), lwd =3)
segments(x0 = c(7.5,8),
         y0 = c(6,6)+1,
         x1 = c(7.5,8),
         y1 = c(12,12)+1,
         lwd =1.3, lty = 3, col = 'blue')
text(6.2, 9+1, 'n', srt=90, col ='blue', cex = 1.4)
text(8.5, 12.8+1, 'm', col = 'red',  cex = 1.4)
text(9, 9+1, '...',  cex = 1.4, col='blue')
text(8.7, 5.0+1, bquote(U[n%*%m]),  cex = 2.5, col= 'blue')
#Singular value diagonal matrix D
segments(x0 = c(12,12,15,15),
         y0 = c(9,12,12,9)+1,
         x1 = c(12,15,15,12),
         y1 = c(12,12,9,9)+1,
         col = c('brown','brown','brown','brown'), lwd =3)
segments(x0 = 12, y0 = 12+1, x1 = 15, y1 = 9+1, lty=3,
         col = c('brown'), lwd =1.3)#diagonal line
text(11.2, 10.5+1, 'm', srt=90, col ='red', cex = 1.4)
text(13.5, 12.8+1, 'm', col = 'red',  cex = 1.4)
text(14.1, 11.3+1, '0', col = 'brown',  cex = 1.4)
text(12.9, 10.0+1, '0', col = 'brown',  cex = 1.4)
text(13.9, 8.0+1, bquote(D[m%*%m]),  cex = 2.5, col='brown')
#Temporal matrix V
segments(x0 = c(17,17,20,20),
         y0 = c(9,12,12,9)+1,
         x1 = c(17,20,20,17),
         y1 = c(12,12,9,9)+1,
         col = c('red','red','red','red'), lwd =3)
segments(x0 = c(17,17),
         y0 = c(11.5,10.8)+1,
         x1 = c(20,20),
         y1 = c(11.5,10.8)+1,
         col = c('red','red'), lty=3, lwd =1.3)
text(16.2, 10.5+1, 'm', srt=90, col ='red', cex = 1.4)
text(18.5, 12.5+1, 'm', col = 'red',  cex = 1.4)
text(19.5, 8+1, bquote((V^t)[m%*%m]),  cex = 2.5, col='red')
text(18.5, 10+1, '...',  col='red', srt=90, cex =1.4)
# Space-time data matrix B when n < m
segments(x0 = c(0,0,6,6),
         y0 = c(0,3,3,0),
         x1 = c(0,6,6,0),
         y1 = c(3,3,0,0),
         col = c('blue','red','blue','red'), lwd =3)
segments(x0 = c(1,2,5),
         y0 = c(0,0,0),
         x1 = c(1,2,5),
         y1 = c(3,3,3),
         lwd =1.3, lty = 3)
text(-0.8, 1.5, 'n', srt=90, col ='blue', cex = 1.4)
text(3, 3.8, 'm', col = 'red',  cex = 1.4)
text(3.5, 1.5, '...',  cex = 1.4)
```

```
text(3, -1.5, bquote(A[n%*%m]),  cex = 2.5)
text(8, 1.5, '=',  cex = 3)
#Spatial matrix U
segments(x0 = c(11,11,14,14),
         y0 = c(0,3,3,0),
         x1 = c(11,14,14,11),
         y1 = c(3,3,0,0),
         col = c('blue','blue','blue','blue'), lwd =3)
segments(x0 = c(11.5,12.2),
         y0 = c(0,0),
         x1 = c(11.5,12.2),
         y1 = c(3,3),
         lwd =1.3, lty = 3, col = 'blue')
text(10.2, 1.5, 'n', srt=90, col ='blue', cex = 1.4)
text(12.5, 3.8, 'n', col = 'blue',  cex = 1.4)
text(13.2, 1.5, '...',  cex = 1.4, col='blue')
text(12.5, -1.5, bquote(U[n%*%n]),  cex = 2.5, col= 'blue')
#Singular value diagonal matrix D
segments(x0 = c(16,16,19,19),
         y0 = c(0,3,3,0),
         x1 = c(16,19,19,16),
         y1 = c(3,3,0,0),
         col = c('brown','brown','brown','brown'), lwd =3)
segments(x0 = 16, y0 = 3, x1 = 19, y1 = 0, lty=3,
         col = c('brown'), lwd =1.3)#diagonal line
text(15.2, 1.5, 'n', srt=90, col ='blue', cex = 1.4)
text(17.5, 3.8, 'n', col = 'blue',  cex = 1.4)
text(18.1, 2.3, '0', col = 'brown',  cex = 1.4)
text(16.9, 1.0, '0', col = 'brown',  cex = 1.4)
text(17.5, -1.5, bquote(D[n%*%n]),  cex = 2.5, col='brown')
#Temporal matrix V
segments(x0 = c(21,21,27,27),
         y0 = c(0,3,3,0),
         x1 = c(21,27,27,21),
         y1 = c(3,3,0,0),
         col = c('red','red','red','red'),
         lwd =3)
segments(x0 = c(21,21),
         y0 = c(2.5,1.8),
         x1 = c(27,27),
         y1 = c(2.5,1.8),
         col = c('red','red'), lty=3, lwd =1.3)
text(20.2, 1.5, 'n', srt=90, col ='blue', cex = 1.4)
text(24, 3.8, 'm', col = 'red',  cex = 1.4)
text(24, -1.5, bquote((V^t)[n%*%m]),  cex = 2.5, col='red')
text(24, 1, '...',  col='red', srt=90, cex =1.4)
dev.off()
```

```
#Python plot Fig. 5.3: Schematic diagram of SVD
import matplotlib.patches as patches
import numpy as np
import pylab as pl
from matplotlib import collections  as mc
lines = [[(0, 7), (0, 13)], [(0, 13), (3, 13)],
         [(3, 13), (3, 7)], [(3, 7), (0,7)]]
c = np.array(['b', 'r', 'b', 'r'])
lc = mc.LineCollection(lines, colors=c, linewidths=3)
fig, ax = pl.subplots()
ax.set_xlim([-3, 28])
ax.set_ylim([-3, 16])
ax.add_collection(lc)
ax.margins(0.1)
plt.plot([0.5,0.5], [7, 13],
         linestyle='dotted', color = 'k')
plt.plot([1, 1], [7, 13],
        linestyle='dotted', color = 'k')
plt.text(13, 15.5,
         r'SVD:␣␣$A==UDV^t$␣when␣n␣>␣m␣or␣n␣<␣m',
        fontsize = 30)
plt.text(-1.2, 10, 'n', color = 'blue',
           fontsize =25, rotation=90)
plt.text(1.1, 13.3, 'm', color = 'red',
           fontsize =25, rotation=0)
plt.text(0.0, 5.5, r'$A_{n\times␣m}$',
           fontsize =35, rotation=0)
plt.text(1.5, 10, '...',
           fontsize =25, rotation=0)
plt.axis('off')
plt.show()
```

This Python code generates the top-left rectangular box in Fig. 5.3. The remaining code for other boxes is highly repetitive and can be found from the book website.

2.9 SVD for the standardized sea level pressure data of Tahiti and Darwin

The Southern Oscillation Index (SOI) is an indicator for El Niño or La Niña. It is computed as the difference of sea level pressure (SLP) of Tahiti (17.75°S, 149.42°W) minus that of Darwin (12.46°S, 130.84°E). An SVD analysis of the SLP data can substantiate this calculation formula.

The following shows the data matrix of the standardized SLP anomalies of Tahiti and Darwin from 2009 to 2015, and its SVD:

```
#R SVD analysis for the weighted SOI from SLP data
setwd("/Users/sshen/climmath")
```

```
Pda<-read.table("data/PSTANDdarwin.txt", header=F)
dim(Pda)
#[1] 65 13 #Monthly Darwin data from 1951-2015
pdaDec<-Pda[,13] #Darwin Dec standardized SLP anomalies data
Pta<-read.table("data/PSTANDtahiti.txt", header=F)
ptaDec=Pta[,13] #Tahiti Dec standardized SLP anomalies
ptada1 = cbind(pdaDec, ptaDec) #space-time data matrix

#Space-time data format
ptada = t(ptada1[59:65,]) #2009-2015 data
colnames(ptada)<-2009:2015
rownames(ptada)<-c("Darwin", "Tahiti")
ptada #6 year of data for two stations
#        2009 2010 2011 2012 2013 2014 2015
#Darwin   0.5 -2.3 -2.2  0.3  0.3  0.1 -0.4
#Tahiti -0.7  2.5  1.9 -0.7  0.4 -0.8 -1.3
svdptd = svd(ptada) #SVD for the 2-by-6 matrix
U=round(svdptd$u, digits=2)
U
#[1,] -0.66 0.75
#[2,]  0.75 0.66
D=round(diag(svdptd$d), digits=2)
D
#[1,]  4.7 0.00
#[2,]  0.0 1.42
V =round(svdptd$v, digits=2)
t(V)
#[1,] -0.18  0.72  0.61 -0.15 0.02 -0.14 -0.15
#[2,] -0.06 -0.06 -0.28 -0.17 0.34 -0.32 -0.82
```

```
#Python SVD analysis of the Darwin and Tahiti SLP data
import os
os.chdir("/Users/sshen/LinAlg")
PDA = np.array(read_table("data/PSTANDdarwin.txt", \
                          header = None, delimiter = "\s+"))
PTA = np.array(read_table("data/PSTANDtahiti.txt", \
                          header = None, delimiter = "\s+"))
pdata = np.stack([PDA[58:65,12], PTA[58:65,12]], axis=0)
print('The␣Darwin␣and␣Tahiti␣SLP␣data␣2009-2015␣=', pdata)
#The Darwin and Tahiti Standardized SLP anomalies =
#[[ 0.5 -2.3 -2.2  0.3  0.3  0.1 -0.4]
# [-0.7  2.5  1.9 -0.7  0.4 -0.8 -1.3]]
u, d, v = np.linalg.svd(pdata)
print('Spatial␣singular␣vectors␣EOFs␣U␣=', np.round(u,2))
#Spatial singular vectors EOFs U = [[-0.66  0.75]
# [ 0.75  0.66]]
print('Diagonal␣matrix␣D␣=', np.round(np.diag(d),2))
#Diagonal matrix D = [[4.7  0.  ]
# [0.   1.42]]
print('Temporal␣singular␣vectors␣PCs␣V=', np.round(v,2))
#Temporal singular vectors PCs V=
#[[-0.18  0.72  0.61 -0.15  0.02 -0.14 -0.15]
# [-0.06 -0.06 -0.28 -0.17  0.34 -0.32 -0.82] ...]
```

One can verify that

$$\mathbf{UDV}^t \tag{2.112}$$

approximately recovers the original data matrix.

The first column vector $(-0.66, 0.75)$ of the spatial pattern matrix $\mathbf{U}$ may be interpreted to be associated with the Southern Oscillation Index (SOI), which puts a negative weight -0.66 on Darwin, and a positive weight 0.75 on Tahiti. The weighted sum is approximately equal to the difference of Tahiti's SLP minus that of Darwin. which the definition of SOI. The index measures large scale ENSO dynamics of the tropical Pacific (Trenberth 2020). The magnitude of the vector $(-0.66, 0.75)$ is approximately one, because U is a unitary matrix. The corresponding singular vector has a distinctly large value 0.72 in December 2010, which was a strong La Niña month. In this month, the Darwin has a strong negative SLP anomaly, while Tahiti has a strong positive SLP anomaly. This situation enhanced the easterly trade winds in the tropical Pacific and caused abnormally high precipitation in Australia in the 2010-2011 La Niña period.

The second column vector $(0.75, 0.66)$ of the spatial pattern matrix $\mathbf{U}$ also has climate implications. The weighted sum with two positive equal weights measures the tropical Pacific dynamics of small scales (Trenberth 2020).

2.10 Chapter summary

A matrix can be regarded as a 2-dimensional $n \times m$ rectangular array of numbers or symbols, or as m column vectors, or as n row vectors. This may be interpreted as climate data at n locations with m time steps. Many mathematical properties of a matrix of climate data have climate interpretations. For example, SVD decomposes a space-time climate data matrix into an orthogonal spatial pattern matrix, an orthogonal temporal pattern matrix, and a diagonal "energy-level" matrix that measures the standard deviation of the temporal pattern:

$$\mathbf{A} = \mathbf{UDV}^t \tag{2.113}$$

The column vectors of the spatial matrix $\mathbf{U}$ are spatial singular vectors, also called EOFs, while those of the temporal matrix $\mathbf{V}$ are temporal singular vectors, also called PCs. The first a few EOFs often have climate dynamic interpretations, such as El Niño Southern Oscillation (ENSO). If EOF1 corresponds to El Niño and shows some typical ENSO properties, such as the opposite signs of SLP anomalies of Darwin and Tahiti, then PC1 shows a temporal pattern, e.g., the extreme values of PC1 indicating both the occurrence time and the strength of El Niño. The diagonal elements of matrix $\mathbf{D}$ are singular values, also known as eigenvalues.

An eigenvector $\mathbf{v}$ of a square matrix $\mathbf{C}$ is such a special vector that $\mathbf{C}$'s action on $\mathbf{v}$ does not change its orientation, i.e., $\mathbf{Cv}$ is parallel to $\mathbf{v}$. This statement

implies the existence of a scaler λ, called eigenvalue (also known as singular value or characteristic value), such that

$$\mathbf{C}\mathbf{v} = \lambda\mathbf{v}. \tag{2.114}$$

We have also discussed the matrix method of solving a system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.115}$$

linear independence of vectors, linear transform, and other basic matrix methods. These methods are useful for the chapters on covariance, EOFs, spectral analysis, regression analysis, and machine learning. You may focus on the computing methods and computer code of the relevant methods. The mathematical proofs of this chapter, although helpful for exploring new mathematical methods, are not necessarily needed to read the other chapters of this book. If you are interested in an in-depth mathematical exploration of matrix theory, you may wish read the books by Horn and Johnson (1985) and Strang (2016).

References and Further Readings

[1] Golub, G.H. and C. Reinsch, 1970: Singular value decomposition and least squares solutions. *Numerische mathematik* 14, 403-420.

This seminal paper established an important method, known as the Golub-Reinsch algorithm, to compute the eigenvalues of a covariance matrix from a space-time matrix A without actually first computing the covariance matrix AA^t. This algorithm makes the SVD computation very efficient, which helps scientists consider SVD as a genuine linear algebra method, not a traditionally regarded statistical method based on a covariance matrix.

[2] Horn, R. A., and C. R. Johnson, 1985: *Matrix Analysis*, Cambridge University Press, 561pp.

This is a comprehensive book on matrix theory and is a good reference for a researcher in climate statistics. It assumes the knowledge of the first course of linear algebra.

[3] Strang, G., 2016: *Introduction to Linear Algebra.* 5th edition, Wellesley-Cambridge Press, Wellesley, MA 02482, 574pp.

Gilbert Strang (1934-) is an American mathematician and educator. His textbooks and pedagogy have been internationally influential. This text is one of the very few basic linear algebra books that includes excellent materials on SVD, probability, and statistics.

[4] Stewart, G.W., 1993: On the early history of the singular value decomposition. *SIAM Review* 35, 551-566.

This paper describes the contributions from five mathematicians in the period of 1814-1955 to the development of the basic SVD theory.

[5] Trenberth, K., and National Center for Atmospheric Research Staff (Eds), 2020: The Climate Data Guide: Southern Oscillation Indices: Signal, Noise and Tahiti/Darwin SLP (SOI). Retrieved from

https://climatedataguide.ucar.edu/climate-data/southern-oscillation-indices-signal-noise-and-tahitidarwin-slp-soi

This site describes the optimal indices for large and small scale dynamics.

[6] Tucker, A., 1993: The growing importance of linear algebra in undergraduate mathematics. *College Mathematics Journal* 24, 3-9.

This paper describes the historical development of linear algebra, such as the term "matrix" being coined by J.J. Sylvester in 1848, and pointed out that "tools of linear algebra find use in almost all academic fields and throughout modern society." The use of linear algebra in the big data era is now even more popular.

Exercises

2.1 Write a computer code to
(a) Read the NOAAGlobalTemp data file, and
(b) Generate a 4×8 space-time data matrix for the December mean surface air temperature anomaly data of four grid boxes and eight years.
Hint: You may find the NOAA Global Surface Temperature (NOAAGlobalTemp) dataset online. You can use either netCDF format or CSV format.

2.2 Write a computer code to find the inverse of the following matrix

```
#      [,1] [,2] [,3]
#[1,]  1.7 -0.7  1.3
#[2,] -1.6 -1.4  0.4
#[3,] -1.5 -0.3  0.6
```

2.3 Write a computer code to solve the following linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.116}$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \tag{2.117}$$

2.4 The following equation

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.118}$$

has infinitely many solutions, and cannot be directly solved by a simple computer command, such as `solve(A, b)`.

(a) Show that the three row vectors of the coefficient matrix are not linearly independent.

(b) Because of the dependence, the linear system has only two independent equations. Thus, reduce the linear system into two equations by treating x_3 as an arbitrary value while treating x_1 and x_2 as variables.

(c) Solve the two equations for x_1 and x_2 and express them in terms of x_3. The infinite possibilities of x_3 imply infinitely many solutions of the original system.

2.5 Ethane is a gas similar to the greenhouse gas methane and can burn with oxygen to form carbon dioxide and water:

$$C_2H_6 + O_2 \longrightarrow CO_2 + H_2O \tag{2.119}$$

Given two ethane molecules, how many molecules of oxygen, carbon dioxide and water should be in order for this chemical reaction equation to be balanced?

Hint: Assume x, y and z molecules of oxygen, carbon dioxide and water, and use the balance of the number of atoms of carbon, hydrogen and oxygen to form linear equations. Solve the system of linear equations.

2.6 Carry out the same procedure as the previous problem but for the burning of methane CH_4.

2.7 (a) Use matrix multiplication to show that the vector

$$\mathbf{u} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \tag{2.120}$$

is not an eigenvector of the following matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 4 \\ -2 & -7 \end{bmatrix} \tag{2.121}$$

noindent (b) Find all the unit eigenvectors of matrix $\mathbf{A}$ in Part (a).

2.8 Use hand-calculation to compute the matrix multiplication of $\mathbf{UDV}^t$ where the data of relevant matrices are given by the following R output:

```
A=matrix(c(1,-1,1,1),nrow=2)
A
#[1,]    1    1
#[2,]   -1    1
svd(A)
#$d
#[1] 1.414214 1.414214
#$u
#[1,] -0.7071068 0.7071068
#[2,]  0.7071068 0.7071068
#$v
#[1,]   -1    0
#[2,]    0    1
```

2.9 Use computer to find the matrices $\mathbf{U}, \mathbf{D}$ and $\mathbf{V}$ of the SVD for the following data matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix} \tag{2.122}$$

2.10 Use the first singular vectors to approximately reconstruct the data matrix $\mathbf{A}$ using

$$B = d_1 \mathbf{u}_1 \mathbf{v}_1^t. \tag{2.123}$$

Describe the goodness of the approximation using text, limited to 20 - 100 words.

2.11 For the following data matrix

$$\mathbf{A} = \begin{bmatrix} 1.2 & -0.5 & 0.9 & -0.6 \\ 1.0 & -0.7 & -0.4 & 0.9 \\ -0.2 & 1.1 & 1.6 & -0.4 \end{bmatrix}, \tag{2.124}$$

(a) Use computer to find the eigenvectors and eigenvalues of matrix $\mathbf{A}\mathbf{A}^t$.
(b) Use computer to find the eigenvectors and eigenvalues of matrix $\mathbf{A}^t\mathbf{A}$.
(c) Use computer to calculate SVD of $\mathbf{A}$.
(d) Compare the singular vectors and singular values in Step (c) with the eigenvalues and eigenvectors computed in Steps (a) and (b). Use text to discuss your comparison.
(e) Use computer to verify that the column vectors of $\mathbf{U}$ and $\mathbf{V}$ in the SVD of Step (c) are orthonormal to each other.

2.12 Make an SVD analysis of the December standardized anomalies data of sea level pressure (SLP) at Darwin and Tahiti from 1961 to 2010. You can find the data from the Internet or from the website of this book.
(a) Write a computer code to organize the data into a 2×50 space-time data matrix.
(b) Make the SVD calculation for this space-time matrix.
(c) Plot the first singular vector in $\mathbf{V}$ against time 1961 to 2010 as a time series curve, which is called the first principal component, denoted by PC1.
(d) Interpret the first singular vector in $\mathbf{U}$, which is called the first empirical orthogonal function (EOF1), as weights of Darwin and Tahiti stations.
(e) Check the historical El Niño events between 1961 and 2010 from the Internet, and interpret the extreme values of PC1.

2.13 Plot PC2 against time from the SVD analysis in the previous problem. Discuss the singular values λ_1 and λ_2 and interpret PC2, in reference to the description entitled "The Climate Data Guide: Southern Oscillation Indices: Signal, Noise and Tahiti/Darwin SLP (SOI)" by Kevin Trenberth (2020).

2.14 Make an SVD analysis similar to the previous two problems for the January standardized SLP anomalies data at Darwin and Tahiti from 1961 to 2010.

2.15 Make an SVD analysis similar to the previous problem for the monthly standardized SLP anomalies data at Darwin and Tahiti from 1961 to 2010. This problem includes anomalies for every month. The space-time data is a 2×600 matrix.

2.16 For the observed data of the monthly surface air temperature at five stations of your interest from January 1961 to December 2010, form a space-time data matrix, compute the SVD of this matrix, and interpret your results from the perspective of climate science. Please plot PC1, PC2, and PC3 against time. You may find your data from the Internet, such as the NOAA Climate Data Online website `https://www.ncdc.noaa.gov/cdo-web`

2.17 Do the same analysis as the previous problem, but for the monthly precipitation data at the same stations in the same time period.

2.18 For an Reanalysis dataset, make an SVD analysis similar to the previous problem for the monthly surface temperature data over 10 grid boxes of your choice in the time period of 1961-2010. The space-time data is a 10×600 matrix. You can use your preferred Reanalysis dataset and download the data from the Internet, such as NCEP/NCAR Reanalysis, and ECMWF ERA.

2.19 Let $\mathbf{C} = \mathbf{A}\mathbf{A}^t$ and $\mathbf{A}$ is any real-valued rectangular matrix. Show that

(a) the eigenvalues of $\mathbf{C}$ are non-negative;

(b) if $\mathbf{v} = \mathbf{A}^t\mathbf{u}$, then $\mathbf{v}$ is a eigenvector of $\mathbf{C}^t = \mathbf{A}^t\mathbf{A}$.

2.20 Given that

$$\mathbf{A} = \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \tag{2.125}$$

write down the second order polynomial corresponding to the following matrix expression

$$P(x_1, x_2) = \mathbf{x}^t\mathbf{A}\mathbf{A}^t\mathbf{x}. \tag{2.126}$$

This is known as the quadratic form of the matrix $\mathbf{A}\mathbf{A}^t$.

2.21 If $\mathbf{x}$ is a unit vector, use calculus to find the maximum value of $P(x_1, x_2)$ in the previous problem. How is your solution related to the eigenvalue of $\mathbf{A}\mathbf{A}^t$.

3 Matrix Applications to Machine Learning

Machine learning (ML) is a branch of science that uses data and algorithms to mimic how human beings learn. The accuracy of the ML results can be gradually improved based on new training data and algorithm update. For example, a baby learns how to pick an orange from a fruit plate containing apples, bananas and oranges. Another baby learns how to sort out different kinds of fruits from a basket into three categories without naming the fruits. Then, how does ML work? It is basically a decision process for clustering, classification, or prediction, based on the input data, decision criteria, and algorithms. It does not stop here. It further validates the decision results and quantifies errors. The errors and the updated data will help update the algorithms and improve the results.

ML has recently become a very popular method in climate science due to the availability of powerful and convenient resources of computing. It has been used to predict weather and climate, and to develop climate models. This chapter is a brief introduction of ML and provides basic ideas and examples. Our materials will help readers understand and improve the more complex ML algorithms used in climate science, so that they can go a step beyond only applying the ML software packages as a black box. We also provide R and Python codes for some basic ML algorithms, such as K-means for clustering, support vector machine for the maximum separation of sets, random forest of decision trees for classification and regression, and neural network training and predictions.

Artificial intelligence (AI) allows computers to automatically learn from past data without human programming, which enables a machine to learn and to have intelligence. Machine learning is a subset of AI. Our chapter here focuses on ML, not the general AI.

3.1 K-means clustering

A few toddlers at a daycare center may learn how to grab a few candies near themselves. It can be a point of fighting for a candy at a location that is not obviously closer to one than another. K-means method can help divide the candies among the toddlers in a fair way.

Based on the historical weather data over a country, can ML decide the climate regimes for the country? Can ML determine the ecoregions of a country? Can ML

define the regimes of wild fire over a region? The K-means clustering method can be useful to answering these questions.

3.1.1 K-means setup and trivial examples

The aim of the K-means clustering is to divide N points into K clusters so that the total within cluster sum of squares (tWCSS) is minimized. Here we use 2D data points to describe tWCSS and the K-means algorithm, although the K-means method can be formulated in higher dimensions. We regard the data $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N)$ as the 2D coordinates of N points. For example, we treat $\mathbf{x}_1 = (1.2, -7.3)$ as observational data. Assume that these points can be divided into K clusters $(C_1, C_2, \cdots, C_K)$, where K is subjectively given by you, the user who wishes to divide the N points into K clusters by the K-means method. Let $\mathbf{x} \in C_i$, i.e., points within cluster C_i. The number of points in cluster C_i is unknown and is to be determined by the K-means algorithm. The total WCSS is defined by the following formula

$$\text{tWCSS} = \sum_{i=1}^{K} \left(\sum_{\mathbf{x} \in C_i} \left| \mathbf{x} - \boldsymbol{\mu}_i \right|^2 \right) \tag{3.1}$$

where

$$\boldsymbol{\mu}_i = \frac{1}{K_i} \sum_{\mathbf{x} \in C_i} \mathbf{x} \tag{3.2}$$

is the mean of the points within the cluster C_i, as K_i is the number of points in C_i. Thus, we have K means, the name of the K-means method. That in the parentheses is the within cluster sum of squares (WCSS)

$$\text{WCSS}_i = \left(\sum_{\mathbf{x} \in C_i} \left| \mathbf{x} - \boldsymbol{\mu}_i \right|^2 \right). \tag{3.3}$$

This is defined for each cluster, and tWCSS is the sum of these WCSS_i for all the K clusters.

Some publications use WCSS or Tot WCSS, instead of tWCSS, to express the right hand side of Eq. (3.1). Be careful when reading literatures concerning the WCSS definition. Some computer software for K-means may even use a different tWCSS definition.

Our K-means computing algorithm is to minimize the tWCSS by optimally organizing the N points into K clusters. This algorithm can assign each data point to a cluster. If we regard μ_i as the centroid or center of cluster C_i, we may say that the points in cluster C_i may have some kind of similarity, such as similar climate or similar ecological characteristics, based on certain criteria. Below we use two simple cases ($N = 2$ and $N = 3$) to illustrate the K-means method and its solutions.

(i) Case $N = 2$:

When $N = 2$, if we assume $K = 2$, then

$$\mu_1 = \mathbf{x}_1, \quad \mu_2 = \mathbf{x}_2 \tag{3.4}$$

and

$$\text{tWCSS} = \sum_{i=1}^{2} \sum_{\mathbf{x} \in C_i} \left|\mathbf{x} - \mu_i\right|^2 = 0. \tag{3.5}$$

This is the global minimum for $K = 2$, and

$$\mathbf{x}_1 \in C_1, \quad \mathbf{x}_2 \in C_2, \quad \mu_1 = \mathbf{x}_1, \quad \mu_2 = \mathbf{x}_2 \tag{3.6}$$

is the unique solution. Of course, this is a trivial solution and bears no meaning.

When $N = 2$, if we assume $K = 1$, then

$$\mu_1 = \frac{\mathbf{x}_1 + \mathbf{x}_2}{2} \tag{3.7}$$

and

$$\text{tWCSS} = \sum_{i=1}^{1} \sum_{\mathbf{x} \in C_i} \left|\mathbf{x} - \mu_i\right|^2 = \frac{1}{4}\left|\mathbf{x}_1 - \mathbf{x}_2\right|^2. \tag{3.8}$$

This is the unique minimum of tWCSS, and the solution is also trivial.

(ii) Case $N = 3$:

When $N = 3$, if we assume $K = 2$, then there are three different cases of clustering

$$C_1 = \{\mathbf{x}_1, \mathbf{x}_2\},\ C_1 = \{\mathbf{x}_1, \mathbf{x}_3\},\ C_1 = \{\mathbf{x}_2, \mathbf{x}_3\}. \tag{3.9}$$

Here, the case

$$C_1 = \{\mathbf{x}_1, \mathbf{x}_2\},\ C_2 = \{\mathbf{x}_3\} \tag{3.10}$$

and

$$C_2 = \{\mathbf{x}_1, \mathbf{x}_2\},\ C_1 = \{\mathbf{x}_3\} \tag{3.11}$$

are considered the same, since their difference is only a matter which cluster is called C_1 or C_2.

For case $C_1 = \{\mathbf{x}_1, \mathbf{x}_2\}, C_2 = \{\mathbf{x}_3\}$, we have

$$\mu_1 = \frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2), \quad \mu_2 = \mathbf{x}_3, \tag{3.12}$$

and

$$\text{tWCSS} = \frac{1}{4}\left|\mathbf{x}_1 - \mathbf{x}_2\right|^2. \tag{3.13}$$

The tWCSS for the other two cases of clustering ($C_1 = \{\mathbf{x}_1, \mathbf{x}_3\},\ C_1 = \{\mathbf{x}_2, \mathbf{x}_3\}$) can be computed in a similar way. Then, the smallest tWCSS from the three determines the final solution, which is one of the three cases of clustering. A numerical example is given below.

Example 3.1 Find the two K-means clusters for the following three points

$$P_1(1,1), \quad P_2(2,1), \quad P_3(3,3.5). \tag{3.14}$$

This is a K-means problem with $N = 3$ and $K = 2$. The K-means clusters can be found by a computer command, e.g., `kmeans(mydata, 2)` in R. Figure 3.1 shows the two K-means clusters with their centers at

$$C_1(1.5,1), \quad C_2(3,3.5) \tag{3.15}$$

and

$$\text{tWCSS} = 0.5. \tag{3.16}$$

Points P_1 and P_2, indicated by red dots, are assigned to cluster C_1 whose center is marked by a red cross. Point P_3, indicated by the sky blue dot, is assigned to cluster C_2, whose center is marked with a blue cross and obviously overlaps with P_3. This result agrees with our intuition since P_1 and P_2 are together. The three points can have two other possibilities of combination $C_1 = \{P_1, P_3\}$ and $C_1 = \{P_2, P_3\}$. The case $C_1 = \{P_1, P_3\}$ leads to tWCSS $= 5.125$, and $C_1 = \{P_2, P_3\}$ to tWCSS $= 3.625$. Both tWCSS are greater than 0.5. Thus, these two combinations should not be a solution of the K-means clustering. These numerical results may be produced by the following computer codes.

```
#tWCSS calculation for N = 3 and K = 2
mydata <- matrix(c(1, 1, 2, 1, 3, 3.5),
                 nrow = N, byrow = TRUE)
x1 = mydata[1, ]
x2 = mydata[2, ]
x3 = mydata[3, ]

#Case C1 = (P1, P2)
c1 = (mydata[1, ] + mydata[2, ])/2
c2 = mydata[3, ]
tWCSS = norm(x1 - c1, type = '2')^2 +
  norm(x2 - c1, type = '2')^2 +
  norm(x3 - c2, type = '2')^2
tWCSS
#[1] 0.5

#Case C1 = (P1, P3)
c1 = (mydata[1, ] + mydata[3, ])/2
c2 = mydata[2, ]
norm(x1 - c1, type = '2')^2 +
  norm(x3 - c1, type = '2')^2 +
  norm(x2 - c2, type = '2')^2
#[1] 5.125

#Case C1 = (P2, P3)
c1 = (mydata[2, ] + mydata[3, ])/2
c2 = mydata[1, ]
norm(x2 - c1, type = '2')^2 +
  norm(x3 - c1, type = '2')^2 +
```

```
2933   norm(x1 - c2, type = '2')^2
2934 #[1] 3.625
2935
2936 #The case C1 = (P1, P2) can be quickly found by
2937 kmeans(mydata, 2)
2938 #Clustering vector:
2939 #[1] 1 1 2 #points P1, P2 in C1
```

```
#Python code: K-means computing for N = 3 and K = 2
N = 3
# create a 3 x 2 matrix of data
mydata = np.array([1,1,2,1,3,3.5]).reshape(3,2)

# first row of data
x1 = mydata[0,:]
# second row of data
x2 = mydata[1,:]
# third row of data
x3 = mydata[2,:]

# case C1 = (P1, P2)
c1 = (x1 + x2)/2
c2 = mydata[2,:]
tWCSS = np.linalg.norm(x1 - c1)**2 +
np.linalg.norm(x2 - c1)**2 +\
        np.linalg.norm(x3 - c2)**2
print(tWCSS)
#0.5

# case C1 = (P1, P3)
c1 = (x1 + x3)/2
c2 = mydata[1,:]
print(np.linalg.norm(x1 - c1)**2 +
np.linalg.norm(x3 - c1)**2 + \
      np.linalg.norm(x2 - c2)**2)

# case C1 = (P2, P3)
c1 = (x2 + x3)/2
c2 = mydata[0,:]
print(np.linalg.norm(x2 - c1)**2 +
np.linalg.norm(x3 - c1)**2 + \
      np.linalg.norm(x1 - c2)**2)
#5.125

# case C1 = (P1, P2) can be quickly found by
kmeans = KMeans(n_clusters=2).fit(mydata)

print(kmeans.labels_)
#[0 0 1] #points P1, P2 in C1
#because Python index begins with 0 while R starts from 1

# number of data points
N = 3
# assume K clusters
K = 2
# create a 3 x 2 matrix of data
mydata =  np.array([1, 1, 2, 1, 3, 3.5]).reshape(3,2)

Kclusters = KMeans(n_clusters=K).fit(mydata)
# cluster means
print(Kclusters.cluster_centers_, "\n")
#[[1.5 1. ]
#[3.  3.5]]
# sum of squares by cluster
print(Kclusters.inertia_)
#0.5
```

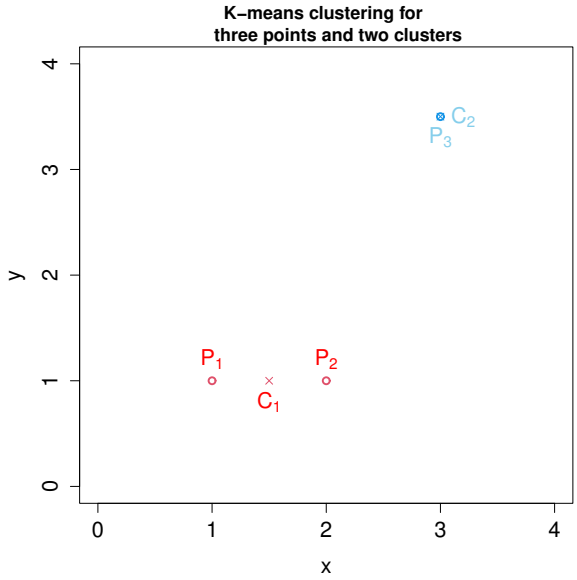


Fig. 3.1 The K-means clustering results for three points ($N = 3$) and two clusters ($K = 2$).

Of course, this problem is trivial and can even be solved analytically by hand. The computer solution of the K-means clustering is necessary when more points are involved and when it is not obvious for a point to belong to a specific cluster.

Figure 3.1 may be generated by the following computer code.

```
# R plot Fig. 9.1: K-means for N = 3 and K = 2
setwd("~/climstats")
N = 3 #The number of data points
K = 2 #Assume K clusters
mydata = matrix(c(1, 1, 2, 1, 3, 3.5),
                  nrow = N, byrow = TRUE)
Kclusters = kmeans(mydata, K)
Kclusters #gives the K-means results,
#e.g., cluster centers and WCSS
#Cluster means:
#[,1] [,2]
#1  1.5  1.0
#2  3.0  3.5
#Within cluster sum of squares by cluster:
#  [1] 0.5 0.0
Kclusters$centers
par(mar = c(4,4,2.5,0.5))
plot(mydata[,1], mydata[,2], lwd = 2,
     xlim =c(0, 4), ylim = c(0, 4),
     xlab = 'x', ylab = 'y', col = c(2, 2, 4),
     main = 'K-means␣clustering␣for
␣␣␣␣␣three␣points␣and␣two␣clusters',
```

```
     cex.lab = 1.4, cex.axis = 1.4)
points(Kclusters$centers[,1], Kclusters$centers[,2],
       col = c(2, 4), pch = 4)
text(1.5, 0.8, bquote(C[1]), col = 'red', cex = 1.4)
text(3.2, 3.5, bquote(C[2]), col = 'skyblue', cex = 1.4)
text(1, 1.2, bquote(P[1]), cex = 1.4, col = 'red')
text(2, 1.2, bquote(P[2]), cex = 1.4, col = 'red')
text(3, 3.3, bquote(P[3]), cex = 1.4, col = 'skyblue')
```

```
#Python plot Fig. 9.1: SVM clustering for N=3 and K=2
#starting from setting up the plot
plt.figure(figsize = (8,8))
plt.ylim([0,4])
plt.yticks([0,1,2,3,4])
plt.xlim([0,4])
plt.xticks([0,1,2,3,4])
plt.title("K-means␣clustering␣for␣\n␣\
␣␣three␣points␣and␣two␣clusters", pad = 10)
plt.xlabel("x", labelpad = 10)
plt.ylabel("y", labelpad = 10)

# plot P1-P3
plt.scatter(mydata[:,0], mydata[:,1],
            color = ('r', 'r', 'dodgerblue'),
            facecolors='none', s = 80)
# plot C1 and C2
plt.scatter((Kclusters.cluster_centers_[0][0],
                Kclusters.cluster_centers_[1][0]),
               (Kclusters.cluster_centers_[0][1],
               Kclusters.cluster_centers_[1][1]),
              marker = 'X', color = ('dodgerblue', 'r'))

# add labels
plt.text(1.43, 0.8, "$C_1$", color = 'r', size = 14)
plt.text(3.1, 3.45, "$C_2$", color = 'dodgerblue', size =14)
plt.text(0.95,1.1, "$P_1$", color = 'r', size = 14)
plt.text(1.95,1.1, "$P_2$", color = 'r', size = 14)
plt.text(2.95,3.3, "$P_3$", color = 'dodgerblue', size = 14)

plt.show()
```

3.1.2 The number of exhaustive K clusters from N points

The search for the minimum tWCSS in the previous sub-section for three points is through an algorithm of exhaustive search. This brute-force algorithm enumerates all possible cases. This algorithm can only work for a small N. When N is large, the number of cases will become too large to be calculated by a computer. For example, if $N = 100$ and $K = 10$, then the number of different clusters is approximately equal to

$$\mathrm{C}(99, 9) = 1.7310 \times 10^{12}. \tag{3.17}$$

It is impossible for a computer to exhaust all these cases to compute tWCSS for each case. The computer code for the combination is

```
choose(99, 9) #R code to choose 9 out of 99
#[1] 1.731031e+12 for N = 100, K =10
```

The number of cluster cases can be computed following the so-called stars-and-bars theorem, also known as the sticks-and-stones theorem, in counting. The stars-and-bars theorem states that when using $K-1$ bars or sticks to separate a line of N stars, what is the number of possible cases of separation? Or put it in another way. N number of candies is divided among K children. How many ways to divide the candies, assuming that each child will receive at least one candy? The answer is

$$\mathrm{C}(N-1, K-1), \tag{3.18}$$

because you can line up all the N stars and $K-1$ bars among them to form K subsets of stars. You first put one star in each subset. After this, you have $N-K$ stars to divide among the K subsets, i.e., K clusters. The $N-K$ stars plus $K-1$ bars form $(N-K)+(K-1)=N-1$ objects that include both stars and bars. Then the problem becomes choosing $K-1$ objects from $N-1$ objects, which has $\mathrm{C}(N-1, K-1)$ cases.

3.1.3 A K-means algorithm

The previous sub-section has demonstrated that the brute-force exhaustive search is not an algorithm choice to find the best K clusters from N points. We introduce the K-means algorithm in this sub-section.

From the above sub-section, we may have already developed a feeling that the K-means clustering method may roughly be divided into the following two steps:

- Assignment Step: Assume a number K and randomly generate K points $\boldsymbol{\mu}_i^{(1)}$ $(i = 1, 2, \cdots, K)$ as the initial K centroids for K clusters. Without replacement, for each point $\mathbf{x}$, compute the distance

$$d_i = \left|\mathbf{x} - \boldsymbol{\mu}_i^{(1)}\right| \quad i = 1, 2, \cdots, K. \tag{3.19}$$

  Assign the point $\mathbf{x}$ to the cluster with the smallest distance. Then, assign the next point to a cluster until every point is assigned. Thus, the initial K clusters are formed: $C_i^{(1)}$ and each $C_i^{(1)}$ containing $K_i^{(1)}$ data points $(i = 1, 2, \cdots, K)$.
- Update Step: Compute the centroids of the initial K clusters

$$\boldsymbol{\mu}_i^{(2)} = \frac{1}{K_i^{(1)}} \sum_{\mathbf{x} \in C_i^{(1)}} \mathbf{x} \tag{3.20}$$

and compute $\text{tWCSS}^{(2)}$

$$\text{tWCSS}^{(2)} = \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i^{(2)}} \left| \mathbf{x} - \boldsymbol{\mu}_i^{(2)} \right|^2 \tag{3.21}$$

These $\boldsymbol{\mu}_i^{(2)}$ $(i = 1, 2, \cdots, K)$ are regarded as the updated centroids. We then repeat the Assignment Step to assign each point to a cluster again according to the rule of the nearest distance. Next, we compute $\boldsymbol{\mu}_i^{(3)}$ $(i = 1, 2, \cdots, K)$, and $\text{tWCSS}^{(3)}$

The K-means algorithm continues the above iterative steps of Assignment and Update until the centroids $\boldsymbol{\mu}_i^{(n)}$ $(i = 1, 2, \cdots, K)$ do not change from those in the previous assignments $\boldsymbol{\mu}_i^{(n-1)}$ $(i = 1, 2, \cdots, K)$. When the assignment centroids do not change, we say that the K-means iterative sequence has converged. It can be rigorously proved that this algorithm always converges.

R and Python have K-means functions. Many software packages are freely available for some modified K-means algorithms using different distance formulas. Climate scientists often just use these functions or packages without actually writing an original computer code for a specific K-means algorithm. For example, the R calculation in the previous sub-section used the command `kmeans(mydata, K)` to calculate the cluster results for three points shown in Fig. 3.1.

The K-means algorithm appears very simple, but it was invented only in the 1950s. Although the above K-means iterative sequence always converges, the final result may depend on the initial assignment of the centroid: $C_i^{(1)}$ $(i = 1, 2, \cdots, K)$, hence each run of the R K-means command `kmeans(mydata, K)` may not always lead to the same solution. Further, a K-means clustering result may not be a global minimum for tWCSS.

Thus, after we have obtained our K-means results, we should try to explain the results and see if they are reasonable. We may also run the code with different K values and check which value K is the most reasonable. The tWCSS score is a function of K We may plot the function $\text{tWCSS}(K)$ and check the shape of the curve. Usually, $\text{tWCSS}(K)$ is a decreasing function, and further,

$$\Delta_K = \text{tWCSS}(K) - \text{tWCSS}(K+1) > 0, \quad K = 1, 2, \cdots \tag{3.22}$$

decreases fast for $K \leq K_e$, and then the decrease suddenly slows down or or even increases at $K = K_e$. In this sense, the curve $\text{tWCSS}(K)$ looks like having an elbow at K_e. We would choose this K_e as the optimal K. See the elbow in the example of the next sub-section.

Different applications may need different optimization criteria for the selection of the best number of clusters. When you work on your own data, you may try different criteria to choose the best number of clusters. For example, the Silhouette method uses the Silhouette score to measure the similarity level of a point with its own cluster compared to other clusters.

A computer code for the K-means clustering may not always obtain the minimum

tWCSS against all partitions. Sometimes, a computer yields a "local" minimum for tWCSS(K).

In short, when using the K-means method to divide given points into clusters, you should run the K-means code multiple times, check the relevant solution results. Although most K-means code yields a unique solution, multiple solutions sometimes appear, then we need to examine which solution is the most reasonable for our purposes.

3.1.4 K-means clustering for the daily Miami weather data

In this sub-section, we present an example using the K-means method to cluster the observed daily weather data at the Miami International Airport, the United States.

3.1.4.1 K-means clustering for $N = 365, K = 2$

Figure 3.2(a) shows a scatter plot of the daily minimum surface air temperature Tmin [in °C] and the direction of the fastest 2-minute wind in a day WDF2 [in degrees] of the Miami International Airport. The wind blowing from the north is zero degree, from the east is 90 degrees, and from the west is 270 degrees. The scatter plot seems to support two clusters or weather regimes: The lower regime showing the wind direction between 0 and 180 degrees, meaning the wind from the east, northeast, or southeast; and and another with the wind direction between 180 degrees and 360 degrees, meaning that the wind came from the west, northwest, or southwest;

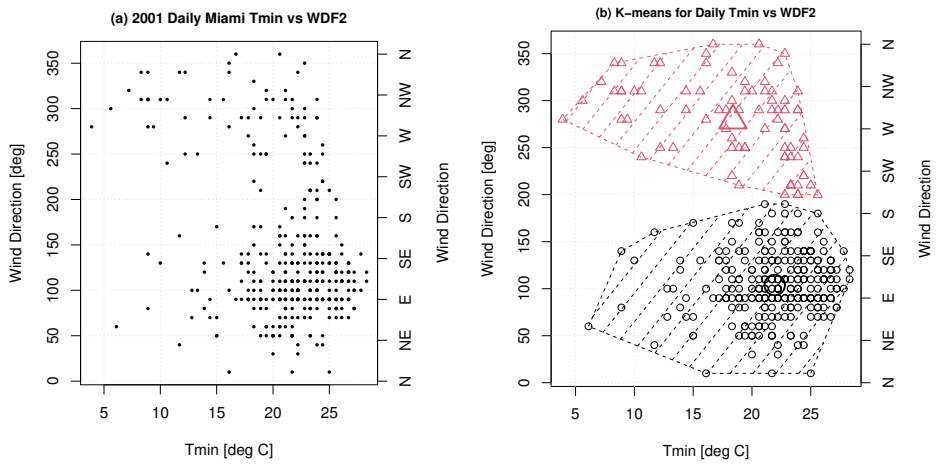


Fig. 3.2 (a) Scatter plot of the daily Tmin vs WDF2 for the Miami International Airport in 2001. (b) The K-means clusters of the the daily Tmin vs WDF2 data points when assuming $K = 2$.

We would like to use the K-means method to make the clustering automatically and justify our above intuitive conclusion from the scatter diagram. This example, although simple, is closer to reality. You can apply the procedures described here to your own data for various clustering purposes, such as defining wildfire zones, ecoregions, climate regimes, and agricultural areas. You can find numerous application examples by an online search using keywords like "K-means clustering for ecoregions", or "K-means clustering for climate regimes."

The daily weather data for the Miami International Airport can be obtained from the Global Historical Climatology Network-Daily (GHCN-D). The station code is USW00012839. The daily data from 2001 to 2020 are included in the book's master data file `data.zip` from the book website `www.climatestatistics.org`. The file name of the Miami data is

`MiamiIntlAirport2001_2020.csv`

You can access the updated data by an online search for " NOAA Climate Data Online USW00012839. " Figure 3.2(b) shows the K-means clustering result for the daily Tmin vs WDF2 for the 2001 daily data at the Miami International Airport. The two clusters generated by the K-means method agree with our intuition by looking at the scatter diagram Fig. 3.2(a). However, the K-means clustering result Fig. 3.2(b) provides a more clear picture: A red cluster with each data point indicated by a small triangle, and a black cluster with each data point indicated by a small circle. The cluster boundaries are linked by dashed lines. The large vertical gap at Tmin around 5°C (in winter) between the two clusters indicates that the fastest 2-minute wind in a day has a WDF2 angle value around 50°, a northeast wind for the red cluster, and around 300°, a northwest wind for the black cluster.

Figure 3.2 may be generated by the following computer codes.

```
#R for Fig. 9.2: K-means clustering for 2001 daily weather
#data at Miami International Airport, Station ID USW00012839
setwd("~/climstats")
dat = read.csv("data/MiamiIntlAirport2001_2020.csv",
                header=TRUE)
dim(dat)
#[1] 7305   29
tmin = dat[,'TMIN']
wdf2 = dat[,'WDF2']
# plot the scatter diagram Tmin vs WDF2
setEPS() #Plot the data of 150 observations
postscript("fig0902a.eps",  width=5, height=5)
par(mar=c(4.5, 4.5, 2, 4.5))
plot(tmin[2:366], wdf2[2:366],
     pch =16, cex = 0.5,
     xlab = 'Tmin [deg C]',
     ylab = 'Wind Direction [deg]', grid())
title('(a) 2001 Daily Miami Tmin vs WDF2',
      cex.main = 0.9, line = 1)
axis(4, at = c(0, 45, 90, 135, 180, 225, 270, 315, 360),
     lab = c('N', 'NE', 'E', 'SE', 'S', 'SW',  'W', 'NW', 'N'))
mtext('Wind Direction', side = 4, line =3)
```

```
dev.off()
#K-means clustering
K = 2 #assuming K = 2, i.e., 2 clusters
mydata = cbind(tmin[2:366], wdf2[2:366])
fit = kmeans(mydata, K) # K-means clustering
#Output the coordinates of the cluster centers
fit$centers
#1 18.38608 278.8608
#2 21.93357 103.9161
fit$tot.withinss # total WCSS
#[1] 457844.9 for # the value may vary for each run

#Visualize the clusters by kmeans.ani()
mycluster <- data.frame(mydata, fit$cluster)
names(mycluster)<-c('Tmin [deg C]',
                    'Wind Direction [deg]',
                    'Cluster')
library(animation)
par(mar = c(4.5, 4.5, 2, 4.5))
kmeans.ani(mycluster, centers = K, pch=1:K, col=1:K,
           hints = '')
title(main=
        "(b) K-means Clusters for Daily Tmin vs WDF2",
      cex.main = 0.8)
axis(4, at = c(0, 45, 90, 135, 180, 225, 270, 315, 360),
     lab = c('N', 'NE', 'E', 'SE', 'S', 'SW',  'W', 'NW', 'N'))
mtext('Wind Direction', side = 4, line =3)
```

```
# Python plot Fig. 9.2: K-means clustering for Tmin and WDF2
# K-means clustering for Fig. 9.2(b)
K = 2
# combine data into two column df
mydata = pd.concat([tmin[1:366],wdf2[1:366]], axis = 1)

# K-means clustering
fit = KMeans(n_clusters=K).fit(mydata)

# cluster center coordinates
print(fit.cluster_centers_, '\n')#enters of the two clusters
print(fit.inertia_) # total WCSS

#kmeans = KMeans(n_clusters=K, random_state=0)
mydata['cluster'] = \
kmeans.fit_predict(mydata[['TMIN', 'WDF2']])
colors = ['black', 'red']
mydata['color'] = \
mydata.cluster.map({0:colors[0], 1:colors[1]})

kmeans = KMeans(n_clusters=K, random_state=0)
mydata['cluster'] = \
kmeans.fit_predict(mydata[['TMIN', 'WDF2']])

colors = ['black', 'red']
mydata['color'] = \
mydata.cluster.map({0:colors[0], 1:colors[1]})

x1 = fit.cluster_centers_[0][0] #Cluster center
x2 = fit.cluster_centers_[1][0]
y1 = fit.cluster_centers_[0][1]
y2 = fit.cluster_centers_[1][1]

plt.title("(b) K-means for Tmin vs WDF2", pad = 10)
plt.xlabel(r"Tmin [$\degree$C]", labelpad = 10)
plt.ylabel(r"Wind Direction [$\degree$]", labelpad = 10)
plt.yticks([0,50,100,200,300])
# plot points
plt.scatter(mydata['TMIN'], mydata['WDF2'],
            s = 10, color = mydata['color']) #s is size

for i in mydata.cluster.unique():
    points = \
    mydata[mydata.cluster == i][['TMIN', 'WDF2']].values
    # get convex hull
    hull = ConvexHull(points)
    # get x and y coordinates
    # repeat last point to close the polygon
    x_hull = np.append(points[hull.vertices,0],
                       points[hull.vertices,0][0])
    y_hull = np.append(points[hull.vertices,1],
                       points[hull.vertices,1][0])
    # plot shape
    plt.fill(x_hull, y_hull, alpha=0.2, c=colors[i])
    # plot centers
plt.scatter([x1,x2],[y1,y2], color = ['red', 'black'],
            linewidth = 2, facecolors= 'none', s = 700)
plt.show()
```

3.1.4.2 Why K = 2 for the Miami weather data?

The K-means computer code for the Miami Tmin and WDF2 data shows two cluster centers at $C_1(18.4°\text{C}, 278.9°)$, and $C_2(21.9°\text{C}, 103.9°)$. The corresponding total WCSS is 457844.9. To justify our choice of $K = 2$, we compute tWCSS, the total WCSS, for different numbers of K, and then use the elbow rule to determine the best K value. Figure 3.3(a) shows the tWCSS scores for $K = 2, 3, \cdots, 8$. We can see that the elbow appears at $K = 2$, since total WCSS decrement suddenly slows down from $K = 2$. Thus, the elbow rule has justified our choice of $K = 2$.

Another method to choose K is the knee rule, which is determined by the variances explained by K clusters. When $K = 1$, tWCSS is the spatial variance of the entire data. When $K = N$, tWCSS $= 0$, where N is the total number of the data points. Usually, we have

$$\text{tWCSS}[1] \geq \text{tWCSS}[K] \geq \text{tWCSS}[N] \tag{3.23}$$

for any $1 \leq K \leq N$. Then,

$$\text{pWCSS}[K] = \frac{\text{tWCSS}[1] - \text{tWCSS}[K]}{\text{tWCSS}[1]} \times 100\% \tag{3.24}$$

is defined as the percentage of variance explained by the K clusters. Usually, pWCSS$[K]$ is an increasing function of K. When the increase rate suddenly slows down from K, a shape like a knee appears in the pWCSS$[K]$ curve. We then use this K as the optimal number of centers for our K-means clustering. This is the so called the knee rule. Figure 3.3(b) shows that the increase of pWCSS$[K]$ dramatically slows down at $K = 2$. Thus, we choose $K = 2$, which provides another justification why $K = 2$.

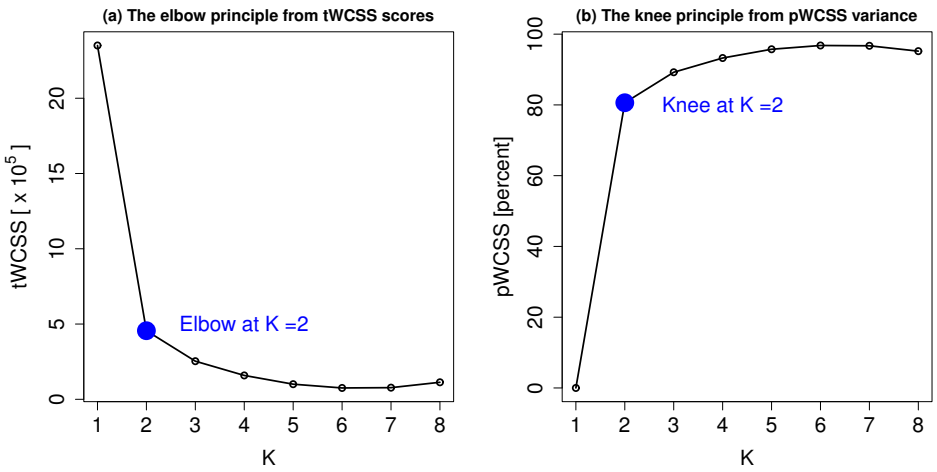


Fig. 3.3 (a) tWCSS scores for different K. (b) pWCSS variances for different K.

Figure 3.3 may be generated by the following computer codes.

```
#R plot Fig. 9.3: tWCSS(K) and pWCSS(K)
twcss = c()
for(K in 1:8){
  mydata=cbind(tmax[2:366], wdf2[2:366])
  twcss[K] = kmeans(mydata, K)$tot.withinss
}
 twcss
par(mar = c(4.5, 6, 2, 0.5))
par(mfrow=c(1,2))
plot(twcss/100000, type = 'o', lwd = 2,
     xlab = 'K', ylab = bquote('tWCSS [ x' ~  10^5 ~ ']'),
     main = '(a) The elbow principle from tWCSS scores',
     cex.lab = 1.5, cex.axis = 1.5)
points(2, twcss[2]/100000, pch =16, cex = 3, col = 'blue')
text(4, 5, 'Elbow at K = 2', cex = 1.5, col = 'blue')
#compute percentage of variance explained
pWCSS = 100*(twcss[1]- twcss)/twcss[1]
plot(pWCSS, type = 'o', lwd = 2,
     xlab = 'K', ylab = 'pWCSS [percent]',
     main = '(b) The knee principle from pWCSS variance',
     cex.lab = 1.5, cex.axis = 1.5)
points(2, pWCSS[2], pch =16, cex = 3, col = 'blue')
text(4, 80, 'Knee at K = 2', cex = 1.5, col = 'blue')
dev.off()
```

```
#Python plot Fig. 9.3: tWCSS(K) and pWCSS(K)
#Plot Fig. 9.3(a): Elbow principle
twcss = np.zeros(9)
for K in range(1,9):
    mydata = pd.concat([tmin[1:366],wdf2[1:366]], axis = 1)
    twcss[K] = KMeans(n_clusters=K).fit(mydata).inertia_
# remove K = 0 value from twcss
twcss = twcss[1:]
# plot elbow
plt.plot(np.linspace(1,8,8), twcss/100000, 'k', linewidth=2)
# plot points
plt.scatter([np.linspace(1,8,8)], [twcss/100000], color='k')

# plot elbow at K = 2
plt.scatter(2,twcss[1]/100000, color = 'blue', s = 500)
# add text
plt.text(2.2, 5.2, 'Elbow at K = 2', color='blue', size=20)
# add labels
plt.title("(a) The elbow principle from tWCSS scores",
          pad = 10)
plt.xlabel("K", labelpad = 10)
plt.ylabel(r"tWCSS [x $10^5$]", labelpad = 10)
plt.show()

#Plot Fig. 9.3(b): Knee principle
# compute percentage of variance explained
pWCSS = 100*(twcss[0]-twcss)/twcss[0]
# plot knee
plt.plot(np.linspace(1,8,8),pWCSS, 'k', linewidth = 2)
# add points
plt.scatter(np.linspace(1,8,8),pWCSS, color = 'k')

# plot knee at K = 2
plt.scatter(2,pWCSS[1], color = 'blue', s = 500)
# add text
plt.text(2.2, 76, 'Knee at K = 2', color = 'blue',
         size = 20)
# add labels
plt.title("(b) The knee princple from pWCSS variance",
          pad = 10)
plt.xlabel("K", labelpad = 10)
plt.ylabel("pWCSS [%]", labelpad = 10)
plt.show()
```

3.1.4.3 Steps of the K-means clustering data analysis

From the previous sub-sections, we may summarize that the K-means method for data analysis consists of the following steps.

(i) Data preparation: Write your data into an $N \times 2$ matrix, where N is the total number of data points, and 2 is for two variables, i.e., in a 2-dimensional space. The real value matrix must have no missing data. Although our

previous examples are for 2-dimensional data, the K-means clustering can be used for higher dimensional data. On the Internet, you can easily find examples of the K-means for data of three or more variables.

(ii) Preliminary data analysis: You may wish to make some preliminary simple analyses before applying the K-means method. These analyses may include plotting a scatter diagram, plot a histogram for each variable, and compute mean, standard deviation and quantiles for each variable. If the data for the two variables have different units, you may convert the data into their standardized anomalies. The standardized anomaly computing procedure is called "scale" in the machine learning community. For example, the R command `scale(x)` yields the standard anomalies of data sequence `x`, i.e, divided by the standard deviation after a subtraction of mean. Mathematically speaking, the scaling procedure is preferred when two variables have different units, because the $\text{unit1}^2 + \text{unit2}^2$ in tWCSS usually do not have a good interpretation. In contrast, standardized anomalies are dimensionless and have no units. Non-dimensional data are often preferred when applying statistical methods. However, our goal of the K-means clustering is to find patterns without labeling the data, which is an unsupervised learning. The K-means results from scaled data and unscaled data may not be the same. As long as the K-means result is reasonable and interpretable, regardless of the scaled or unscaled data, we adopt the result. For example, in our Miami Tmin and WDF2 example in the previous sub-section, we found the K-means clusters for both scaled and unscaled data, but we have chosen the result from the unscaled data from our view point of interpretation. Therefore, we keep in mind that the K-means is a result-oriented method.

(iii) Test K-means clustering: Apply a computer code of K-means to the prepared $N \times 2$ data matrix with a K value inspired from the scatter diagram or science background of the data. You may apply the code once to the scaled data and another to the unscaled data, and see if the results make sense. Run your code a few times and see if the results vary.

(iv) Determine the optimal K by the elbow rule: Compute the tWCSS scores for different K and find the elbow location of the tWCSS(K) curve. Apply the K-means computer code for this optimal K value. Compare this K with the K in the previous step. Check the details of the clustering results, such as cluster assignment, tWCSS, centers.

(v) Visualize the K-means clustering results: Use a computer code to plot the K clusters. Different visualization codes are available for your choice.

(vi) Interpret the final result: Identify the characteristics of the clusters and interpret them with text or more figures.

Applying the K-means computer code is very easy, but in the K-means data analysis, we should avoid the following common mistakes:

- Skipped the preliminary data analysis step: Some people may quickly apply the

K-means computer code and treat the K-means results as unique, hence regard their their plot of the clusters as the end of the analysis.

- Missed the interpretation: Interpretation step is needed, short or long. This is our purpose anyway.
- Forgot to scale the data: We should always analyze the scaled data. At the same time, because of the interpretation requirement, we should also check the results from the unscaled data, and see which result is more reasonable.
- Overlooked the justification of the optimal K: A complete K-means clustering should have this step, which helps us to explore all the possible clusters.

In addition to using the standard K-means computer code you can find from the Internet or that built in R or Python, you sometimes may wish to make special plots for your clustering results with some specific features, such as a single cluster on a figure with the marked data order, as shown in Fig. 3.4. Convex hull is a simple method to plot a cluster. By definition, the convex hull of a set of points is the smallest convex polygon that encloses all of the points in the set. Below is the a computer code to plot Fig. 3.4 using the convex hull method.

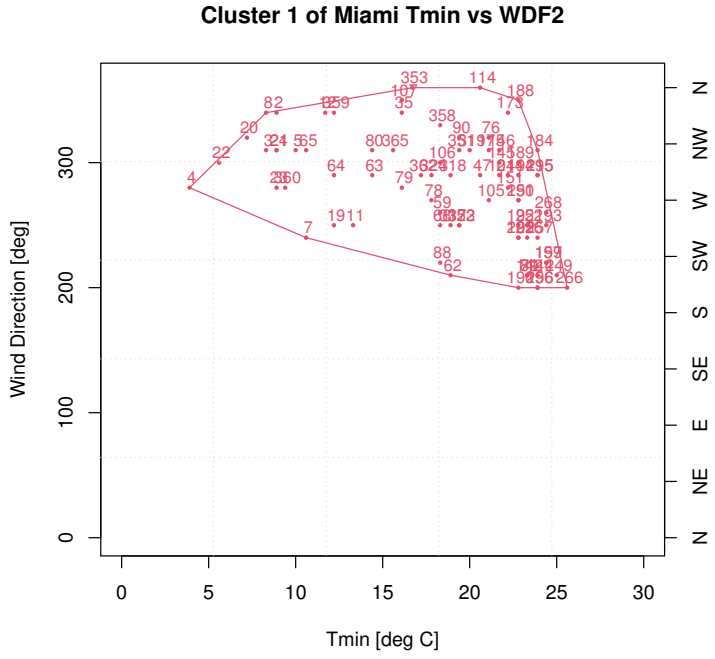


Fig. 3.4 Convex hull for Cluster 1. The number above each data point is the day number in a year, e.g., January 1 being 1, February 2 being 33, and December 31 being 365 in 2001. The day numbers help identify the seasonality of the points, such as, the left most tip of the cluster being Day 4, a specific winter day.

Figure 3.4 may be generated by the following computer codes.

```
#R plot Fig. 9.4: Convex hull for a cluster
setwd("~/climstats")
dat = read.csv("data/MiamiIntlAirport2001_2020.csv",
               header=TRUE)
dim(dat)
#[1] 7305   29
tmin = dat[,'TMIN']
wdf2 = dat[,'WDF2']
#K-means clustering
K = 2 #assuming K = 2, i.e., 2 clusters
mydata = cbind(tmin[2:366], wdf2[2:366]) #2001 data
clusterK = kmeans(mydata, K) # K-means clustering
mycluster <- data.frame(mydata, clusterK$cluster)
plotdat = cbind(1:N, mycluster)

par(mar = c(4.5, 6, 2, 0.5)) #set up the plot margin
i = 1 # plot Cluster 1
N = 365 #Number of data points
X = plotdat[which(mycluster[,3] == i), 1:3]
colnames(X)<-c('Day', 'Tmin [deg C]', 'WDF2 [deg]')
plot(X[,2:3], pch = 16, cex = 0.5, col = i,
     xlim = c(0, 30), ylim = c(0, 365),
     main = 'Cluster 1 of Miami Tmin vs WDF2' )
grid(5, 5)
#chull() finds the boundary points of a convex hull
hpts = chull(X[, 2:3])
hpts1 = c(hpts, hpts[1]) #close the boundary
lines(X[hpts1, 2:3], col = i)
for(j in 1:length(X[,1])){
  text(X[j,2], X[j,3] + 8, paste("", X[j,1]),
       col = i, cex = 0.8)
} #Put the data order on the cluster
```

```
# Python plot Fig. 9.4: Convex hull for a cluster
K = 2
clusterK = KMeans(n_clusters=K).fit(mydata)
mydata['cluster'] = \
       kmeans.fit_predict(mydata[['TMIN', 'WDF2']])
mydata['day']= np.arange(1, 366)
subset = mydata[mydata['cluster']==0]
# set up plot
plt.title("Cluster 1 of Miami Tmin vs WDF2", pad = 10)
plt.xlabel(r"Tmin [$\degree$C]", labelpad = 10)
plt.ylabel(r"WDF2 [$\degree$]", labelpad = 10)

plt.xlim([0,30])
plt.ylim([0,400])
plt.yticks([0,100,200,300])

# plot points
plt.scatter(subset["TMIN"],subset["WDF2"], color = 'k')
# add labels
for i in range(len(subset["TMIN"])):
    plt.text(subset["TMIN"].values[i],
             subset["WDF2"].values[i],
             subset["day"].values[i], size=12)
# add boundary
for i in subset.cluster.unique():
    points = \
      subset[subset.cluster == i][['TMIN', 'WDF2']].values
    # get convex hull
    hull = ConvexHull(points)
    # get x and y coordinates
    # repeat last point to close the polygon
    x_hull = np.append(points[hull.vertices,0],
                       points[hull.vertices,0][0])
    y_hull = np.append(points[hull.vertices,1],
                       points[hull.vertices,1][0])
    # plots hape
    plt.fill(x_hull, y_hull, alpha=0.2, c=colors[i])
```

The K-means method can not only be used for weather data clustering and climate classification, but also for many other purposes, such as weather forecasting, storm identification, and more. In many applications, K-means is part of a machine learning algorithm, and takes a modified version, such as the incremental K-means for weather forecasting and the weighted K-means for climate analysis.

3.2 Support vector machine

The K-means clustering can organize a suite of given data points into K clusters. The data points are not labeled. This is like organizing basket of fruits into K piles

according to the minimum tWCSS principle. After the clustering, we may name the clusters, say $1, 2, \cdots, K$, or apples, oranges, peaches, and grapes. ML often uses the term "cluster label " in lieu of "cluster name." So, we say that we label a cluster when we name a cluster. The learning process for a un-labeled data is called unsupervised learning. If a basket of fruits is sorted out by a baby who cannot tell the fruit names according to wishes of his mother, he is performing an unsupervised learning.

In contrast, support vector machine (SVM) is in general a supervised learning, working on the labelled data, e.g., a basket of apples, oranges, peaches and grapes. Based on the numerical data, SVM has two purposes. First, SVM determines the maximum "distances" among the sets of labeled data, say apples, oranges, peaches and grapes. This is a training process and helps a computer learn the differences among different labeled sets, e.g., different fruits, different weather, or different climate regimes. Second, SVM makes a prediction, which, based on the trained model, predicts the labels (i.e., names) of the new and unlabeled data. Thus, SVM is a system to learn the maximum differences among the labelled clusters of data, and to predict the labels of the new unlabeled data. This process of training and prediction mimics a human's learning experience. For example, through years a child is trained by his parents to learn the knowledge of apples, oranges and other fruits. At a certain time, the child is well trained and ready to independently tell (i.e., predict) whether a new fruit is apple or orange. During this learning process, the fruits are labeled (e.g., apples, oranges, etc). His parents help input the data into his brain through a teaching and learning process. His brain was trained to maximize the difference between an apple and an orange. In the prediction stage, the child independently imports the data using his eyes, and make the prediction whether a fruit is an apple or orange. SVM mimics this process by maximizing the difference between two or more labeled categories or clusters in the training process, then by using the trained model, SVM predicts the categories for the new data. The key is how to quantify the difference between any two categories.

In the following we will illustrate these SVM learning ideas using three examples: (i) A trivial case of two data points, (ii) training and prediction for a system of three data points, and (iii) a more general case of many data points.

3.2.0.1 Maximize the difference between two labelled points

We formulate a two-point learning question as follows. We have an apple and an orange. We know all the data about the apple and orange. We teach a baby how to use the data to quantify the maximum difference between the apple and orange. In the machine learning language, "apple" and "orange" are called labels. The data are called the training data, such as color and surface smoothness. The maximum difference will be used to predict whether a new fruit is an apple or orange.

An abstract description of the above may be put in the following way. Given the data for an object labeled A and also the data for another object labeled B, quantify the maximum difference between A and B. A simple example corresponding to this

statement is that given two points $P_1(1,1)$ and $P_2(3,3)$ which are labeled $y_1 = -1$ and $y_2 = 1$, quantify the maximum difference between y_1 and y_2. For this simple problem, the obvious answer is that the two categories labeled by y_1 and y_2 are divided by the perpendicular bisector of the two points $P1$ and P_2 as shown in Fig. 3.5. The bisector is called the *separating hyperplane* . A hyperplane means a plane in a space of more than three dimensions. In a 2-dimensional space, it is a straight line. In a 3-dimensional space, it is just a regular flat plane. The real ML applications are often for higher dimensional data, e.g., a comprehensive description of an apple may need data of color, surface smoothness, stem size, skin thickness, and more, which form a p-dimensional vector.

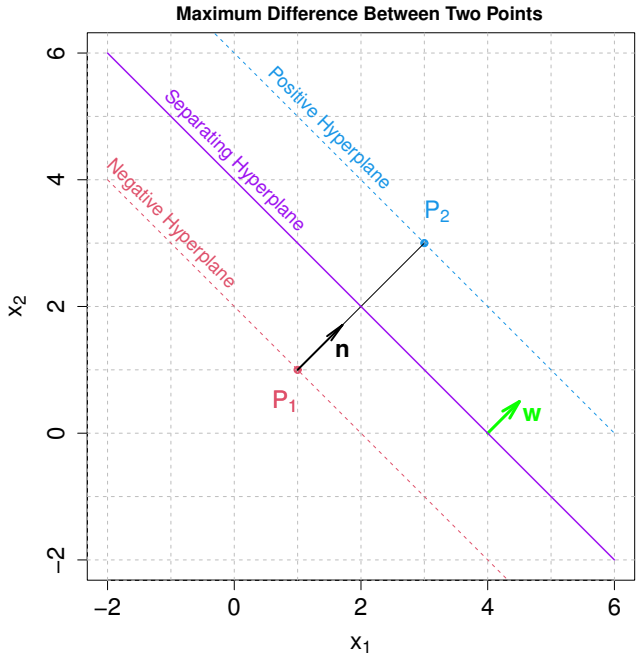


Fig. 3.5 Quantify the maximum difference between two points.

The maximum margin of difference between the two categories are bounded by two lines parallel to the bisector, one line passing through P_1 and another P_2. One line is called the negative hyperplane, and another the positive hyperplane. The data points on these hyperplanes are called *support vectors*. The support vector machine (SVM) has its name from these vectors.

The equation for the separating hyperplane (i.e., the purple line) is

$$x_1 + x_2 = 4, \tag{3.25}$$

that for the positive hyperplane (i.e., the blue dashed line) is

$$x_1 + x_2 = 6, \tag{3.26}$$

and that for the negative hyperplane (i.e., the red dashed line) is

$$x_1 + x_2 = 2. \tag{3.27}$$

The unit vector in the perpendicular bisector direction from P_1 to P_2 is

$$\mathbf{n} = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} \right). \tag{3.28}$$

This vector is a normal vector for the separating hyperplane. The equation of the separating hyperplane may be expressed in the normal vector form

$$\mathbf{n} \cdot \mathbf{x} = 2\sqrt{2}. \tag{3.29}$$

This can further be written in the following form

$$\mathbf{w} \cdot \mathbf{x} - b = 0, \tag{3.30}$$

where

$$\mathbf{w} = (0.5, 0.5) \tag{3.31}$$

and

$$b = 2. \tag{3.32}$$

An advantage of this form is that the equations for the positive and negative hyperplanes can be simply written in the form

$$\mathbf{w} \cdot \mathbf{x} - b = \pm 1, \tag{3.33}$$

and that for the separating hyperplane can be written as

$$\mathbf{w} \cdot \mathbf{x} - b = 0. \tag{3.34}$$

In the above, $\mathbf{w}$ is called a normal vector and is related with the unit normal vector $\mathbf{n}$ as follows:

$$\mathbf{w} = -\frac{1}{\sqrt{2}}\, \mathbf{n}. \tag{3.35}$$

The distance between the positive and negative hyperplanes is

$$D_m = \frac{2}{|\mathbf{w}|} = 2\sqrt{2}, \tag{3.36}$$

is the margin size. The maximum difference between P_1 and P_2 is quantified by D_m. The primary goal of SVM is to maximize D_m when training data are given.

The distance of the origin to the separating hyperplane is

$$D_o = -\frac{b}{|\mathbf{w}|} = 2\sqrt{2}. \tag{3.37}$$

These formulations of the hyperplanes and the quantities $\mathbf{w}, b, D_m$ can be used for the general SVM mathematical formulation, which are described in the next sub-section.

Figure 3.5 may be generated by the following computer code.

```
#R plot Fig. 9.5: Maximum difference between two points
x = matrix(c(1, 1, 3, 3),
           ncol = 2, byrow = TRUE)#Two points
y= c(-1, 1) #Two labels -1 and 1
#Plot the figure and save it as a .eps file
setEPS()
postscript("fig0905.eps", height=7, width=7)
par(mar = c(4.5, 4.5, 2.0, 2.0))
plot(x, col = y + 3, pch = 19,
     xlim = c(-2, 6), ylim = c(-2, 6),
     xlab = bquote(x[1]), ylab = bquote(x[2]),
     cex.lab = 1.5, cex.axis = 1.5,
     main = "Maximum␣Difference␣Between␣Two␣Points")
axis(2, at = (-2):6, tck = 1, lty = 2,
     col = "grey", labels = NA)
axis(1, at = (-2):6, tck = 1, lty = 2,
     col = "grey", labels = NA)
segments(1, 1, 3, 3)
arrows(1, 1, 1.71, 1.71, lwd = 2,
       angle = 9, length= 0.2)
text(1.71, 1.71-0.4, quote(bold('n')), cex = 1.5)
arrows(4, 0, 4 + 0.5, 0 + 0.5, lwd = 3,
       angle = 15, length= 0.2, col = 'green' )
text(4.7, 0.5 -0.2, quote(bold('w')),
     cex = 1.5, col = 'green')
x1 = seq(-2, 6, len = 31)
x20 = 4 - x1
lines(x1, x20, lwd = 1.5, col = 'purple')
x2m = 2 - x1
lines(x1, x2m, lty = 2, col = 2)
x2p = 6 - x1
lines(x1, x2p, lty = 2, col = 4)
text(1-0.2,1-0.5, bquote(P[1]), cex = 1.5, col = 2)
text(3+0.2,3+0.5, bquote(P[2]), cex = 1.5, col = 4)
text(1-0.2,1-0.5, bquote(P[1]), cex = 1.5, col = 2)
text(0,4.3, 'Separating␣Hyperplane',
     srt = -45, cex = 1.2, col = 'purple')
text(1.5, 4.8, 'Positive␣Hyperplane',
     srt = -45, cex = 1.2, col = 4)
text(-1, 3.3, 'Negative␣Hyperplane',
     srt = -45, cex = 1.2, col = 2)
dev.off()
```

```
#Python plot Fig. 9.5: Maximum difference between two points
# define the two points
x = np.array([1,1,3,3]).reshape(2,2)
# define the two labels
y = (-1,1)
# set up plot
plt.figure(figsize = (10,10))
plt.title("Maximum Difference Between Two Points", pad = 10)
plt.xlabel("$x_1$", labelpad = 10)
plt.ylabel("$x_2$", labelpad = 10)
plt.xlim(-2.2,6.2)
plt.xticks([-2,-1,0,1,2,3,4,5,6],[-2,'',0,'',2,'',4,'',6])
plt.ylim(-2.2,6.2)
plt.yticks([-2,-1,0,1,2,3,4,5,6],[-2,'',0,'',2,'',4,'',6])
plt.grid()
# plot points
plt.scatter(x[0],x[0], color = 'r')
plt.scatter(x[1],x[1], color = 'dodgerblue')
# connect points
plt.plot([1,3],[1,3], 'k')
# add dashed lines
plt.plot([0,6],[6,0], color ='dodgerblue', linestyle ='--')
plt.plot([-2,6],[6,-2], color = 'purple', linestyle = '--')
plt.plot([-2,4],[4,-2], 'r', linestyle = '--')
# plot arrows
plt.arrow(1,1,0.5,0.5, head_width = 0.12, color='k')
plt.arrow(4,0,0.3,0.3, head_width = 0.12, color='limegreen')
# add text
plt.text(0.5,4.3, "Positive Hyperplane", rotation = -45,
         size = 14, color = 'dodgerblue')
plt.text(3.2,3.2, "$P_2$", size = 15, color = 'dodgerblue')
plt.text(-1.2,3.8, "Separating Hyperplane", rotation = -45,
         size = 14, color = 'purple')
plt.text(4.4,0.1, "w", size = 15, color = 'limegreen')
plt.text(1.5,1.2, "n", size = 15)
plt.text(-2,2.7, "Negative Hyperplane", rotation = -45,
         size = 14, color = 'r')
plt.text(0.7, 0.6, "$P_1$", size = 15, color = 'r')
plt.show()
```

3.2.1 SVM for a system of three points labeled in two categories

Let us progress from a two-point system to a three-point system and build an SVM using a computer code. Figure 3.6 shows the three points labeled in two categories. The two blue points are $P_1(1,1), P_2(2,1)$ which are in the first category corresponding to the categorical value $y = 1$. The red point $P_3(3, 3.5)$ is in the second category corresponding to $y = -1$. Training these data is to maximize the margin size D_m, which results in the following SVM parameters.

$$w = (-0.28 - 0.69), \quad b = 2.24, \quad D_m = 2.69. \tag{3.38}$$

The support vectors are $P_2(2,1)$ and $P_3(3,3.5)$.

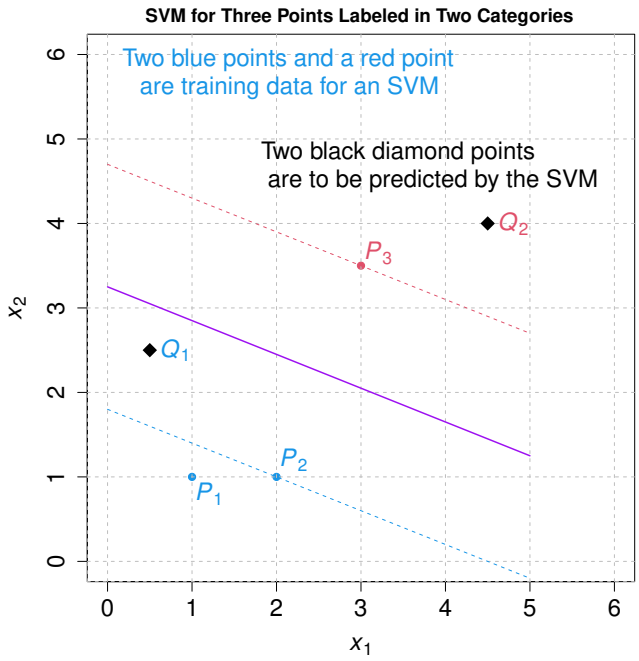


Fig. 3.6 SVM training from three data points, and the prediction by the trained SVM.

We acquire two new data points $Q_1(0.5, 2.5)$ and $Q_2(4.5, 4.0)$. We wish to use the trained SVM to predict whether $Q_1(0.5, 2.5)$ corresponds to 1 (i.e., the first category) or -1 (i.e., the second category). The SVM prediction results are shown in Fig. 3.6: $Q_1(0.5, 2.5)$ belongs to the first category, and $Q_2(4.5, 4.0)$ to the second category. Geometrically, this prediction result in Fig. 3.6 is obvious and has nothing surprising. The significance of this example is its systematic use of the SVM method and its computer algorithm. Although this system is simple and has only three points, solving this problem by hand is very difficult. The following computer code solves this problem and generates Fig. 3.6.

```
#R plot Fig. 9.6: SVM for three points
#training data x
x = matrix(c(1, 1, 2, 1, 3, 3.5),
           ncol = 2, byrow = TRUE)
y = c(1, 1, -1) #two categories 1 and -1
plot(x, col = y + 3, pch = 19,
     xlim = c(-2, 8), ylim = c(-2, 8))
library(e1071)
dat = data.frame(x, y = as.factor(y))
svm3P = svm(y ~ ., data = dat,
            kernel = "linear", cost = 10,
            scale = FALSE,
            type = 'C-classification')
svm3P #This is the trained SVM
```

```
xnew = matrix(c(0.5, 2.5, 4.5, 4),
              ncol = 2, byrow = TRUE) #New data
predict(svm3P, xnew) #prediction using the trained SVM
# 1  2
# 1 -1

# Find hyperplane, normal vector, and SV (wx + b = 0)
w = t(svm3P$coefs) %*% svm3P$SV
w
#[1,] -0.2758621 -0.6896552
b = svm3P$rho
b
#[1] -2.241379
2/norm(w, type ='2') #maximum margin of separation
#[1] 2.692582

x1 = seq(0, 5, len = 31)
x2 = (b - w[1]*x1)/w[2]
x2p = (1 + b - w[1]*x1)/w[2]
x2m = (-1 + b - w[1]*x1)/w[2]
x20 = (b - w[1]*x1)/w[2]
#plot the SVM results
setEPS()
postscript("fig0906.eps", height=7, width=7)
par(mar = c(4.5, 4.5, 2.0, 2.0))
plot(x, col = y + 3, pch = 19,
     xlim = c(0, 6), ylim = c(0, 6),
     xlab = bquote(x[1]), ylab = bquote(x[2]),
     cex.lab = 1.5, cex.axis = 1.5,
     main = 'SVM␣for␣three␣points␣labeled␣in␣two␣categories')
axis(2, at = (-2):8, tck = 1, lty = 2,
     col = "grey", labels = NA)
axis(1, at = (-2):8, tck = 1, lty = 2,
     col = "grey", labels = NA)
lines(x1, x2p, lty = 2, col = 4)
lines(x1, x2m, lty = 2, col = 2)
lines(x1, x20, lwd = 1.5, col = 'purple')
xnew = matrix(c(0.5, 2.5, 4.5, 4),
              ncol = 2, byrow = TRUE)
points(xnew, pch = 18, cex = 2)
for(i in 1:2){
  text(xnew[i,1] + 0.5, xnew[i,2] , paste('Q',i),
       cex = 1.5, col = 6-2*i)
}
text(2.2,5.8, "Two␣blue␣points␣and␣a␣red␣point
are␣training␣data␣for␣an␣SVM␣",
     cex = 1.5, col = 4)
text(3.5,4.7, "Two␣black␣diamond␣points
␣␣␣␣␣␣␣␣␣are␣to␣be␣predicted␣by␣the␣SVM",
     cex = 1.5)
dev.off()
```

```
#Python plot Fig. 9.6: SVM for three points
# define x for ease of plotting
x = np.array([1,1,2,1,3,3.5])
# redefine x for svm
X = [[1, 1],[2,1],[3,3.5]]
y = [1, 1, -1]
# define SVM
svm3P = svm.SVC(kernel='linear')
# train SVM
svm3P.fit(X, y)
# predict new data using the trained SVM
print('The prediction is: ',
      svm3P.predict([[0.5, 2.5],[4.5,4]]), '\n')
# find hyperplane, normal vector, and SV (wx + b = 0)
w = svm3P.coef_[0]
print('w is: ',w, '\n')
b = -svm3P.intercept_ #intercept
print('b is: ', b, '\n')
# find the maximum margin of separation
print(2/np.linalg.norm(w))
x1 = np.linspace(0,5,5)
x2 = (b - w[0]*x1)/w[1]
x2p = (1 + b - w[0]*x1)/w[1]
x2m = (-1 + b - w[0]*x1)/w[1]
x20 = (b - w[0]*x1)/w[1]
# set up plot
plt.figure(figsize = (10,10))
plt.title("SVM for three points labeled in two categories",
          pad = 10)
plt.xlabel("$x_1$", labelpad = 10)
plt.ylabel("$x_2$", labelpad = 10)
plt.xlim(-0.2,5.2)
plt.ylim(-0.2,6.2)
plt.grid()
# plot original points
plt.scatter([x[::2]],[x[1::2]],
            color=['dodgerblue','dodgerblue','red'])
# add lines
plt.plot(x1,x2p, color = 'dodgerblue', linestyle = '--')
plt.plot(x1, x2m, color = 'red', linestyle = '--')
plt.plot(x1, x20, color = 'purple')
# plot diamonds
plt.scatter([0.5,4.5], [2.5,4], color = 'k', marker = 'D')
# add text
plt.text(0.65,2.45,"$Q_1$", size = 16, color = 'dodgerblue')
plt.text(4.65,3.95, "$Q_2$", size = 16, color = 'r')
plt.text(0.3,5.4, "Two blue points and a red point \n \
are training data for an SVM",
        size = 16, color = 'dodgerblue')
plt.text(1.7,4.5, "Two black diamond points \n \
          are to be predicted by the SVM",
        size = 16, color = 'k')
plt.show()
```

3509

3.2.2 SVM mathematical formulation for a system of many points in two categories

The previous quantification of maximum separation between two points or three points can be generalized to the separation of many points of two categories labeled $(1, -1)$, or $((1, 2)$, or (A, B), or another way. What is a systematic formulation of mathematics to make the best separation, by maximizing the margin? How can we use a computer code to implement the separation? This sub-section attempts to answer these questions.

We have n training data points:

$$(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), , \cdots, (\mathbf{x}_n, \mathbf{y}_n), \tag{3.39}$$

where $\mathbf{x}_i$ is a p-dimensional vector (i.e., using p parameters to describe every data point in a category), and $\mathbf{y}_i$ is equal to 1 or -1, a category indicator, $i = 1, 2, \cdots, n$.

The separating hyperplane has its linear equation as follows

$$\mathbf{w} \cdot \mathbf{x} - b = 0. \tag{3.40}$$

Our SVM algorithm is to find the p-dimensional normal vector $\mathbf{w}$ and a real scaler b so that the distance

$$D_m = \frac{2}{|\mathbf{w}|} \tag{3.41}$$

between the positive and negative hyperplanes

$$\mathbf{w} \cdot \mathbf{x} - b = \pm 1 \tag{3.42}$$

is maximized.

The maximization is the domain in the p-dimensional space below the negative hyperplane and above the positive hyperplane, i.e.,

$$\mathbf{w} \cdot \mathbf{x}_i - b \geq 1 \quad \text{when} \quad y_i = 1 \tag{3.43}$$

and

$$\mathbf{w} \cdot \mathbf{x}_i - b \leq 1 \quad \text{when} \quad y_i = -1. \tag{3.44}$$

This definition of the maximization implies that no training data points are located between the positive and negative hyperplanes. Thus, the solution of the maximization of $D_m = 2/|\mathbf{w}|$ must occur at the boundary of the domain, i.e., the positive and negative hyperplanes. The points on these hyperplanes are called the support vectors (SV). R or Python SVM code can find these SVs.

Figure 3.7 shows an example of an SVM training with 20 training data points, and an SVM prediction for 3 data points. The data are given below:

```
   X1  X2 y
1   1 6.0 1
2   2 8.0 1
3   3 7.5 1
```

```
4   1 8.0 1
5   4 9.0 1
6   5 9.0 1
7   3 7.0 1
8   5 9.0 1
9   1 5.0 1
10  5 3.0 2
11  6 4.0 2
12  7 4.0 2
13  8 6.0 2
14  9 5.0 2
15 10 6.0 2
16  5 0.0 2
17  6 5.0 2
18  8 2.0 2
19  2 2.0 2
20  1 1.0 2
```

The new data to be predicted are

```
[1,]  0.5  2.5
[2,]  7.0  2.0
[3,]  6.0  9.0
```

The following computer code can do the SVM calculation and plot Fig. 3.7.

We acquire three new data points $Q_1(0.5, 2.5)$, $Q_2(7.5, 2.0)$ and $Q_3(6.0, 9.0)$. We wish to use the trained SVM to find out which point belongs to what category. The SVM prediction result is shown in Fig. 3.7: $Q_1(0.5, 2.5)$ and $Q_2(7.5, 2.0)$ belong to Category 1, and $Q_3(6.0, 9.0)$ to Category 2. Again, this prediction result is obvious to us when the new data points are plotted, our eyes can see the points, and our brains can make the decisions based on our trained intelligence. The SVM applications in the real world often involve data of higher dimensions, in which case the visual decision is impossible, and requires us to predict the category for the new data without plotting. The prediction is based on the training data, new data, and SVM algorithm without visualization. For example, the R prediction of this problem can be done by the command `predict(svmP, xnew)` where `xnew` is the new data in a 3×2 matrix, and `svmP` is the trained SVM. This command is included in the computer code for generating Fig. 3.7.

```
#R plot Fig. 9.7: SVM for many points
#Training data x and y
x = matrix(c(1, 6, 2, 8, 3, 7.5, 1, 8, 4, 9, 5, 9,
             3, 7, 5, 9, 1, 5,
             5, 3, 6, 4, 7, 4,   8, 6, 9, 5, 10, 6,
             5, 0, 6, 5, 8, 2, 2, 2, 1, 1),
           ncol = 2, byrow = TRUE)
y= c(1, 1, 1, 1, 1, 1,
```

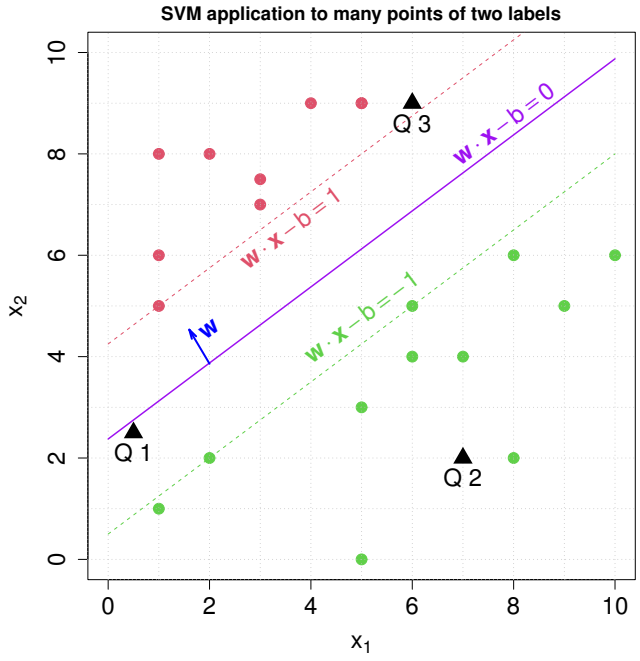


Fig. 3.7 SVM training from 20 data points, and the prediction of 3 new data points by the trained SVM.

```
      1, 1, 1,
      2, 2, 2, 2, 2, 2 ,
      2, 2, 2, 2, 2)

library(e1071)
dat = data.frame(x, y = as.factor(y))
svmP = svm(y ~ ., data = dat,
           kernel = "linear", cost = 10,
           scale = FALSE,
           type = 'C-classification')
svmP
#Number of Support Vectors:  3
svmP$SV #SVs are #x[9,], x[17,], x[19,]
#9   1  5
#17  6  5
#19  2  2

# Find SVM parameters: w, b, SV (wx+c=0)
w = t(svmP$coefs) %*% svmP$SV
# In essence this finds the hyper plane that separates our points
w
#[1,] -0.39996 0.53328
b = svmP$rho
b
#[1] 1.266573
2/norm(w, type ='2')
```

```
#[1] 3.0003 is the maximum margin
x1 = seq(0, 10, len = 31)
x2 = (b - w[1]*x1)/w[2]
x2p = (1 + b - w[1]*x1)/w[2]
x2m = (-1 + b - w[1]*x1)/w[2]
x20 = (b - w[1]*x1)/w[2]

#plot the svm results
setEPS()
postscript("fig0907.eps", height=7, width=7)
par(mar = c(4.5, 4.5, 2.0, 2.0))
plot(x, col = y + 9, pch = 19, cex =1.5,
     xlim = c(0, 10), ylim = c(0, 10),
     xlab = bquote(x[1]), ylab = bquote(x[2]),
     cex.lab = 1.5, cex.axis = 1.5,
     main = 'SVM␣application␣to␣many␣points␣of␣two␣labels')
axis(2, at = 0:10, tck = 1, lty = 3,
     col = "grey", labels = NA)
axis(1, at = 0:10, tck = 1, lty = 3,
     col = "grey", labels = NA)
lines(x1, x2p, lty = 2, col = 10)
lines(x1, x2m, lty = 2, col = 11)
lines(x1, x20, lwd = 1.5, col = 'purple')
thetasvm = atan(-w[1]/w[2])*180/pi
thetasvm
#[1] 36.8699 #36.9 deg angle of the hyperplane
#linear equations for the hyperplanes
delx = 1.4
dely = delx * (-w[1]/w[2])
text(5 + 2*delx, 6.5 + 2*dely, bquote(bold(w%.%x) - b == 0),
     srt = thetasvm, cex = 1.5, col = 'purple')
text(5 - delx, 7.6 - dely, bquote(bold(w%.%x) - b == 1),
     srt = thetasvm, cex = 1.5, col = 10)
text(5, 4.8, bquote(bold(w%.%x) - b == -1),
     srt = thetasvm, cex = 1.5, col = 11)
#normal direction of the hyperplanes
arrows(2, 3.86, 2 + w[1], 4 + w[2], lwd = 2,
       angle = 15, length= 0.1, col = 'blue' )
text(2 + w[1] + 0.4, 4 + w[2], bquote(bold(w)),
     srt = thetasvm, cex = 1.5, col = 'blue')

#new data points to be predicted
xnew = matrix(c(0.5, 2.5, 7, 2, 6, 9),
              ncol = 2, byrow = TRUE)
points(xnew, pch = 17, cex = 2)
predict(svmP, xnew) #Prediction
#1 2 3
#2 2 1

for(i in 1:3){
  text(xnew[i,1], xnew[i,2] - 0.4 ,
       paste('Q',i), cex = 1.5)
}
dev.off()
```

```
#Python for Fig. 9.7: SVM for a system of many points
# define x for plotting
x = np.array([1,6,2,8,3,7.5,1,8,
              4,9,5,9,3,7,5,9,1,5,
              5,3,6,4,7,4,8,6,9,5,
              10,6,5,0,6,5,8,2,2,2,1,1])
# define y
y = [1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2]
# redefine x for SVM
X = [[1,6],[2,8],[3,7.5],[1,8],
              [4,9],[5,9],[3,7],[5,9],[1,5],
              [5,3],[6,4],[7,4],[8,6],[9,5],
              [10,6],[5,0],[6,5],[8,2],[2,2],[1,1]]

svmP = svm.SVC(kernel='linear')
# train SVM
svmP.fit(X, y)

# predict new data using trained SVM
print('The␣support␣vectors␣are:␣',
      svmP.support_vectors_, '\n')

# find hyperplane, normal vector, and SV (wx + b = 0)
w = -svmP.coef_[0]
print('w␣is:␣',w, '\n')

# rho is the negative intercept in R
# intercept is the positive intercept in Python
b = svmP.intercept_
print('b␣is:␣',b, '\n')

# find the maximum margin of separation
print(2/np.linalg.norm(w))
x1 = np.linspace(0,10,31)
x2 = (b - w[0]*x1)/w[1]
x2p = (1 + b - w[0]*x1)/w[1]
x2m = (-1 + b - w[0]*x1)/w[1]
x20 = (b - w[0]*x1)/w[1]

thetasvm = math.atan(-w[0]/w[1])*180/np.pi
# degree angle of the hyperplane
print(thetasvm)

delx = 1.4
dely = delx * (-w[0]/w[1])

# new predicted data points
svmP.predict([[0.5,2.5],[7,2],[6,9]])
```

3664

```
#Python plot Fig. 9.7: SVM for a system of many points
#The plotting part of  Fig. 9.7
plt.figure(figsize = (10,10))#set up plot
plt.xlim(-0.2,10.2)
plt.ylim(-0.2,10.2)
plt.title("SVM␣application␣to␣many␣points␣of␣two␣labels",
          pad = 10)
plt.xlabel("$x_1$", labelpad = 10)
plt.ylabel("$x_2$", labelpad = 10)

# plot points
color = ['r','r','r','r','r','r','r','r','r','r','forestgreen',
         'forestgreen','forestgreen','forestgreen',
  'forestgreen','forestgreen','forestgreen','forestgreen',
         'forestgreen','forestgreen','forestgreen',]
plt.scatter(x[::2],x[1::2], color = color)
# plot newly predicted points
plt.scatter([0.5,7,6],[2.5,2,9], marker = "^",
            color = 'k', s = 90)
# plot lines
plt.plot(x1,x2p, color = 'red', linestyle = '--')
plt.plot(x1,x2m, color = 'forestgreen', linestyle = '--')
plt.plot(x1,x20,color = 'purple')

# add text
plt.text(5+2*delx,6.5+2*dely,'$w␣\cdot␣x␣-␣b␣=␣0$',
         color = 'purple', rotation = thetasvm, size = 15)
plt.text(5-delx, 7.6-dely, '$w␣\cdot␣x␣-␣b␣=␣1$',
         color = 'red', rotation = thetasvm, size = 15)
plt.text(5, 4.8,'$w␣\cdot␣x␣-␣b␣=␣-1$',
    color = 'forestgreen', rotation = thetasvm, size = 15)
plt.text(1.8,4.3,'w', color = 'blue', size = 15)
plt.text(0.3,2.1,'$Q_1$', color = 'k', size = 15)
plt.text(6.8,1.6,'$Q_2$', color = 'k', size = 15)
plt.text(5.8,8.6,'$Q_3$', color = 'k', size = 15)

# add normal direction of the hyperplanes
plt.arrow(2,3.86,w[0],w[1],color = 'blue', head_width = 0.1)
plt.show()
```

SVM maximizes $2/|\mathbf{w}|$, i.e., minimizes $|\mathbf{w}|$. The above computer code shows that $\mathbf{w} = (-0.39996, 0.53328)$. This optimization is reached at the three SVs $P_9(1,5), P_17(6,5)$ and $P_19(2,2)$. We can verify that the end points of these three SVs are on the two hyperplanes, by plugging in these coordinates into the following two linear equations

$$\mathbf{w} \cdot \mathbf{x} - b = \pm 1. \tag{3.45}$$

Reversely, these equations have three parameters w_1, w_2, b, which can be determined by three support vectors (SVs).

The linear SVM separation can have nonlinear extensions, such as circular or spherical hypersurfaces. Nonlinear SVM is apparently needed when the data cannot

be easily separated by linear hyperplanes. Both R and Python codes have function parameters for the nonlinear SVM. Another SVM extension is from two classes to many classes. The multi-class SVM and nonlinear SVM are beyond the scope of this book. The interested readers are referred to the designated machine learning books, such as those listed at the section of References and Further Readings at the end of this chapter.

3.3 Random forest method for classification and regression

Random forest (RF) is a popular machine learning algorithm and belongs to the class of supervised learning. It uses many decision trees trained from the given data, called the training data. The training data usually include categorical data, such as weather types (e.g., rainy, cloudy, and sunny) as labels, and numeric data, such as historical instrument observations (e.g., atmospheric pressure, humidity, wind speed, wind direction, and air temperature). These data form many logical decision trees that decide under what sets of conditions the weather is rainy, cloudy, or sunny. Many decision trees form a forest. Multiple decision nodes and branches for each tree are determined and trained by using a set of random sampling procedures. The random sampling and the decision trees lead to the name of random forest. The training step results in an RF model which are a set of decision trees. In the prediction step, you submit your new data to the the trained RF model whose each decision tree makes a vote for a category based on your new data. If the category "tomorrow is rainy" receives the most votes, then you have predicted a rainy day tomorrow. In this sense, RF is an ensemble learning method. Because of this nature of ensemble predictions, an RF algorithm should ensure the votes among the trained trees should have as little correlation as possible.

The above is a layman's description of RF for classification, when the RF objective is for categorical data, or called factor in data types. RF can also be used to fill in the missing numeric real values. The missing values, or missing data in climate science, are often denoted by `NA`, `NaN`, or `-99999`. RF makes a prediction for the missing data. The prediction is an ensemble mean from the trained RF trees. This is an RF regression, which produces numerical results.

3.3.1 RF flower classification for a benchmark iris dataset

We use the popular iris flower dataset (Fisher 1936) frequently used for the RF teaching as an example to explain the RF computing procedures and to interpret the RF results. Sir Ronald A. Fisher (1890 – 1962) was a British statistician and biologist. The Fisher dataset contains three iris species: *setosa, versicolor, virginica*. The data are the length and width of sepal and petal of a flower. The first two rows of the total 150 rows of the dataset are below.

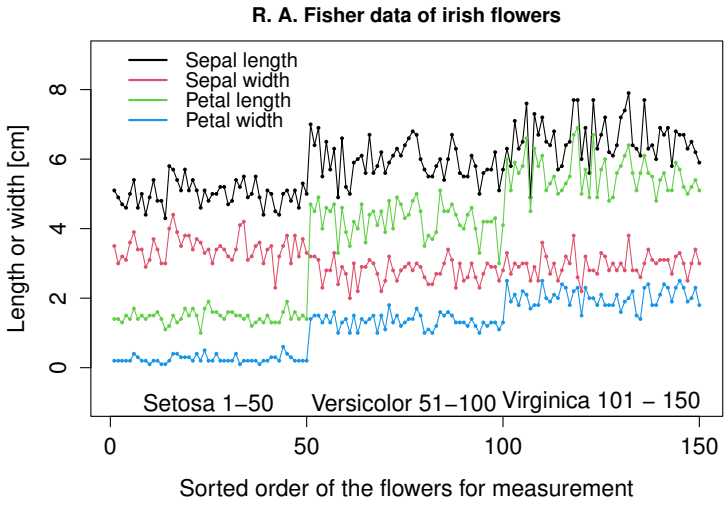


Fig. 3.8 The R. A. Fisher iris dataset of sepal length, sepal width, petal length, and petal width for flower species (Fisher 1936).

```
#    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#1            5.1         3.5          1.4         0.2  setosa
#2            4.9         3.0          1.4         0.2  setosa
```

Figure 3.8 shows the entire dataset.

Figure 3.8 may be generated by the following computer code.

```
#R plot Fig. 9.8: R.A. Fisher data of three iris species
setwd('~/climstats')
data(iris) #read the data already embedded in R
dim(iris)
#[1] 150   5
iris[1:2,] # Check the first two rows of the data
#    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#1            5.1         3.5          1.4         0.2  setosa
#2            4.9         3.0          1.4         0.2  setosa

str(iris) # Check the structure of the data
#'data.frame':  150 obs. of  5 variables:
#$ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
#$ Species     : Factor w/ 3 levels "setosa","versicolor",...

setEPS() #Plot the data of 150 observations
postscript("fig0908.eps",  width=7, height=5)
par(mar= c(4.5, 4.5, 2.5, 0.2))
plot(iris[,1], type = 'o', pch = 16, cex = 0.5, ylim = c(-1, 9),
     xlab = 'Sorted order of the flowers for measurement',
     ylab = 'Length or width [cm]',
     main = 'R. A. Fisher data of irish flowers', col = 1,
     cex.lab = 1.3, cex.axis = 1.3)
```

```
lines(iris[,2], type = 'o', pch = 16, cex = 0.5, col=2)
lines(iris[,3], type = 'o', pch = 16, cex = 0.5, col=3)
lines(iris[,4], type = 'o', pch = 16, cex = 0.5, col=4)
legend(0, 9.5, legend = c('Sepal length', 'Sepal width',
                          'Petal length', 'Petal width'),
       col = 1:4, lty = 1, lwd = 2, bty = 'n',
       y.intersp = 0.8, cex = 1.2)
text(25, -1, 'Setosa 1-50', cex = 1.3)
text(75, -1, 'Versicolor 51-100', cex = 1.3)
text(125, -1, 'Virginica 101 - 150', cex = 1.3)
dev.off()
```

```
# Python plot Fig. 9.8: R.A. Fisher data of iris species
iris = datasets.load_iris()
# notice "target" is equivalent to "species"
iris = pd.DataFrame(data=np.c_[iris['data'],iris['target']],
                columns= iris['feature_names'] + ['target'])

# check the structure of the data
print(iris.info(), '\n')

# check the first two rows of the data
iris.iloc[0:2,]

# set up plot
plt.title("R. A. Fisher data of iris flowers", pad = 10)
plt.xlabel("Sorted order of the flowers for measurement",
           labelpad = 10)
plt.ylabel("Length or width [cm]", labelpad = 10)
plt.xticks([0,50,100,150])
plt.ylim(-2,9)
plt.yticks([0,2,4,6,8])

# plot the data of 150 observations
x = np.linspace(0,149,150)
plt.plot(x,iris.iloc[:,0],'ko-', label = 'Sepal length')
plt.plot(x,iris.iloc[:,1],'ro-', label = 'Sepal width')
plt.plot(x,iris.iloc[:,2],'go-', label = 'Petal length')
plt.plot(x,iris.iloc[:,3],'bo-', label = 'Petal width')

# add legend
plt.legend()

# add text
plt.text(13,-1, "Setosa 1-50", size = 15)
plt.text(57,-1, "Versicolor 51-100", size = 15)
plt.text(107,-1, "Virginica 101-150", size = 15)

plt.show()
```

The first 50 are setosa, which are the smallest flower, and the last 50 for virginica that have the largest petals. Given the data of sepal and petal sizes, one may correctly detect their corresponding flower species. However, not all the flowers have

the same size. This makes the species detection more difficult. The random forest algorithm can make the detection with a small error. We make an RF experiment with this dataset. We randomly select 120 rows of data as the training data to obtained a trained RF model with 500 decision trees, use the remained 30 rows as the new data for our detection. These decision trees to vote on each of the 30 rows of the new data. The most species votes a row receives from the trained RF trees is the RF detected result. By default, the decision trees in an RF algorithm are built randomly, the RF result from each run based on the same data may be slightly different. The following computer code shows the RF run and its result of this experiment for Fisher's iris data. The code also generates Fig. 3.9.

```
#R code: RF prediction using the Fisher iris data
#install.packages("randomForest")
library(randomForest)
#set.seed(8)  # run this line to get the same result
#randomly select 120 observations as training data
train_id = sort(sample(1:150, 120, replace = FALSE))
train_data = iris[train_id, ]
dim(train_data)
#[1] 120   5
#use the remaining 30 as the new data for prediction
new_data = iris[-train_id, ]
dim(new_data)
#[1] 30  5

#train the RF model
classifyRF = randomForest(x = train_data[, 1:4],
                    y = train_data[, 5], ntree = 800)
classifyRF #output RF training result
#Type of random forest: classification
#Number of trees: 800
#No. of variables tried at each split: 2

#OOB estimate of  error rate: 4.17%
#Confusion matrix:
#               setosa versicolor virginica class.error
#setosa         41          0         0  0.00000000
#versicolor      0         34         2  0.05555556
#virginica       0          3        40  0.06976744

plot(classifyRF) #plot the errors vs RF trees Fig. 9.9a

classifyRF$importance #classifyRF$ has many outputs
#             MeanDecreaseGini
#Sepal.Length         8.313131
#Sepal.Width          1.507188
#Petal.Length        31.075960
#Petal.Width         38.169763
varImpPlot(classifyRF) #plot the importance result Fig. 9.9b

#RF prediction for the new data based on the trained trees
predict(classifyRF, newdata = new_data[,1:4])
# 2          4         10         11         12         14
#setosa     setosa     setosa     setosa     setosa     setosa
```

```
#It got two wrong: 120 versicolor and 135 versicolor

#Another version of the randomForest() command
anotherRF = randomForest(Species ~ .,
                         data = train_data, ntree = 500)
```

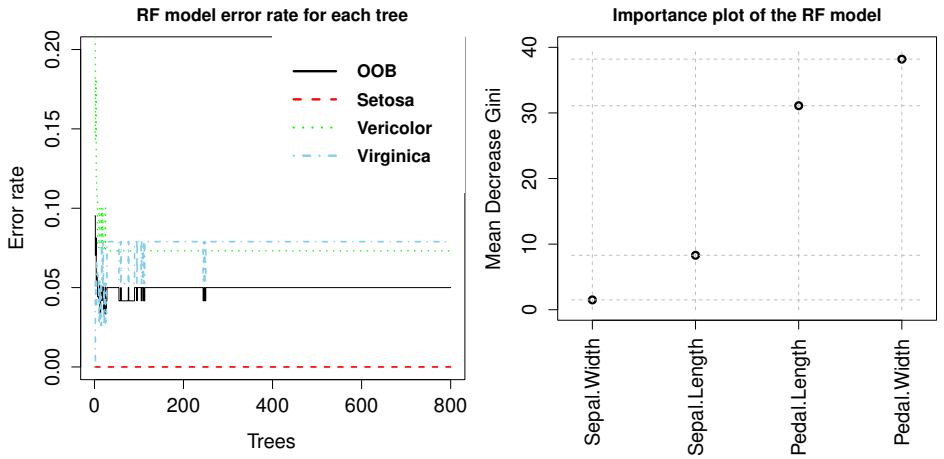
```
#Python code: RF prediction with the Fisher iris data
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import metrics
# randomly select 120 observations as training data
train_id = random.sample(range(0, 150), 120)
train_data = iris.iloc[train_id,:]
print(train_data.shape)# dimensions of training data
# use the remaining 30 as the new data for prediction
new_data = iris.drop(train_id)
print(new_data.shape)# dimensions of new data
# train the RF model
classifyRF = RandomForestClassifier(n_estimators=800,
                                    oob_score=True)
training = classifyRF.fit(train_data.iloc[:,:4],
                  train_data.iloc[:,4])
predictions = classifyRF.predict(new_data.iloc[:,:4])
print(predictions)
#[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 2.
#1. 2. 1. 1. 2. 1. 1. 2. 2. 2. 2.
# 2. 2. 2. 2. 2. 2.]
#0 = setosa, 1 = versicolor, 2 = virginica
confusion_M = confusion_matrix(new_data.iloc[:,4],
                               predictions)
print(confusion_M)
#[[ 9  0  0] #all correct predictions for setosa
# [ 0  8  3]  #3 wrong predictions for versicolor
# [ 0  0 10]] #all correct predictions for virginica
accuracy_score(new_data.iloc[:,4],
                               predictions)
#0.9
```

```
#Python lot Fig. 9.9(a): RF mean accuracy
n_estimators = 100
classifyRF = RandomForestClassifier(n_estimators,
                                    oob_score=True)
RFscore = []
for i in range(1, n_estimators + 1):
    classifyRF.set_params(n_estimators=i)
    classifyRF.fit(train_data.iloc[:,:4],
                   train_data.iloc[:,4])
    RFscore.append(classifyRF.score(train_data.iloc[:,:4],
                   train_data.iloc[:,4]))
plt.plot(RFscore)
plt.title("Random␣forest␣score␣of␣the␣mean␣accuracy")
plt.xlabel("Number␣of␣estimators")
plt.ylabel("OOB␣score")
plt.show()
```

```
#Python plot Fig. 9.9(b): RF importance plot
# Python shows the importance results differently from R
plt.plot(np.linspace(1,4,4),
         classifyRF.feature_importances_,
         'ko', markersize = 15)
plt.title("Importance␣plot␣of␣RF␣model", pad = 10)
plt.ylabel("Mean␣decrease␣in␣impurity", labelpad = 12)
plt.xticks([1,2,3,4], ['Sepal.Length','Sepal.Width',
     'Petal.Length', 'Petal.Width'], rotation = 90)
plt.yticks(classifyRF.feature_importances_,
           ['0.01','0.11','0.39','0.48'])
plt.grid()
plt.show()
```

The following explains the RF output data and figures.

- Confusion matrix: This matrix displays correct and wrong classifications based on the training data. The columns are truth and rows are RF classifications. The diagonals are corrected classifications, and off diagonals are the wrong ones. The 41 setosas are all correctly classified. Among the 37 versicolors, 34 are correctly classified, but 3 are wrongly classified as virginica. Among the 37 versicolors, 34 are correctly classified, but 3 are wrongly classified as virginica.
- Classification error: The last column in the confusion matrix is the classification error which is equal to the sum of the off-diagonal elements in that row divided by the sum of the entire row, i.e., the ratio of the number of wrong classifications to the total number of classifications. The errors shows how good is the trained RF model.
- OOB estimate of error rate: OOB stands for out-of-bag, a term for a statistical sampling process that puts a part of the data for training the RF model and the remainder for validation. This remainder is referred to as the OOB set



(a) (b)

Fig. 3.9 (a) RF errors vs. trees. (b) The RF's importance plot.

of data. The OOB error rate is defined as the ratio of the number of wrong decisions based on the majority vote to the size of OOB set.

- No. of variables tried at each split: This is the number of variables randomly sampled for growing trees, and is denoted by `mtry`. The Fisher iris data have four variables ($p = 4$). It is recommended that `mtry` $= \sqrt{p}$ for RF classification, and $= p/3$ for RF regression.
- Number of trees: We want to build enough trees for RF so that the RF errors shown in Fig. 3.9(a)is stabilized. Too few trees may yield results of large differences at different runs.
- RF errors vs. trees: The colored lines show errors of different types of validation procedures. As RF users, we pay attention to whether these errors are stabilized. The detailed error definitions are beyond the scope of this chapter.
- Mean decrease Gini scores and the importance plot (Fig. 3.9(b)): A higher mean decrease Gini (MDG) score corresponds to more importance of the variable in the model. In our example, the petal width is the most important variable. This agrees with our intuition from Fig. 3.8. The figure shows that the petal width has clear distinctions among the three species and have little variance. The petal length also has clear distinctions, but has larger variances, and is the second most important variable. The sepal width has almost no distinctions among the species and has the smallest MDG score 1.5. It is the least important variable.
- RF prediction: Finally, the RF predictions were made for the 30 rows of new data. The prediction results for the first six rows of the new data are shown here. These are the final results. Among the 30 rows, RF correctly predicted 28 and got only two wrong: 120 and 135 should be virginica, but RF predicted versicolor for both. In weather forecast, these can be the public weather outlook

of the 9th day from today (e.g., sunny, rainy, or cloudy) for 30 different locations in a country.

3.3.2 RF regression for the daily ozone data of New York City

Similar to Fisher's iris data, the daily air quality data, the file name `airquality`, of New York City (NYC) from 1 May 1973 - 30 September 1973 (153 days) is another benchmark data for the random forest algorithms. The dataset contains the following weather parameters: Ozone concentration in the ground-level air, cumulative solar radiation, average wind speed, and daily maximum air temperature. When the ozone concentration is higher than 86 parts per billion (ppb), the air is considered unhealthy. The ozone concentration is related to solar radiation, wind and temperature. The following list provides more information on this `airquality` dataset:

- The ozone data in ppb observed between 1300 and 1500 hours at Roosevelt Island of the New York City. Among the 153 days, 37 had no data and are denoted by NA. The data range is 1 - 168 ppb. The maximum ozone level of this dataset is 168 ppb, occurred on 25 August 1973, when the cumulative solar radiation was high at 238 Lang, the average wind speed low at 3.4 mph, and Tmax is moderate 81°F.
- Cumulative solar radiation from 0800 to 1200 hours with units in Langley (Lang or Ly) (1 Langley = 41, 868 Watt·sec/m^2, or 1 Watt/m^2 = 0.085985 Langley/hour) in the lightwave length range 4,000 - 7,700 Angstroms at Central Park of the New York City. The data range is 7 - 334 Lang.
- Average wind speed in miles per hour (mph) at 0700 and 1000 hours at LaGuardia Airport, less than 10 km from Central Park. The data range is 1.7 - 20.7 mph.
- Maximum daily temperature Tmax in °F at La Guardia Airport. The data range is 56 - 97 °F. The daily Tmax data can also be downloaded from the NOAA NCEI website by an online search using the words: `NOAA Climate Data Online LaGuardia Airport`.

While RF analysis for this `airquality` dataset can be for many purposes, our example is to fill the 37 missing ozone data with RF regression results. RF use the above four parameters and their data to build decision trees. Figure 3.10(a) shows the original 116 observed data points in black dots and lines, and the RF regression estimates for the 37 missing data (the blue dots and lines). A single blue dot means only an isolated day of missing data. A blue line indicates missing data in successive days. For better visualization, Fig. 3.10(b) shows the complete NYC daily ozone time series that join the observed data with the result data from an RF regression.

Figure 3.10 may be generated by the following computer code.

```
#R plot Fig. 9.10: RF regression for ozone data
library(randomForest)
airquality[1:2,] #use R's RF benchmark data "airquality"
#  Ozone Solar.R Wind Temp Month Day
```

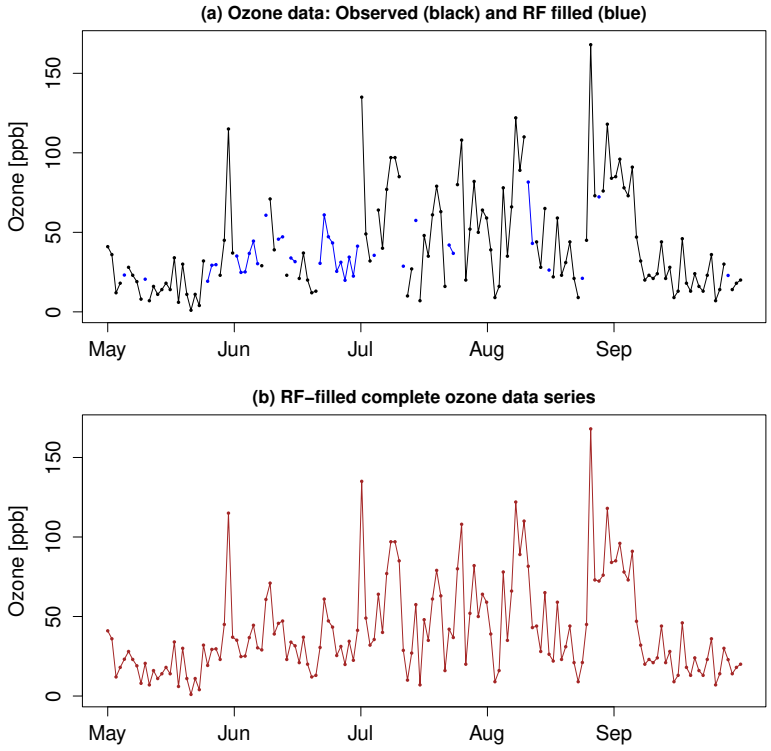


Fig. 3.10 (a) New York City ozone data from 1 May to 30 September 1973. Among the 153 days, 116 had observed data (black dots and lines) and 37 had missing data. The missing data are filled by RF regression results (blue dots and lines). (b) The complete ozone data time series as a continuous curve with dots when the missing data are replaced by the RF regression.

```
#1     41      190   7.4    67      5    1
#2     36      118   8.0    72      5    2
dim(airquality)
#[1] 153   6
ozoneRFreg = randomForest(Ozone ~ ., data = airquality,
               mtry = 2, ntree = 500, importance = TRUE,
               na.action = na.roughfix)
#na.roughfix allows NA to be replaced by medians
#to begin with when training the RF trees
ozonePred = ozoneRFreg$predicted #RF regression result
t0 = 1:153
n1 = which(airquality$Ozone > 0) #positions of data
n0 = t0[-n1] #positions of missing data
ozone_complete = ozone_filled= airquality$Ozone
ozone_complete[n0] = ozonePred[n0] #filled by RF
ozone_filled = ozonePred #contains the RF reg result
ozone_filled[n1] <- NA #replace the n1 positions by NA
```

```
t1 = seq(5, 10, len = 153) #determine the time May - Sept

par(mfrow = c(2, 1))
par(mar = c(3, 4.5, 2, 0.1))
plot(t1, airquality$Ozone,
     type = 'o', pch = 16, cex = 0.5, ylim = c(0, 170),
     xlab = '', ylab = 'Ozone [ppb]', xaxt="n",
 main = '(a) Ozone data: Observed (black) and RF filled (blue)',
     col = 1, cex.lab = 1.3, cex.axis = 1.3)
MaySept = c("May","Jun", "Jul", "Aug", "Sep")
axis(side=1, at=5:9, labels = MaySept, cex.axis = 1.3)
points(t1, ozone_filled, col = 'blue',
       type = 'o', pch = 16, cex = 0.5)#RF filled data

#Plot the complete data
par(mar = c(3, 4.5, 2, 0.1))
plot(t1, ozone_complete,
     type = 'o', pch = 16, cex = 0.5, ylim = c(0, 170),
     xlab = '', ylab = 'Ozone [ppb]',
     xaxt="n", col = 'brown',
     main = '(b) RF-filled complete ozone data series',
     cex.lab = 1.3, cex.axis = 1.3)
MaySept = c("May","Jun", "Jul", "Aug", "Sep")
axis(side=1, at=5:9, labels = MaySept, cex.axis = 1.3)
```

```
# Python plot Fig. 9.10: RF regression for ozone data
from sklearn.ensemble import RandomForestRegressor
# Read in the airquality data and create a copy
airquality = pd.read_csv("data/airquality.csv",
                         index_col = 'Unnamed: 0')
airquality_copy = airquality.copy()
print(airquality.head())#print the first 5 rows of data
print(np.shape(airquality))#data matrix dim (153, 6)
# Create list of integer from 1 to 153
t0 = list(np.linspace(1,153,153, dtype=int))
# positions of recorded data
n1 = np.where(airquality["Ozone"] > 0)
n1 = n1[0].tolist()
n1 = [x+1 for x in n1]
# positions of unknown 'NaN' data
n0=[]
for x in t0:
    if x not in n1:
        n0.append(x)
# Replace NA with medians in order to train the RF trees
airquality['Solar.R'].fillna(
    value=airquality['Solar.R'].median(), inplace=True)
airquality['Wind'].fillna(
    value=airquality['Wind'].median(), inplace=True)
airquality['Temp'].fillna(
    value=airquality['Temp'].median(), inplace=True)
airquality['Day'].fillna(
    value=airquality['Day'].median(), inplace=True)
airquality['Month'].fillna(
    value=airquality['Month'].median(), inplace=True)
airquality['Ozone'].fillna(
    value=airquality['Ozone'].median(), inplace=True)
# Create or features X and our target y
X = airquality[['Solar.R', 'Wind', 'Temp', 'Month', 'Day']]
y = airquality[['Ozone']]
# split our data into training and test sets
X_train = X.loc[n1,:]
X_test = X.loc[n0,:]
y_train = y.loc[n1,:]
y_test = y.loc[n0,:]
#create the RF Regressor model with 500 esitmators
ozoneRFreg = RandomForestRegressor(n_estimators=500,
                   oob_score=True, max_features = 2)
# fit model to our training set
ozoneRFreg.fit(X_train, y_train.values.ravel())
#create our prediction
ozonePrediction = ozoneRFreg.predict(X_test)
#create data frame with our results
result = X_test
result['Ozone'] = y_test
result['Prediction'] = ozonePrediction.tolist()
# create data frame with our predictions
ozone_filled = pd.DataFrame(None, index = np.arange(153),
                            columns = ["Prediction"] )
ozone_filled.loc[n0, 'Prediction'] = result["Prediction"]
ozone_filled.index += 1
```

```
#Python plot Fig. 9.10(a)
t1 = np.linspace(5,10,153)
mfrow = [2,1]
mar = [3, 4.5, 2, 0.1]
# original known data
ar = pd.read_csv("data/airquality.csv",
                          index_col = 'Unnamed: 0')
plt.plot(t1, ar["Ozone"],
         'ko-', markersize = 3)
plt.ylim(0, 170)
#unknown Predicted data
plt.plot(t1, ozone_filled['Prediction'],
         'bo-', markersize = 3)
plt.ylabel("Ozone [ppb]", labelpad = 12)
MaySept = ["May","Jun", "Jul", "Aug", "Sep"]
plt.xticks([5,6,7,8,9], MaySept)
plt.title("(a) Observed (black) and RF filled (blue)")
plt.show()
```

```
#Python plot Fig. 9.10(b)
# combine unknown and known data into one dataframe
result_copy = result.copy()
result_copy = result_copy.rename(
    columns = {'Ozone':'y_tested', 'Prediction':'Ozone'})
ozone_complete = airquality_copy[
    "Ozone"].fillna(result_copy["Ozone"])
#plot the combined data
t1 = np.linspace(5,10,153)
mfrow = [2,1]
mar = [3, 4.5, 2, 0.1]
plt.plot(t1, ozone_complete, 'ro-', markersize = 3)
plt.ylim(0, 170)
plt.ylabel("Ozone [ppb]", labelpad = 12)
MaySept = ["May","Jun", "Jul", "Aug", "Sep"]
plt.xticks([5,6,7,8,9], MaySept)
plt.title("(b) RF-filled complete ozone data series")
plt.show()
```

3.3.3 What does a decision tree look like?

We have learned that the RF prediction is the result of majority votes from the trained decision trees in a random forest. Then, what does a decision tree look like? Figure 3.11 shows a decision tree in the RF computing process for the training data of iris flowers. The gray box on top shows the percentage of each species in the 120 rows of training data. This particular set of training data was randomly selected from the entire R.A. Fisher dataset which has 150 rows. Among the 120 rows of training data, 41 rows are setosa (34%), 42 rows versicolor (35%), and 37

rows (31%) virginica. The first branch of the decision tree grows from a condition

$$\text{Petal Length} < 2.5 \text{ [cm]}. \tag{3.46}$$

If "yes", this iris is setosa. All the 41 setosa flowers (34% of the entire training data) have been detected. This decision is clearly supported by the real petal length data (the green line) shown in Fig. 3.8, which shows that only setosa has petal length less than 2.5.

If "no", then two possibilities exist. This allows us to grow a new branch of the tree. The condition for this branch is

$$\text{Petal Width} < 1.8 \text{ [cm]}. \tag{3.47}$$

This condition determines whether a flower is versicolor or virginica. After the isolation of 41 setosa flowers, the remaining 79 flowers are 42 versicolor (53%) and 37 virginica (47%). The "yes" result of condition (3.47) leads to versicolor. This result is 93% correct and 7% incorrect. The "no" result implies virginica. This conclusion is 97% correct, and 3% incorrect. These conclusions are supported by the real petal width data (the blue line) shown in Fig. 3.8. The petal width of versicolor iris flowers is in general less than 1.8 [cm]. However, the data have some fluctuations in both the versicolor and virginica sections in Fig. 3.8. These fluctuations lead to the errors of decision.

The entire RF process for our iris specicies example had grown 800 such decision trees using the 120 rows of training data. Different trees use different branch conditions. RF algorithms grow these trees following various kinds of optimization principles, which involve some tedious mathematics not covered here.

Figure 3.11 may be generated by the following computer code.

```
#R plot of Fig. 9.11: A tree in a random forest
#install.packages('rpart')
library(rpart)
#install.packages('rpart.plot')
library(rpart.plot)

setwd('~/climstats')
setEPS() #Plot the data of 150 observations
postscript("fig0911.eps",  width=6, height=6)
par(mar = c(0, 2, 1, 1))
iris_tree = rpart( Species ~. , data = train_data)
rpart.plot(iris_tree,
   main = 'An␣decision␣tree␣for␣the␣RF␣training␣data')
dev.off()
```

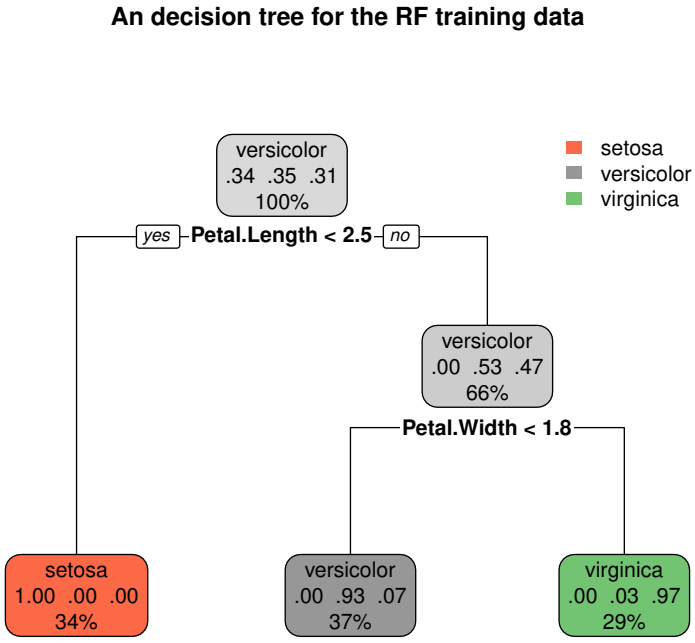


Fig. 3.11 A trained tree in the random forest for the iris data of R.A. Fisher (1936).

```
# Python plot Fig. 9.11: A tree in a random forest
classifyRF = RandomForestClassifier(n_estimators=1,
                                    oob_score=True)
training = classifyRF.fit(train_data.iloc[:,:4],
              train_data.iloc[:,4])
features = ['sepal␣length␣(cm)', 'sepal␣width␣(cm)',
            'petal␣length(cm)', 'petal␣width␣(cm)']
class_name = ['setosa', 'versicolor', 'virginica']
# set up figure
plt.figure(figsize = (15,15))
plot_tree(classifyRF.estimators_[0],
          feature_names = features,
          class_names = class_name, filled = True,
          fontsize = 12)
plt.show()
```

3.4 Neural network and deep learning

A neural network (NN) consists of a series of data fitting based on the training data and an application of the fitted NN model for test data. The NN outputs categorical predictions, such as a sunny or rainy day, or numerical predictions as a regression. NN is also known as an artificial neural network (ANN) or simulated neural network (SNN). It is a popular machine learning method, and is a fundamental building block the deep learning algorithms that often involve the data fitting of multiple layers, referred to as hidden layers.

Why is it called neural network? How does a data fitting process have anything to do with "neural" and/or "network"? Artificial neurons were first proposed by Warren Sturgis McCulloch (1898 - 1969), an American neurophysiologist, and Walter Harry Pitts (1923 - 1969), an American logician, in their 1943 paper entitled " A logical calculus of ideas immanent in nervous activity." This mathematical paper used terms "neuron", "action", "logic expression", and "net", which are among the keywords in the modern NN writings. The ten theorems of the paper formulate a suite of logic expressions based on data. The word "neuron" was more a graphic indication for actions and logic expressions than a biological reference. A layman's understanding of an NN machine learning is often put incorrectly toward biological neurons and a biological neural network. Therefore, ANN may be a more appropriate term for NN to avoid confusion.

This book attempts to make such a brief NN introduction that you can use our R or Python code for your data and objectives, interpret your NN computing results, and understand the basic principles of mathematical formulations of an NN algorithm. However, we do not attempt to derive mathematical details of the NN theory. This section has two sub-sections: (i) A simple NN example of a decision system, and (ii) An example of NN prediction using the benchmark data of Fisher's iris flowers.

3.4.1 An NN model for an automized decision system

3.4.1.1 An overall idea of a neural network decision system for recruitment

A senior human resource manager of an IT company recruited three new employees among six candidates. Her hiring decisions were made based on the technical knowledge scores (TCS) and communication skills scores (CSS) given in Table 9.1. TCS and CSS were the interview results from company's working groups. A junior human resource manager wishes to use an NN model to help him make his recruitment decisions consistent with those made by the senior manager.

The junior manager has received the following scores for the three new job applicants A, B, and C, whose TKS and CSS scores are as follows: TKS (30, 51, 72), and CSS (85, 51, 30). The senior manager's ruling seems to suggest that a candidate is

Table 9.1 TCS and CSS data and recruitment decisions

TKS	20	10	30	20	80	30
CSS	90	20	40	50	50	80
Decision	Hire	Reject	Reject	Reject	Hire	Hire

recruited when either TKS score or CSS score is high. Thus, this junior manager has an easy decision for Candidate A whose CSS 85 is high, so to be recruited. However, the decision for Candidate B is difficult. Neither TKS 51 nor CSS 51 of Candidate B is high, but both are higher than those of the rejected, and further the sum of TKS and CSS is 102, lower than the minimum of the totals of the recruited, 110. Should the junior manager hire Candidate B? The decision for Candidate C is also difficult. Candidate C seems weaker than the recruited case TKS 30 and CSS 80, but not too much weaker. Should the junior manager hire Candidate C? The following NN computer code may learn from the data of the senior manager and suggest an NN decision that may serve as a useful reference for the junior manager.

3.4.1.2 An NN code, results, and their interpretation

```
#R code for NN recruitment decision and Fig. 9.12
#Ref: https://www.datacamp.com/tutorial/neural-network-models-r
TKS = c(20,10,30,20,80,30)
CSS = c(90,20,40,50,50,80)
Recruited = c(1,0,0,0,1,1)
#make a data frame for the NN function neuralnet
df = data.frame(TKS, CSS, Recruited)

require(neuralnet) # load 'neuralnet' library
# fit neural network
set.seed(123)
nn = neuralnet(Recruited ~ TKS + CSS, data = df,
               hidden = 5,  act.fct = "logistic",
               linear.output = FALSE)
plot(nn) #Plot Fig 9.12: A neural network

TKS=c(30,51,72) #new data for decision
CSS=c(85,51,30) #new data for decision
test=data.frame(TKS,CSS)
Predict=neuralnet::compute(nn,test)
Predict$net.result #the result is probability
#[1,] 0.99014936
#[2,] 0.58160633
#[3,] 0.01309036

# Converting probabilities into decisions
ifelse(Predict$net.result > 0.5, 1, 0) #threshold = 0.5
#[1,]    1
#[2,]    1
#[3,]    0

#print bias and weights
nn$weights[[1]][[1]]
```

```
#print the last bias and weights before decision
nn$weights[[1]][[2]]
#print the random start bias and weights
nn$startweights
#print error and other technical indices of the nn run
nn$result.matrix #results data
```

```
#Python code for the NN recruitment decision and Fig. 9.12
#imports to build neural network model and visualize
#You need to install the following Python environments:
#tensorflow and keras-models
#!pip3 install ann_visualizer
#!pip3 install keras-models
#from a terminal: pip install tensorflow
from ann_visualizer.visualize import ann_viz
#from graphviz import Source
#from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense

TKS = [20,10,30,20,80,30]
CSS = [90,20,40,50,50,80]
Recruited = [1,0,0,0,1,1]
#combine multiple columns into a single set
df = pd.DataFrame({'TKS': TKS,'CSS': CSS,
     'Recruited': Recruited})
df.head()
X = df[["TKS", "CSS"]]
y = df[["Recruited"]]
#random.seed(123).
#The model is random due to too little training data

nn = Sequential()
nn.add(Dense(5, input_dim = 2, activation = "sigmoid"))
nn.add(Dense(1,activation = "sigmoid"))
#nn.add(Dense(2,activation = "sigmoid"))
nn.compile(optimizer = "adam", loss = "BinaryCrossentropy",
           metrics = "BinaryAccuracy")
#fit neural network to data
nn.fit(X,y)
#visualize the neural network
ann_viz(nn, title = "Recruitment Decision")
#output the nn model weights and biases
print(nn.get_weights())

TKS = [30,51,72] #new data for decision
CSS = [85,51,30] #new data for decision
test = pd.DataFrame({'TKS': TKS,'CSS': CSS})
prediction = nn.predict(test)
print(prediction)
#[[0.70047873]
# [0.66671777]
# [0.38699177]]
#Converting probabilities into decisions
for i in range(3):
  if prediction[i] >= 0.5:
    print("Hire")
  else:
    print("Reject")
#Hire
#Hire
#Reject
```

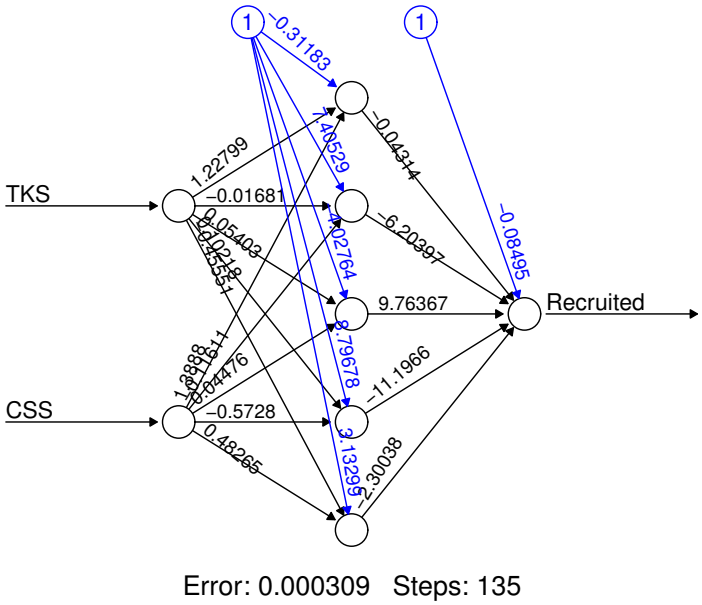


Fig. 3.12 A simple neural network of five neurons in a hidden layer for an NN hiring decision system.

Figure 3.12 shows a simple neural network plotted by the above computer code. The black numbers are called weights w_{ij} that are multiplied by the input data x_i for neuron j. The blue numbers are called biases b_j, associated with neurons j, indicated by the circles pointed by a blue arrow. The last circle on the right is the result, or called output layer. The first two circles on the left indicate input data, and form the input layer. The weights and biases are mathematically aggregated in the following way:

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j, \tag{3.48}$$

where z_j are for fitting an activation function at neuron j when all the training data are used. In the above code, the activation function is logistic. A logistic function is defined as

$$g(z) = \frac{1}{1 + \exp(-k(z - z_0))}, \tag{3.49}$$

where k is called the logistic growth rate, and z_0 is the midpoint. This function can also be written as

$$g(z) = \frac{1}{2} + \frac{1}{2} \tanh\left(-\frac{k(z - z_0)}{2}\right), \tag{3.50}$$

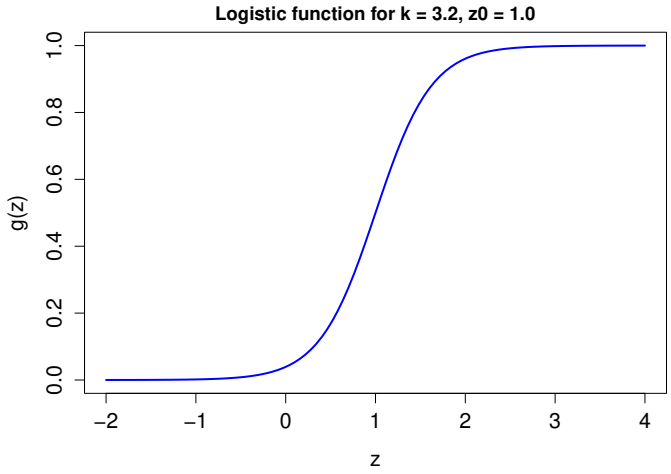


Fig. 3.13 Logistic function with growth rate $k = 3.2$ and midpoint $y_0 = 1.0$.

where the tanh function is defined as

$$\tanh(t) = \frac{\exp(t) - \exp(-t)}{\exp(t) + \exp(-t)}. \tag{3.51}$$

A curve for a logistic function is shown in Fig. 3.13. The curve has a property that $f(-\infty) = 0$ and $f(\infty) = 1$. This property makes the logistic activation function useful for categorical assignment: False for 0 and True for 1.

Figure 3.13 can be plotted by the following computer code.

```
#R plot Fig. 9.13: Curve of a logistic function
y = seq(-2, 4, len = 101)
k = 3.2
y0 = 1
setEPS() #Automatically saves the .eps file
postscript("fig0913.eps", height=5, width=7)
par(mar = c(4.2, 4.2, 2, 0.5))
plot(y, 1/(1 + exp(-k*(y - y0))),
     type = 'l', col = 'blue', lwd =2,
     xlab = 'y', ylab = 'f(y)',
     main = 'Logistic function for k = 3.2, y0 = 1.0',
     cex.lab = 1.3, cex.axis = 1.3)
dev.off()
```

```
#Python plot Fig. 9.13: Curve of a logistic function
# define variables
y = np.linspace(-2,4,101)
k = 3.2
y0 = 1

# plot figure
plt.plot(y, 1/(1+ np.exp(-k*(y-y0))), 'b', linewidth = 2.5)
# add labels
plt.title('Logistic function for k = 3.2, z0 = 1.0',
          pad = 10)
plt.xlabel("z", labelpad = 10)
plt.ylabel("g(z)", labelpad = 10)
plt.show()
```

When the growth rate $k = 1$ and midpoint $y_0 = 0$, the logistic function is often referred to as a sigmoid function. Some Python or R code uses sigmoid to represent the activation function.

The idea of data-based decision process is that you integrate all the data, develop a model through data fitting with a specified optimization, and apply the model to new data to generate predictions. The neural network integrates data by assigning each datum a weight as Eq. (3.48) and assigning each weighted sum a bias. The weighted data z_j and the decision data, 0 or 1, from the training dataset form a series of data pairs (z, d). These data pairs are used for the logistic data fitting many times to train a neural network model. Each time, the NN algorithm will try to optimize its data fitting, such as minimizing mean square errors (MSE) of the fitting. This optimization updates the weights and biases. This optimization process is called back-propagation in an NN algorithm. This makes the NN learning an optimization process. When the fitting errors do not change much, the optimization process has converged, and a logistic function model has been trained. You can now plot the trained neural network, denoted by `nn` in the above code, and produce Fig. 3.12.

The new data are aggregated together as a data frame denoted by `test` in the above code. You apply the trained `nn` model to `test` and obtain the NN prediction result as probabilities, a higher probability suggesting a positive decision, i.e., recruited. You may use 0.5 as your hiring probability threshold and convert the probability result into a decision result, indicated by 1(recruit) or 0 (reject). In this example, Candidates A and B are hired, and Candidate C is rejected, suggested by this NN model.

When running the computer code for this example of hiring data, you may have noticed that each run has a different prediction result. This makes the prediction unreliable. To improve the situation, we need more training data and more optimizations by using several layers of neurons. The approach of multiple hidden layers is a type of deep learning. It is intuitive why more training data are needed to train a more reliable model, since few data cannot cover all possibilities. In the next sub-section, we will present an example of the NN deep learning using more

than six groups of training data. Specifically, we will use 75 groups of the Fisher iris flower data.

3.4.1.3 Dissect NN results and errors

Logistic regression fits weighted data to a logistic function. Logistic regression can be a standalone statistics course or a chapter in a statistics book. Interested readers are referred to those books and courses.

In addition, there are other types of activation functions, such as linear, rectified linear, leaky rectified linear, exponential linear, and SoftPlus. You can find these functions from the specialized ML books or online.

Usually a software package of neural network can output the weights and bias, such as using the following R command

```
nn$weights[[1]][[1]]}
```

in the R NN package `neuralnet`. R can also output the initial assignment of weights and bias

```
nn$weights[[1]][[2]]}
```

The initial weights and biases are randomly assigned, which makes the NN result different for different runs when having insufficient amount of data, or insufficient number of hidden layers.

An NN computer package can also output many other modeling results, such as an R command

```
nn$result.matrix
```

These output data can be used to analyze the quality of your trained model. So, it is unfair to say that NN is a blackbox. Yet, analyzing a trained model for a complex NN is very difficult and requires some nontrivial mathematical preparation.

Most NN users may not have the mathematical ability to analyze the trained model based on mathematical theories. Instead, they apply the model to the new data and see if the NN prediction makes sense based on their experience or common sense. If not, train the model again with different parameters, or even feed more training data.

3.4.2 An NN prediction of iris species

We wish to use the Fisher iris data to show an example of NN with a sufficiently large size of training data and a highly reliable result. The Fisher iris data is a

150×5 matrix with three iris species: 50 setosa, 50 virginica, and 50 versicolor, as described in the random forest section (Fisher 1936). We wish to use a percentage of the data as the training data and the rest as the test data. The following computer code is for 50% of the data to be the training data, i.e., $p = 0.5$ in the code. We choose 10 neurons, and 10 hidden layers. The NN prediction result shows that NN correctly predicted all the 27 setosa irises, got 19 correct among the 20 versivolor irises but got one versicolor incorrectly as virginica, and got 26 correct among the 28 virginica irises but got 2 incorrectly as versicolor. Since the 75 groups of training data are plenty, compared to the 6 groups of training data for the hiring decision, the the trained NN model for the iris flowers does not vary much for different runs. The prediction results are quite reliable. Using more neurons and hidden layers may also help achieve reliable results. Although the results from different runs of the NN code do not change much, small differences still exist. For example, you may get a result of predicting 27 virginica.

When you change $p = 0.8$, you will have 120 rows of training data and 30 rows of test data. This is the same as what we used in the random forest example section. This increase of training data size helps further stabilize the prediction results, i.e., the results from different runs have little differences. You can make some runs of the code on your own computer, and see how many incorrect predictions are there. Our example runs shown below show only one or two incorrect predictions among the 30 rows of test data. For example, in one run, the 9 setosa and 12 versicolor irises were correctly predicted, and the 9 virginica irises had 8 correct predictions and one wrong. The level of the NN prediction accuracy seems similar to that of the random forest prediction for this particular dataset.

```
#R NN code for the Fisher iris flower data
#Ref: https://rpubs.com/vitorhs/iris
data(iris) #150-by-5 iris data
#attach True or False columns to iris data
iris$setosa = iris$Species == "setosa"
iris$virginica = iris$Species == "virginica"
iris$versicolor = iris$Species == "versicolor"
p = 0.5 # assign 50% of data for training
train.idx = sample(x = nrow(iris), size = p*nrow(iris))
train = iris[train.idx,] #determine the training data
test = iris[-train.idx,] #determine the test data
dim(train) #check the train data dimension
#[1] 75  8

#training a neural network
library(neuralnet)
#use the length, width, True and False data for training
iris.nn = neuralnet(setosa + versicolor + virginica ~
                      Sepal.Length + Sepal.Width +
                      Petal.Length + Petal.Width,
                    data = train, hidden=c(10, 10),
                    rep = 5, err.fct = "ce",
                    linear.output = F, lifesign = "minimal",
                    stepmax = 1000000, threshold = 0.001)
```

```
plot(iris.nn, rep="best") #plot the neural network

#Prediction for the rest data
prediction = neuralnet::compute(iris.nn, test[,1:4])
#prediction$net.result is 75-by-3 matrix
prediction$net.result[1:2,]
#  [,1]          [,2]          [,3]
#2    1 3.531638e-09 6.745692e-137
#4    1 3.527872e-09 6.756521e-137

#find which column is for the max of each row
pred.idx <- apply(prediction$net.result, 1, which.max)
pred.idx
#2  4  6  8  9 10
#1  1  1  1  1  1

#Assign 1 for setosa, 2 for versicolor, 3 for virginica
predicted <- c('setosa', 'versicolor', 'virginica')[pred.idx]
predicted[1:6] #The prediction result
#[1] "setosa" "setosa" "setosa" "setosa" "setosa" "setosa"

#Create confusion matrix: table(prediction,observation)
table(predicted, test$Species)
#predicted    setosa versicolor virginica
#setosa         27          0         0
#versicolor      0         19         2
#virginica       0          1        26
```

```
#Python NN code for the iris flower prediction
from ann_visualizer.visualize import ann_viz
from graphviz import Source
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split

iris = pd.read_csv("data/iris.csv") #150-by-5 iris data
#attach True or False columns to iris data
iris['setosa'] = iris['class'].map(lambda x:
                                x == "Iris-setosa")
iris['versicolor'] = iris['class'].map(lambda x:
                                x == "Iris-versicolor")
iris['virginica'] = iris['class'].map(lambda x:
                                x == "Iris-virginica")

X = iris.iloc[:,0:4]#legnth and width data
y = iris.iloc[:,5:8]#species data
y["setosa"] = y["setosa"].astype(int)
y["versicolor"] = y["versicolor"].astype(int)
y["virginica"] = y["virginica"].astype(int)

#Randomly generate training and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                        test_size=0.5)
nn = Sequential() #define neural network nn
nn.add(Dense(10, activation='relu', input_dim = 4))
nn.add(Dense(10, activation='relu', input_dim = 10))
nn.add(Dense(3, activation='sigmoid'))
nn.compile(optimizer = "sgd", loss = "binary_crossentropy",
           metrics = ["accuracy"])
nn.fit(X_train, y_train, epochs = 5)#train nn
prediction = nn.predict(X_test) #use nn to predict
#output prediction result
p = prediction
for i in range(0, p.shape[0]):
    p[i] = np.where(p[i] == np.max(p[i]), True, False)
np.hstack((p, y_test))
#array([[1., 0., 0., 1., 0., 0.],
#       [1., 0., 0., 1., 0., 0.],
#       [1., 0., 0., 1., 0., 0.],
#       [1., 0., 0., 1., 0., 0.],
#       [0., 0., 1., 0., 0., 1.],
#       [0., 0., 1., 0., 0., 1.],
#       [1., 0., 0., 1., 0., 0.],
#       [0., 0., 1., 0., 0., 1.],
#       [1., 0., 0., 1., 0., 0.],
#       [0., 0., 1., 0., 1., 0.],
#The left 3 columns are the predictions results, and
#the right 3 columns are the validation data.
#The first 9 predictions are correct and the 10th is wrong
```

3.5 Chapter summary

This chapter has described four methods of machine learning: K-means, support vector machine (SVM), random forest (RF), and neural network (NN). In the description, we have used the following datasets: the daily weather data at the Miami International Airport, the United States, the daily air quality data of New York City, and R. A. Fisher's iris flower data of species. We have provided both R and Python codes for these algorithms and their applications examples. The following provides a brief summary of our descriptions about the four methods.

(i) K-means clustering: Simply speaking, this clustering method can fairly divide N identical candies on a table for K kids according to the candy locations. It is normally used as an unsupervised learning method. The K-means algorithm minimizes the total within cluster sum of squares (tWCSS) through iterations. As an example, we applied the K-means clustering method to the data of the daily minimum surface air temperature Tmin [in °C] and the direction of the fastest 2-minute wind in a day WDF2 [in degrees] of the Miami International Airport. Our tests suggested that the data have two clusters, one corresponding to the prevailing wind from east, and another from west.

(ii) Support vector machine for the maximum separation of different sets: The SVM algorithm is built on the principle of the maximum differences among the labelled groups of data. The maximum separation of different groups is measured by the distance between positive and negative hyperplanes. SVM can also predict the labels of the new unlabeled data. SVM is usually used as a tool of supervised learning.

(iii) Random forest of decision trees: An RF model is a set of decision trees which form a "forest of decisions." It is usually used as a tool of supervised learning. It has two steps: training and prediction. RF can make both classification and regression. A benchmark RF example is used in our RF classification description: The separation and prediction of the R.A. Fisher iris species dataset. We have also included an example on RF regression using the daily air quality data of New York City.

(iv) Neural network of multiple data fitting: An NN model is a series of data fitting for a given activation function. NN can also make classifications and numerical predictions as a regression. The logistic function is a popular activation function, a smooth transition between category 0 to category 1. Because of randomness is involved in the NN algorithm, different runs of R or Python code may yield different results. We have included an example of using the NN model to help make a hiring decision using the existing data. To compare with the RF model, we have also applied the NN model to the R.A. Fisher flower dataset of iris species.

References and Further Readings

[1] Boehmke, B., and B. Greenwell, 2019: *Hands-on Machine Learning with R*. Chapman and Hall/CRC, Boca Raton, USA.

> This machine learning book has many R code and examples. The book website is `https://bradleyboehmke.github.io/HOML/`.

[2] Chambers, J. M., W.S. Cleveland, B. Kleiner, and P. A. Tukey, 1983: *Graphical Methods for Data Analysis*. Wasworth International Group, Belmont, USA.

> This book contains the daily ozone and weather data from 1 May 1973 to 30 September 1973 of New York. This dataset serves as a benchmark dataset in random forest algorithm. John M. Chambers is a Canadian statistician, who developed the S programming language, and is a core member of the R programming language project.

[3] Fisher, R.A., 1936: The use of multiple measurements in taxonomic problems. *Annals of Eugenics*. 7(2), 179-188.

> This paper contained the iris data frequently used in the teaching of machine learning. The current name of the journal *Annals of Eugenics* is *Annals of Human Genetics*. The author, Ronald Aylmer Fisher (1890 962), was a British mathematician, statistician, biologist, and geneticist. Modern description of this dataset and its applications in statistics and machine learning can be found from numerous websites, such as
>
> `https://www.angela1c.com/projects/iris_project/the-iris-dataset/`

[4] Géron, A., 2017: *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts: Tools, and Techniques to Build Intelligent Systems*. O'Reilly, Boston, USA.

> This machine learning book has many Python code and examples.

[5] McCulloch, W. and W. Pitts, 1943: A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5, 115 -133.

This seminal work is considered the first paper that created the modern NN theory. The paper is highly mathematical, contains ten theorems, and includes a figure of neurons.

[6] Hastie, T., R. Tibshirani, and J.H. Friedman, 2017: *The Elements of Statistical Learning: Data mining, Inference, and Prediction* . 2nd Ed., Springer, New York, USA.

This is a very famous machine learning book, which has in depth descriptions of the commonly used ML algorithms.

Exercises

3.1 Use the K-means method to make a cluster analysis for the following five points in the 2D space: $P_1(1,1), P_2(2,2), P_3(2,3), P_4(3,4), P_5(4,4)$. Assume $K = 2$. Plot a figure similar to Fig. 9.1. What is the final tWCSS equal to?

3.2 Given the following three points $P_1(1,1), P_2(2,2), P_3(2,3)$ and given $K = 2$, make a K-means cluster analysis following the method presented in Sub-Section 9.1.1. Plot your K-means clustering result.

3.3 Use the K-means method to make a cluster analysis for the daily TMIN and WDF2 data of the Miami International Airport in 2015, following the method presented in Sub-sections 9.1.3. Here, TMIN is the daily minimum temperature, and WDF2 denotes the direction [in degrees] of the fastest 2-minute wind in a day. You can obtain the data from the NOAA Climate Data Online website, or from the file

`MiamiIntlAirport2001_2020.csv`

in the `data.zip` file downloadable from the book website

`www.climatestatistics.org`

3.4 Use the K-means method to make a cluster analysis for the daily TMAX and WDF2 data of the Miami International Airport in 2015.

3.5 Use the K-means method to make a cluster analysis for the daily TMIN and Tmax data of the Miami International Airport in 2015.

3.6 Use the K-means method to make a cluster analysis for the daily TMIN and PRCP data of the Miami International Airport in 2015. Here, PRCP denotes the daily total precipitation.

3.7 Identify two climate parameters of your interest, find the corresponding data online, and make a K-means cluster analysis similar to the previous problem for your data.

3.8 Following the method presented in Sub-section 9.2.1., make an SVM analysis for the following five points in a 2D space: $P_1(1,1), P_2(2,2), P_3(2,3), P_4(3,4), P_5(4,4)$. The first three points are labeled 1 and the last two are labeled 2. What are w, b and D_m? What points are the support vectors?

3.9 Two new points are introduced to the previous problem: $Q_1(1.5,1)$ and $Q_2(3,3)$. Use the SVM trained in the previous problem to find out which point belongs to what category.

3.10 From the Internet, download the historical daily data of minimum temperature (TMIN) and average wind speed (AWND) for a month and a location of your interest. Label your data as 1 if the daily precipitation is greater than 0.5 millimeter, and 0 otherwise. Make an SVM analysis for your labeled data.

3.11 SVM forecast: From the Internet, download the historical daily data of minimum temperature and sea level pressure for a month and a location of your interest. Label your data as 1 if the total precipitation of the next day is greater than 0.5 millimeter, and 0 otherwise. Make an SVM analysis for your labeled data. Given the daily data of minimum temperature and sea level pressure of a day in the same month but a different year, use your trained SVM model to forecast whether the next day was rainy or not, i.e., to determine whether the next day is 1 or 0.

3.12 In order to improve the accuracy of your prediction, can you use the data of multiple years to train your SVM model? Perform numerical experiments and show your results.

3.13 The first 50 in the 150 rows of the R.A. Fisher iris data are for setosa, 51-100 are for versocolor, and 101-150 are for virginica, use the data 1-40, 51-90, and 101-140 to train an RF model, following the method in Sub-Section 9.3.1. Use the RF model to predict the species of the remaining data of lengths and widths of petal and sepal. You can download the R.A. Fisher dataset `iris.csv` from the book website or use the data already built in R or Python software packages.

3.14 For the 150 rows R.A. Fisher iris data, use only 20% of the data from each species to train your RF model. Select another 10% of the riris data of lengths and widthes as the new data for prediction. Then use the RF model to predict the species of the new data. Discuss the errors of your RF model and your prediction.

3.15 Plot a decision tree like Fig. 3.11 for the previous exercise problem.

3.16 For the same R.A. Fisher dataset, design your own RF training and prediction. Discuss the RF prediction accuracy for this problem.

3.17 RF forecast: From the Internet, download the historical daily data of minimum temperature and sea level pressure for a month and a location of your interest. Label your data as 1 if the total precipitation of the next day is greater than 0.5 millimeter, and 0 otherwise. Make an RF analysis for your

labeled data. Given the daily data of minimum temperature and sea level pressure of a day in the same month but a different year, use your trained RF model to forecast whether the next day was rainy or not. Is your forecast accurate? How can you improve your forecast?

3.18 Find a climate data time series with missing data and fill in the missing data using the RF regression method described in Sub-Section 9.3.2.

3.19 The first 50 in the 150 rows of the R.A. Fisher iris data are for setosa, 51-100 are for versocolor, and 101-150 are for virginica, use the data 1-40, 51-90, and 101-140 to train an NN model, following the method in Sub-Section 9.4.2. Use the NN model to predict the species of the remaining data of lengths and widths of petal and sepal.

3.20 For the 150 rows R.A. Fisher iris data, use only 20% of the data from each species to train your NN model. Select another 10% of the riris data of lengths and widthes as the new data for prediction. Then use the NN model to predict the species of the new data. Discuss the errors of your NN model and your prediction.

3.21 NN forecast: From the Internet, download the historical daily data of minimum temperature and sea level pressure for a month and a location of your interest. Label your data as 1 if the total precipitation of the next day is greater than 0.5 millimeter, and 0 otherwise. Make an NN analysis for your labeled data. Given the daily data of minimum temperature and sea level pressure of a day in the same month but a different year, use your trained NN model to forecast whether the next day was rainy or not. Is your forecast accurate? Compare your NN forecast with your RF and SVM forecasts.

3.22 Design a machine learning project for yourself or others. What are your training data? What are your test data? What is your training model? What is your training model error? How would you assess your prediction error?

4 Matrix Applications to Regression Models

The word "regression" means "a return to a previous and less advanced or worse form, state, condition, or way of behaving," according to the Cambridge dictionary. The first part "regress" of the word originates from the Latin "regressus," past participle of regredi ("to go back"), from re- ("back") + gradi ("to go"). Thus, "regress" means "return, to go back" and is in contrast to the commonly used word "progress." The *regression* in statistical data analysis refers to a process of returning from the irregular and complex data to a simpler and less perfect state, which is called a model and can be expressed as a curve, a surface, or a function. The function or curve, less complex or less advanced than the irregular data pattern, describes a way of behaving or a relationship. This chapter covers linear models in both uni- and multivariate regressions, least-square estimations of parameters, confidence intervals and inference of the parameters, and fittings of polynomials and other nonlinear curves. By running diagnostic studies on residuals we explain the assumptions of a linear regression model: linearity, homogeneity, independence, and normality. As usual, we use examples of real climate data and provide both R and Python codes.

4.1 Simple linear regression

The simplest model of regression is a straight line, i.e., a linear model. This involves only two variables x and y. Therefore, a *simple linear regression* means going from the data pairs $(x_i, y_i)(i = 1, 2, \cdots, n)$ on the xy-plane back to a simple straight line model

$$y = a + bx. \tag{4.1}$$

The data will determine the values of a and b by the criterion of the best fit. The model can be used for prediction: predict y when a value of x is given.

4.1.1 Temperature lapse rate and an approximately linear model

It is known that as an approximation, in a mountain terrain temperature decreases linearly according to elevation. The decrease rate is called the temperature *lapse rate* in meteorology. The linear relationship and the lapse rate are shown in Fig. 4.1 for Colorado, which is a Rocky Mountains state of the United States and has

a large elevation range from 1010 to 4401 meters, according to Colorado Tourism `www.colorado.com`. The dots in the figure are the scatter plot of the data of elevation and 1981-2010 average July surface air temperature Tmean of the 24 Colorado stations in the United States Historical Climatological Network (USHCN) (Menne et al. 2009). The straight line of the figure corresponds to the linear model

$$y = 33.48 - 0.0070x. \tag{4.2}$$

This equation is a result from regression of data, referred to as observations in climate science, or sample data in statistics.

Thus, Figure 4.1 shows an exhibit of a scattered and complex data pattern regressed to a simpler form, in this case, a straight line model. This is called the simple linear regression process. The regression result has climate science interpretations, such as the lapse rate of temperature.

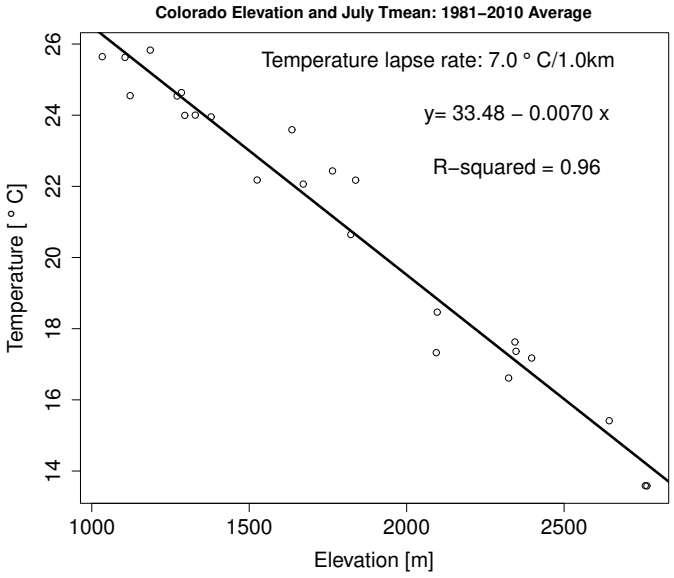


Fig. 4.1 The dots are the scatter plot of the data of elevation and 1981-2010 average July surface air temperature Tmean taken from the 24 USHCN stations. The thick straight line is the linear regression model computed from the data.

The horizontal axis of Fig. 4.1 is for elevation. The elevation data in units of meters for the 24 stations are as follows:

```
1671.5, 1635.6, 2097.0, 1295.4, 1822.7, 2396.9, 2763.0, 1284.7,
1525.2, 1328.6, 1378.9, 2323.8, 2757.8, 1033.3, 1105.5, 1185.7,
2343.9, 1764.5, 1271.0, 2347.3, 2094.0, 2643.2, 1837.9, 1121.7
```

The vertical axis is for temperature, which is the 30-year average of the July daily

mean temperature (Tmean) from 1981-2010, computed from the USHCN monthly data, which have been adjusted for the time of observation bias (TOB). Some stations had missing data denoted by -9999. When computing the 30-year average, the entries of -9999 were omitted. Thus, some averages were computed from fewer than 30 years. The resulting average temperature data in the unit of °C for the 24 stations are as follows:

```
22.064, 23.591, 18.464, 23.995, 20.645, 17.175, 13.582, 24.635,
22.178, 24.002, 23.952, 16.613, 13.588, 25.645, 25.625, 25.828,
17.626, 22.433, 24.539, 17.364, 17.327, 15.413, 22.174, 24.549
```

With the above data, Figure 4.1 can be generated by the following computer code

```
#R plot Fig. 4.1: Colorado temperature lapse rate
x= c(
1671.5, 1635.6, 2097.0, 1295.4, 1822.7, 2396.9, 2763.0, 1284.7,
1525.2, 1328.6, 1378.9, 2323.8, 2757.8, 1033.3, 1105.5, 1185.7,
2343.9, 1764.5, 1271.0, 2347.3, 2094.0, 2643.2, 1837.9, 1121.7)
y= c(
22.064, 23.591, 18.464, 23.995, 20.645, 17.175, 13.582, 24.635,
22.178, 24.002, 23.952, 16.613, 13.588, 25.645, 25.625, 25.828,
17.626, 22.433, 24.539, 17.364, 17.327, 15.413, 22.174, 24.549)
setEPS() # save the .eps figure
postscript("fig0401.eps",  width = 8)
par(mar=c(4.5,4.5,2.5,0.5))
plot(x,y,
     xlab="Elevation [m]",
     ylab=expression("Temperature ["~degree~"C]"),
  main="Colorado Elevation and July Tmean: 1981-2010 Average",
     cex.lab=1.5, cex.axis=1.5, cex.main =1.2)
reg=lm(y~x)
reg
#(Intercept)            x
# 33.476216    -0.006982   #-7.0 degC/1km
summary(reg)
#R-squared:  0.9631
abline(reg,lwd=3)
text(2100, 25.5,
expression("Temperature lapse rate: 7.0"~degree~"C/1.0km"),
     cex=1.5)
text(2350, 24, "y= 33.48 - 0.0070 x", cex=1.5)
text(2350, 22.5,"R-squared = 0.96", cex=1.5)
dev.off()
```

```
# Python plot Fig. 4.1: Colorado temperature lapse rate
x = np.array([
1671.5, 1635.6, 2097.0, 1295.4, 1822.7,2396.9,2763.0,1284.7,
1525.2, 1328.6, 1378.9, 2323.8, 2757.8,1033.3,1105.5,1185.7,
2343.9, 1764.5, 1271.0, 2347.3, 2094.0,2643.2,1837.9,1121.7
])
y = np.array([
22.064, 23.591, 18.464, 23.995, 20.645,17.175,13.582,24.635,
22.178, 24.002, 23.952, 16.613, 13.588,25.645,25.625,25.828,
17.626, 22.433, 24.539, 17.364, 17.327,15.413,22.174,24.549
])
# calculate correlation coefficients
corrMatr = np.corrcoef(x,y)
# R-squared
Rsqu = corrMatr[0,1]**2
# trend line
reg1 = np.array(np.polyfit(x, y, 1))
abline = reg1[1] + x*reg1[0]
fig, ax = plt.subplots(figsize=(12,12))
ax.plot(x,y, 'ko');
ax.plot(x,abline, 'k-');
ax.set_title("Colorado Elevation vs. July Tmean: \n \
1981 - 2010 Average",
             size = 25, fontweight = 'bold', pad = 20)
ax.set_xlabel("Elevation [$m$]", size = 25, labelpad = 20)
ax.set_ylabel("Temperature [$\degree$C]",
              size = 25, labelpad = 20);
ax.tick_params(length=6, width=2, labelsize=20);
ax.set_xticks(np.round(np.linspace(1000, 3000, 5), 2))
ax.set_yticks(np.round(np.linspace(12, 26, 8), 2))
ax.text(1750, 25.5,
        r"Temperature lapse rate: 7.0 $\degree$C/km",
        color= 'k', size = 20)
ax.text(2250, 24.5, r"$y = 33.48 - 0.0070~ x$" % Rsqu,
        color= 'k', size = 20)
ax.text(2250, 23.5, r"$R-squared = %.2f$" % Rsqu,
        color= 'k', size = 20)
plt.show()
```

The Colorado July temperature lapse rate (TLR) 7.0 °C/1.0km is consistent with earlier studies using 104 stations on the west slope of Colorado for an elevation range of 1,500- 3,600 meters: 6.9 °C/1.0km (Fall 1997). For the annual mean temperature, the LTR is 6.0 °C/1.0km. These two results are comparable to an LTR study for the Southern Ecuadorian Andes in the elevation range of 2,610-4,200 meters (Cordova et al. 2016). The annual mean temperature TLR is 6.88 °C/1.0km.

TLR may be applied to approximately predict the temperature of a mountain region at a given location, in case it is hard to maintain a weather station at some locations due to high elevation or complex terrain, while it is relatively easy to obtain the elevation data based on the Geographical Information System (GIS) or a digital elevation model (DEM) dataset.

4.1.2 Assumptions and formula derivations of the single variate linear regression

This subsection describes the statistical concepts and mathematical theory of the above linear regression procedure. We pay special attention to the assumptions about the linear regression model.

4.1.2.1 Linear model and its assumptions

The linear model for TLR may be written as

$$Y = a + bx + \epsilon. \tag{4.3}$$

Here,

(i) x is an independent variable, also called explanatory variable, and is deterministic (not random);
(ii) Y is the dependent variable, also called the response variable, and is a random variable with a constant variance $\mathrm{Var}[Y|x] = \sigma^2$ for any x;
(iii) ϵ is the random error term, which is assumed to have zero mean $\mathrm{E}[\epsilon] = 0$, and constant variance σ^2: $\mathrm{Var}(\epsilon|x) = \sigma^2$, and is uncorrelated with each other: $\mathrm{Cor}(\epsilon|x_1, \epsilon|x_2) = 0$ if $x_1 \neq x_2$;
(iv) a, b and σ are called parameters and are to be estimated from data $(x_i, y_i)(i = 1, 2, \cdots, n)$, also called samples, or sample data; and
(v) The expected value of Y for a given x is $a + bx$, i.e.,

$$\mathrm{E}[Y|x] = a + bx. \tag{4.4}$$

The linear model Eq. (4.3) at the given points x_i is

$$Y_i = a + bx_i + \epsilon_i, \quad i = 1, 2, \cdots, n. \tag{4.5}$$

where both ϵ_i and Y_i are random variables with

$$\mathrm{E}[Y_i] = a + bx_i, \tag{4.6}$$
$$\mathrm{E}[\epsilon_i] = 0, \tag{4.7}$$
$$\mathrm{Var}(Y_i) = \mathrm{E}[(Y_i - (a + bx_i))^2] = \sigma^2, \tag{4.8}$$
$$\mathrm{Var}(\epsilon_i) = \mathrm{E}[(\epsilon_i - 0)^2] = \sigma^2, \tag{4.9}$$
$$\mathrm{Cor}(\epsilon_i, \epsilon_j) = \delta_{ij}, \tag{4.10}$$

where δ_{ij} is the Kronecker delta

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{4.11}$$

The last equation means that the error terms ϵ_i at the observational points x_i are uncorrelated with each other.

The observations at the given points x_i are y_i. Both are fixed values, such as temperature $y_1 = 22.064°\text{C}$ at the given elevation $x_1 = 1671.5$ meters in the Colorado July temperature example. Here, y_1 is a sample value for the random variable Y_1. The corresponding error datum is e_1, which is called a residual and is regarded a sample value of ϵ_1 with respect to the linear model.

4.1.2.2 Estimating a and b from data

Given a set of sample data $(x_i, y_i), i = 1, 2, \cdots, n$, the parameters a and b in the regression model can be estimated as $\hat{a}, \hat{b}$. Thus, $\hat{a}, \hat{b}$ correspond to a particular dataset, i.e., another dataset $(u_i, v_i), i = 1, 2, \cdots, m$ will yield a pair of different estimators that may be denoted by $\hat{a}_{uv}, \hat{b}_{uv}$. In contrast, a, b are the general notations of constant coefficients for a linear regression model.

The estimated linear model may be written in the following way:

$$y_i = \hat{a} + \hat{b}x_i + e_i, \quad i = 1, 2, \cdots, n, \tag{4.12}$$

where

$$e_i = y_i - (\hat{a} + \hat{b}x_i) \tag{4.13}$$

are called residuals of the linear model, and are equal to the observed values y_i minus the predicted values

$$\hat{y}_i = \hat{a} + \hat{b}x_i. \tag{4.14}$$

The mean of the above n equations (4.12) yields

$$\bar{y} = \hat{a} + \hat{b}\bar{x}, \tag{4.15}$$

where

$$\bar{x} = \frac{x_1 + x_2 + \cdots x_n}{n}, \tag{4.16}$$

$$\bar{y} = \frac{y_1 + y_2 + \cdots y_n}{n}. \tag{4.17}$$

Because of the unbiased model assumption $E[\epsilon] = 0$, we require that

$$\bar{e} = \frac{e_1 + e_2 + \cdots e_n}{n} = 0, \tag{4.18}$$

equivalently,

$$\sum_{i=1}^{n} e_i = 0. \tag{4.19}$$

Equations (4.15) and (4.18) give an estimate for a from data:

$$\hat{a} = \bar{y} - \hat{b}\bar{x}, \tag{4.20}$$

but $\hat{b}$ is to be estimated by

$$\hat{b} = \frac{\mathbf{y}_a \cdot \mathbf{x}_a}{\mathbf{x}_a \cdot \mathbf{x}_a}. \tag{4.21}$$

Here,

$$\mathbf{x}_a = \begin{bmatrix} x_{a,1} \\ x_{a,2} \\ \vdots \\ x_{a,n} \end{bmatrix} \quad \mathbf{y}_a = \begin{bmatrix} y_{a,1} \\ y_{a,2} \\ \vdots \\ y_{a,n} \end{bmatrix} \quad \mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}, \tag{4.22}$$

with

$$x_{a,i} = x_i - \bar{x}, \tag{4.23}$$

$$y_{a,i} = y_i - \bar{y}. \tag{4.24}$$

In the last two formulas, $x_{a,i}$ and $y_{a,i}$ may be regarded as anomaly data, hence with the subscript "a." Climate scientists often uses the concept of anomaly data, departures from a climate normal.

The $\hat{b}$ estimate formula (4.21) can be derived from the best fit condition which minimizes the mean square errors (MSE)

$$\text{MSE} = \frac{|\mathbf{e}|^2}{n} = \frac{\sum_{i=1}^n e_i^2}{n} = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}, \tag{4.25}$$

equivalently minimizing the sum of the square errors (SSE)

$$\text{SSE} = |\mathbf{e}|^2 = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = n \times \text{MSE}, \tag{4.26}$$

where $|\mathbf{e}|$ denotes the Euclidean length of vector $\mathbf{e}$. Thus, this method is also called the least square estimate. The least square condition is equivalent to the following orthogonality condition (See Fig. 4.2)[1]

$$\mathbf{e} \cdot \mathbf{x}_a = 0. \tag{4.27}$$

Inserting Eq. (4.20) into the linear model with data (4.12) leads to

$$y_i - \bar{y} = \hat{b}(x_i - \bar{x}) + e_i, \quad i = 1, 2, \cdots, n. \tag{4.28}$$

or

$$y_{a,i} = \hat{b} x_{a,i} + e_i, \quad i = 1, 2, \cdots, n, \tag{4.29}$$

[1] SSE can be written as

$$\text{SSE} = |\mathbf{e}|^2 = |\mathbf{y_a} - \hat{b}\mathbf{x_a}|^2.$$

When $\hat{b}$ is optimized to minimize SSE, the derivative of the above equation with respect to $\hat{b}$ is zero.

$$\frac{\mathrm{d}}{\mathrm{d}\hat{b}}\text{SSE} = (\mathbf{y_a} - \hat{b}\mathbf{x_a}) \cdot \mathbf{x_a} = \mathbf{e} \cdot \mathbf{x_a} = 0.$$

This is the orthogonality condition. This condition $\mathbf{e} \cdot \mathbf{x}_a = 0$ can also be derived from geometric point of view. The three vectors $\hat{b}\mathbf{x}_a, \mathbf{e}$ and $\mathbf{y}_a$ in $\mathbf{y}_a = \hat{b}\mathbf{x}_a + \mathbf{e}$ form a triangle. The side $\mathbf{e}$, depending on $\hat{b}$, is the shortest when $\hat{b}\mathbf{x}_a$ is perpendicular to $\mathbf{e}$ by adjusting parameter $\hat{b}$.

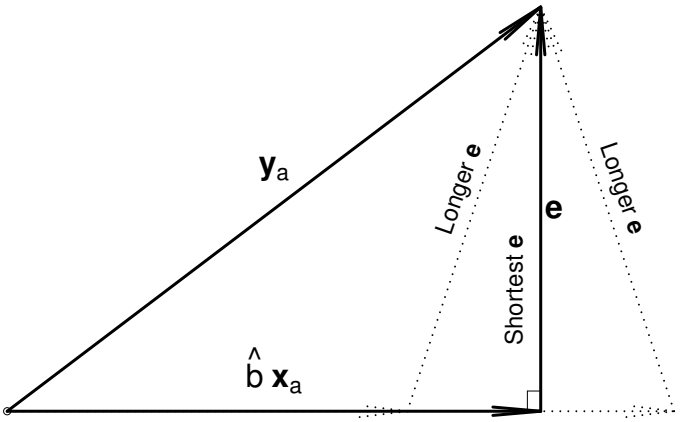


Fig. 4.2 The condition of the least sum of the residual squares SSE is equivalent to the orthogonality condition $\mathbf{e} \cdot \mathbf{x}_a = 0$.

or

$$\mathbf{y}_a = \hat{b}\mathbf{x}_a + \mathbf{e}. \tag{4.30}$$

The dot product of both sides of Eq. (4.29) with $\mathbf{x}_a$ yields

$$\hat{b} = \frac{\mathbf{y}_a \cdot \mathbf{x}_a - \mathbf{e} \cdot \mathbf{x}_a}{\mathbf{x}_a \cdot \mathbf{x}_a}. \tag{4.31}$$

With the orthogonality condition Eq. (4.27) $\mathbf{e} \cdot \mathbf{x}_a = 0$, Eq. (4.31) is reduced to the least square estimate of b: Eq. (4.21).

In summary, the estimates $\hat{a}$ and $\hat{b}$ are derived by requiring (i) zero mean of the residuals, and (ii) minimum sum of the residual squares.

As shown in the computer code for Fig. 4.1, the R command for the simple linear regression is `lm(y ~ x)`. The TLR dataset yields the estimate of the intercept $\hat{a} = 33.476216$ and slope $\hat{b} = -0.006982$:

```
lm(y ~ x)
#(Intercept)            x
#33.476216      -0.006982
```

The estimated linear model is thus

$$y = 33.476216 - 0.006982x, \tag{4.32}$$

or

$$\text{Temp } [^\circ\text{C}] = 33.476216 - 0.006982 \times \text{Elevation [m]}. \tag{4.33}$$

The corresponding Python code is as follows:

```
#Python linear regression as the first order polynomial fit
reg = np.polyfit(x, y, 1)
print(reg)
#[-6.98188456e-03  3.34763004e+01]
```

The 24 residuals $e_i, i = 1, 2, \cdots, 24$ are

```
reg = lm(y ~ x)
round(reg$residuals, digits = 5)
#0.25792  1.53427 -0.37129 -0.43697 -0.10542  0.43358
#-0.60335  0.12833 -0.64953 -0.19817  0.10302 -0.63880
#-0.63366 -0.61692 -0.13283  0.63012  0.51454  1.27623
#-0.06333  0.27628 -1.52923  0.39122  1.52971 -1.09572
```

The mean of the 24 residuals is zero, demonstrated by an R code below

```
mean(reg$residuals)
#[1] 1.62043e-17
```

The least square condition $\mathbf{e} \cdot \mathbf{x}_a = 0$ is demonstrated by the following R code

```
xa = x - mean(x)
sum(xa*reg$residuals)
#[1] -2.83773e-13
```

The unbiased MSE is

$$s^2 = \frac{1}{n-2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{4.34}$$

Here, subtracting 2 is due to the two constraints of estimating a and b. The unbiased MSE s^2 the Colorado TLR regression can be computed using the following R code

```
sum((reg$residuals)^2)/(length(y) -2)
#[1] 0.6096193
```

Thus, one can use the data to estimate $\hat{b}$ first using (4.21), then $\hat{a}$ using (4.20), and finally $\hat{\sigma}^2$, also denoted by s^2, using (4.34).

The above computer code is summarized as follows.

```
#R code for the Colorado TLR regression analysis
lm(y ~ x)
#(Intercept)            x
#33.476216    -0.006982

reg = lm(y ~ x)
round(reg$residuals, digits = 5)
mean(reg$residuals)
#[1] 1.62043e-17

xa = x - mean(x)
sum(xa*reg$residuals)
#[1] -2.83773e-13
```

```
# Python code for the Colorado TLR regression analysis
reg = np.polyfit(x, y, 1)
print(reg) #slope and intercept
#[-6.98188456e-03  3.34763004e+01]

regfit = np.polyval(reg,x)
regresiduals = y - regfit
print(np.round(regresiduals,5))
print(np.mean(regresiduals))
#-1.8503717077085940e-17 #should be almost zero

xa = np.array(x) - np.mean(x)
np.sum(xa*regresiduals)
print(np.dot(xa, regresiduals)) #orthogonality
#-1.4892975741531700e-11 #should be almost zero

np.sum((regresiduals)**2)/(y.size -2)
#0.6096193452251238 #unbiased MSE
```

Once again, we emphasize that two assumptions are used in the derivation of the formulas to estimate $\hat{a}$ and $\hat{b}$:
(a) The unbiased model assumption: The mean residual is zero $\bar{e} = 0$, and
(b) The optimization assumption: The residual vector $\mathbf{e}$ is perpendicular to the x-anomaly vector $\mathbf{x}_a = x - \bar{x}$.

Under these two assumptions, the estimators $\hat{a}$ and $\hat{b}$ are highly sensitive to the Y outliers, particularly the outlier data corresponding to the smallest and largest x values. One outlier can completely change the $\hat{a}$ and $\hat{b}$ values, in agreement with our intuition. This is an end point problem often encountered in data analysis. To suppress the sensitivity, many robust regression methods have been developed and R packages are available, such as the Robust Regression, by UCLA institute for Digital Research & Education:

`https://stats.idre.ucla.edu/r/dae/robust-regression/`

Figure 4.2 can be generated by the following R code.

```
#R plot Fig. 4.2: Geometric derivation of the least squares
par(mar=c(0.0,0.5,0.0,0.5))
plot(0,0, xlim=c(0,5.2), ylim=c(0,2.2),
     axes = FALSE, xlab="", ylab="")
arrows(0,0,4,0, angle=5, code=2, lwd=3, length=0.5)
arrows(4,0,4,2, angle=5, code=2, lwd=3, length=0.5)
arrows(0,0,4,2, angle=5, code=2, lwd=3, length=0.5)
arrows(5,0,4,2, angle=7, code=2, lwd=2, lty=3, length=0.5)
arrows(0,0,5,0, angle=7, code=2, lwd=2, lty=3, length=0.5)
arrows(3,0,4,2, angle=7, code=2, lwd=2, lty=3, length=0.5)
arrows(0,0,3,0, angle=7, code=2, lwd=2, lty=3, length=0.5)
segments(3.9,0, 3.9, 0.1)
segments(3.9, 0.1, 4.0, 0.1)
text(2,0.2, expression(hat(b)~bold(x)[a]), cex=2)
text(2,1.2, expression(bold(y)[a]), cex=2)
text(4.1,1, expression(bold(e)), cex=2)
```

```
text(3.8,0.6, expression(paste("Shortest␣",bold(e))),
     cex=1.5, srt=90)
text(3.4,1.1, expression(paste("Longer␣",bold(e))),
     cex=1.5, srt=71)
text(4.6,1.1, expression(paste("Longer␣",bold(e))),
     cex=1.5, srt=-71)
```

```
# Python plot Fig. 4.2: Geometry of the least squares
plt.figure(figsize=(10,8))# Define figure size
plt.xlim([0, 5.2]); plt.ylim([0, 2.2])
plt.axis('off')
plt.annotate('', ha = 'center', va = 'bottom',
             xytext = (0, 0),xy = (4,  0),
            arrowprops = {'facecolor' : 'black'})
plt.annotate('', ha = 'center', va = 'bottom',
             xytext = (4, 0),xy = (4,  2),
            arrowprops = {'facecolor' : 'black'})
plt.annotate('', ha = 'center', va = 'bottom',
             xytext = (0, 0),xy = (4,  2),
            arrowprops = {'facecolor' : 'black'})
plt.plot([4, 5], [0,  0], color ='k', linewidth = 12,
        dashes = (3, 1))
plt.plot([3,4], [0,  2], color ='k', linewidth = 3,
        dashes = (3, 1))
plt.plot([5,4], [0,  2], color ='k', linewidth = 3,
        dashes = (4,1))
plt.plot([0, 3], [0,  0], color ='k', linewidth = 12,
        dashes = (4,1))
plt.plot([3.9, 3.9], [0,  0.1], color ='k')
plt.plot([3.9, 4.0], [0.1,  0.1], color ='k')
plt.text(2, 0.1, r'$\hat{b}~\mathbf{x}_a$', fontsize = 30)
plt.text(2, 1.2, r'$\mathbf{y}_a$', fontsize = 30)
plt.text(4.08, 0.8, r'$\mathbf{e}$', fontsize = 30)
plt.text(3.8, 0.6, r'Shortest␣$\mathbf{e}$',
         rotation=90, fontsize = 20)
plt.text(3.3, 1.1, r'Longer␣$\mathbf{e}$',
         rotation=71, fontsize = 20)
plt.text(4.3, 1.1, r'Longer␣$\mathbf{e}$',
         rotation= -71, fontsize = 20)
plt.show()
```

4.1.2.3 Relationship between slope and correlation

The slope $\hat{b}$ and correlation r_{xy} are related in the following way

$$\hat{b}|\mathbf{x}_a| = r_{xy}|\mathbf{y}_a|. \tag{4.35}$$

This has two extreme cases: $r_{xy} = 1$ and $r_{xy} = 0$.

Case (a). Perfect correlation $r_{xy} = 1$ when the points on the scatter plot of the x, y data lie exactly on a straight line, which implies perfect prediction

$$\hat{b}|\mathbf{x}_a| = |\mathbf{y}_a|. \tag{4.36}$$

Case (b). Perfect noise with zero correlation $r_{xy} = 0$, which implies

$$\hat{b} = 0, \tag{4.37}$$

i.e., the best prediction is the mean.

Formula (4.35) can be derived as follows. The slope estimate formula (4.21) can be re-written to provide geometric and science interpretations.

(i) Geometric projection interpretation:

$$\hat{b} = \frac{\mathbf{y}_a \cdot (\mathbf{x}_a/|\mathbf{x}_a|)}{|\mathbf{x}_a|}. \tag{4.38}$$

Since $\mathbf{x}_a/|\mathbf{x}_a|$ is a unit vector in the direction of $\mathbf{x}_a$, the slope is the projection of the y anomaly data vector on the x anomaly data vector and then normalized by the x anomaly data vector.

Or we write the $\hat{b}$ formula in the following way:

$$\hat{b} = \frac{\mathbf{y}_a}{|\mathbf{x}_a|} \cdot \frac{\mathbf{x}_a}{|\mathbf{x}_a|}. \tag{4.39}$$

The first fraction on the right hand side is the y anomaly data vector normalized by $|\mathbf{x}_a|$. Thus, the slope is the projection of this normalized y anomaly data vector onto the unit x anomaly data vector.

(ii) Correlation interpretation:

$$\hat{b} = r_{xy} \frac{|\mathbf{y}_a|}{|\mathbf{x}_a|} \tag{4.40}$$

where r_{xy} is the correlation coefficient, or simply called the correlation, defined by

$$r_{xy} = \frac{\mathbf{y}_a \cdot \mathbf{x}_a}{|\mathbf{x}_a||\mathbf{y}_a|} \tag{4.41}$$

The slope depends directly on the correlation coefficient between the x and y anomaly data. The correlation is scaled by the ratio of the length of the y anomaly data vector to the length of the x anomaly data vector.

Below is the computer code corresponding to the two interpretations for the Colorado July TLR example.

```
#R code for estimating regression slope b

#Method 1: Using vector projection
xa = x - mean(x)  #Anomaly the x data vector
nxa = sqrt(sum(xa^2)) #Norm of the anomaly data vector
ya = y - mean(y)
nya=sqrt(sum(ya^2))
sum(ya*(xa/nxa))/nxa #Compute b
#[1] -0.006981885  #This is an estimate for b

#Method 2:  Using correlation
corxy=cor(xa, ya) #Compute the correlation between xa and ya
corxy
#[1] -0.9813858 #Very high correlation
```

```
corxy*nya/nxa #Compute b
#[1] -0.006981885 #This is an estimate for b
```

```
#Python code for estimating regression slope b

#Method 1: Using vector projection
xa = x - np.mean(x)  #Anomaly the x data vector
nxa = np.sqrt(np.sum(xa**2)) #Norm of the anomaly vector
ya = y - np.mean(y)
nya = np.sqrt(sum(ya**2))
print(np.sum(ya*(xa/nxa))/nxa) #Compute b
#[1] -0.006981885  #This is an estimate for b

#Method 2:  Using correlation
from scipy.stats import pearsonr
corxy, _ = pearsonr(xa, ya) #Correlation between xa and ya
print(corxy)
#[1] -0.9813858 #Very high correlation
print(corxy*nya/nxa) #Compute b
#[1] -0.006981885 #This is an estimate for b
```

4.1.2.4 Percentage of variance explained: $R^2 \times 100\%$

The unbiased variance of the model-predicted data $\hat{y}_i (i = 1, 2, \cdots, n)$ is defined as

$$\text{MV} = \frac{\sum_{i=1}^{n} \left[(\hat{y}_i - \bar{\hat{y}} \right]^2}{n-1}, \tag{4.42}$$

where $\bar{\hat{y}}$ is the mean of $\hat{y}_i (i = 1, 2, \cdots, n)$.

The unbiased variance of the Y data

$$\text{YV} = \frac{\sum_{i=1}^{n} \left[(y_i - \bar{y}) \right]^2}{n-1}, \tag{4.43}$$

where $\bar{y}$ is the mean of the original station temperature data $y_i (i = 1, 2, \cdots, n)$.

We wish to measure how good the model is in terms of variance, and thus define the ratio of MV to YV, which is named R^2

$$R^2 = \frac{\text{MV}}{\text{YV}}, \tag{4.44}$$

The value $R^2 \times 100\%$ indicates the percentage of variance explained by the linear model. A larger R^2 value suggests a better model.

The variance of the model-predicted data can be re-written as

$$\begin{aligned}\text{MV} &= \frac{\sum_{i=1}^{n}\left[(\hat{a}+\hat{b}x_i)-(\hat{a}+\hat{a}\bar{x})\right]^2}{n-1}\\ &= \hat{b}^2\frac{\sum_{i=1}^{n}[x_i-\bar{x}]^2}{n-1}\\ &= \left[r_{xy}\frac{|\mathbf{y}_a|}{|\mathbf{x}_a|}\right]^2\frac{|\mathbf{x}_a|^2}{n-1}\\ &= r_{xy}^2\frac{|\mathbf{y}_a|^2}{n-1}\\ &= r_{xy}^2\times \text{YV}. \end{aligned} \tag{4.45}$$

Therefore,

$$R^2 = r_{xy}^2. \tag{4.46}$$

The R^2 value is equal to the square of the correlation coefficient computed from the data $(x_i, y_i)(i = 1, 2, \cdots, n)$ with $0 \le R^2 \le 1$.

For the Colorado TLR data used earlier, we have $r_{xy} = -0.9813858$, and $R^2 = 0.9631181$.

The variances MV, YV, and R^2 can be computed by the following computer code in multiple ways.

```
#R code for computing MV
var(reg$fitted.values)
#[1] 15.22721
#Or another way
yhat = reg$fitted.values
var(yhat)
#[1] 15.22721
#Or still another way
n = 24
sum((yhat - mean(yhat))^2)/(n-1)
#[1] 15.22721

#R code for computing YV
sum((y - mean(y))^2)/(n-1)
# [1] 15.81033
#Or another way
var(y)
#[1] 15.81033

#R code for computing R-squared value
var(reg$fitted.values)/var(y)
#[1] 0.9631181  #This is the R-squared value

cor(x,y)
#[1] -0.9813858
(cor(x,y))^2
#[1] 0.9631181 #This is the R-squared value
```

```
#Python code for Compute MV, YV, and R^2
from statistics import variance
reg = np.polyfit(x, y, 1)
yhat = np.polyval(reg,x)
#computing MV
print('MV = ', variance(yhat))
#MV =  15.227212262175959
#Or another way
n = 24
print('MV = ',
      np.sum((yhat - np.mean(yhat))**2)/(n-1))
#MV =  15.227212262175959

#computing YV
print('YV = ', np.sum((y - np.mean(y))**2)/(n-1))
# [1] 15.81033
#Or another way
print('YV = ', variance(y))
#YV = 15.81032641847826

#computing R-squared value
print('R-squared = ', variance(yhat)/variance(y))
#R-squared =  0.9631181456430407

#computing correlation and R-squared
from scipy.stats import pearsonr
corxy, _ = pearsonr(x, y)
print('Correlation r_xy = ', corxy)
#Correlation r_xy =  -0.9813858291431772
print('R-squared = ', corxy**2)
#R-squared =  0.9631181456430413
```

4.1.2.5 The regression estimates by the least square or the maximum likelihood

A more complicated and commonly used derivation is based on the minimization of the sum of squared errors (SSE)

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} [y_i - (\hat{a} + \hat{b}x_i)]^2. \tag{4.47}$$

This approach can be found in most statistics books and from the Internet. The terminology of "least square" regression comes from this minimization principle.

This minimization is with respect to $\hat{a}$ and $\hat{b}$. The minimization condition leads to two linear equations that determine $\hat{a}$ and $\hat{b}$. The "least square" minimization condition for $\hat{b}$ is equivalent to the orthogonality condition (4.27). Geometrically, the minimum distance of a point to a line is defined as the length of the line segment that is orthogonal to the line and connects the point to the line, that is, it

is the minimum distance between the point and the line (See Fig. 4.2). Thus, the orthogonality condition and minimization condition are equivalent.

Another commonly used method to estimate the parameters is to maximize a likelihood function defined as

$$L(a, b, \sigma) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp(-(y_i - (a + bx_i))^2/(2\sigma^2)). \tag{4.48}$$

The solution for the maximum $L(a, b, \sigma)$ yields the estimate of $\hat{a}$, $\hat{b}$, and $\hat{\sigma}^2$. The result is exactly the same as the ones obtained by the method of least squares or perpendicular projection.

However, the maximum likelihood approach shown here explicitly assumes the normal distribution of the error term ϵ_i and the response variable Y_i. For the least squares approach, the assumption of normal distribution is needed only when for the inference of the parameters $\hat{a}$, $\hat{b}$, and $\hat{\sigma}^2$, but is not needed to estimate them.

4.1.3 Statistics of slope and intercept: Distributions, confidence intervals, and inference

4.1.3.1 Mean and variance of the slope

The normal distribution of a and b is now assumed in this section. This assumption was not needed earlier to estimate $\hat{a}$ and $\hat{b}$ and to compute R-squared.

Different datasets will yield different estimates of $\hat{a}$, $\hat{b}$, and $\hat{\sigma}^2$, which form a distribution corresponding to the random datasets. Thus, we may regard $\hat{a}$, $\hat{b}$, and $\hat{\sigma}^2$ as random variables. Their expected values are a, b and σ^2 if the random datasets satisfy the linear model assumptions (linearity, constant variance, independent errors, and normal distribution). However, in practical applications, we just have one dataset, estimate $\hat{a}$, $\hat{b}$, and $\hat{\sigma}^2$ once, and then interpret the results.

Following Eq. (4.21), instead of using data y_i to estimate the slope, we use the corresponding random variables Y_i to define the slope B as a random variable:

$$\begin{aligned} B &= \frac{\mathbf{Y}_a \cdot \mathbf{x}_a}{\mathbf{x}_a \cdot \mathbf{x}_a} \\ &= \frac{\sum_{i=1}^{n}(Y_i - \bar{Y})(x_i - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\ &= \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \right) Y_i, \end{aligned} \tag{4.49}$$

because

$$\sum_{i=1}^{n}(x_i - \bar{x}) = 0. \tag{4.50}$$

The above shows that B is a linear combination of n terms Y_i. If $Y_i \sim N(a+bx, \sigma^2)$, then B is also normally distributed when n is large, because of the Central Limit Theorem.

The expected value of B is b, as shown below:

$$\begin{aligned}
\mathrm{E}[B] &= \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \right) E[Y_i] \\
&= \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \right) (a + bx_i) \\
&= b \times \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \right) x_i \\
&= b \times \left(\frac{\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \right) \\
&= b. \qquad (4.51)
\end{aligned}$$

The variance of B is

$$\mathrm{Var}(B) = \frac{\sigma^2}{||\mathbf{x}||_a^2}. \qquad (4.52)$$

This can be shown as follows:

$$\begin{aligned}
\mathrm{Var}[B] &= \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \right)^2 \mathrm{Var}(Y_i) \\
&= \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\
&= \frac{\sigma^2}{||\mathbf{x}||_a^2}. \qquad (4.53)
\end{aligned}$$

Statistics literature often denotes

$$S_{xx} = ||\mathbf{x}||_a^2. \qquad (4.54)$$

Hence,

$$\mathrm{Var}(B) = \frac{\sigma^2}{S_{xx}}. \qquad (4.55)$$

When n goes to infinity, S_{xx} also goes to infinity. Thus,

$$\lim_{n \to \infty} \mathrm{Var}(B) = 0. \qquad (4.56)$$

This means that more data points yield a better estimate for the slope, supporting our intuition.

4.1.3.2 Mean and variance of the intercept

Instead of estimating the intercept using data y_i following formula (4.20), we define the intercept as a random variable using the same formula but with random variable Y:

$$A = \bar{Y} - B\bar{x}. \qquad (4.57)$$

Below we show that $E[A] = a$:

$$\begin{aligned} \mathrm{E}[A] &= E[\bar{Y}] - E[B]\bar{x} \\ &= \frac{\sum_{i=1}^{n} Y_i}{n} - b\frac{\sum_{i=1}^{n} x_i}{n} \\ &= \frac{\sum_{i=1}^{n}(Y_i - bx_i)}{n} \\ &= \frac{\sum_{i=1}^{n} a}{n} \\ &= a. \end{aligned} \tag{4.58}$$

The variance of A is equal to

$$\mathrm{Var}[A] = \frac{\sigma^2}{n}\left(1 + \frac{\bar{x}^2}{\sigma_x^2}\right), \tag{4.59}$$

where

$$\sigma_x^2 = \frac{S_{xx}}{n} \tag{4.60}$$

may be considered the estimated variance of the x_i data. The is result can be derived as follows:

$$\begin{aligned} \mathrm{Var}[A] &= \mathrm{Var}[\bar{Y}] + \mathrm{Var}[B]\bar{x}^2 \\ &= \frac{\sigma^2}{n} + \frac{\sigma^2}{S_{xx}}\bar{x}^2 \\ &= \frac{\sigma^2}{n}\left(1 + \frac{\bar{x}^2}{S_{xx}/n}\right). \end{aligned} \tag{4.61}$$

This expression implies that as n goes to infinity, the variance $\mathrm{Var}[A]$ also goes to zero. Thus, the intercept estimate is better when having more data points.

4.1.3.3 Confidence intervals of slope and intercept

The $(1-\alpha)\times 100\%$ confidence interval of the slope is

$$\left(\hat{b} - t_{\alpha/2,n-2}\frac{s}{\sqrt{S_{xx}}},\quad \hat{b} + t_{\alpha/2,n-2}\frac{s}{\sqrt{S_{xx}}}\right), \tag{4.62}$$

where

$$s = \sqrt{\frac{SSE}{n-2}} \tag{4.63}$$

is the unbiased estimator of σ, and the degrees of freedom (dof) of the t-distribution is $n-2$.

If the confidence interval does not include zero, then the slope is significantly different from zero at the $\alpha \times 100\%$ significance level. The Colorado TLR example has $\hat{b} = -0.006982$. For the 95% confidence interval, the R's quantile function `qt` of the t-distribution can produce the quantile interval corresponding to probabilities 0.025 and 0.975:

```
qt(c(.025, .975), df=22)
#[1] -2.073873  2.073873
```

The estimated standard error for $\hat{b}$ is

$$\mathrm{SE}[\hat{b}] = \frac{s}{\sqrt{S_{xx}}} = 0.0002913 \tag{4.64}$$

based on the R output from `summary(reg)`. Thus, the confidence interval for $\hat{b}$ is

$$\begin{aligned}&(-0.0069818 - 2.073873 \times 0.0002913, -0.0069818 + 2.073873 \times 0.0002913)\\ &= (-0.0076, -0.0064).\end{aligned} \tag{4.65}$$

This confidence interval does not include zero, and hence the slope is significantly less zero at the 5% confidence level. The 95% confidence interval for TLR is (6.4, 7.6) °C/km based on the Colorado data used in this chapter.

The $(1-\alpha) \times 100\%$ confidence interval of the intercept is

$$\left(\hat{a} - t_{\alpha/2,n-2}\ s\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}},\quad \hat{a} + t_{\alpha/2,n-2}\ s\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}} \right). \tag{4.66}$$

The R output from `summary(reg)` gives the standard error of $\hat{a}$

$$\mathrm{SE}[\hat{a}] = s\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}} = 0.5460279 \tag{4.67}$$

Thus, the confidence interval for $\hat{a}$ is

$$\begin{aligned}&(33.4762157 - 2.073873 \times 0.5460279, 33.4762157 + 2.073873 \times 0.5460279)\\ &= (32.3438, 34.6086).\end{aligned} \tag{4.68}$$

The mathematical expressions of the confidence interval can be derived as follows. Formulas (4.51), (4.55) (4.58) and (4.59) imply that

$$A \sim N\left(a, \frac{\sigma^2}{n}\left(1 + \frac{\bar{x}^2}{\sigma_x^2}\right)\right), \tag{4.69}$$

$$B \sim N\left(b, \frac{\sigma^2}{S_{xx}}\right) \tag{4.70}$$

$$\sigma_x^2 = S_{xx}/n \tag{4.71}$$

for a given σ^2.

When σ^2 is unknown and is to be estimated by s^2, then A and B follow a t-distribution. To be exact, the t-statistics with $n-2$ dof can be defined as

$$T_a = \frac{\hat{a} - a}{s\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}}}, \tag{4.72}$$

$$T_b = \frac{\hat{b} - b}{s/\sqrt{S_{xx}}}, \tag{4.73}$$

where a and b are given, and $\hat{a}$ and $\hat{b}$ are estimated from data. The confidence

interval $(-t_{\alpha/2,n-2}, t_{\alpha/2,n-2})$ for T_a and T_b leads to the confidence intervals for $\hat{a}$ and $\hat{b}$. Thus, these two statistics T_a and T_b can be used for a hypothesis test.

4.1.3.4 Hypothesis test for the slope using the t-test

We wish to test the hypothesis whether the slope is equal to a given constant β. The null and alternative hypotheses are

$$H_0: \quad b = \beta, \tag{4.74}$$

$$H_1: \quad b \neq \beta. \tag{4.75}$$

We define a t-statistic

$$T_b = \frac{\hat{b} - \beta}{s/\sqrt{S_{xx}}}. \tag{4.76}$$

At the $\alpha \times 100\%$ significance level, we reject H_0 if

$$|T_b| > t_{1-\alpha/2,n-2}. \tag{4.77}$$

For the Colorado July TLR example, if we want to test whether the TLR is equal to 7.3 °C/km, we calculate

$$T_b = \frac{-0.0069818 - (-0.0073)}{0.0002913} = 1.092345. \tag{4.78}$$

The quantile

$$t_{0.975,22} = 2.073873 \tag{4.79}$$

Thus, $|T_b| < t_{0.975,22}$, and hence the null hypothesis is not rejected at the 5% significance level. Namely, the obtained TLR 7.0 °C/km is not significantly different from the given value 7.3 °C/km.

4.1.3.5 Confidence intervals of prediction: The fitted model and the response variable

For an assigned point x^*, which is not on the data points x_i, the fitted model prediction is

$$\hat{Y} = \hat{a} + \hat{b}x^*. \tag{4.80}$$

The variance of this model prediction is

$$\mathrm{Var}[\hat{a} + \hat{b}x^*] = \sigma^2 \left[\frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}\right]. \tag{4.81}$$

Thus, the $(1-\alpha) \times 100\%$ confidence interval for $\hat{a} + \hat{b}x^*$ is given by

$$\hat{a} + \hat{b}x^* \pm t_{1-\alpha/2,n-2} s \sqrt{\frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}}. \tag{4.82}$$

This interval can be computed for any $x = x^*$ and is shown by the red lines in Fig. 4.3. It means that 95% of the chance for the model prediction to be between the red lines. Note that the red lines diverge as x^* deviates from $\bar{x}$. This implies that the confidence interval becomes larger as we go out of the range of the data.

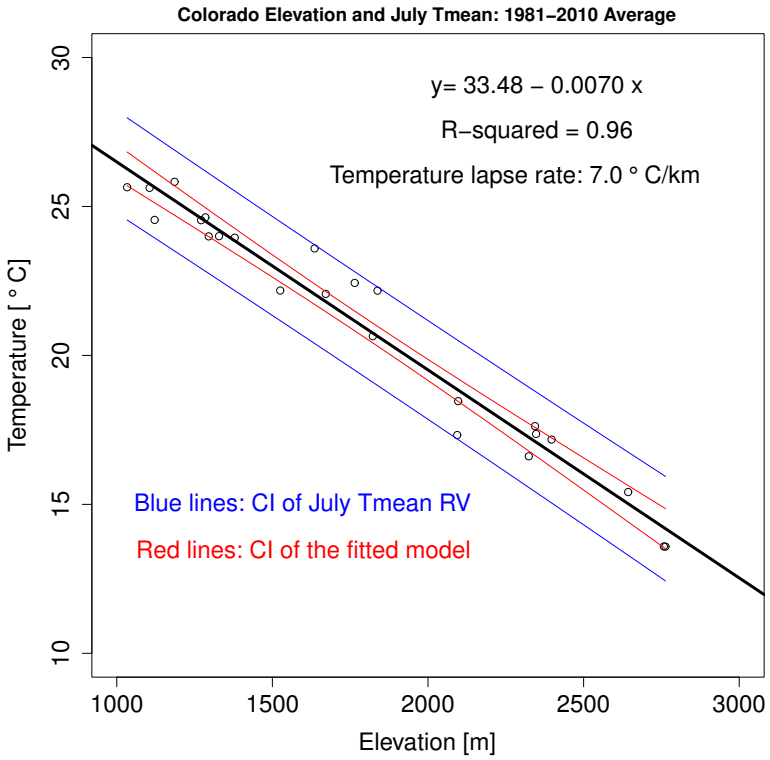


Fig. 4.3 The confidence interval of the fitted model based on the Colorado July mean temperature data and their corresponding station elevations (Red lines). The confidence interval of the response variable Tmean (Blue lines).

The error of the prediction at x^* is

$$e^* = Y - \hat{Y} = Y - (\hat{a} + \hat{b}x^*) \tag{4.83}$$

is normally distributed if all the assumptions for the linear model are satisfied. The error's mean is zero and variance is

$$\begin{aligned} \mathrm{Var}[e^*] &= \mathrm{Var}[Y - \hat{a} + \hat{b}x^*] \\ &= \mathrm{Var}[Y] + \mathrm{Var}[\hat{a} + \hat{b}x^*] \\ &= \sigma^2 + \sigma^2 \left[\frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}} \right]. \end{aligned} \tag{4.84}$$

This result implies that the confidence interval for the response variable Y at an

assigned x^* is

$$\hat{a} + \hat{b}x^* \pm t_{1-\alpha/2,n-2} s \sqrt{1 + \frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}}. \tag{4.85}$$

The confidence interval is shown by the blue lines in Fig. 2.2. It means that 95% of the chance for the Y values to be in between the blue lines. As expected, almost all the Colorado TLR observed data lie between the blue lines. Only 5% of chance a data point would lie outside the blue lines. According to Eq. (4.85), the blue lines also diverge, but very slowly because

$$\frac{(x^* - \bar{x})^2}{S_{xx}} < 0.1310$$

is much smaller than $1 + \frac{1}{n} = 1.0417$.

Figure 4.3 can be produced by the following computer code.

```
#R plot Fig. 4.3: Confidence intervals of a regression model
setwd("/Users/sshen/climstats")
#Confidence interval of the linear model
x1 = seq(max(x), min(x),len=100)
n = 24
xbar = mean(x)
reg = lm(y ~ x)
SSE = sum((reg$residuals)^2)
s_squared = SSE/(length(y)-2)
s = sqrt(s_squared)
modTLR = 33.476216 + -0.006982*x1
xbar = mean(x)
Sxx = sum((x-xbar)^2)
CIupperModel= modTLR +
   qt(.975, df=n-2)*s*sqrt((1/n)+(x1-xbar)^2/Sxx)
CIlowerModel= modTLR -
   qt(.975, df=n-2)*s*sqrt((1/n)+(x1-xbar)^2/Sxx)
CIupperResponse= modTLR +
   qt(.975, df=n-2)*s*sqrt(1+(1/n)+(x1-xbar)^2/Sxx)
CIlowerResponse= modTLR -
   qt(.975, df=n-2)*s*sqrt(1+(1/n)+(x1-xbar)^2/Sxx)

setEPS() #Plot the figure and save the file
postscript("fig0403.eps", height = 8, width = 8)
par(mar=c(4.5,4.5,2.0,0.5))
plot(x,y,
     ylim=c(10,30), xlim=c(1000,3000),
     xlab="Elevation␣[m]",
     ylab=bquote("Temperature␣["~degree~"C]"),
   main="Colorado␣Elevation␣and␣July␣Tmean:␣1981-2010␣Average",
     cex.lab=1.5, cex.axis=1.5)
lines(x1,CIupperModel,type="l",col='red')
lines(x1,CIlowerModel,type="l",col='red')
lines(x1,CIupperResponse,type="l",col='blue')
lines(x1,CIlowerResponse,type="l",col='blue')
abline(reg,lwd=3)
text(2280, 26,
bquote("Temperature␣lapse␣rate:␣7.0"~degree~"C/km"),
cex=1.5)
```

```
text(2350, 27.5,"R-squared = 0.96", cex=1.5)
text(2350, 29, "y= 33.48 - 0.0070 x", cex=1.5)
text(1600, 15,"Blue lines: CI of July Tmean RV",
     col="blue", cex=1.5)
text(1600, 13.5,"Red lines: CI of the fitted model",
     col="red", cex=1.5)
dev.off()
```

```
#Python plot Fig. 4.3: Confidence intervals of regression
reg1 = np.array(np.polyfit(x, y, 1))#regression
abline = reg1[1] + x*reg1[0]
# get confidence intervals
yl,yu,xd,rl,ru = linregress_CIs(x,y)
# plot the figure
fig, ax = plt.subplots(figsize=(13,12))
ax.plot(x,y, 'ko');
ax.plot(x, abline, 'k-');
ax.plot(xd, yu, 'r-')
ax.plot(xd, yl, 'r-')
ax.plot(xd, ru, 'b-')
ax.plot(xd, rl, 'b-')
ax.set_title("Colorado Elevation vs. July Tmean \n \
1981 - 2010 Average",size=25, pad = 20,
             fontweight = 'bold')
ax.set_xlabel("Elevation $[m]$", size = 25, labelpad = 20)
ax.set_ylabel(r"Temperature [$\degree$C]",
              size = 25, labelpad = 20);
ax.tick_params(length=6, width=2, labelsize=20);
ax.set_xticks(np.round(np.linspace(1000, 3000, 5), 2))
ax.set_yticks(np.round(np.linspace(10, 30, 5), 2))
ax.text(1600, 25, r"Temp lapse rate: 7.0$\degree$C/km",
        color= 'k', size = 20)
ax.text(1750, 27,
        r"$y = %1.2f %1.4f x$"%(reg1[1], reg1[0]),
        size = 20)
ax.text(1800, 26, r"$R-squared = %.2f$" % Rsqu,
        color= 'k', size = 20)
ax.text(1050, 15, "Blue lines: CI of July Tmean RV",
        color= 'b', size = 20)
ax.text(1050, 14, "Red lines: CI of the fitted model",
        color= 'r', size = 20)
plt.show()
```

4.1.3.6 When the assumptions for the linear regression model are violated

The computing and plotting of a linear regression is quite straightforward, and have been used widely in data analysis. However, users often are unaware of the assumptions behind the computing and results. You may wish to know what are the assumptions and how they are violated.

(a) Review of the assumptions of linear regression

The linear regression model and its inference have four major assumptions:

(i) Approximate linearity: The data support an approximate linear relationship between x and Y;
(ii) Normality: Both the model error term ϵ and Y are normally distributed;
(iii) Constant variance: ϵ and Y have variance σ^2, which does not change with respect to x;
(iv) Independence of the error term: $\text{Cor}(\epsilon_i, \epsilon_j) = \delta_{ij}$.

Although the derivation of the estimation formulas for $\hat{a}$ and $\hat{b}$ does not need any of the above assumptions, the interpretation and inferences on $\hat{a}$, $\hat{b}$, $\hat{a} + \hat{b}x$, and other quantities are often based on these assumptions. If one or more assumptions are violated, the inference may not work. There are a variety of checking methods and resolution methods in literature, such as using a nonlinear relationship (e.g., a polynomial fitting model). R has a package `lmtest` to test linear regression models. A residual scatter plot (x_i, e_i) may be a first good step to intuitively identify possible violations of the assumptions (See Fig. 4.4). Ideally, the plot shows uniform noise for a good model. Figure 4.4 shows the residuals of the Colorado temperature regression against elevation, and seems showing this kind of noise without an obvious pattern. We may safely conclude that the assumptions of linearity and constant variance are satisfied.

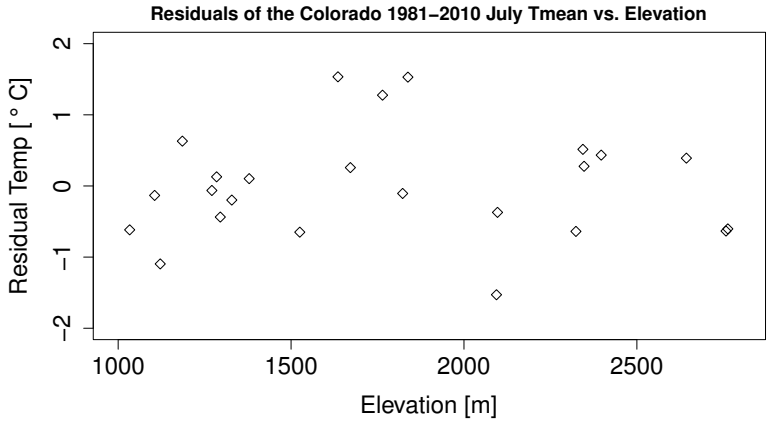


Fig. 4.4 Scatter plot of the residuals of the linear regression for the Colorado July mean temperature data against the elevations. The data are from the 24 USHCN stations in Colorado, and the July Tmean is the 1981-2010 average temperature.

To be rigorous on the statistical inferences about the results of a linear regression, a comprehensive residual analysis may be made to assess whether the studied dataset satisfies the linear regression assumptions. If some assumptions are violated for a given dataset, one may seek other methods to remediate the problems. One way is to make a data transformation or to use a new regression function so that the assumptions are satisfied for the transformed data or the new functional model.

Another way is to use a non-parametric method, which does not assume any distribution. The comprehensive residual analysis methods and the corresponding R and Python codes can be found online. This book is limited to the discussion of a few examples.

Violation of independence assumption (iv) often occurs in cases of y being a time series. In this case there could be correlations from one time to later times. This serial correlation effect leads to fewer independent time intervals, i. e., the degrees of freedom (dof) can be reduced, and the statistical inference needs special attention.

```
#R plot Fig. 4.4: Regression residuals
reg = lm(y~x)
setEPS() #Plot the figure and save the file
postscript("fig0404.eps", height = 4.5, width = 8)
par(mar=c(4.5,4.5,2.0,0.5))
plot(x, reg$residuals, pch=5,
     ylim=c(-2,2), xlim=c(1000,2800),
     xlab="Elevation␣[m]",
     ylab=bquote("Residual␣Temp␣["~degree~"C]"),
     main="Residuals␣of␣the␣Colorado␣July␣Tmean␣vs.␣Elevation",
     cex.lab=1.5, cex.axis=1.5, cex.main = 1.2)
dev.off()
```

```
# Python plot Fig. 4.4: Regression residuals
reg1 = np.array(np.polyfit(x, y, 1))#regression
abline = reg1[1] + x*reg1[0]
# calculate residuals
r = y - abline
fig, ax = plt.subplots(figsize=(12,8))
ax.plot(x, r, 'kd')
ax.set_title("Residuals␣of␣the␣Colorado␣1981␣-␣2010␣July␣\n\
Tmean␣vs.␣Elevation",fontweight = 'bold', size=25, pad = 20)
ax.set_xlabel("Elevation␣$[m]$", size = 25, labelpad = 20)
ax.set_ylabel(r"Residual␣Temp␣[$\degree$C]",
              size = 25, labelpad = 20);
ax.tick_params(length=6, width=2, labelsize=20);
ax.set_xticks(np.linspace(1000, 3000, 5))
ax.set_yticks(np.linspace(-2,2,5))
plt.show()
```

(b) Test for normality

The normality and independence assumptions may not show in the residual scatter plot. You may use a Q-Q plot to test the normality, and Durbin-Watson test to check the independence. In the Colorado TLR example, the Q-Q plot is shown in Fig. 4.5. The residual data quantile points overlap fairly well with the theoretical line for the normal distribution. We may conclude that the data satisfy the normality assumption.

However, the QQ-plot is only a subjective visual test. A quantitative test may be used, such as the Kolmogorov-Smirnov (KS) test or Shapiro-Wilk (SW) test. The KS-test statistic and its p-value for the Colorado TLR data are $D = 0.20833$

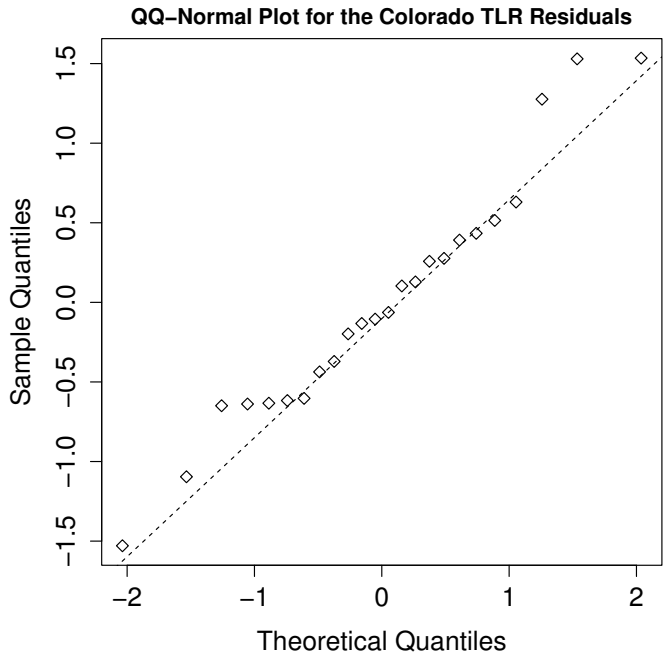


Fig. 4.5 Q-Q plot of the linear regression residuals for Colorado July mean temperature data at the USHCN 24 stations. The diagonal straight line is the theoretical line for the standard normal distribution.

and p-value = 0.686. The null hypothesis is not rejected, i.e., the temperature residuals from the linear regression are not significantly different from the normal distribution. The normality assumption is valid.

Figure 4.5 can generated by the following computer code.

```
#R plot Fig. 4.5: Q-Q plot of quantiles
reg = lm(y~x)
setEPS() #Plot the figure and save the file
postscript("fig0405.eps", height = 6, width = 6)
par(mar=c(4.5,4.5,2.0,0.5))
qqnorm(reg$residuals, pch=5,
       main="QQ-Normal Plot for the Colorado TLR Residuals",
       cex.lab = 1.4, cex.axis = 1.4)
qqline(reg$residuals, lty=2)
dev.off()
```

```
#Python plot Fig. 4.5: Q-Q plot of quantiles
fig, ax = plt.subplots(figsize=(12,8))#fig setup
reg1 = np.array(np.polyfit(x, y, 1))#regression
abline = reg1[1] + x*reg1[0]
r = y - abline# calculate residuals
#Q-Q plot as a probability plot: quantiles vs quantiles
pp1 = scistats.probplot(r, dist="norm", plot=ax,)
ax.set_title("QQ-Normal␣Plot␣for␣Colorado␣TLR␣Residuals",
             fontweight = 'bold',size = 25, pad = 20);
ax.set_ylabel("Sample␣Quantiles", size = 25, labelpad = 20);
ax.set_xlabel("Theoretical␣Quantiles",
              size = 25, labelpad = 20);
ax.tick_params(length=6, width=2, labelsize=20);
ax.set_xticks(np.round(np.linspace(-2, 2, 5), 2))
plt.show()
```

(c) Serial correlation

The R package `lmtest` has a function to test for the independence of the error term, using the Durbin-Watson (DW) test statistic defined as

$$DW = \frac{\sum_{i=2}^{n}(e_i - e_{i-1})^2}{\sum_{i=1}^{n} e_i^2}. \tag{4.86}$$

The DW test is meant for a time series. We thus sort the Tmean data according to the ascending order of elevation, and then compute DW.

When $\text{cor}(e_i, e_j) = \sigma^2\delta_{ij}$, we have $DW \approx 2$. For the Colorado July TLR data, $DW = 2.3072$ and $p - value = 0.7062$, which can be computed by the following code:

```
#R code for the DW-test for independence
#install.packages("lmtest")
library(lmtest)
ElevTemp=cbind(x,y, 1:24)
#Sort the data for ascending elevation
ElevTemp=ElevTemp[order(x),]
reg1=lm(ElevTemp[,2] ~ ElevTemp[,1])
dwtest(reg1)
#DW = 2.3072, p-value = 0.7062
```

```
#Python code for the DW-test for independence
from statsmodels.stats.stattools import durbin_watson
orderET = np.arange(1,25)
ElevTemp = np.stack((x,y, orderET), axis = -1)
sorted_ind = np.argsort(ElevTemp[:,0],kind='mergesort')
dat1 = ElevTemp[sorted_ind]
# use the first order polynomial fit for linear regression
reg1 = np.array(np.polyfit(dat1[:,0], dat1[:,1], 1))
abline = reg1[1] + dat1[:,0]*reg1[0]
r = dat1[:,1] - abline# calculate residuals
#perform Durbin-Watson test
durbin_watson(r)
#2.307190038542777
```

The large p-value 0.7062 implies that the null hypothesis of no serial correlation (i.e., independence) is not rejected, because DW 2.3072 is not too far away from 2, which indicates independence of the error terms. Some literatures use $1.5 < \text{DW} < 2.5$ to conclude independence without checking the p-value.

When the independence assumption is violated, the data have serial correlation, which reduces dof and hence enlarges the confidence interval for the regression results and makes the results not as reliable.

(d) Non-parametric trend inference and heteroskedasty

Even when some of the four assumptions for the linear regression model are invalid, one can still estimate $\hat{a}$ and $\hat{b}$ and the linear model $\hat{a} + \hat{b}x$ in the same way and can still interpret the study subject using the domain knowledge. In this case, the statistical inference based on the normality assumption may not make any sense, and the confidence interval calculations based on the t-distribution are generally invalid. Now, you may use a non-parametric test to make an inference on the residuals. For example, a non-parametric test for the trend can be applied, such as Mann-Kendall (MK) trend test, and Theil-Sen's trend estimation and test. The trend, if presenting, can be linear or nonlinear. These tests are meant for time series. Thus, before applying these tests, the data should be sorted according to the ascending order of the explainable variable. For the Colorado TLR example, the MK-test for the linear regression residual can be computed by the following code.

```
#R code for the Mann-Kendall test
#install.packages("trend")
library(trend)
ElevTemp=cbind(x, y, 1:24)
#Sort the data for ascending elevation
ElevTemp=ElevTemp[order(x),]
reg1=lm(ElevTemp[,2] ~ ElevTemp[,1])
ElevTemp[,3]=reg1$residuals
mk.test(ElevTemp[,3])
#data:  ElevTemp[, 3]
#z = 0.47128, n = 24, p-value = 0.6374
mk.test(ElevTemp[,2])
#z = -5.9779, n = 24, p-value = 2.261e-09
```

```
#Python code for the Mann-Kendall test
import pymannkendall as mk
orderET = np.arange(1,25)
ElevTemp = np.stack((x,y, orderET), axis = -1)
sorted_ind = np.argsort(ElevTemp[:,0],kind='mergesort')
dat1 = ElevTemp[sorted_ind]
# use the first order polynomial fit for linear regression
reg1 = np.array(np.polyfit(dat1[:,0], dat1[:,1], 1))
abline = reg1[1] + dat1[:,0]*reg1[0]
r = dat1[:,1] - abline# calculate residuals
#perform Durbin-Watson test
dat1[:,2] = r
print(mk.original_test(dat1[:,2]))#test for residual trend
#p=0.6374381847429542, z=0.47128365713220055
print(mk.original_test(dat1[:,1]))#test for temp trend
#p=2.260863496417187e-09, z=-5.97786112467686
```

The large p-value `0.6374` of the MK-test for residuals implies that the residuals have no trend, linear or nonlinear. The small p-value `2.261e-09` of the MK-test for the sorted temperature implies that the sorted July Tmean according to elevation has a significant trend.

The linear trend is frequently used to explain the global average temperature. The temperature data with respect to time is clearly nonlinear. When the scatter plot shows a clear pattern, then one or more of the four assumptions are usually violated. When that happens, you may use different ways to remediate this situation. For example, when linearity is violated, the problem may be intrinsically nonlinear and a polynomial model may be a better fit than a linear model. When the constant variance is violated, known as heteroskedastic data in statistics literature, then the data standardization (i.e., the anomaly data being divided by the data standard deviation) or logarithmic transform for the positive-valued data, may be used to transform the data and to make the data homoskedastic, i.e., constant variance.

4.2 Multiple linear regression

This section uses the method of matrices, which will be described in Chapter 5. If you are not familiar with matrices, you may come back to this section after reading the first two sections of Chapter 5.

4.2.1 Calculating the Colorado TLR when taking location coordinates into account

In the previous simple linear regression, the July temperature was assumed to linearly depend only on the vertical coordinate of the station: elevation. However,

the temperature may also depend on the horizontal coordinates of the station: latitude and longitude. This subsection deals with this kind of problem of more than one explanatory variable. A multivariate linear regression model (or called multiple linear regression model) can be expressed as follows:

$$Y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_m x_m + \epsilon, \tag{4.87}$$

where $x_1, x_2, \cdots, x_m$ are m explanatory variables, which are non-random and deterministic variables; ϵ is the model error with zero mean and variance σ^2 being a constant; Y is the response variable, which is random and has its variance equal to σ^2, and expected value equal to

$$\mathrm{E}[Y] = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_m x_m; \tag{4.88}$$

and $b_0, b_1, b_2, \cdots, b_m$ are parameters to be estimated from data $(x_{ij}, y_j), i = 1, 2, \cdots, m$, and $j = 1, 2, \cdots, n$.

In the Colorado July TLR example, we have x_1 as latitude, x_2 as longitude, x_3 as elevation, Y as the July air temperature, $m = 3$, and $n = 24$. The latitude and longitude data are as follows:

```
lat=c(
39.9919, 38.4600, 39.2203, 38.8236, 39.2425, 37.6742,
39.6261, 38.4775, 40.6147, 40.2600, 39.1653, 38.5258,
37.7717, 38.0494, 38.0936, 38.0636, 37.1742, 38.4858,
8.0392, 38.0858, 40.4883, 37.9492, 37.1786, 40.0583
)
lon=c(
-105.2667, -105.2256, -105.2783, -102.3486, -107.9631, -106.3247,
-106.0353, -102.7808, -105.1314, -103.8156, -108.7331, -106.9675,
-107.1097, -102.1236, -102.6306, -103.2153, -105.9392, -107.8792,
-103.6933, -106.1444, -106.8233, -107.8733, -104.4869, -102.2189
)
```

The elevation and temperature data are the same as those at the beginning of this chapter.

The computer code for the multiple linear regression of three variables is as follows.

```
#R code for the TLR multivariate linear regression
elev = x; temp = y #The x and y data were entered earlier
dat = cbind(lat, lon, elev, temp)
datdf = data.frame(dat)
datdf[1:2,] #Show the data of the first two stations
#       lat        lon   elev     temp
# 39.9919 -105.2667 1671.5 22.064
# 38.4600 -105.2256 1635.6 23.591

#Multivariate linear regression
reg=lm(temp ~ lat + lon + elev, data = datdf)
```

```
summary(reg)  #Display the regression results
#                 Estimate   Std. Error   t value  Pr(>|t|)
#(Intercept) 36.4399561  9.4355746   3.862 0.000971 ***
#  lat         -0.4925051  0.1320096  -3.731 0.001319 **
#  lon         -0.1630799  0.0889159  -1.834 0.081564 .
#  elev        -0.0075693  0.0003298 -22.953 7.67e-16 ***
#Residual standard error: 0.6176 on 20 degrees of freedom
#Multiple R-squared:  0.979
```

```
#Python code for the TLR multivariate linear regression
from sklearn import linear_model
lat=np.array([
39.9919, 38.4600, 39.2203, 38.8236, 39.2425, 37.6742,
39.6261, 38.4775, 40.6147, 40.2600, 39.1653, 38.5258,
37.7717, 38.0494, 38.0936, 38.0636, 37.1742, 38.4858,
8.0392, 38.0858, 40.4883, 37.9492, 37.1786, 40.0583
])
lon=np.array([
-105.2667,-105.2256,-105.2783,-102.3486,-107.9631,-106.3247,
-106.0353,-102.7808,-105.1314,-103.8156,-108.7331,-106.9675,
-107.1097,-102.1236,-102.6306,-103.2153,-105.9392,-107.8792,
-103.6933,-106.1444,-106.8233,-107.8733,-104.4869,-102.2189
])
elev = x; temp = y #The x and y data were entered earlier
dat = np.stack((lat, lon, elev), axis = -1)
print(dat[0:2,:]) #Show the data of the first two stations
#[[  39.9919 -105.2667 1671.5   ]
# [  38.46   -105.2256 1635.6   ]]
#Multivariate linear regression
xdat = dat
ydat = temp
regr = linear_model.LinearRegression()
multi_reg = regr.fit(xdat, ydat)
print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)
```

According to this three-variable linear regression, the TLR is the regression coefficient for elevation, i.e., -0.0075694. This means 7.6 °C/km, larger than the one estimated earlier using only one variable: elevation.

The quantile of $t_{0.975,20} = 2.085963$. The dof is 20 now because four parameters have been estimated. The confidence interval of TLR can be computed below

$$\begin{aligned}&(-0.0075694 - 2.085963 \times 0.0003298, -0.0075694 + 2.085963 \times 0.0003298)\\ &= (-0.008257351, -0.006881449).\end{aligned} \tag{4.89}$$

The confidence interval for TLR at 95% confidence level is $(6.9, 8.3)$°C/km.

The R^2 values is very large: 0.979, which means that the linear model data can explain 98% of the variance of observed temperature data.

4.2.2 Formulas for estimating parameters in the multiple linear regression

The n groups of data for the explainable $x_1, x_2, \cdots, x_m$ and response variable Y are as follows:

$$x_{ij}, i = 1, 2, \cdots, m, \quad \text{and} \quad j = 1, 2, \cdots, n; \tag{4.90}$$

and

$$y_1, y_2, \cdots, y_n \tag{4.91}$$

The data and their corresponding regression coefficients are written in the matrix form as follows:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{4.92}$$

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{m1} \\ 1 & x_{12} & x_{22} & \cdots & x_{m2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1n} & x_{2n} & \cdots & x_{mn} \end{bmatrix} \tag{4.93}$$

$$\hat{\mathbf{b}} = \begin{bmatrix} \hat{b}_0 \\ \hat{b}_1 \\ \hat{b}_2 \\ \vdots \\ \hat{b}_n \end{bmatrix} \tag{4.94}$$

The data, linear model prediction, and the corresponding residuals (i.e., the prediction errors of the linear model) can be written as follows:

$$\mathbf{y}_{n\times 1} = \mathbf{X}_{n\times(m+1)}\hat{\mathbf{b}}_{(m+1)\times 1} + \mathbf{e}_{n\times 1} \tag{4.95}$$

where the residual vector is

$$\mathbf{e}_{n\times 1} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \tag{4.96}$$

Multiplying Eq. (4.95) by $\mathbf{X}^t$ from the left and enforcing

$$(\mathbf{X}^t)_{(m+1)\times n}\mathbf{e}_{n\times 1} = \mathbf{0}, \tag{4.97}$$

you can obtain the estimate of the regression coefficients

$$\hat{\mathbf{b}}_{(m+1)\times 1} = \left[(\mathbf{X}^t)_{(m+1)\times n}\mathbf{X}_{n\times(m+1)}\right]^{-1}(\mathbf{X}^t)_{(m+1)\times n}\mathbf{y}_{n\times 1}. \tag{4.98}$$

Condition (4.97) means the each column vector of the $\mathbf{X}_{n\times(m+1)}$ matrix is perpendicular to the residual vector $\mathbf{e}_{n\times 1}$. For the first column, the condition corresponds to the assumption of zero mean of the model error:

$$\sum_{i=1}^{n} e_i = 0. \tag{4.99}$$

For the remaining columns, the condition means that the residual vector is perpendicular to the data vector for each explanatory variable. This implies that the residual vectors Euclidean distance to the data vectors are minimized, i.e., the minimizing sum of squared errors (SSE). Therefore, Equation (2.83) is the least square estimate of the regression coefficients.

As illustrated in the previous subsection, to implement the R estimate of the regression coefficients, we put data $\mathbf{X}$ and $\mathbf{y}$ in a single matrix in the form of `datdf=data.frame(cbind(X,y))` with proper column names, such as

```
colnames(datdf) <- c('x1', 'x2', 'x3', 'y')
```

for the case of three explanatory variables. Then, use the R command to make the linear model estimate

```
reg=lm(y ~ x1 + x2 + x3, data = datdf)
```

Finally, `summary(reg)` outputs all the important regression results.

The R command `reg$` allows to display the specific result, such as regression coefficients

```
round(reg$coefficients, digits=5)
# (Intercept)         lat         lon        elev
#    36.43996    -0.49251    -0.16308    -0.00757
```

Thus, the multiple linear model for the Colorado July temperature is

$$\text{Temp}[^\circ\text{C}] = 36.43527 - 0.49255\times\text{Lat} - 0.16314\times\text{Lon} - 0.00757\times\text{Elev [m]}. \tag{4.100}$$

The confidence interval for the linear model

$$\hat{Y} = b_0 + b_1x_1 + b_2x_2 + \cdots + b_mx_m \tag{4.101}$$

at a given point $\mathbf{x}^* = (1, x_1^*, x_2^*, \cdots, x_m^*)$ is given by the following formula

$$\hat{\mathbf{b}}^t\mathbf{x}^* \pm t_{1-\alpha/2,n-m}\, s\sqrt{(\mathbf{x}^*)^t(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{x}^*)}. \tag{4.102}$$

The confidence interval for the response variable Y at a given point

$$\mathbf{x}^* = (1, x_1^*, x_2^*, \cdots, x_m^*)$$

is

$$\hat{\mathbf{b}}^t\mathbf{x}^* \pm t_{1-\alpha/2,n-m}\, s\sqrt{1 + (\mathbf{x}^*)^t(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{x}^*)}, \tag{4.103}$$

where s is the same as that defined earlier for the simple linear regression:

$$s = \sqrt{\frac{SSE}{n-2}} \tag{4.104}$$

If $m = 1$, the above two formulas are the same as those for the confidence intervals given by Eqs. (4.82) and (4.85) for the simple linear regression.

4.3 Nonlinear fittings using the multiple linear regression

4.3.1 Diagnostics of linear regression: An example of global temperature

When the data have strong nonlinearity, the scatter plot of residuals will show obvious patterns, such as that shown in Fig. 4.6 for the linear regression of the global average annual mean surface air temperature anomalies from 1880 to 2018 with respect to the 1971-2000 climatology. The linearity assumption of the simple linear regression is clearly violated. The independence assumption is also violated.

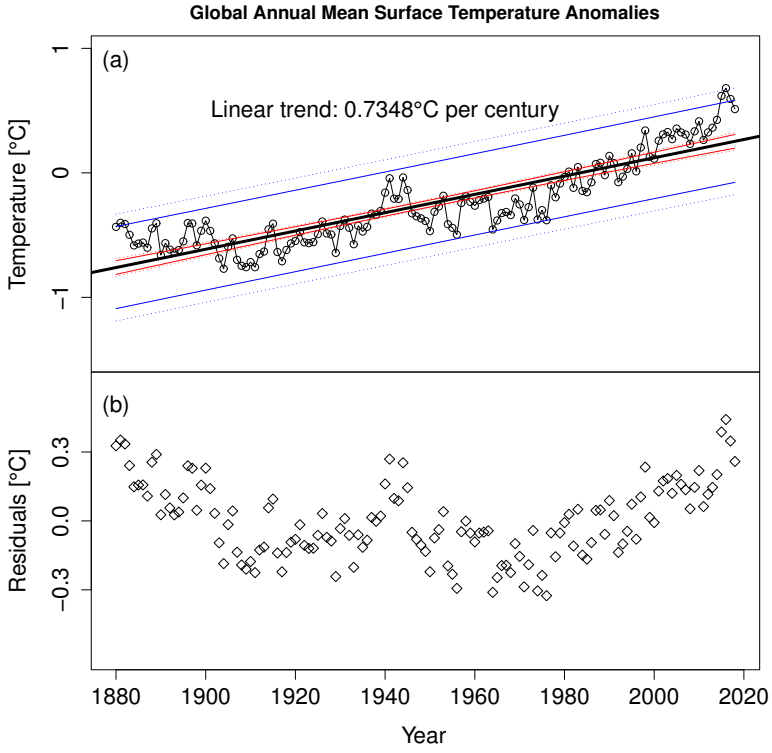


Fig. 4.6 (a) Linear regression of the global average annual mean land and ocean surface air temperature anomalies with respect to the 1971-2000 climatology based on the NOAAGlobalTemp dataset (Zhang et al. 2019); (b) Scatter plot of the linear regression residuals.

The following computer code can plot Fig. 4.6 and make the diagnostics of the linear regression.

```
#R plot Fig. 4.6: Regression diagnostics
```

```
setwd("/Users/sshen/climstats")
dtmean<-read.table(
  "data/aravg.ann.land_ocean.90S.90N.v5.0.0.201909.txt",
  header=F)
dim(dtmean)
#[1] 140   6
x = dtmean[1:139,1]
y = dtmean[1:139,2]
reg =  lm(y ~ x) #linear regression
reg
#(Intercept)       yrtime
#-14.574841     0.007348

#Confidence interval of the linear model
xbar = mean(x)
SSE = sum((reg$residuals)^2)
s_squared = SSE/(length(y)-2)
s = sqrt(s_squared)
modT = -14.574841 + 0.007348 *x
xbar = mean(x)
Sxx = sum((x-xbar)^2)
n = length(y)
CIupperModel= modT +
  qt(.975, df=n-2)*s*sqrt((1/n)+(x-xbar)^2/Sxx)
CIlowerModel= modT -
  qt(.975, df=n-2)*s*sqrt((1/n)+(x-xbar)^2/Sxx)
CIupperResponse= modT +
  qt(.975, df=n-2)*s*sqrt(1+(1/n)+(x-xbar)^2/Sxx)
CIlowerResponse= modT -
  qt(.975, df=n-2)*s*sqrt(1+(1/n)+(x-xbar)^2/Sxx)

CIupperModelr= modT +
  qt(.975, df=5)*s*sqrt((1/n)+(x-xbar)^2/Sxx)
CIlowerModelr= modT -
  qt(.975, df=5)*s*sqrt((1/n)+(x-xbar)^2/Sxx)
CIupperResponser= modT +
  qt(.975, df=5)*s*sqrt(1+(1/n)+(x-xbar)^2/Sxx)
CIlowerResponser= modT -
  qt(.975, df=5)*s*sqrt(1+(1/n)+(x-xbar)^2/Sxx)

setEPS() #Plot the figure and save the file
postscript("fig0406.eps", height = 8, width = 8)
par(mfrow=c(2,1))
par(mar=c(0,4.5,2.5,0.7))
plot(x, y,  ylim = c(-1.5, 1),
     type="o", xaxt="n", yaxt="n",
     cex.lab=1.4, cex.axis=1.4,
     xlab="Year", ylab=bquote("Temperature␣["~degree~"C]"),
     main="Global␣Annual␣Mean␣Surface␣Temperature␣Anomalies",
     cex.lab=1.4, cex.axis=1.4
)
axis(side = 2, at = c(-1.0, 0, 1.0), cex.axis = 1.4)
abline(reg, col="black", lwd=3)
lines(x,CIupperModel,type="l",col='red')
lines(x,CIlowerModel,type="l",col='red')
lines(x,CIupperResponse,type="l",col='blue')
```

```
5244 lines(x,CIlowerResponse,type="l",col='blue')
5245
5246 lines(x,CIupperModelr,type="l", lty = 3, col='red')
5247 lines(x,CIlowerModelr,type="l", lty = 3, col='red')
5248 lines(x,CIupperResponser,type="l",lty = 3, col='blue')
5249 lines(x,CIlowerResponser,type="l",lty = 3, col='blue')
5250
5251 text(1940, 0.5,
5252      bquote("Linear␣trend:␣0.7348"~degree~"C␣per␣century"),
5253      col="black",cex=1.4)
5254 text(1880, 0.9, "(a)", cex=1.4)
5255 par(mar=c(4.5,4.5,0,0.7))
5256 plot(x, reg$residuals, ylim = c(-0.6,0.6),
5257      pch=5, cex.lab=1.4, cex.axis=1.4,
5258      yaxt = 'n', xlab="Year",
5259      ylab=bquote("Residuals␣["~degree~"C]"))
5260 axis(side = 2, at = c(-0.3, 0, 0.3), cex.axis = 1.4)
5261 text(1880, 0.5, "(b)", cex=1.4)
5262 dev.off()
```

```
#Python plot Fig. 4.6: Regression diagnostics
# Change current working directory
os.chdir("/Users/sshen/climstats/")
# Read the data
dtmean = np.array(read_table(
    "data/aravg.ann.land_ocean.90S.90N.v5.0.0.201909.txt",
              header = None, delimiter = "\s+"))
xDT = dtmean[0:139, 0]
yDT = dtmean[0:139, 1]
# Make a linear fit, i.e., linear regression
regLin = np.array(np.polyfit(xDT, yDT, 1))
ablinereg = regLin[1] + xDT*regLin[0]
yld,yud,xdd,rld,rud = linregress_CIs(xDT,yDT)

# Plot the figure
fig, ax = plt.subplots(2, 1, figsize=(12,12))
ax[0].plot(xDT, yDT, 'ko-')
ax[0].plot(xDT, ablinereg, 'k-')
ax[0].plot(xdd, yld, 'r-')
ax[0].plot(xdd, yud, 'r-')
ax[0].plot(xdd, rld, 'b-')
ax[0].plot(xdd, rud, 'b-')
ax[1].plot(xDT, yDT - ablinereg, 'kd')
ax[0].set_title(
    "Global␣Annual␣Mean␣Land␣and␣Ocean␣Surface␣\n\
Temperature␣Anomolies", fontweight = 'bold',
                size = 25, pad = 20)
ax[0].set_ylabel("Temperature␣[$\degree$C]",
                 size = 25, labelpad = 20)
ax[0].tick_params(length=6, width=2, labelsize=20);
ax[0].set_yticks(np.round(np.linspace(-1, 1, 5), 2))
ax[0].text(1880, 0.4,
        "Linear␣temp␣trend␣0.7348␣deg␣C␣per␣century",
           color = 'k', size = 20)
ax[0].text(1877, 0.6, "(a)", size = 20)
ax[0].axes.get_xaxis().set_visible(False)
ax[1].tick_params(length=6, width=2, labelsize=20);
ax[1].set_yticks(np.round(np.linspace(-0.5, 0.5, 5), 2))
ax[1].set_ylabel("Residuals␣[$\degree$C]",
                 size = 25, labelpad = 20)
ax[1].text(1877, 0.4, "(b)", size = 20)
ax[1].set_xlabel("Year", size = 25, labelpad = 20)
fig.tight_layout(pad=-1.5)
plt.show()
```

Based on the visual check of Fig. 4.6(b), the constant variance assumption seems satisfied. The KS-test shows that the normality assumption is also satisfied.

```
#Kolmogorov-Smirnov (KS) test for normality
library(fitdistrplus)
resi_mean = mean(reg$residuals)
resi_sd = sd(reg$residuals)
test_norm = rnorm(length(reg$residuals),
                  mean = 0, sd = 1)
```

```
testvar = (reg$residuals - resi_mean)/resi_sd
ks.test(testvar, test_norm)
#D = 0.057554, p-value = 0.9754
#The normality assumption is accepted

#Diagnostics on independence and normality
# Durbin-Watson (DW) test for independence
dwtest(reg)
#DW = 0.45235, p-value < 2.2e-16
#The independence assumption is rejected

#degrees of freedom and critical t values
rho1 = acf(y)[[1]][2] #Auto-correlation function
rho1 #[1] 0.9270817
edof = (length(y) - 2)*(1 - rho1)/(1 + rho1)
edof #[1] 5.183904 effective degrees of freedom
qt(.975, df=137) #[1] 1.977431 critical t value
qt(.975, df=5) #[1] 2.570582 critical t value
```

```
#Kolmogorov-Smirnov (KS) test for normality
import statistics
from scipy.stats import kstest
resi = yDT - ablinereg
testvar = (resi - np.mean(resi))/statistics.stdev(resi)
kstest(testvar, 'norm')#
#KstestResult(statistic=0.0751, pvalue=0.3971)
#The normality assumption is accepted

#perform Durbin-Watson test for independence
from statsmodels.stats.stattools import durbin_watson
durbin_watson(resi)
#0.45234915992769864 not in (1.5, 2.5)
#The independence assumption is rejected

#calculate autocorrelations of temp data for edof
import statsmodels.api as sm
autocorr = sm.tsa.acf(yDT)
rho1 = autocorr[1]
edof = (yDT.size - 2)*(1 - rho1)/(1 + rho1)
edof #[1] 5.183904 effective degrees of freedom

#Compare the crtical t values with different edof
import scipy.stats
scipy.stats.t.ppf(q=.975, df=137)
#critical t value 1.9774312122928936
scipy.stats.t.ppf(q=.975, df=5)
##critical t value 2.5705818366147395
```

The DW-test shows that the independence assumption is violated. This implies the existence of serial correlation, which in turn implies a smaller dof, and hence larger $t_{1-\alpha,\text{dof}}$. Thus, the confidence intervals for model $\hat{Y}$ are wider than the red lines in Fig. 4.6(a), and those for the Y values are wider than the blue lines in

the same figure. Therefore, the regression results are less reliable. In this case, you need to compute the effective dof (edof), which is less than $n-2$. For the NOAAGlobalTemp time series, the one-year time lag serial correlation is $\rho_1 = 0.9271$, the edof under the assumption of an autoregression one process AR(1) is approximately equal to

$$\text{edof} = \frac{1-\rho_1}{1+\rho_1} \times \text{dof}. \tag{4.105}$$

Consequently, the effective dof is reduced to 5, compared to the non-serial correlation case dof 137. R can compute `qt(.975, df=137)` and yield 1.977431, and compute `qt(.975, df=5)` and yield 2.570582. Thus, the actual confidence interval at the 95% confidence level for the linear regression is about 25% wider. Namely, the dotted color lines are wider than the solid color lines.

The W pattern of the residuals shown in Fig. 4.6(b) implies that the linearity assumption is violated. To remediate the nonlinearity, we fit the data to a nonlinear model and hope the residuals do not show clear patterns. We will use the third-order polynomial to illustrate the procedure.

4.3.2 Fit a third order polynomial

Figure 4.7(a) shows the third order polynomial fitting to the NOAAGlobalTemp dataset:

$$y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + \epsilon. \tag{4.106}$$

This fitting can be estimated by the method of multiple linear regression with three variables

$$x_1 = x, x_2 = x^2, x_3 = x^3. \tag{4.107}$$

Then, a computer code for the multiple linear regression can be used to estimate the coefficients of the polynomial and plot Fig. 4.7.

```
#R plot Fig. 4.7: Polynomial fitting
x1=x
x2=x1^2
x3=x1^3
dat3=data.frame(cbind(x1,x2,x3,y))
reg3 = lm(y ~ x1 + x2 + x3, data=dat3)
# simply use
# reg3 = lm(y ~ x + I(x^2) + I(x^3))
setEPS() #Plot the figure and save the file
postscript("fig0407.eps", height = 8, width = 8)
par(mfrow=c(2,1))
par(mar=c(0,4.5,2.5,0.7))
plot(x, y,  type="o", xaxt="n",
     cex.lab=1.4, cex.axis=1.4, xlab="Year",
     ylab=bquote("Temperature␣["~degree~"C]"),
     main="Global␣Annual␣Mean␣Surface␣Temperature␣Anomalies",
     cex.lab=1.4, cex.axis=1.4)
lines(x, predict(reg3), col="black", lwd=3)
reg3
```

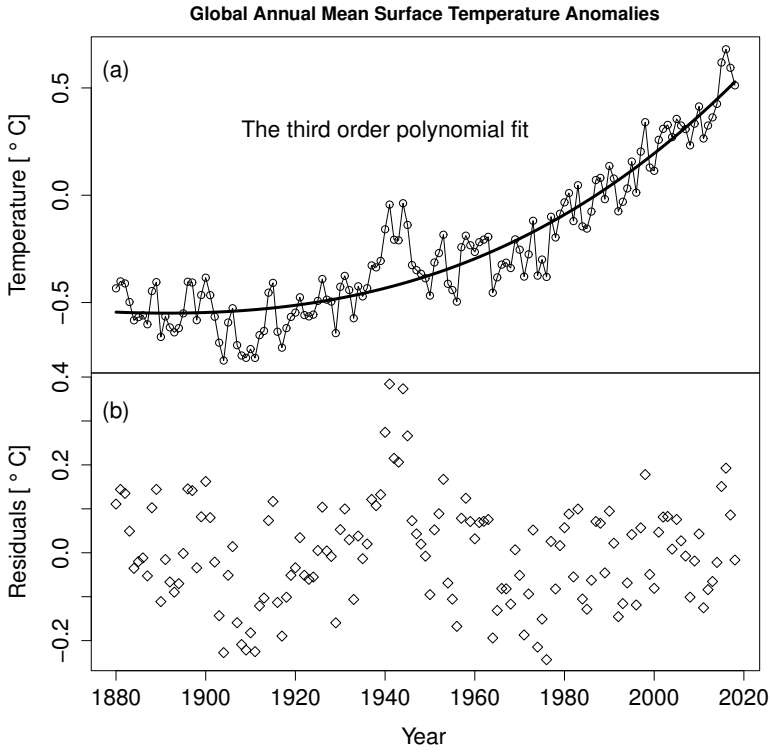


Fig. 4.7 (a) Fit a third order polynomial to the global average annual mean land and ocean surface air temperature anomalies with respect to the 1971-2000 climatology based on the NOAAGlobalTemp dataset (Zhang et al. 2019); (b) Scatter plot of the corresponding residuals.

```
#(Intercept)            x1            x2            x3
#-1.426e+03    2.333e+00   -1.271e-03    2.308e-07
text(1940, 0.3,
     "The␣third␣order␣polynomial␣fit",
     col="black",cex=1.4)
text(1880, 0.58, "(a)", cex=1.4)
par(mar=c(4.5,4.5,0,0.7))
plot(x1, reg3$residuals,
     pch=5, cex.lab=1.4, cex.axis=1.4,
     xlab="Year", ylab=bquote("Residuals␣["~degree~"C]"))
text(1880, 0.32, "(b)", cex=1.4)
dev.off()
```

```
os.chdir("/Users/sshen/climstats/")
# Read the data
dtmean = np.array(read_table(
    "data/aravg.ann.land_ocean.90S.90N.v5.0.0.201909.txt",
                header = None, delimiter = "\s+"))
xDT = dtmean[0:139, 0]
yDT = dtmean[0:139, 1]
# Create trend line
reg3 = np.array(np.polyfit(xDT, yDT, 3))
ablineDT3 = reg3[3] + xDT*reg3[2] + \
           (xDT**2)*reg3[1] + (xDT**3)*reg3[0]
fig, ax = plt.subplots(2, 1, figsize=(12,12))
ax[0].plot(xDT, yDT, 'ko-')
ax[0].plot(xDT, ablineDT3, 'k-')

ax[1].plot(xDT, yDT - ablineDT3, 'kd')
ax[0].set_title("Global␣Annual␣Mean␣Land␣and␣Ocean␣\n\
␣␣␣␣␣␣␣␣␣␣␣␣Surface␣Temperature␣Anomolies",
             fontweight = 'bold', size = 25, pad = 20)
ax[0].tick_params(length=6, width=2, labelsize=20);
ax[0].set_yticks(np.round(np.linspace(-1, 1, 5), 2))
ax[0].set_ylabel("Temperature␣[$\degree$C]",
                 size = 25, labelpad = 20)
ax[0].text(1900, 0.3, "The␣third-order␣polynomial␣fit",
           color = 'k', size = 20)
ax[0].text(1877, 0.6, "(a)", size = 20)
ax[0].axes.get_xaxis().set_visible(False)
ax[1].tick_params(length=6, width=2, labelsize=20);
ax[1].set_yticks(np.round(np.linspace(-0.4, 0.4, 5), 2))
ax[1].set_ylabel("Residuals␣[$\degree$C]",
                 size = 25, labelpad = 20)
ax[1].set_xlabel("Year", size = 25, labelpad = 20)
ax[1].text(1877, 0.35, "(b)", size = 20)
fig.tight_layout(pad=-1.5)
plt.show()
```

R has another simpler command for the third order polynomial regression:

```
lm(y ~ poly(x, 3, raw=TRUE))
```

where, `raw=TRUE` means using the raw polynomial form written like formula (4.106). Another option is `raw=FALSE`, which means that the data are fitted to an orthogonal polynomial.

Comparison of Figs. 4.6(b) and 4.7(b) shows that the W-shape pattern of the residuals in the third-order polynomial fitting is weaker than that for the linear regression. Nonetheless, the W-shape pattern is still clear. Figure 4.7(b) visually suggests that the constant variance assumption is satisfied. The KS-test shows that the normality assumption is also satisfied. However, the DW-test shows the existence of serial correlation. The computing of these tests are left as exercise problems.

You can try to fit a ninth order orthogonal polynomial. This can eliminate the W-

shape nonlinear pattern of the residuals. You can also try to fit other polynomials or functions.

4.4 Linear Regression by Weighted Least Squares

The usual regression minimizes the mean square error:

$$\epsilon^2 = \sum_i (y_i - \hat{y}_i)^2. \tag{4.108}$$

However, sometimes the square erorr may have different importance for different observations i. The importance is quantified by a weight w_i. Thus, the weight least square error is defined as

$$\varepsilon^2 = \sum_i w_i (y_i - \hat{y}_i)^2. \tag{4.109}$$

The weights w_i are often required to be non-negative and sometimes to be normalized

$$\sum_i w_i = 1, \quad w_i \geq 0, \quad i = 1, 2, 3, \cdots \tag{4.110}$$

As an example of weights, the gridded global climate data are often defined on a uniform latitude-longitude grid. A grid box covers more area over the equatorial region than a polar region. From the spherical coordinates, you can find that the area-weight is $\cos\phi_i$, where ϕ_i is the latitude of the grid box's centroid.

The weighted least squares regression can be converted into the usual least square regression for the weighted field and data:

$$U_i = \sqrt{w_i} y_i, \quad \hat{U}_i = \sqrt{w_i}\hat{y}_i, \quad i = 1, 2, 3, \cdots \tag{4.111}$$

Here, $\sqrt{w_i}$ are often referred to as weight-factor. In climate science, $\sqrt{w_i} = \sqrt{\cos\phi_i}$ are called area-factor. In terms of the weighted data, the weighted least squares becomes the usual least squares:

$$\varepsilon^2 = \sum_i (U_i - \hat{U}_i)^2. \tag{4.112}$$

When considering 3D data in space, we may need to introduce the volume-factor $\sqrt{V_i}$. When heat capacity, mass-density and other physical or chemical properties in addition to geometry need to be considered, then a more complex weight should be used. For more examples, see Bui et al. (2023), Lafarga et al. (2023)), and Shen and North (2023).

4.5 Chapter summary

This chapter has introduced three regression methods commonly used in climate science: (a) simple linear regression of a single variate, (b) multiple linear regression, and (c) nonlinear fitting. For (a) and (b) we used the data of surface air temperature and elevation of the 24 USHCN stations in the state of Colorado, USA. For (c) we used the NOAAGlobalTemp's global average annual mean temperature data from 1880 to 2018.

A linear regression model has four fundamental assumptions:

(i) Approximate linearity between x and Y,
(ii) Normal distribution of ϵ and Y,
(iii) Constant variance of ϵ and Y, and
(iv) Independence of error terms: $\text{Cor}(\epsilon_i, \epsilon_j) = \sigma^2 \delta_{ij}$.

Assumptions (i) and (iiii) can be verified by visually examining the scatter plot of residuals. Assumption (ii) may be verified by the KS-test. Assumption (iv) can be verified by the DW-test. When one or more assumptions are violated, you may improve your model, such as using a nonlinear model, or transforming the data.

References and Further Readings

[1] Cordova, M., R. Celleri, C.J. Shellito, J. Orellana-Alvear, A. Abril, and G. Carrillo-Rojas, 2016: Near-surface air temperature lapse rate over complex terrain in the Southern Ecuadorian Andes: implications for temperature mapping. *Arctic, Antarctic, and Alpine Research*, 48, 673 - 684.

Figure 3 of this paper shows the linear regression with $R^2 = 0.98$ for the temperature data of nine stations whose elevations are in the range from 2,610 to 4,200 meters. The regression implies a lapse rate of 6.88 °C/km for the annual mean temperature.

[2] Fall, P.L., 1997: Timberline fluctuations and late Quaternary paleoclimates in the Southern Rocky Mountains, Colorado. *Geological Society of America Bulletin*, 109, 1306-1320.

Figure 2a of this paper shows a lapse rate of 6.9 °C/km for the mean July temperature based on the data of 104 stations and a linear regression with $R^2 = 0.86$. Figure 2b shows the annual mean temperature TLR equal to 6.0 °C/km with $R^2 = 0.80$.

[3] Graybill, F.A., H.K. Iyer, 1994: *Regression Analysis: Concepts and Applications.* Duxbury Press, Belmont, California, 701pp.

This book clearly outlines the assumptions of regression. It focuses on concepts and applications about solving practical statistical problems.

[4] Menne, M.J., C.N. Williams, and R.S. Vose, 2009: The United States Historical Climatology Network monthly temperature data Version 2. *Bulletin of the American Meteorological Society*, 90, 993-1007.

This paper is one of the series of papers on the USHCN datasets prepared at the NOAA National Centers for Environmental Information and have been widely used since the early 1990s. The network includes 1,218 stations and has both monthly and daily data of temperature and precipitation publicly available. URL: `https://www.ncdc.noaa.gov/ushcn`

[5] Zhang, H.-M., B. Huang, J. Lawrimore, M. Menne, T. M. Smith, 2019: NOAA Global Surface Temperature Dataset (NOAAGlobalTemp), Version 5.0 (Time Series). NOAA National Centers for Environmental Information. doi:10.7289/V5FN144H [Access date: March 2021].

NOAA Merged Land Ocean Global Surface Temperature Analysis (NOAAGlobalTemp) dataset was produced and maintained by the NOAA National Centers for Environmental Information. This is monthly anomaly data with respect to the 1971-2000 climatology. It covers time from January 1880 to present, and has a spatial resolution $5° \times 5°$. The data can be visualized at `www.4dvd.org`.

Exercises

4.1 (a) Compute the temperature lapse rate for August using the TOB mean temperature data from the 24 USHCN stations in Colorado during the period of 1981-2010.
(b) Plot the figure similar to Fig. 4.1 for the data and results of Part (a).

4.2 Repeat the previous problem but for January. Compare your January results with those of July in Fig. 4.1 and its related text.

4.3 (a) Compute the temperature lapse rate for the annual mean temperature based on the 24 USHCN stations in Colorado during the period of 1981-2010.
(b) Plot the figure similar to Fig. 4.1 for the data and results of Part (a).

4.4 (a) Compute the annual mean temperature lapse rate for a high mountainous region and for a period of your interest.
(b) Plot the figure similar to Fig. 4.1 for the data and results of Part (a).

4.5 Show that the orthogonality condition Eq. (4.27)

$$\mathbf{e} \cdot \mathbf{x}_a = 0 \tag{4.113}$$

is equivalent to the condition of minimizing SSE given the condition (4.18)

$$\sum_{i=1}^{n} e_i = 0. \tag{4.114}$$

4.6 Show that when $m = 1$, the confidence interval formula for the multiple linear regression (4.102) is reduced to the confidence interval formula for the simple linear regression (4.82).

4.7 Examine the global average December temperature anomalies from 1880 to 2018 in the dataset of the NOAAGlobalTemp.
(a) Make a linear regression of the temperature anomalies against time.
(b) Compute the confidence intervals of the fitted model at 95% confidence

level.

(c) Compute the confidence intervals of the anomaly data at 95% confidence level.

(d) On the same figure similar to Fig. 4.6(a), plot the scatter plot of the anomaly data against time, and plot the confidence intervals computed in (b) and (c).

4.8 Make a diagnostic analysis for the regression results of the previous problem.

(a) Produce a scatter plot of the residuals against time.

(b) Visually check whether the assumptions of linearity and constant variance are satisfied.

(c) Use the KS-test to check the normality assumption on residuals.

(d) Use the DW-test to check the independence assumption on residuals.

(e) When serial correlation is considered, find the effective degrees of freedom (edof).

(f) Compute the confidence intervals in Steps (b) and (c) in the above problem using edof.

(g) Produce a scatter plot of the anomaly data against time, and plot the confidence intervals on the same figure using the results of Step (f).

4.9 (a) Fit the NOAAGlobalTemp's global average annual mean data from 1880 to 2018 to a ninth order orthogonal polynomial.

(b) Plot the data and the fitted polynomial function on the same figure.

(c) Produce a scatter plot of the residuals of the fitting against time.

4.10 Make a diagnostic analysis for the above regression and examine the regression assumptions. In particular, use the KS-test to verify the normality assumption, and the DW-test to verify the independence assumption.

4.11 (a) Use multiple linear regression to compute the 12th order polynomial fitting of the NOAAGlobalTemp's global average annual mean data from 1880 to 2018:

$$T = b_0 + b_1 t + b_2 t^2 + \cdots + b_{12} t^{12} + \epsilon. \tag{4.115}$$

(b) Plot the data and the fitted polynomial function on the same figure.

(c) Produce a scatter plot of the residuals of the fitting against time.

4.12 Make a diagnostic analysis for the above regression and examine the regression assumptions. In particular, use the KS-test to verify the normality assumption, and the DW-test to verify the independence assumption.

4.13 (a) Fit the global average **January** monthly mean temperature anomaly data from 1880 to 2018 in the NOAAGlobalTemp dataset to a third order orthogonal polynomial. The global average monthly mean NOAAGlobalTemp time series data are included in the book's master dataset named `data.zip`. You can also download the updated data from the Internet.

(b) Plot the data and the fitted polynomial function on the same figure.

(c) Produce a scatter plot of the residuals of the fitting against time in another figure.

4.14 Make a diagnostic analysis for the previous **January** data fit following the

procedures in this chapter. In particular, use the KS-test to verify the normality assumption, and the DW-test to verify the independence assumption.

4.15 (a) Fit the global average **July** monthly mean temperature anomaly data from 1880 to 2018 in the NOAAGlobalTemp dataset to a third order orthogonal polynomial.

(b) Plot the data and the fitted polynomial function on the same figure.

(c) Produce a scatter plot of the residuals of the fitting against time in another figure.

4.16 Make a diagnostic analysis for the previous **July** data fit following the procedures in this chapter. In particular, use the KS-test to verify the normality assumption, and the DW-test to verify the independence assumption.

4.17 Use the gridded monthly NOAAGlobalTemp dataset and make a third-order polynomial fit to the **January** monthly mean temperature anomaly data from 1880 to 2018 in a grid box that covers Tokyo, Japan. The gridded monthly NOAAGlobalTemp dataset is included in the book's master data file `data.zip`. You can also download the updated data from the Internet.

4.18 Use the gridded monthly NOAAGlobalTemp dataset and make a third-order polynomial fit to the **January** monthly mean temperature anomaly data from 1880 to 2018 in a grid box that covers Bonn, Germany.

4.19 Use the gridded monthly NOAAGlobalTemp dataset and make a fifth-order polynomial fit to the monthly mean temperature anomaly data for a grid box, a month, and a period of time of your own interest. For example, you may choose your hometown grid box, the month you were born, and the period of 1900-1999.

4.20 Make a diagnostic analysis for the previous data fit following the procedures in this chapter.

4.21 Use the January time series data of an USHCN station and fit a third order polynomial. Make a diagnostic analysis for the fit. The updated monthly USHCN station data may be downloaded from the Internet.

A Appendix A A Tutorial of R and RStudio

The book uses R and the R Notebook. This chapter explains the installation of R and R Studio and demonstrates some basic uses of R.

Equivalent Python codes and their Jupyter Notebooks may be found at the website `www.climatemathematics.org`.

A.1 Download and install R and R-Studio

For Windows users, visit the website
`https://cran.r-project.org/bin/windows/base/`
to find the instructions for R program download and installations.

For Mac users, visit
`https://cran.r-project.org/bin/macosx/`

If you experience difficulties, please refer to online resources, Google or YouTube. A recent 3-minute YouTube instruction for R installation for Windows can be found from the following link:
`https://www.youtube.com/watch?v=Ohnk9hcxf9M`

The same author also has a YouTube instruction about R installation for Mac (2 minutes):
`https://www.youtube.com/watch?v=uxuuWXU-7UQ`

When R is installed, one can open R. The `R Console` window will appear. See Fig. A.1. One can use `R Console` to perform calculations, such as typing 2+3 and hitting return. However, most people today prefer using RStudio as the interface. To install RStudio, visit
`https://www.rstudio.com/products/rstudio/download/`
This site allows one to choose Windows, or Mac OS, or Unix.

After both R and RStudio are installed, one can use either R or RStudio, or both, depending on one's interest. However, RStudio will not work without R. Thus, always install R first.

When opening RStudio, four windows will appear as shown in Fig. A.2: The top left window is called R script, for writing the R code. The green arrow on

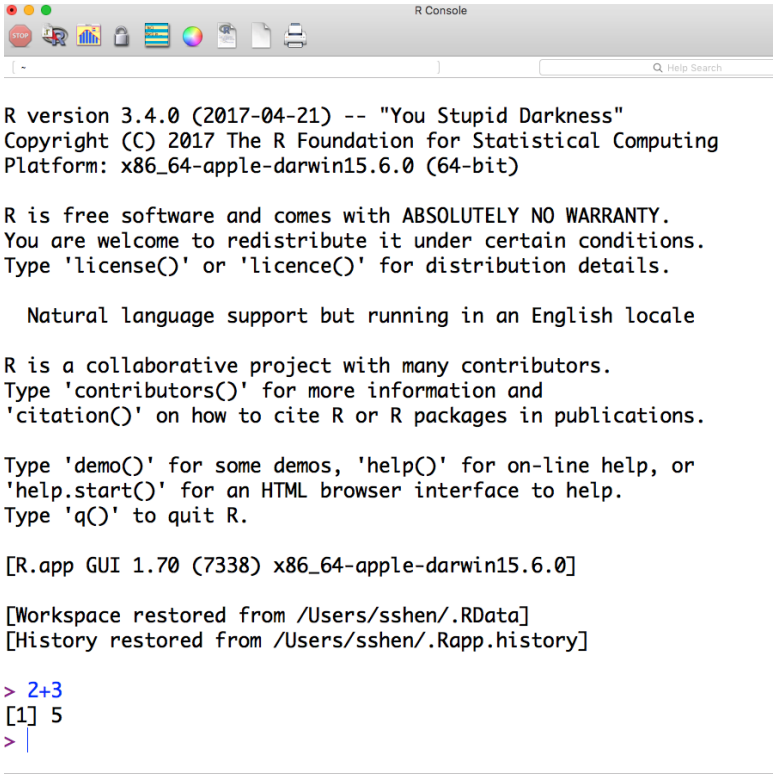


Fig. A.1 R Console window after opening R.

top of the window can be clicked to run the code. Each run is shown in the lower left R Console window, and recorded on the upper right R History window. When plotting, the figure will appear in the lower right R Plots window. For example, `plot(x,x*x)` renders the eight points in the Plots window, because `x=1:8` defines a sequence of numbers from 1 to 8. `x*x` yields a sequence from 1^2 to 8^2.

A.2 R Tutorial

Many excellent tutorials for quickly learning R programming, using a few hours or a few evenings, are available online and in YouTube, such as
`http://ww2.coastal.edu/kingw/statistics/R-tutorials/`.
You can easily perform an Internet search and find your preferred tutorials.

It can be extremely difficult for the beginners of R to navigate through the official, formal, detailed, and massive R-Project documentation:
`https://www.r-project.org/`

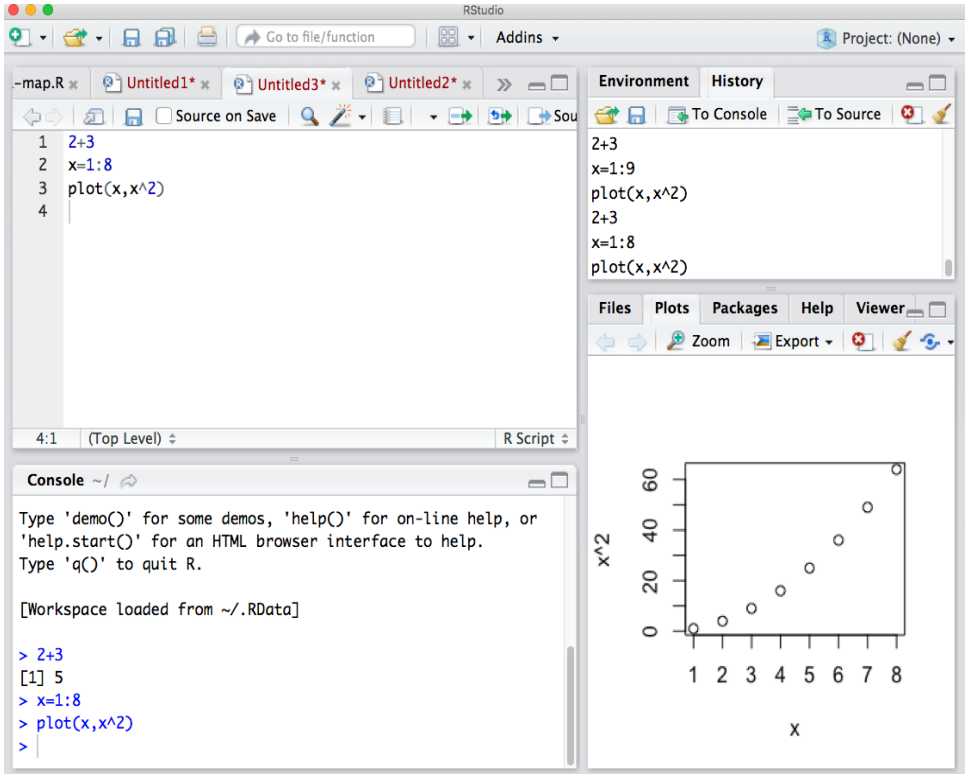


Fig. A.2 R Studio windows.

A.2.1 R as a smart calculator

R can be used like a smart calculator that allows more elaborate calculations than those done with ordinary calculators.

```
1+4
#[1] 5  This is the result
2+pi/4-0.8
#[1] 1.985398
x<-1
y<-2
z<-4
t<-2*x^y-z
t
#[1] -2
u=2         # "=" sign and "<-" are almost equivalent
v=3         # The text behind the "#" sign is comments
u+v
#[1] 5
```

```
sin(u*v)    # u*v = 6 in the sine function is considered radian by R
#[1] -0.2794155
```

R programming uses assignment operator `a <- b` to assign b to a. Often the equal operator `a=b` can do the same job or vice versa. The two operators are equivalent in general. However, certain R formulas have specific meanings for `=` and cannot be replaced by `<-` . Most veteran R users use `<-` for assignment and `=` for defined R formulas.

A.2.2 Define a sequence in R

Directly enter a sequence of daily maximum temperature data at San Diego International Airport (Lat: 32.7336°N, Lon: 117.1831°W) during 1-7 May 2017 [unit: $^{\circ}F$].

```
tmax <- c(77, 72, 75, 73, 66, 64, 59)
```

The data are from the Daily Summary of the Local Climatological Data (LCD), National Centers for Environmental Information (NCEI)

`https://www.ncdc.noaa.gov/cdo-web/datatools/lcd`

The command `c( )` is used to hold a data sequence and is named `tmax`. Entering the `tmax` command will render the temperature data sequence:

```
tmax
#[1] 77 72 75 73 66 64 59
```

You can generate different sequences using R, e.g.,

```
 1: 8 #Generates a sequence 1,2,...,8
```

Here the pound sign `#` begins R comments which are not executed by R calculations. The same sequence can be generated by different commands, such as

```
seq(1,8)
seq(8)
seq(1,8, by=1)
seq(1,8, length=8)
seq(1,8, length.out =8)
```

The most useful sequence commands are `seq(1,8, by=1)` and `seq(1,8, length=8)` or `seq(1,8, len=8)`. The former is determined by a begin value, end value, and step size, and the latter by a begin value, end value, and number of values in the sequence. For example, `seq(1951,2016, len=66*12)` renders a sequence of all the months from January 1951 to December 2016.

A.2.3 Define a function in R

The function command is
`name <- function(var1, var2, ...) expression of the function.`
For example,

```
samfctn <- function(x) x*x
samfctn(4)
#[1] 16
fctn2 <- function(x,y,z) x+y-z/2
fctn2(1,2,3)
#[1] 1.5
```

A.2.4 Plot with R

R can can plot all kinds of curves, surfaces, statistical plots, and maps. Below are a few very simple examples for R beginners. For adding labels, ticks, color, and other features to a plot, you will learn them from later parts of this book, and you may also carry out an Internet search on R plot to find the commands for the proper inclusion of the desired features.

R plotting is based on the coordinate data. The following command plots the seven days of San Diego Tmax data above:
`plot(1:7, c(77, 72, 75, 73, 66, 64, 59))`
The resulting graph is shown in Fig. A.3.

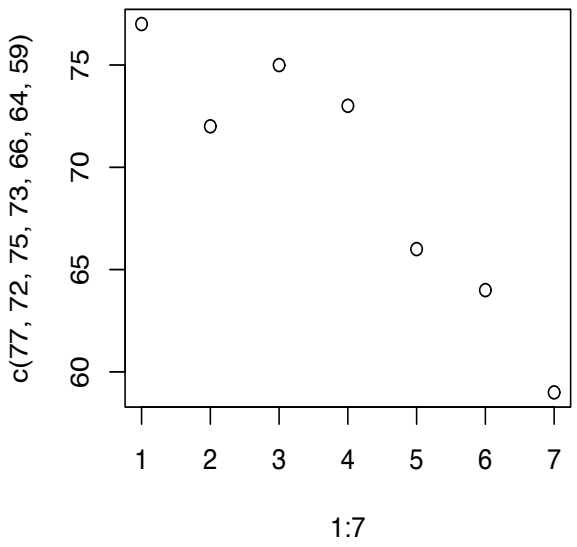


Fig. A.3 The daily maximum temperature during 1-7 May 2017 at San Diego International Airport.

```
plot(sin, -pi, 2*pi)   #plot the curve of y=sin(x) from -pi to 2 pi

square <- function(x) x*x   #Define a function
plot(square, -3,2)   # Plot the defined function

 # Plot a 3D surface
x <- seq(-1, 1, length=100)
y <- seq(-1, 1, length=100)
z <- outer(x, y, function(x, y)(1-x^2-y^2))
#outer (x,y, function) renders z function on the x, y grid
persp(x,y,z, theta=330)
# yields a 3D surface with perspective angle 330 deg

#Contour plot\index{contour plot}
contour(x,y,z) #lined contours
filled.contour(x,y,z) #color map of contours
```

The color map of contours resulting from the last command is shown in Fig. A.4.

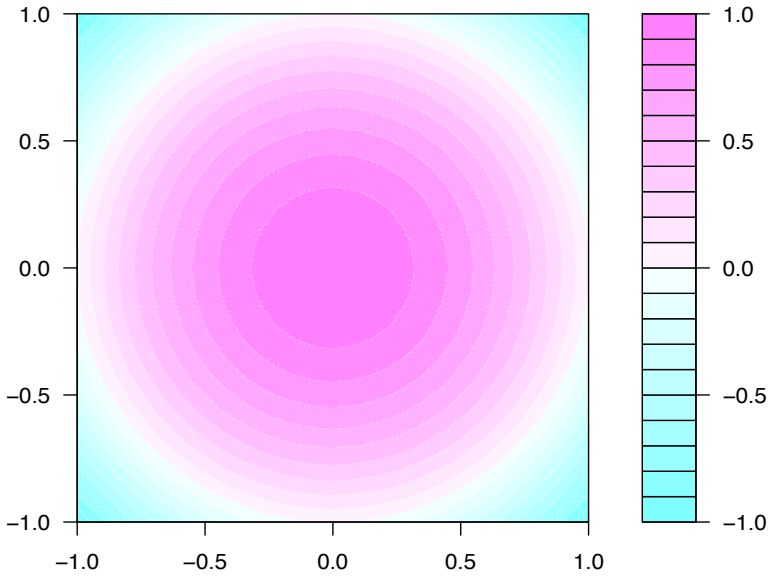


Fig. A.4 The color map of contours for the function $z = 1 - x^2 - y^2$.

A.2.5 Symbolic calculations by R

Many people once thought that R can handle only numbers. Actually R can also do symbolic calculations, such as finding a derivative, although at present R is not the

best symbolic calculation tool. One can use WolframAlpha, SymPy, and Yacas for free symbolic calculations or use the paid software packages Maple or Mathematica. Carry out an Internet search on symbolic calculation for calculus to find a long list of symbolic calculation software packages, e.g., https://en.wikipedia.org/wiki/List_of_computer_algebra_systems.

```
D(expression(x^2,'x'), 'x')
# Take derivative of x^2 w.r.t. x
2 * x #The answer is 2x

fx= expression(x^2,'x')  #assign a function
D(fx,'x') #differentiate the function w.r.t. x
2 * x  #The answer is 2x

fx= expression(x^2*sin(x),'x')
#Change the expression and use the same derivative command
D(fx,'x')
2 * x * sin(x) + x^2 * cos(x)

 fxy = expression(x^2+y^2, 'x','y')
#One can define a function of 2 or more variables
 fxy #renders an expression of the function in terms of x and y
#expression(x^2 + y^2, "x", "y")
D(fxy,'x') #yields the partial derivative with respect to x: 2 * x
D(fxy,'y') #yields the partial derivative with respect to y: 2 * y

square = function(x) x^2
integrate (square, 0,1)
#Integrate x^2 from 0 to 1 equals to 1/3 with details below
#0.3333333 with absolute error < 3.7e-15

integrate(cos,0,pi/2)
#Integrate cos(x) from 0 to pi/2 equals to 1 with details below
#1 with absolute error < 1.1e-14
```

The above two integration examples are for definite integrals. It seems that no efficient R packages are available for finding antiderivatives, or indefinite integrals.

A.2.6 Vectors and matrices

R can handle all kinds of operations involving vectors and matrices.

```
c(1,6,3,pi,-3) #c() gives a vector and is considered a 4X1 column vector\index{column vect
#[1]  1.000000  6.000000  3.000000  3.141593 -3.000000
```

```
seq(2,6) #Generate a sequence from 2 to 6
#[1] 2 3 4 5 6
seq(1,10,2) # Generate a sequence from 1 to 10 with 2 increment
#[1] 1 3 5 7 9
x=c(1,-1,1,-1)
x+1 #1 is added to each element of x
#[1] 2 0 2 0
2*x #2 multiplies each element of x
#[1]  2 -2  2 -2
x/2 # Each element of x is divided by 2
#[1]  0.5 -0.5  0.5 -0.5
y=seq(1,4)
x*y  # This multiplication * multiples each pair of elements
#[1]  1 -2  3 -4
x%*%y #This is the dot product of two vectors and yields
#     [,1]
#[1,]   -2
t(x)  # Transforms x into a row 1X4 vector
#     [,1] [,2] [,3] [,4]
#[1,]    1   -1    1   -1
t(x)%*%y #This is equivalent to dot product and forms 1X1 matrix
#     [,1]
#[1,]   -2
x%*%t(y) #This column times row yields a 4X4 matrix\index{matrix}
#     [,1] [,2] [,3] [,4]
#[1,]    1    2    3    4
#[2,]   -1   -2   -3   -4
#[3,]    1    2    3    4
#[4,]   -1   -2   -3   -4
my=matrix(y,ncol=2)
#Convert a vector into a matrix of the same number of elements
#The matrix elements\index{matrix!elements} go by column, the first column, second, etc
#Command matrix(y,ncol=2, nrow=2), or matrix(y, ncol=2)
#or matrix(y,2), or matrix(y,2,2) does the same job
my
#     [,1] [,2]
#[1,]    1    3
#[2,]    2    4
dim(my)  #find dimensions of a matrix
#[1] 2 2

bigM=matrix(1:100, nrow=10) #Generate a 10-by-10 matrix
subM=bigM[4:6,3:7] #Extract a sub-matrix from a big matrix
subM
```

```
#     [,1] [,2] [,3] [,4] [,5]
#[1,]   24   34   44   54   64
#[2,]   25   35   45   55   65
#[3,]   26   36   46   56   66

as.vector(my) #Convert a matrix to a vector, again via columns
#[1] 1 2 3 4
mx <- matrix(c(1,1,-1,-1), byrow=TRUE,nrow=2)
mx*my #multiplication between each pair of elements
#     [,1] [,2]
#[1,]    1    3
#[2,]   -2   -4
mx/my #division between each pair of elements
#     [,1]       [,2]
#[1,]  1.0  0.3333333
#[2,] -0.5 -0.2500000
mx-2*my
#     [,1] [,2]
#[1,]   -1   -5
#[2,]   -5   -9
mx%*%my #This is the real matrix multiplication\index{matrix multiplication} in matrix the
#     [,1] [,2]
#[1,]    3    7
#[2,]   -3   -7
det(my) #determinant\index{determinant}
#[1] -2
myinv = solve(my) #yields the inverse of a matrix
myinv
#     [,1] [,2]
#[1,]   -2  1.5
#[2,]    1 -0.5
myinv%*%my #verifies the inverse of a matrix
#     [,1] [,2]
#[1,]    1    0
#[2,]    0    1
diag(my) #yields the diagonal vector of a matrix
#[1] 1 4
myeig=eigen(my) #yields eigenvalues\index{eigenvalue} and unit eigenvectors\index{eigenvec
myeig
myeig$values
#[1]  5.3722813 -0.3722813
myeig$vectors
#           [,1]       [,2]
#[1,] -0.5657675 -0.9093767
```

```
#[2,] -0.8245648  0.4159736
mysvd = svd(my) #SVD decomposition of a matrix M=UDV'
#SVD can be done for a rectangular matrix of mXn
mysvd$d
#[1] 5.4649857 0.3659662
mysvd$u
#           [,1]       [,2]
#[1,] -0.5760484 -0.8174156
#[2,] -0.8174156  0.5760484
mysvd$v
#           [,1]       [,2]
#[1,] -0.4045536  0.9145143
#[2,] -0.9145143 -0.4045536

ysol=solve(my,c(1,3))
#solve linear equations\index{linear equations} matrix %*% x = b
ysol  #solve(matrix, b)
#[1]  2.5 -0.5
my%*%ysol #verifies the solution
#     [,1]
#[1,]    1
#[2,]    3
```

A.2.7 Simple statistics by R

R was originally designed to do statistical calculations. Thus, R has a comprehensive set of statistics functions and software packages. This sub-section gives a few basic commands. More will be described in the statistics chapter of this book.

```
x=rnorm(10) #generate 10 normally distributed numbers
x
#[1]  2.8322260 -1.2187118  0.4690320 -0.2112469  0.1870511
#[6]  0.2275427 -1.2619005  0.2855896  1.7492474 -0.1640900
mean(x)
#[1] 0.289474
var(x)
#[1] 1.531215
sd(x)
#[1] 1.237423
median(x)
#[1] 0.2072969
quantile(x)
#        0%        25%        50%        75%       100%
#-1.2619005 -0.1994577  0.2072969  0.4231714  2.8322260
```

```
range(x) #yields the min and max of x
#[1] -1.261900  2.832226
max(x)
#[1] 2.832226

boxplot\index{boxplot}(x) #yields the box plot of x
w=rnorm(1000)

summary(rnorm(12)) #statistical summary of the data sequence
#   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#-1.9250 -0.6068  0.3366  0.2309  1.1840  2.5750

hist(w)
#yields the histogram\index{histogram} of 1000 random numbers with a normal distribution

#Linear regression\index{linear regression} and linear trend\index{linear trend} line
#2007-2016 data of the global temperature\index{anomalies!temperature} anomalies
#Source: NOAAGlobalTemp data
t=2007:2016
T=c(.36,.30, .39, .46, .33, .38, .42, .50, .66, .70)
lm(T ~ t) #Linear regression model of temp vs time
#(Intercept)            t
#-73.42691      0.03673
#Temperature change rate is 0.03673  deg C/yr or 0.37 deg C/decade
plot(t,T, type="o",xlab="Year",ylab="Temperature [deg C]",
     main="2007-2016 Global Temperature\index{anomalies!temperature} Anomalies\index{anoma
          and Their Linear Trend [0.37 deg C/decade] ")
abline(lm(T ~ t), lwd=2, col="red") #Regression line
```

The global temperature data from 2007-2016 in the above R code example are displayed in Fig. A.5, together with their linear trend line

$$T = -73.42691 + 0.03673t. \tag{A.1}$$

A.3 Online Tutorials

Numerous online R tutorials are available. Several are relatively efficient for learning climate mathematics and are recommended below.

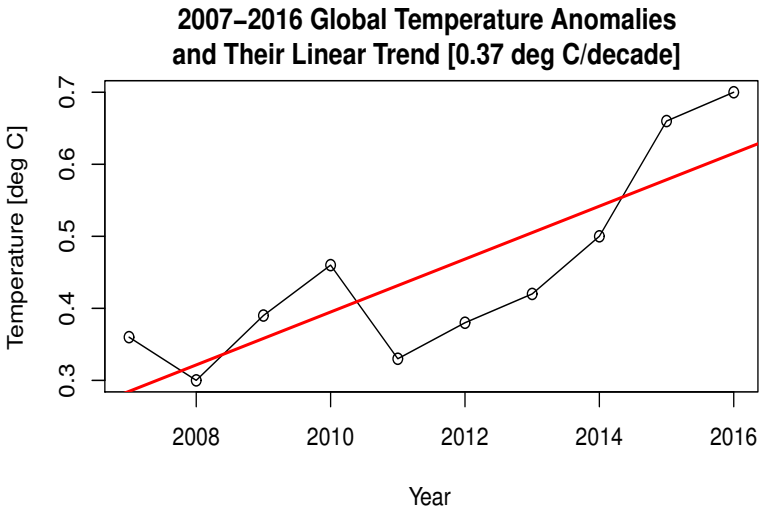


Fig. A.5 The 2007-2016 global average annual mean surface air temperature anomalies with respect to the 1971-2000 climate normal. The red line is a linear trend line computed from a linear regression model.

A.3.1 YouTube tutorial: for true beginners

This is a very good and slow-paced 22-minute YouTube tutorial: Chapter 1. An Introduction to R

`https://www.youtube.com/watch?v=suVFuGET-0U`

A.3.2 YouTube tutorial: for some basic statistical summaries

This is a 9-minute tutorial by Layth Alwan.

`https://www.youtube.com/watch?v=XjOZQN-Nre4`

A.3.3 YouTube tutorial: Input data by reading a csv file into R

An excel file can be saved as csv file: xxxx.csv. This 15-minute YouTube video by Layth Alwan shows how to read a csv file into R. He also shows linear regression.

`https://www.youtube.com/watch?v=QkE8cp0B9gg`

R can input all kinds of data files, including xlsx, txt, netCDF, MatLab data, Fortran file, and SAS data. Some commands are below. One can search the Internet to find proper data reading commands for any particular data format.

```
mydata <- read.csv("mydata.csv")
# read csv file named "mydata.csv"
```

```
mydata <- read.table("mydata.txt")
# read text file named "my data.txt"

library(gdata)                    # load the gdata package
mydata = read.xls("mydata.xls")  # read an excel file

library(foreign)                  # load the foreign package
mydata = read.mtp("mydata.mtp")  # read from .mtp file

library(foreign)                  # load the foreign package
mydata = read.spss("myfile", to.data.frame=TRUE)

ff <- tempfile()
cat(file = ff, "123456", "987654", sep = "\n")
read.fortran(ff, c("F2.1","F2.0","I2")) #read a fotran file

library(ncdf4)
ncin <- ncdf4::nc_open("ncfname")  # open a NetCDF file
lon <- ncvar_get(ncin, "lon") #read data "lon" from a netCDF file into R
```

Many more details of reading and reformatting of .nc files will be discussed later when dealing with NCEP/NCAR Reanalysis data.

Some libraries are not in the R project. For example,

```
library(ncdf4) #The following error message pops up
Error in library(ncdf4) : there is no package called cdf4
```

You can install the R package by

```
install.packages("ncdf4")
```

After this installation, `library(ncdf4)` will run, and the functions in the ncdf4 package will work.

You only need to install the package once on your computer, but in each new R session you must run `library(package)` in order to activate the package functions. Many examples will be shown in the rest of this book.

The R packages and the datasets used in this book are listed below and can be downloaded and installed first before proceeding to the R codes in the rest of the book.

```
#R packages: animation, chron, e1071, fields, ggplot2, lattice,
#latticeExtra, maps, mapdata, mapproj, matrixStats, ncdf4,
#NLRoot, RColorBrewer, rgdal, rasterVis, raster, sp, TTR

#The zipped data:
```

```
#https://cambridge.org/climatemathematics/data.zip

#To load a single package, such as "animation", you can do
library(animation)

#You can also load all these packages in one shot using
# pacman

install.packages("pacman")
library(pacman)
pacman::p_load(animation, chron, e1071, fields, ggplot2, lattice,
            latticeExtra, maps, mapdata, mapproj, matrixStats, ncdf4,
            NLRoot, RColorBrewer, rgdal, rasterVis, raster, sp, TTR)
```

On your computer, you can create a directory called `climmath` under your user name. The one used in the book is `Users/sshen/climmath`. You unzip the data and move the data folder under the `Users/sshen/climmath` directory. A data folder will created: `Users/sshen/climmath/data`. The data folder contains about 400 MB of data. Place all the R codes in the directory `Users/sshen/climmath`. Then, you can run all the codes in this book after replacing `sshen` by your user name on your own computer.

A.4 Chapter summary

This chapter has described the following basic aspects of the R programming language

(i) Installation of both R and R Studio.
(ii) Layout of R Studio panels and functions.
(iii) R commands for defining and computing vectors, matrices, and functions.
(iv) Simple statistics using R: mean, median, standard deviation, variance, histogram, boxplot, scatterplot, and linear regression.
(v) Online resources for R tutorial material.

We suggest that reading and practicing the R basics included this chapter might require about three hours. You might then use perhaps five to ten hours to do the exercise problems of this chapter. After that introduction to R, we estimate that you should be able to develop and carry out projects using R independently. Familiarizing yourself with additional R commands may require spending some time online with a search engine.

More sophisticated examples of using R for the analysis and visualization of the data from climate models and observations are included in Chapters 9-11. These

chapters also provide research-level R-graphics codes for climate sciences, R programs for advanced statistical analysis of climate data, such as empirical orthogonal functions computed from the climate model datasets, and R programming techniques for analyzing datasets with missing data, either in space or time. To develop R projects at a more advanced level, you will need to read these chapters or search for the R code for the specific analysis or graphics tasks described in these chapters. These tasks include examples such as the preparation of data from a global climate model (GCM) for a singular value decomposition (SVD) analysis.

The R programming language was created by statisticians Ross Ihaka and Robert Gentleman in New Zealand and was first released in 1995. R is named partly after the first names of the two original two authors of R, and partly as a play on the name of the S programming language for statistics. Currently, R is a popular computer programming language used in almost every field of science and engineering. Among the top programming languages ranked by the Institute of Electric and Electronic Engineers (IEEE) in 2017, R ranks sixth, following Python, C, Java, C++, and C#. For a climate science student or professional who is not a specialist in computer programming or information technology, R is easy to learn, and it offers numerous public-domain software packages that are free to use. As an alternative to R, equivalent Python codes for our entire book are available online at the book website `www.climatemathematics.org`.

References and Further Readings

[1] King, K.B., 2016: *R Tutorials*, Coastal Carolina University. URL:

`http://ww2.coastal.edu/kingw/statistics/R-tutorials/`

This easy-to-learn tutorial is for beginners of R, and does not require any computer programming background. It contains many statistics examples.

[2] Jost, S., 2018: *Introduction to R: An R Tutorial for Data Analysis and Regression*. De Paul University. URL:

`http://facweb.cs.depaul.edu/sjost/csc423/`

This is a very brief R tutorial from the perspective of computer programming for beginners. One needs only about one hour to go through the entire tutorial.

[3] Vallis, G.K., 2016: Geophysical fluid dynamics: whence, whither and why? *Proceedings of the Royal Society A*, 472: 20160140. http://dx.doi.org/10.1098/rspa.2016.0140

In this stimulating article, Vallis discusses the role of geophysical fluid dynamics in understanding the dynamics of atmospheres and oceans. Geoffrey K. Vallis, a Professor in the Department of Mathematics at the University of Exeter, is an expert in climate dynamics, the circulation of planetary atmospheres, and dynamical meteorology and oceanography. He is the author of *Atmospheric and Oceanic Fluid Dynamics: Fundamentals and Large-Scale Circulation*. This widely praised standard text is a magisterial treatment of geophysical fluid dynamics. The book was first published in 2006, and its second edition in 2017 contains nearly 1,000 pages.

Exercises

A.1 Use R to define a data sequence `t=seq(2015,2018, length=100)`, and then plot the following two functions on the same figure: $y = \sin(2\pi(t - 0.1))$ and $y = \cos^2(2\pi t)$.

A.2 (a) Use R to make a contour plot of the function $z = \sin^2 x \cos^2(y - \pi)$ over the domain of $[0, 2\pi] \times [0, 2\pi]$.

(b) Use R to plot a color contour map for the same function on the same domain.

A.3 Use R to solve the following linear equations:

$$\begin{cases} 9x + 8y = 87 \\ 6x - 20y = 126 \end{cases}$$

A.4 Use R to solve the following linear equations:

$$\begin{cases} -3x + 2y + z = 1 \\ -2x - y + z = 2 \\ 2x + y - 4z = 0 \end{cases}$$

A.5 For some purposes, climatology or climate is defined as the mean state, or normal state, of a climate parameter, and is calculated from data over a period of time called the climatology period (e.g., 1961-1990). Thus the surface air temperature climate or climatology at a given location may be calculated by averaging observational temperature data over a period such as 1961 through 1990. Thirty years are often considered in the climate science community as the standard length of a climatology period. Due to the relatively high density of weather stations in 1961-1990, compared to earlier periods, investigators have often used 1961-1990 as their climatology period, although some may now choose 1971-2000 or 1981-2010. Surface air temperature (SAT) is often defined as the temperature inside a white-painted louvered instrument container or box, known as a Stevenson screen located on a stand about 2 meters above the ground. The purpose of the Stevenson screen is to shelter the instruments from radiation, precipitation, animals, leaves, etc, while allowing the air to circulate freely inside the box. The daily maximum temperature (Tmax) is the maximum temperature measured inside the screen box by a maximum temperature thermometer within 24 hours. The daily minimum temperature (Tmin) is the minimum temperature within 24 hours. The daily mean temperature (Tmean) is the average of Tmax and Tmin.

Go to the USHCN website

`http://cdiac.ornl.gov/epubs/ndp/ushcn/ushcn_map_interface.html`

and download the monthly Tmax, Tmin, and Tmean data of the Cuyamaca

station (USHCN Site No. 042239) near San Diego, California, USA. Or download the data of this book at

`www.cambridge.org/climatemathematics/data.zip`

Unzip the data and find the data from the file named `CA042239T.csv`

(a) Use R to arrange the monthly Cuyamaca Tmax data from January 1961 to December 1990 as a matrix with each row as year and each column as month.

(b) Do the same for Tmin.

(c) Do the same for Tmean.

A.6 (a) Use R to calculate the August climatology of Tmax, Tmin, and Tmean for the Cuyamaca station according to the 1961-1990 climatology period.

(b) Use R to compute the standard deviation of Tmax, Tmin, and Tmean of the Cuyamaca station for January during the 1961-1990 climatology period.

A.7 (a) Use R to plot the the Cuyamaca January Tmin time series from 1951 to 2010 with a continuous curve.

(b) Use R to plot the linear trend lines of Tmin on the same plot as (a) in the following time periods:

(i) 1951-2010,

(ii) 1961-2010,

(iii) 1971-2010, and

(iv) 1981-2010.

(c) Finally, what is the temporal trend per decade for each of the four periods above?

A.8 Use R to plot the time series and its trend line for P.D. Jones' global average annual mean temperature anomaly data: `JonesGlobalT.txt` . This data file can be found from the book's data.zip file downloaded from the book website.

(a) Plot the global average annual mean temperature from 1880 to 2015.

(b) Find the linear trend of the temperature from 1880 to 2015. Plot the trend line on the same figure as (a).

(c) Find the linear trend from 1900 to 1999. Plot the trend line on the same figure as (a).

A.9 Use the gridded NOAA global monthly temperature anomaly data NOAAGlobalTemp from the following website or another data source

`https://www.ncdc.noaa.gov/data-access/marineocean-data/noaa-global-surface-temperature-noaaglobaltemp`

Or use the `NOAAGlobalT.csv` data file from the book's data.zip file downloaded from the book website. Choose two 5-by-5 degrees lat-lon grid boxes of your interest. Plot the temperature anomaly time series of the two boxes on the same figure using two different colors.

A.10 Use the same NOAAGlobalTemp dataset, choose sufficiently many grid boxes that cover the state of Texas, USA. Compute the average temperature anomalies of these boxes for each month. Then plot the monthly average temperature anomalies as a function of time. Plot a linear trend line on the same figure.

A.11 Choose a 5-by-5 degrees grid box that covers Edmonton, Canada, and another grid box that covers San Diego, USA.

(a) Use R and 30 years of the January NOAAGlobalTemp data from January 1981 to January 2010 to compute the standard deviations for each grid box for January.

(b) Do the same for February, March, $\cdots$, December.

(c) Use R to write your standard deviation results in a 12-by-2 matrix with each row for a month, and each column for a grid box ID.

(d) Describe the main differences between the values of the two columns.

B Appendix B Visualization of Matrices

People talk about climate data frequently, also read or imagine climate data, and yet rarely play with them and use them, because people often think that it takes a computer expert to do that. However, that has changed. With today's technology, now anyone can use a computer to play with climate data, such as a sequence of temperature values of a weather station at different observed times, a matrix of data for a station for temperature, air pressure, precipitation; wind speed, and wind direction at different times; and an array of temperature data on a 5-degree latitude-longitude grid for the entire world for different months. The first is a vector. The second is a variable-time matrix, and a space-time 3-dimensional array. When considering temperature variation in time at different air pressure levels and different water depth, we need to add one more dimension: the altitude. The temperature data for ocean and atmosphere for the Earth is a 4-dimensional array, with 3D space and 1D time. This chapter attempts to provide basic statistical and computing methods to describe and visualize some simple climate datasets. As the book progresses, more complex statistics and data visualization will be introduced.

We use both R and Python computer codes in this book for computing and visualization. Our method description is stated in R. A Python code following each R code is included in a box with a light yellow background. You can also learn the two computer languages and their applications to climate data from the book "*Climate Mathematics: Theory and Applications*" (Shen and Somerville 2019) and its website `www.climatemathematics.org`.

The climate data used in this book are included in the `data.zip` file downloadable from our book website `www.climatestatistics.org`. You can also obtain the updated data from the original data providers, such as `www.esrl.noaa.gov` and `www.ncei.noaa.gov`.

After learning this chapter, a reader should be able to analyze simple climate datasets, compute data statistics, and plot the data in various ways.

B.1 Global temperature anomalies from 1880 to 2018: data visualization and statistical indices

In a list of popular climate datasets, the global average annual mean surface air temperature anomalies might be on top. Here, the *anomalies* means the temperature

departures from the normal temperature that is called *climatology*. Climatology is usually defined as the mean of temperature data in a given period of time, such as from 1971 to 2020. Thus, the temperature anomaly data are the differences of the temperature data minus the climatology.

This section will use the global average annual mean surface air temperature anomaly dataset as an example to describe some basic statistical and computing methods.

B.1.1 The NOAAGlobalTemp dataset

The 1880-2018 global average annual mean surface air temperature (SAT) anomaly data are shown as follows:

```
  [1] -0.37 -0.32 -0.32 -0.40 -0.46 -0.47 -0.45 -0.50 -0.40 -0.35 -0.57
 [12] -0.49 -0.55 -0.56 -0.53 -0.47 -0.34 -0.36 -0.50 -0.37 -0.31 -0.39
 [23] -0.49 -0.58 -0.66 -0.53 -0.46 -0.62 -0.69 -0.68 -0.63 -0.68 -0.58
 [34] -0.57 -0.39 -0.32 -0.54 -0.56 -0.45 -0.45 -0.46 -0.39 -0.47 -0.46
 [45] -0.50 -0.39 -0.31 -0.40 -0.42 -0.54 -0.34 -0.32 -0.36 -0.49 -0.35
 [56] -0.38 -0.36 -0.26 -0.27 -0.26 -0.15 -0.05 -0.09 -0.09  0.04 -0.08
 [67] -0.25 -0.30 -0.30 -0.30 -0.41 -0.26 -0.22 -0.15 -0.36 -0.38 -0.44
 [78] -0.19 -0.13 -0.18 -0.22 -0.16 -0.15 -0.13 -0.39 -0.32 -0.27 -0.26
 [89] -0.27 -0.15 -0.21 -0.32 -0.22 -0.08 -0.32 -0.24 -0.32 -0.05 -0.13
[100] -0.02  0.02  0.06 -0.06  0.10 -0.10 -0.11 -0.01  0.13  0.13  0.05
[111]  0.19  0.16  0.01  0.03  0.09  0.21  0.08  0.27  0.39  0.20  0.18
[122]  0.30  0.35  0.36  0.33  0.41  0.37  0.36  0.30  0.39  0.45  0.33
[133]  0.38  0.42  0.50  0.66  0.70  0.60  0.54
```

The data are part of the dataset named as the NOAA Merged Land Ocean Global Surface Temperature Analysis (NOAAGlobalTemp) V4. The dataset was generated by the NOAA National Centers for Environmental Information (NCEI) (Smith et al. 2008; Vose et al. 2012). Here, NOAA stands for the United States National Oceanic and Atmospheric Administration.

The anomalies are with respect to the 1971-2000 climatology, i.e., 1971-2000 mean. An anomaly of a weather station datum is defined by the datum minus the station climatology. The first anomaly datum `-0.37` °C, indexed by `[1]`, in the above data table corresponds to 1880 and the last to 2018, a total of 139 years. The last row is indexed from `[133]` to `[139]`.

One might be interested in various kinds of statistical characteristics of the data, such as mean, variance, standard deviation, skewness, kurtosis, median, 5th percentile, 95th percentile, and other quantiles. Is the data's probabilistic distribution approximately normal? What does the box plot look like? Are there any outliers? What is a good graphic representation of the data, i.e., popularly known as a climate figure?

When considering global climate changes, why do scientists often use anomalies,

instead of the full values directly from the observed thermometer readings? This is because that the observational estimates of the global average annual mean surface temperature are less accurate than the similar estimates for the changes from year to year. There is a concept of characteristic spatial correlation length scale for a climate variable, such as surface temperature. The length scale is often computed from anomalies.

The use of anomalies is also a way of reducing or eliminating individual station biases. A simple example of such biases is that due to station location, which is usually fixed in a long period of time. It is easy to understand, for instance, that a station located in the valley of a mountainous region might report surface temperatures that are higher than the true average surface temperature for the entire region and cannot be used to describe the behavior of climate change in the region. However, the anomalies at the valley station may synchronously reflect the characteristics of the anomalies for the region. Many online materials give justifications and examples on the use of climate data anomalies, e.g., NOAA NCEI (2021).

The global average annual mean temperature anomalies quoted above are also important for analyzing climate simulations. When we average over a large scale, many random errors cancel out. When we investigate the response of such large scale perturbations as the variations of Sun's brightness or atmospheric carbon dioxide, these averaged data can help validate and improve climate models. See the examples in the book by Hennemuth et al. (2013) that includes many statistical analyses of both observed and model data.

B.1.2 Visualize the data of global average annual mean temperature

Many different ways have been employed to visualize the global average annual mean temperature anomalies. The following three are popular ones appearing in scientific and news publications: (a) a simple point-line graph, (b) a curve of staircase steps, and (c) a color bar chart, as shown in Figs. B.1 - B.3. This subsection shows how to generate these figures by R and Python computer programming languages. The Python codes are in yellow boxes.

B.1.2.1 Plot a point-line graph of time series data

Figure B.1 is a simple line graph that connects all the data points, denoted by the small circles, together to form a curve showing the historical record of the global temperature anomalies. It is plotted from the the NOAAGlobalTemp V4 data quoted above. The figure can be generated by the following R code.

```
# R plot Fig. 1.1: A simple line graph of data
# go to your working directory
setwd("/Users/sshen/climstats")
# read the data file from the folder named "data"
NOAAtemp = read.table(
  "data/aravg.ann.land_ocean.90S.90N.v4.0.1.201907.txt",
```

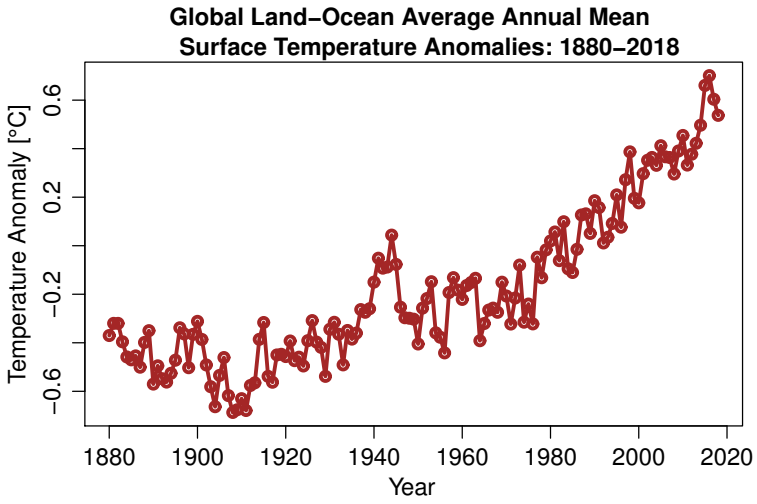


Fig. B.1 Point-line graph of the 1880-2018 global average annual mean temperature anomalies with respect to the 1971-2000 climatology, based on the NOAAGlobalTemp V4 data.

```
6205    header=FALSE) #Read from the data folder
6206  dim(NOAAtemp)# check the data matrix dimension
6207  #[1] 140   6  #140 years from 1880 to 2019
6208  #2019 will be excluded since data only up to July 2019
6209  #col1 is year, col2 is anomalies, col3-6 are data errors
6210  par(mar=c(3.5,3.5,2.5,1), mgp=c(2,0.8,0))
6211  plot(NOAAtemp[1:139,1], NOAAtemp[1:139,2],
6212       type ="l", col="brown", lwd=3, cex.lab=1.2, cex.axis=1.2,
6213       main ="Global␣Land-Ocean␣Average␣Annual␣Mean
6214  ␣␣␣␣␣Surface␣Temperature␣Anomalies:␣1880-2018",
6215       xlab="Year",
6216       ylab=expression(
6217         paste("Temperature␣Anomaly␣[", degree,"C]")))
```

```
# Python plot Fig. 1.1:  A simple line graph of data
# Go to your working directory
os.chdir("/Users/sshen/climstats")
# read the data file from the folder named "data"
NOAAtemp = read_table(\
"data/aravg.ann.land_ocean.90S.90N.v4.0.1.201907.txt",
    header = None, delimiter = "\s+")
# check the data matrix dimension
print("The dimension of our data table is:", NOAAtemp.shape)
x = np.array(NOAAtemp.loc[0:138, 0])
y = np.array(NOAAtemp.loc[0:138, 1])
plt.plot(x, y, 'brown', linewidth = 3);
plt.title("Global Land-Ocean Average Annual Mean \n \
Surface Temperature Anomaly: 1880 - 2018", pad = 15)
plt.xlabel("Year", size = 25, labelpad = 20)
plt.ylabel("Temperature Anomaly [$\degree$C]",
           size = 25, labelpad = 20)
plt.show() # display on screen
```

B.1.2.2 Plot a staircase chart

The staircase chart Fig. B.2 shows both data and the year-to-year annual changes of temperature. One can clearly see that some years have a larger change from their previous year, while other years have a smaller change. This information can be used for further climate analysis.

Denote the global average annual mean temperature by $T(t)$ where t stands for time in year, the 1971-2000 climatology is by C, and the anomalies by $A(t)$. We have

$$T(t) = C + A(t). \tag{B.1}$$

The temperature change from its previous year is

$$\Delta T = T(t) - T(t-1) = A(t) - A(t-1) = \Delta A. \tag{B.2}$$

This implies that an anomaly change is equal to its corresponding temperature change. We do not need to evaluate the 1971-2000 climatology C of the global average annual mean, when studying changes, as discussed earlier.

Figure B.2 can be generated by the following R code

```
#R plot Fig. 1.2: Staircase chart of data
plot(NOAAtemp[1:139,1], NOAAtemp[1:139,2],
     type="s", #staircase curve for data
     col="black", lwd=2,
     main="Global Land-Ocean Average Annual Mean
     Surface Temperature Anomalies: 1880-2018",
     cex.lab=1.2,cex.axis=1.2,
     xlab="year",
     ylab=expression(paste(
       "Temperature Anomaly [", degree,"C]"))
)
```

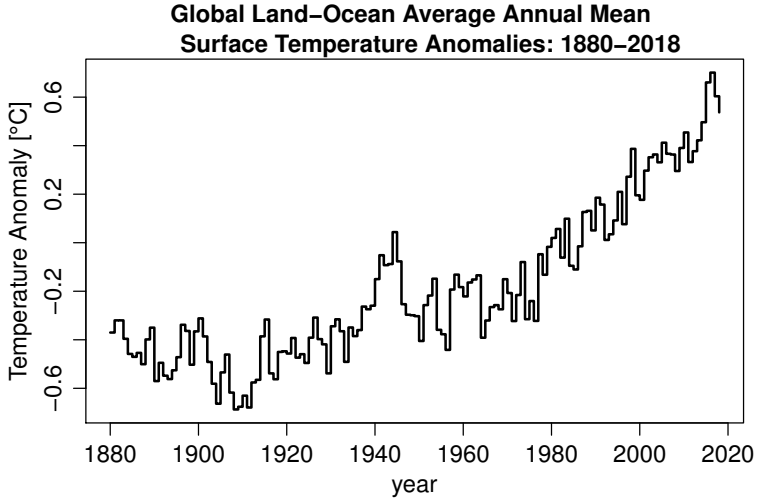


Fig. B.2 Staircase chart of the 1880-2018 global average annual mean temperature anomalies.

The corresponding Python code is in the following box.

```
# Python plot Fig. 1.2: A staircase chart of data
# keyword arguments
kwargs = {'drawstyle' : 'steps'}
plt.plot(x, y, 'black', linewidth = 3, **kwargs);
plt.title("Global␣Land-Ocean␣Average␣Annual␣Mean␣\n␣␣\
Surface␣Temperature␣Anomaly:␣1880␣-␣2018", pad = 15)
plt.xlabel("Year", size = 25, labelpad = 20)
plt.ylabel("Temperature␣Anomaly␣[$\degree$C]",
           size = 25, labelpad = 20)
plt.show()
```

B.1.2.3 Plot a bar chart with colors

Figure B.3 shows the anomaly size for each year by a color bar: red for positive anomalies and blue for negative anomalies. This bar chart style has an advantage of visualizing climate extremes, since these extremes may leave a distinct impression on viewers. The black curve is a 5-point moving average of the annual anomaly data, computed for each year by the mean of that year, together with two years before and two years after. The moving average smooths the high frequency fluctuations to reveal the long-term trends of temperature variations.

```
# R plot Fig. 1.3: A color bar chart of data
x  <-  NOAAtemp[,1]
y  <-  NOAAtemp[,2]
z  <-  rep(-99, length(x))
# compute 5-point moving average
```

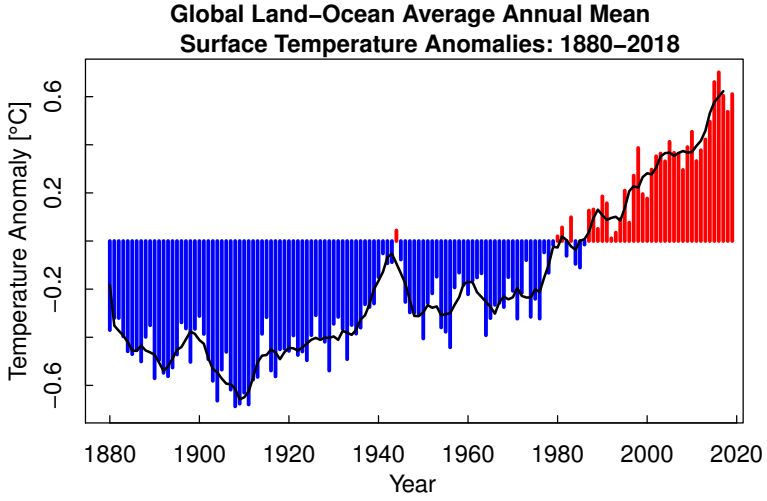


Fig. B.3 Color bar chart of the 1880-2018 global average annual mean temperature anomalies.

```
for (i in 3:length(x)-2) z[i] =
  mean(c(y[i-2],y[i-1],y[i],y[i+1],y[i+2]))
n1 <- which(y>=0); x1 <- x[n1]; y1 <- y[n1]
n2 <- which(y<0); x2 <- x[n2]; y2 <- y[n2]
x3 <- x[2:length(x)-2]
y3 <- z[2:length(x)-2]
plot(x1, y1, type="h", #bars for data
     xlim = c(1880,2016), lwd=3,
     tck = 0.02,  #tck>0 makes ticks inside the plot
     ylim = c(-0.7,0.7), xlab="Year", col="red",
     ylab = expression(paste(
       "Temperature␣Anomaly␣[", degree,"C]")),
     main ="Global␣Land-Ocean␣Average␣Annual␣Mean
␣␣␣␣␣Surface␣Temperature␣Anomalies:␣1880-2018",
     cex.lab = 1.2, cex.axis = 1.2)
lines(x2, y2, type="h",
      lwd = 3, tck = -0.02, col = "blue")
lines(x3, y3, lwd = 2)
```

```
# Python plot Fig. 1.3: A color bar chart of data
# define an array z with y number of ones
z = np.ones(y.size)
z[0] = -99
z[1] = -99
# computer 5-point moving average
for i in range(2, z.size - 2):
    z[i] = np.mean(y[i-2:i+3])
z[z.size - 2] = -99
z[z.size - 1] = -99
# define variables based on range
y1 = [y[i] for i in range(y.size) if y[i] >= 0]
x1 = [x[i] for i in range(y.size) if y[i] >= 0]
y2 = [y[i] for i in range(y.size) if y[i] < 0]
x2 = [x[i] for i in range(y.size) if y[i] < 0]
y3 = z[2:x.size-2]
x3 = x[2:x.size-2]

# plot the moving average
plt.plot(x3, y3, 'black', linewidth = 3)
plt.title("Global␣Land-Ocean␣Average␣Annual␣Mean␣\n␣\
Surface␣Temperature␣Anomaly:␣1880␣-␣2018", pad = 20)

# create bar chart
plt.bar(x1, y1, color = 'red');
plt.bar(x2, y2, color = 'blue');
plt.xlabel("Year", size = 25, labelpad = 20)
plt.ylabel("Temperature␣Anomaly␣[$\degree$C]",
           size = 25, labelpad = 20)
plt.show()
```

B.1.3 Statistical indices

The commonly used basic statistical indices include mean, variance , standard deviation, skewness, kurtosis, and quantiles. We first use R to calculate these indices for the global average annual mean temperature anomalies. Then we describe their mathematical formulas and interpret the numerical results.

```
#R code for computing statistical indices
setwd("/Users/sshen/climstats")
NOAAtemp = read.table(
  "data/aravg.ann.land_ocean.90S.90N.v4.0.1.201907.txt",
  header=FALSE)
temp2018=NOAAtemp[1:139,2] #use the temp data up to 2018
head(temp2018) #show the first six values
#[1] -0.370221 -0.319993 -0.320088 -0.396044 -0.458355 -0.470374
mean(temp2018) #mean
#[1] -0.1858632
sd(temp2018) #standard deviation
#[1] 0.324757
var(temp2018) #variance
#[1] 0.1054671
```

```
library(e1071)
#This R library is needed to compute the following parameters
#install.packages("e1071") #if it is not in your computer
skewness(temp2018)
#[1] 0.7742704
kurtosis(temp2018)
#[1] -0.2619131
median(temp2018)
#[1] -0.274434
quantile(temp2018,probs= c(0.05, 0.25, 0.75, 0.95))
#        5%        25%        75%        95%
#  -0.5764861 -0.4119770  0.0155245  0.4132383
```

We use $x = \{x_1, x_2, \cdots, x_n\}$ to denote the sampling data for a time series. The statistical indices computed above by R are based on the following mathematical formulas for mean, variance, standard deviation, skewness, and kurtosis:

$$\text{Mean: } \mu(x) = \frac{1}{n}\sum_{k=1}^{n} x_k, \tag{B.3}$$

$$\text{Variance by unbiased estimate: } \sigma^2(x) = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \mu(x))^2, \tag{B.4}$$

$$\text{Standard deviation: } \sigma(x) = (\sigma^2(x))^{1/2}, \tag{B.5}$$

$$\text{Skewness: } \gamma_3(x) = \frac{1}{n}\sum_{k=1}^{n}\left(\frac{x_k - \mu(x)}{\sigma}\right)^3, \tag{B.6}$$

$$\text{Kurtosis: } \gamma_4(x) = \frac{1}{n}\sum_{k=1}^{n}\left(\frac{x_k - \mu(x)}{\sigma}\right)^4 - 3. \tag{B.7}$$

A *quantile* cuts a sorted sequence of data. For example, the 25th quantile, also called 25th percentile or 25% quantile, is the value that has 25% of the sorted data smaller than this value and 75% larger than this value. The 50th percentile is also known as the median.

```
# Python code for computing statistical indices
temp2018 = np.array(NOAAtemp.loc[0:138, 1]) # data string
# arithmetic average of the data
print("The␣Mean␣is␣%f.\n" % stats.mean(temp2018))
# standard deviation of the data
print("The␣Standard␣Deviation␣is␣%f.\n" %
      stats.stdev(temp2018))
# variance of the data
print("The␣Variance␣is␣%f.\n" % stats.variance(temp2018))
# skewness of the data
print("The␣Skewness␣is␣%f.\n" % skewness(temp2018))
# kurtosis of the data
print("The␣Kurtosis␣is␣%f.\n" % kurtosis(temp2018))
# median of the data
print("The␣Median␣is␣%f.\n" % stats.median(temp2018))
# percentiles
print("The␣5th,␣25th,␣75th␣and␣95th␣percentiles␣are:")
probs = [5, 25, 75, 95]
print([round(np.percentile(temp2018, p),5) for p in probs])
print()
```

The meaning of these indices may be explained as follows. The mean is the simple average of samples. The variance of climate data reflects the strength of variations of a climate system and has units of the square of the data entries, such as $[^\circ\ \mathrm{C}]^2$. You may have noticed the denominator $n-1$ instead of n, which is for an estimate of unbiased sample variance. The standard deviation describes the spread of the sample entries and has the same units as the data. A large standard deviation means that the samples have a broad spread.

Skewness is a dimensionless quantity. It measures the asymmetry of sample data. Zero skewness means a symmetric distribution about the sample mean. For example, the skewness of a normal distribution is zero. Negative skewness denotes a skew to the left, meaning the existence of a long tail on the left side of the distribution. Positive skewness implies a long tail on the right side.

The words "Kurtosis" and "kurtic" are Greek in origin and indicate peakedness. Kurtosis is also dimensionless and indicates the degree of peakedness of a probability distribution. The kurtosis of a normal distribution[1] is zero when 3 is subtracted as in Eq. (B.7). Positive kurtosis means a high peak at the mean, thus the distribution shape is slim and tall. This is referred to as leptokurtic. "Lepto" is Greek in origin and means thin or fine. Negative kurtosis indicates a low peak at the mean, thus the distribution shape is fat and short, referred to as platykurtic. "Platy" is also Greek in origin and means flat or broad.

For the 139 years of the NOAAGlobalTemp global average annual mean temperature anomalies, mean is -0.1959°C, which means that the 1880-2018 average is lower than the average during the climatology period 1971-2000. During the clima-

[1] Chapter 2 will have a detailed description of the normal distribution and other probabilisitic distributions.

tology period, the temperature anomaly average is approximately zero, and can be computed by the R command `mean(temp2018[92:121])`.

The variance of the data in the 139 years from 1880 to 2018 is 0.1055 [° C]^2, and the corresponding standard derivation is 0.3248 [° C]. The skewness is 0.7743, meaning skew to the right with a long tail on the right, thus with more extreme high temperatures than extreme low temperatures, as shown by Fig. B.4. The kurtosis is -0.2619, meaning the distribution is flatter than a normal distribution, as shown in the histogram Fig. B.4.

The median is -0.2744°C and is a number characterizing a set of samples such that 50% of the sample values are less than the median, and another 50% are greater than the median. To find the median, sort the samples from the smallest to the largest. The median is then the sample value in the middle. If the number of the samples is even, then the median is equal to the mean of the two middle sample values.

Quantiles are defined in a way similar to median by sorting. For example, 25-percentile (also called the 25th percentile) -0.4120°C is a value such that 25% of sample data are less than this value. By definition, 60-percentile is thus larger than 50-percentile. Here, percentile is a description of quantile relative to 100. Obviously, 100-percentile is the largest datum, and 0-percentile is the smallest one. Often, a box plot is used to show the typical quantiles (see Fig. B.5).

The 50-percentile (or 50th percentile) -0.2744°C is equal to the median. If the distribution is symmetric, then the median is equal to mean. Otherwise these two quantities are not equal. If the skew is to the right, then the mean is on the right of the median: the mean is greater than the median. If the skew is to the left, then the mean is on the left of the median: the mean is less than the median. Our 139 years of temperature data are right skewed and have mean equal to -0.1959°C, greater than their median equal to -0.2969°C.

B.2 Commonly used climate statistical plots

We will use the 139 years of NOAAGlobalTemp temperature data and R to illustrate some commonly used statistical figures: histogram, boxplot, q-q plot, and linear regression trend line.

B.2.1 Histogram of a set of data

A histogram of the NOAAGlobalTemp global average annual mean temperature anomalies data from 1880-2018 is shown in Fig. B.4, which can be generated by the following R code.

```
#R plot Fig. 1.4: Histogram and its fit
par(mar=c(3.5,3.5,2.5,1), mgp=c(2,0.8,0))
h <- hist(NOAAtemp[1:139, 2],
```

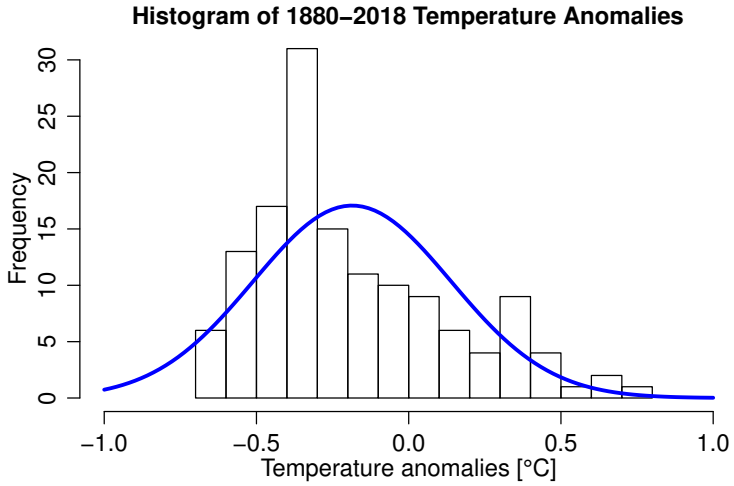


Fig. B.4 Histogram and its normal distribution fit of the global average annual mean temperature anomalies from 1880-2018. Each small interval in the horizontal coordinate is called a bin. The frequency in the vertical coordinate is the number of temperature anomalies in a given bin. For example, the frequency for the bin [-0.5, -0.4]°C is 17.

```
6380            main="Histogram␣of␣1880-2018␣Temperature␣Anomalies",
6381            xlab=expression(paste(
6382                    "Temperature␣anomalies␣[", degree, "C]")),
6383            xlim=c(-1,1), ylim=c(0,30),
6384            breaks=10, cex.lab=1.2, cex.axis=1.2)
6385  xfit <- seq(-1, 1, length=100)
6386  areat <- sum((h$counts)*diff(h$breaks[1:2]))#Normalization area
6387  yfit <- areat*dnorm(xfit,
6388                      mean=mean(NOAAtemp[1:139,2]),
6389                      sd=sd(NOAAtemp[1:139,2]))
6390  #Plot the normal fit on the histogram
6391  lines(xfit, yfit, col="blue", lwd=3)
```

```
# Python plot Fig. 1.4: Histogram and its fit
mu, std = scistats.norm.fit(y)
# create an evenly spaced sequence of 100 points in [-1,1]
points = np.linspace(-1, 1, 100)
plt.hist(y, bins = 20, range=(-1,1), color ='white',
         edgecolor = 'k', density = True);
plt.plot(points, scistats.norm.pdf(points, mu, std),
         color = 'b', linewidth = 3)
plt.title(r"Histogram of 1880 - 2018 Temperature Anomalies",
          pad = 20)
plt.xlabel("Temperature Anomalies [$\degree$C]",
           size = 25, labelpad = 20)
plt.ylabel("Frequency", size = 25, labelpad = 20)
plt.show()
```

The shape of the histogram agrees with the characteristics predicted by the statistical indices in the previous sub-section:

(i) The distribution is asymmetric and skewed to the right with skewness equal to 0.7743.

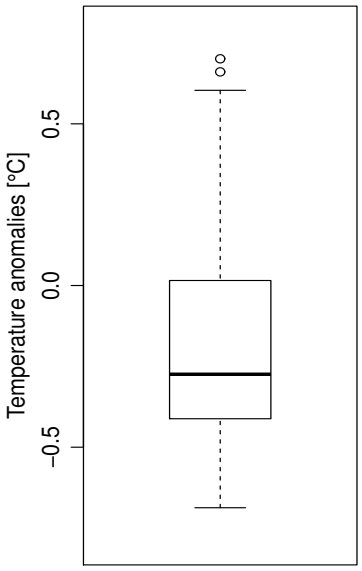
(ii) The distribution is platykurtic with a kurtosis equal to -0.2619, i.e., it is flatter than the standard normal distribution indicated by the blue curve.

B.2.2 Box plot

Figure B.5 is the box plot of the 1880-2018 NOAAGlobalTemp global average annual mean temperature anomaly data, and can be made from the following R command

```
#R plot Fig. 1.5: Box plot
boxplot(NOAAtemp[1:139, 2], ylim = c(-0.8, 0.8),
        ylab=expression(paste(
          "Temperature anomalies [", degree, "C]")),
        width=NULL, cex.lab=1.2, cex.axis=1.2)
```

The rectangular box's mid line indicates the median, which is -0.2744°C. The rectangular box's lower boundary is the first quartile, i.e., 25th percentile -0.4120°C. The box's upper boundary is the third quartile, i.e., the 75th percentile 0.0155°C. The box's height is the third quartile minus the first quartile, and is called the interquartile range (IQR). The upper "whisker" is the third quartile plus 1.5 IQR. The lower whisker is supposed to be at the first quartile minus 1.5 IQR. However, this whisker would then be lower than the lower extreme. In this case, the lower whisker takes the value of the lower extreme, which is -0.6872 °C. The points outside of the two whiskers are considered outliers. Our dataset has two outliers, which are 0.6607 and 0.7011 °C, and are denoted by two small circles in the box plot. The two outliers occurred in 2015 and 2016, respectively.



1

Fig. B.5 Box plot of the global average annual mean temperature anomalies from 1880-2018.

```
#Python plot Fig. 1.5: Box plot
medianprops = dict(linestyle='-', linewidth=2.5, color='k')
whiskerprops = dict(linestyle='--',linewidth=2.0, color='k')
plt.boxplot(y, medianprops = medianprops,
            whiskerprops = whiskerprops);
plt.title("Boxplot of 1880 - 2018 Temperature Anomalies",
          pad = 20)
plt.ylabel("Temperature Anomalies [$\degree$C]",
           size = 25, labelpad = 20)
y_ticks = np.linspace(-0.5,0.5,3)
plt.yticks(y_ticks)
plt.show()
```

B.2.3 Q-Q plot

Figure B.6 shows Quantile-Quantile (Q-Q) plots, also denoted by by q-q plots, qq-plots, or QQ-plots.

The function of a Q-Q plot is to compare the distribution of a given set of data with a specific reference distribution, such as a standard normal distribution with zero mean and standard deviation equal to one, denoted by $N(0,1)$. A Q-Q plot lines up the percentiles of data on the vertical axis and the same number of

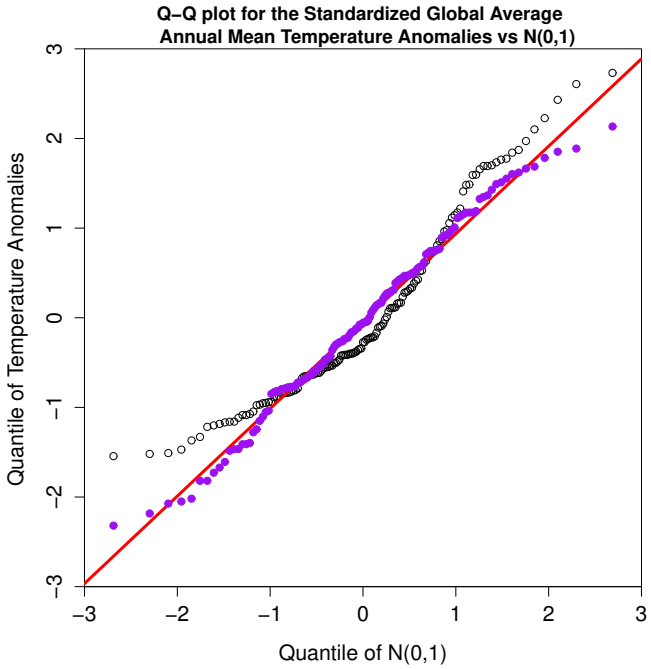


Fig. B.6 Black empty-circle points are the Q-Q plot of the standardized global average annual mean temperature anomalies vs. standard normal distribution. The purple points are the Q-Q plot for the data simulated by `rnorm(139)`. The red is the distribution reference line of $N(0,1)$.

percentiles of the specific reference distribution on the horizontal axis. The pairs of the quantiles $(x_i, y_i), i = 1, 2, \cdots, n$ determine the points on the Q-Q plot. Here, x_i and y_i correspond to the same cumulative percentage or probability p_i for both x and y variables, where p_i monotonically increases from approximately zero to one as i goes from 1 to n. A red Q-Q reference line is plotted as if the vertical axis values are also the quantiles of the given specific distribution. Thus, the Q-Q reference line should be diagonal.

The black empty circles in Fig.B.4 compare the quantiles of the standardized global average annual mean temperature anomalies marked on the vertical axis with those of the standard normal distribution marked on the horizontal axis. The standardized anomalies are equal to anomalies divided by the sample standard deviation. The purple dots shows a Q-Q plot of a set of 139 random numbers simulated by the standard normal distribution. As expected, the simulated points are located close to the red diagonal line, which is the distribution reference line of $N(0,1)$. On the other hand, the temperature Q-Q plot shows a considerable degree of scattering of the points away from the reference line. We may intuitively conclude that the global average annual temperature anomalies from 1880 to 2018 are not exactly distributed according to a normal distribution, also known as Gaussian) distribu-

tion. However, we may also conclude that the distribution of these temperatures is not very far away from the normal distribution either, because the points on the Q-Q plot are not very far away from the distribution reference line, and also because even the simulated $N(0,1)$ points are noticeably off the reference line for the extremes.

Figure B.6 can be generated by the following R code.

```
# R plot Fig. 1.6: Q-Q plot for the standardized
# global average annual mean temperature anomalies
temp2018 <- NOAAtemp[1:139,2]
tstand <- (temp2018 - mean(temp2018))/sd(temp2018)
set.seed(101)
qn <- rnorm(139) #simulate 139 points by N(0,1)
qns <- sort(qn) # sort the points
qq2 <- qqnorm(qns,col="blue", lwd = 2)

setEPS() #Automatically saves the eps file
postscript("fig0106.eps", height=7, width=7)
par(mar = c(4.5,5,2.5,1), xaxs = "i", yaxs = "i")
qt = qqnorm(tstand,
  main = "Q-Q␣plot␣for␣the␣Standardized␣Global␣Average
␣␣Annual␣Mean␣Temperature␣Anomalies␣vs␣N(0,1)",
   ylab="Quantile␣of␣Temperature␣Anomalies",
   xlab="Quantile␣of␣N(0,1)",
  xlim=c(-3,3), ylim = c(-3,3),
           cex.lab = 1.3, cex.axis = 1.3)
qqline(tstand, col = "red", lwd=3)
points(qq2$x, qq2$y, pch = 19,
       col ="purple")
dev.off()
```

In the R code, we first standardize, also called normalize, the global average annual mean temperature data by subtracting the data mean and dividing by the data's standard deviation. Then, we use these 139 years of standardized global average annual mean temperature anomalies to generate a Q-Q plot, which is shown in Fig. B.6.

```
#Python plot Fig. 1.6: Q-Q plot for the standardized
# global average annual mean temperature anomalies
NOAAtemp = read_table(
     "data/aravg.ann.land_ocean.90S.90N.v4.0.1.201907.txt",
                        header = None, delimiter = "\s+")
x = np.array(NOAAtemp.loc[0:138, 0])
y = np.array(NOAAtemp.loc[0:138, 1])
line = np.linspace(-3, 3, y.size)
tstand = np.sort((y - np.mean(y))/np.std(y))
# simulate 139 points following N(0,1)
qn = np.random.normal(size = y.size)
qns = np.sort(qn) # sort the points
qq2 = sm.qqplot(qns)
fig = plt.figure(figsize=(12,12)) # set up figure
sm.qqplot(tstand, color = "k", linewidth = 1)
# plot diagonal line
plt.plot(line, line, 'r-', linewidth = 3)
# Q-Q plot of standard normal simulations
plt.plot(line, qns, 'mo')
plt.tick_params(length=6, width=2, labelsize=20)
plt.title("Q-Q␣plot␣for␣the␣Standardized␣Global␣\n␣\
Temperature␣Anomolies␣vs␣N(0,1)", pad = 20)
plt.xlabel("Quantile␣of␣N(0,1)", size = 25, labelpad = 20)
plt.ylabel("Quantile␣of␣Temperature␣Anomalies", size = 25)
plt.show()
```

B.2.4 Plot a linear trend line

Climate data analysis often involves plotting a linear trend line for time series data, such as the linear trend for the global average annual mean surface temperature anomalies, shown in Fig. B.7. The R code for plotting a linear trend line of data sequence `y` and time sequence `t` is `abline(y  t)`.

Figure B.7 can be generated by the following computer code.

```
# R plot Fig. 1.7: Data line graph with a linear trend line
par(mar=c(3.5,3.5,2.5,1), mgp=c(2,0.8,0))
plot(NOAAtemp[1:139,1], NOAAtemp[1:139,2],
     type="l", col="brown", lwd=3,
     main="␣Global␣Land-Ocean␣Average␣Annual␣Mean
Surface␣Temperature␣Anomaly:␣1880-2018",
     cex.lab=1.2,cex.axis=1.2,
     xlab="Year",
     ylab=expression(paste(
       "Temperature␣Anomaly␣[", degree,"C]"))
)
abline(lm(NOAAtemp[1:139,2] ~ NOAAtemp[1:139,1]),
       lwd=3, col="blue")
lm(NOAAtemp[1:139,2] ~ NOAAtemp[1:139,1])
#       (Intercept)  NOAAtemp[1:139, 1]
#-13.872921             0.007023
#Trend 0.7023 degC/100a
text(1930, 0.5,
```

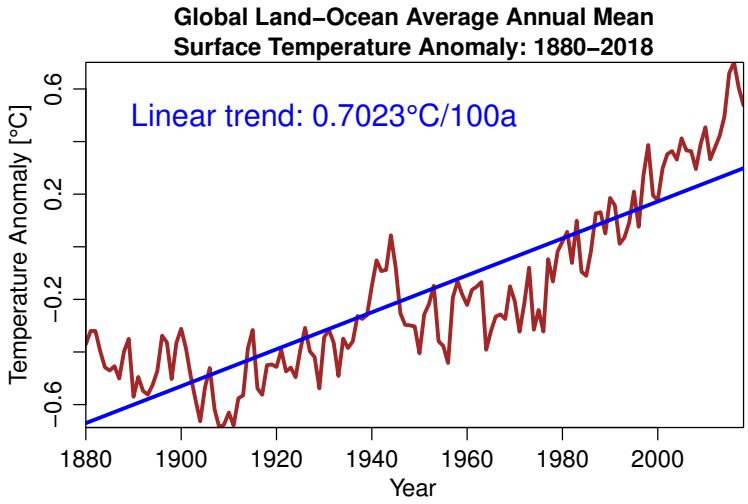


Fig. B.7 Linear trend line with the 1880-2018 global average annual mean surface temperature based on the NOAAGlobalTemp V4.0 dataset.

```
expression(paste("Linear␣trend:␣0.7023",
                 degree,"C/100a")),
    cex = 1.5, col="blue")
```

```
# Python plot Fig. 1.7: Data line graph with a trend line
trend = np.array(np.polyfit(x, y, 1))
abline = trend[1] + x*trend[0]
plt.plot(x, y, 'k-',
         color = 'tab:brown', linewidth = 3);
plt.plot(x, abline, 'k-', color = 'b', linewidth = 3);
plt.title("Global␣Land-Ocean␣Average␣Annual␣Mean␣\n␣\
Surface␣Temperature␣Anomaly:␣1880␣-␣2018", pad = 20);
plt.text(1880, 0.5, r"Linear␣trend:␣0.7023$\degree$C/100a",
         color= 'b', size = 28)
plt.xlabel("Year", size = 25, labelpad = 20)
plt.ylabel("Temperature␣Anomaly␣[$\degree$C]",
           size = 25, labelpad = 20)
```

B.3 Read netCDF data file and plot spatial data maps

B.3.1 Read netCDF data

Climate data are at spatiotemporal points, such as at the grid points on the Earth's surface and at a sequence of time. NetCDF (Network Common Data Form) is a

popular file format for modern climate data with spatial locations and temporal records. The gridded NOAAGlobalTemp data has a netCDF version, and can be downloaded from

`https://www.esrl.noaa.gov/psd/data/gridded/data.noaaglobaltemp.html`

The data are written into a 3D array, with 2D latitude-longitude for space, and 1D for time. R and Python can read and plot the netCDF data. We use the NOAAGlobalTemp as an example to illustrate the netCDF data reading and plotting. Figure B.8 displays a temperature anomaly map for the entire globe for December 2015.

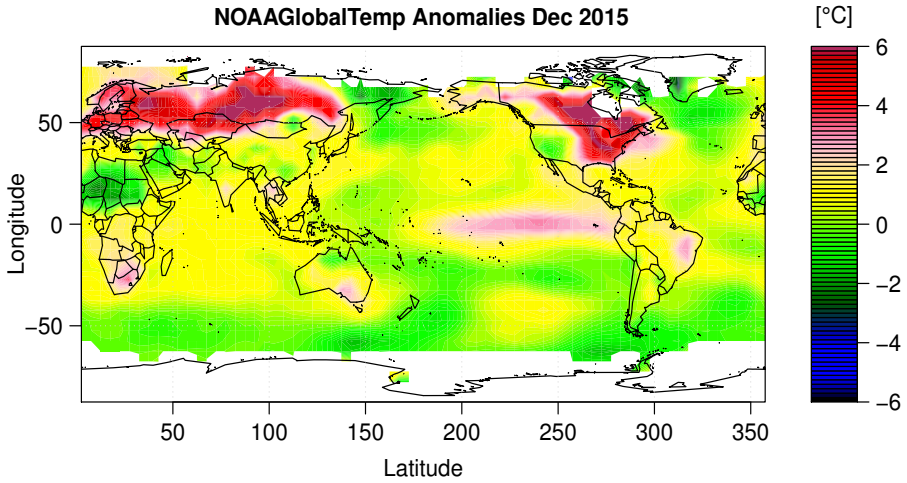


Fig. B.8 The surface temperature anomalies of December 2015 with respect to the 1971-2000 climatology. Data source: The NOAAGlobaTemp V4.0 gridded monthly data.

Figure B.8 can be generated by the following computer code.

```
# R read the netCDF data: NOAAGlobalTemp
setwd("/Users/sshen/climstats")
#install.packages("ncdf4")
library(ncdf4)
nc = ncdf4::nc_open("data/air.mon.anom.nc")
nc # describes details of the dataset
Lat <- ncvar_get(nc, "lat")
Lat # latitude data
#[1] -87.5 -82.5 -77.5 -72.5 -67.5 -62.5
Lon <- ncvar_get(nc, "lon")
Lon # longitude data
#[1]   2.5   7.5  12.5  17.5  22.5  27.5
Time <- ncvar_get(nc, "time")
head(Time) # time data in Julian days
#[1] 29219 29250 29279 29310 29340 29371
library(chron) # convert Julian date to calendar date
nc$dim$time$units # .nc base time for conversion
#[1] "days since 1800-1-1 00:00:0.0"
month.day.year(29219,c(month = 1, day = 1, year = 1800))
```

```
#1880-01-01 # the beginning time of the dataset
tail(Time)
#[1] 79988 80019 80047 80078 80108 80139
month.day.year(80139,c(month = 1, day = 1, year = 1800))
#2019-06-01 # the end time of the dataset

# extract anomaly data in (lon, lat, time) coordinates
NOAAgridT <- ncvar_get(nc, "air")
dim(NOAAgridT) # dimensions of the data array
#[1]  72   36 1674 #5-by-5, 1674 months from Jan 1880-Jun 2019
```

```
#Python read the netCDF data: NOAAGlobalTemp
import netCDF4 as nc # import netCDF data reading package
# go to the working directory
os.chdir('/Users/sshen/climstats')
# read a .nc file from the folder named "data"
nc = nc.Dataset('data/air.mon.anom.nc')
# get the detailed description of the dataset
print(nc)
# extract latitude, longitude, time and temperature
lon = nc.variables['lon'][:]
lat = nc.variables['lat'][:]
time = nc.variables['time'][:]
air = nc.variables['air']
# covert Julian date to calendar date
from netCDF4 import num2date
from datetime import datetime, date, timedelta
from matplotlib.dates import date2num, num2date
units = nc.variables['time'].units
print(units)
dtime = num2date(time)
```

B.3.2 Plot a spatial map of temperature

The NOAAGlobalTemp anomaly data on a 5-degree latitude-longitude grid for a given time can be represented by a color map. Figure B.8 shows the temperature anomaly map for December 2015, an El Niño month. The eastern tropical Pacific has positive anomalies, which is a typical El Niño signal. That particular month also exhibited a very large anomaly across Europe, and the eastern United States and Canada. The white areas over the high latitude regions lack data.

The figure can be generated by the following R code.

```
#R plot Fig. 1.8: Dec 2015 surface temp anomalies map
library(maps)# requires maps package
mapmat=NOAAgridT[,,1632]
# Julian date time 1632 corresponds to Dec 2015
mapmat=pmax(pmin(mapmat,6),-6) # put values in [-6, 6]
int=seq(-6, 6, length.out=81)
rgb.palette=colorRampPalette(c('black','blue',
   'darkgreen', 'green', 'yellow','pink','red','maroon'),
                             interpolate='spline')
```

```
par(mar=c(3.5, 4, 2.5, 1), mgp=c(2.3, 0.8, 0))
filled.contour(Lon, Lat, mapmat,
               color.palette=rgb.palette, levels=int,
  plot.title=title(main="NOAAGlobalTemp␣Anomalies␣Dec␣2015",
            xlab="Latitude", ylab="Longitude", cex.lab=1.2),
  plot.axes={axis(1, cex.axis=1.2, las=1);
     axis(2, cex.axis=1.2, las=2); map('world2', add=TRUE); grid()},
  key.title=title(main=expression(paste("[", degree, "C]"))),
  key.axes={axis(4, cex.axis=1.2)})
```

```
# Python plot Fig. 1.8: Dec 2015 surface temp anomalies map
dpi = 100
fig = plt.figure(figsize = (1100/dpi, 1100/dpi), dpi = dpi)
ax  = fig.add_axes([0.1, 0.1, 0.8, 0.9])
# create map
dmap = Basemap(projection = 'cyl', llcrnrlat = min(lat),
               urcrnrlat = max(lat), resolution = 'c',
               llcrnrlon = min(lon), urcrnrlon = max(lon))
# draw coastlines, state and country boundaries, edge of map
dmap.drawcoastlines()
dmap.drawstates()
dmap.drawcountries()
# convert latitude/longitude values to plot x/y values
x, y = dmap(*np.meshgrid(lon, lat))
# draw filled contours
cnplot = dmap.contourf(x, y, mapmat, clev, cmap=myColMap)
# tick marks
ax.set_xticks([0, 50, 100, 150, 200, 250, 300, 350])
ax.set_yticks([-50,0,50])
ax.tick_params(length=6, width=2, labelsize=20)
# add colorbar
# pad: distance between map and colorbar
cbar = dmap.colorbar(cnplot, pad = "4%",drawedges=True,
           shrink=0.55, ticks = [-6,-4,-2,0,2,4,6])
# add colorbar title
cbar.ax.set_title('[$\degree$C]', size= 17, pad = 10)
cbar.ax.tick_params(labelsize = 15)
# add plot title
plt.title('NOAAGlobalTemp␣Anomalies␣Dec␣2015',
          size = 25, fontweight = "bold", pad = 15)
# label x and y
plt.xlabel('Longitude', size = 25, labelpad = 20)
plt.ylabel('Latitude', size = 25, labelpad = 10)
# display on screen
plt.show()
```

B.3.3 Panoply plot of a spatial map of temperature

You can also use the Panoply software package to plot the map (See Fig. B.9). This is a very powerful data visualization tool developed by NASA specifically for

displaying netCDF files. The software package is free and can be downloaded from `www.giss.nasa.gov/tools/panoply` .

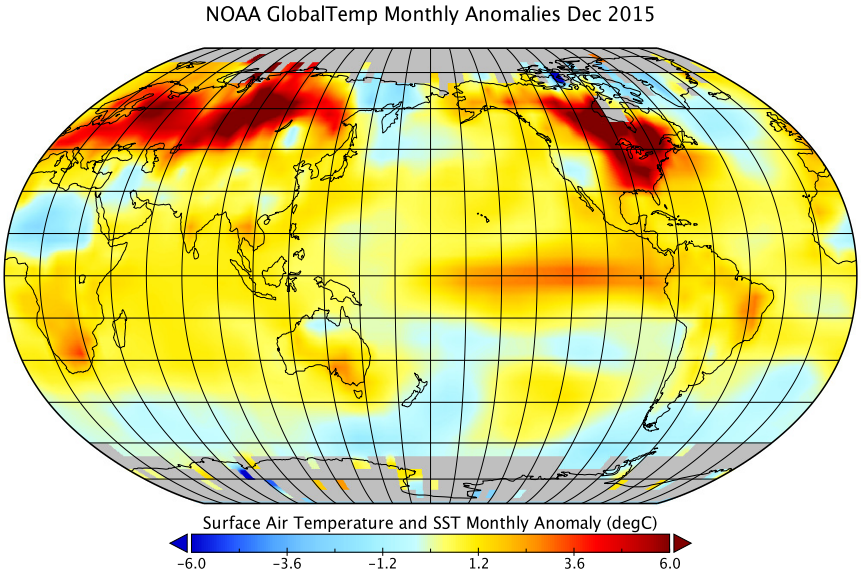


Fig. B.9 A Panoply plot of Robinson projection map for the surface temperature anomalies of December 2015.

To make a Panoply plot, open Panoply and choose `Open` from the `File` menu. `Open` dropdown which will allow you to go to the right directory to find the netCDF file you wish to plot. In our case, the file is `air.mon.anom.nc`. Choose the climate parameter `air`. Click on `Create Plot`. A map will show up. Then you have many choices to modify the map, ranging from the data `Array` , `Scale`, and `Map`, $\cdots$, and finally produce the figure. You can then tune the figure by choosing different graphics parameters underneath the figure, such as `Array(s)` to choose which month to plot, `Map` to choose the map projection types and the map layout options, and `Labels` to type proper captions and labels.

To learn more about Panoply, please use an online Panoply tutorial, such as the NASA Panoply help page `https://www.giss.nasa.gov/tools/panoply/help/`.

Compared with the R or Python map Fig. B.5, the visualization effect of the Panoply map seems more appealing. However, R and Python have an advantage of flexibility and can deal with all kinds of data. For example, the Plotly graphing library in R `https://plotly.com/r/` and in Python `https://plotly.com/python/` can even make interactive and 3D graphics. You may find some high quality figures from the Intergovernmental Panel on Climate Change (IPCC) report (2021) `www.ipcc.ch` and reproduce them using the computing tools described here.

B.4 1D-space-1D-time data and Hovmöller diagram

A very useful climate data visualization technique is the Hovmöller diagram. It displays how a climate variable varies with respect to time along a given line section of latitude, longitude or altitude. It usually has the abscissa for time and ordinate for the line section. A Hovmöller diagram can conveniently show time evolution of a spatial pattern, such as wave motion from south to north or from west to east.

Figure B.10 is a Hovmöller diagram for the sea surface temperature (SST) anomalies at a longitude equal to 240°, i.e., 120°W, in a latitude interval [30°S, 30°N], with time range from January 1989 to December 2018. When the red strips become strong from the south to north, a strong El Niño occurs, such as those in the 1997-1998 and 2015-2016 winters.

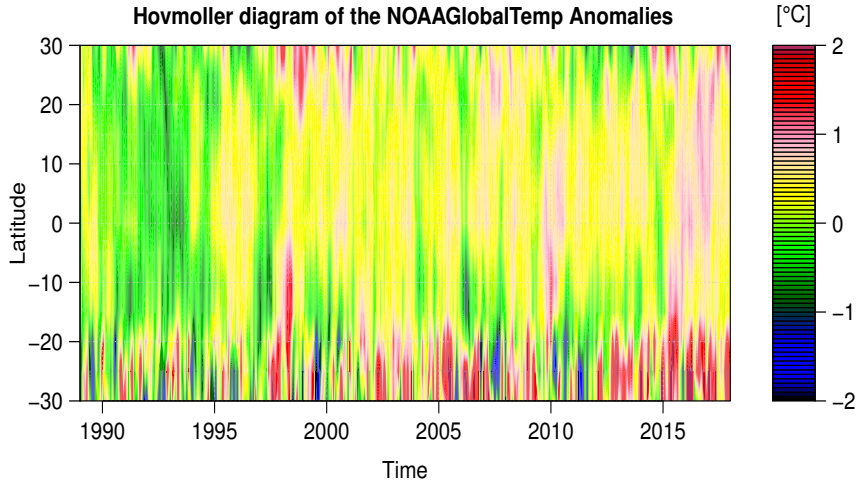


Fig. B.10 Hovmöller diagram for the gridded NOAAGlobalTemp monthly anomalies at longitude 120°W and a latitude interval [30°S, 30°N].

The Hovmöller diagram Fig. B.10 may be plotted by the following R code.

```
#R plot Fig. 1.10: Hovmoller diagram
library(maps)
mapmat=NOAAgridT[30,12:24,1309:1668]
#Longitude= 240 deg, Lat =[-30 30] deg
#Time=Jan 1989-Dec 2018: 30 years
mapmat=pmax(pmin(mapmat,2),-2) # put values in [-2,2]
par(mar=c(4,5,3,0))
int=seq(-2,2,length.out=81)
rgb.palette=colorRampPalette(c('black','blue',
 'darkgreen','green', 'yellow','pink','red','maroon'),
                              interpolate='spline')
par(mar=c(3.5,3.5,2.5,1), mgp=c(2.4, 0.8, 0))
x = seq(1989, 2018, len=360)
y = seq(-30, 30, by=5)
```

```
filled.contour(x, y, t(mapmat),
       color.palette=rgb.palette, levels=int,
       plot.title=title(main=
    "Hovmoller diagram of the NOAAGlobalTemp Anomalies",
                xlab="Time",ylab="Latitude", cex.lab=1.2),
      plot.axes={axis(1, cex.axis=1.2);
                 axis(2, cex.axis=1.2);
                 map('world2', add=TRUE);grid()},
        key.title=title(main =
                expression(paste("[", degree, "C]"))),
        key.axes={axis(4, cex.axis=1.2)})
```

```
# Python plot Fig. 1.10: Hovmoller diagram
mapmat2 = NOAAgridT[1308:1668,11:23,29]

# define values between -2 and 2
mapmat2 = np.array([[j if j < 2 else 2 for j in i]
                    for i in mapmat2])
mapmat2 = np.array([[j if j > -2 else -2 for j in i]
                    for i in mapmat2])
# find dimensions
mapmat2 = np.transpose(mapmat2)
print(mapmat2.shape)
lat3 = np.linspace(-30,30, 12)
print(lat3.shape)
time = np.linspace(1989, 2018, 360)
print(time.shape)

# plot functions
myColMap = LinearSegmentedColormap.from_list(
    name='my_list',
    colors=['black','blue','darkgreen','green','lime',
            'yellow', 'pink','red','maroon'],  N=100)
clev2 = np.linspace(mapmat2.min(), mapmat2.max(), 501)
contf = plt.contourf(time, lat3, mapmat2,
                     clev2, cmap=myColMap);

plt.text(2019.2, 31.5,
         "[$\degree$C]", color='black', size = 23)
plt.title("Hovmoller diagram of the \n \
NOAAGlobalTemp Anomalies",
          fontweight = "bold", size = 25, pad = 20)
plt.xlabel("Time", size = 25, labelpad = 20)
plt.ylabel("Latitude", size = 25, labelpad = 12)
colbar = plt.colorbar(contf, drawedges=False,
                      ticks = [-2,-1,0,1,2])
```

B.5 4D netCDF file and its map plotting

The 4D climate data means 3D spatial dimensions and 1D time. For example, the NCEP Global Ocean Data Assimilation System (GODAS) monthly water temperature data are at 40 depth levels ranging from 5 meters to 4478 meters and at 1/3 degree latitude by 1 degree longitude horizontal resolution and are from January 1980. The NOAA-CIRES 20th Century Reanalysis (20CR) monthly air temperature data are at 24 different pressure levels ranging from 1000 mb to 10 mb and at 2-degree latitude and longitude horizontal resolution and are from January 1851.

GODAS data can be downloaded from NOAA ESRL
`www.esrl.noaa.gov/psd/data/gridded/data.godas.html`
The data for each year is a netCDF file and is about 140MB. The following R code can read the GODAS 2015 data into R.

```
# R read a 4D netCDF file: lon, lat, level, time
setwd("/Users/sshen/climstats")
library(ncdf4)
# read GODAS data 1-by-1 deg, 40 levels, Jan-Dec 2015
nc=ncdf4::nc_open("data/godas2015.nc")
nc
Lat <- ncvar_get(nc, "lat")
Lon <- ncvar_get(nc, "lon")
Level <- ncvar_get(nc, "level")
Time <- ncvar_get(nc, "time")
head(Time)
#[1] 78527 78558 78586 78617 78647 78678
library(chron)
month.day.year(78527,c(month = 1, day = 1, year = 1800))
# 2015-01-01
# potential temperature pottmp[lon, lat, level, time]
godasT <- ncvar_get(nc, "pottmp")
dim(godasT)
#[1] 360 418  40  12,
#i.e., 360 lon, 418 lat, 40 levels, 12 months=2015
t(godasT[246:250, 209:210, 2, 12])
#Dec level 2 (15-meter depth) water temperature [K] of
#a few grid boxes over the eastern tropical Pacific
#         [,1]     [,2]     [,3]     [,4]     [,5]
#[1,] 300.0655 299.9831 299.8793 299.7771 299.6641
#[2,] 300.1845 300.1006 299.9998 299.9007 299.8045
```

```
# Python read a netCDF data file
import netCDF4 as nc
# go to your working directory
os.chdir('/Users/sshen/climstats')
# read a .nc file from the folder named "data"
nc1 = nc.Dataset('data/godas2015.nc')
# get the detailed description of the dataset
print(nc1)
# dimensions of the 2015 pottem data array
godasT = nc1.variables['pottmp'][:]
print(godasT.shape)
# (12, 40, 418, 360)
```

Figure B.11 shows the 2015 annual mean water temperature at 195 meters depth based on the GODAS data. At this depth level, the equatorial upwelling appears: The deep ocean cooler water in the equatorial region upwells and makes the equatorial water cool. The equatorial water is not the hottest anymore at this level, and is cooler than the water in some subtropical regions as shown in Fig. B.11.

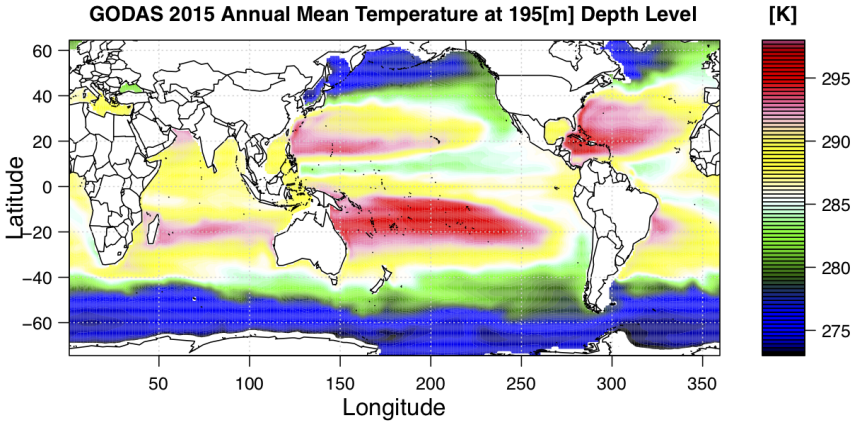


Fig. B.11 The 2015 annual mean water temperature at 195 meters depth based on GODAS data.

Figure B.11 can be generated by the following R code.

```
# R plot Fig. 1.11: The ocean potential temperature
# the 20th layer from surface: 195 meters depth
# compute 2015 annual mean temeprature at 20th layer
library(maps)
climmat=matrix(0,nrow=360,ncol=418)
sdmat=matrix(0,nrow=360,ncol=418)
Jmon<-1:12
for (i in 1:360){
  for (j in 1:418){
    climmat[i,j] = mean(godasT[i,j,20,Jmon]);
    sdmat[i,j]=sd(godasT[i,j,20,Jmon])
```

```
                    }
                  }
int=seq(273,298,length.out=81)
rgb.palette=colorRampPalette(c('black','blue',
    'darkgreen','green', 'white','yellow',
    'pink','red','maroon'), interpolate='spline')
par(mar=c(3.5, 3.5, 2.5, 0), mgp=c(2, 0.8, 0))
filled.contour(Lon, Lat, climmat,
     color.palette=rgb.palette, levels=int,
    plot.title=title(main=
  "GODAS␣2015␣Annual␣Mean␣Temperature␣at␣195[m]␣Depth␣Level",
            xlab="Longitude",ylab="Latitude",
            cex.lab=1.3, cex.axis=1.3),
plot.axes={axis(1); axis(2); map('world2', add=TRUE);grid()},
            key.title=title(main="[K]"))
```

```
# Python plot Fig. 1.11: A spatial map from 4D data
climmat = np.zeros((360, 418))
for i in range(360):
    for j in range(418):
      climmat[i,j] = np.mean(godasT[:, 20, j, i])
climmat = np.transpose(climmat)
lat4 = np.linspace(-75, 65, 418)
long = np.linspace(0, 360, 360)

myColMap = LinearSegmentedColormap.from_list(
    name='my_list',
    colors=['black','blue','darkgreen','green',
            'white','yellow','pink','red','maroon'],
          N=100)
plt.figure(figsize=(18,12));
clev3 = np.arange(godasT.min(), godasT.max(), 0.1)
contf = plt.contourf(long, lat4, climmat,
                     clev3, cmap=myColMap);
plt.text(382, 66, "[K]", fontsize=23, color='black')
plt.tick_params(length=6, width=2, labelsize=20)
m = Basemap(projection='cyl', llcrnrlon=0,
    urcrnrlon=360, resolution='l', fix_aspect=False,
    suppress_ticks=False, llcrnrlat=-75, urcrnrlat=65)
m.drawcoastlines(linewidth=1)
plt.title("GODAS␣2015␣Annual␣Mean␣Temperature␣\n␣\
at␣195[m]␣Depth␣Level",
           size = 25, fontweight = "bold", pad = 20)
plt.xlabel("Longitude", size = 25, labelpad = 20)
plt.ylabel("Latitude", size = 25, labelpad = 15)
plt.tick_params(length=6, width=2, labelsize=30)
colbar = plt.colorbar(contf, drawedges=False,
                      ticks = [275,280,285,290,295])
```

B.6 Paraview and 4DVD

Besides using R, Python and Panoply to plot climate data, other software packages may also be used to visualize data for some specific purposes, such as Paraview for 3D visualization and 4DVD for fast climate diagnostics and data delivery.

B.6.1 Paraview

Paraview is an open-source data visualization software package, available at `www.paraview.org/download`. There are many online tutorials, such as `www.paraview.org/tutorials`.

You may use the software ParaView to plot the GODAS data in a 3D view as shown in Fig. B.12 for the December 2015 water temperature.

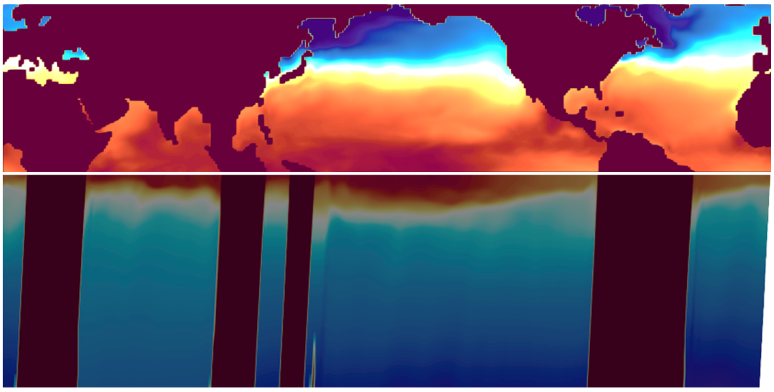


Fig. B.12 A 3D view of the December 2015 annual mean water temperature based on the GODAS data: The surface of the Northern Hemisphere and the cross-sectional map along the equator from surface to the 2,000 meters depth. The dark color indicates land. The red, yellow and blue colors indicate temperature.

B.6.2 4DVD

4DVD (4-dimensional visual delivery of big climate data) is a fast data visualization and delivery software system `www.4dvd.org`. It optimally harnesses the power of distributed computing, database and data storage to allow a large number of general public users to quickly visualize climate data. For example, teachers and students can use 4DVD to visualize climate model data in classrooms and download the visualized data instantly. The 4DVD website has a tutorial for users.

Here, we provide an example of 4DVD visualization of the NOAA-CIRES 20th Century Reanalysis climate model data for atmosphere. 4DVD can display a temperature map at a given time and given pressure level as shown in Fig. B.13 for

January 1851 and 750 millibar pressure level, or one can obtain a time series of the monthly air temperature for a specific grid point from January 1851. In fact, 4DVD can show multiple time series for the same latitude-longitude location but at different pressure levels. The 4DVD not only allows a user to visualize the data, but also download the data for the figure shown in the 4DVD system. In this sense, 4DVD is like a machine that plays data, while the regular DVD player machine, popular for about 30 years since 1980s, play DVD discs for music and movies.

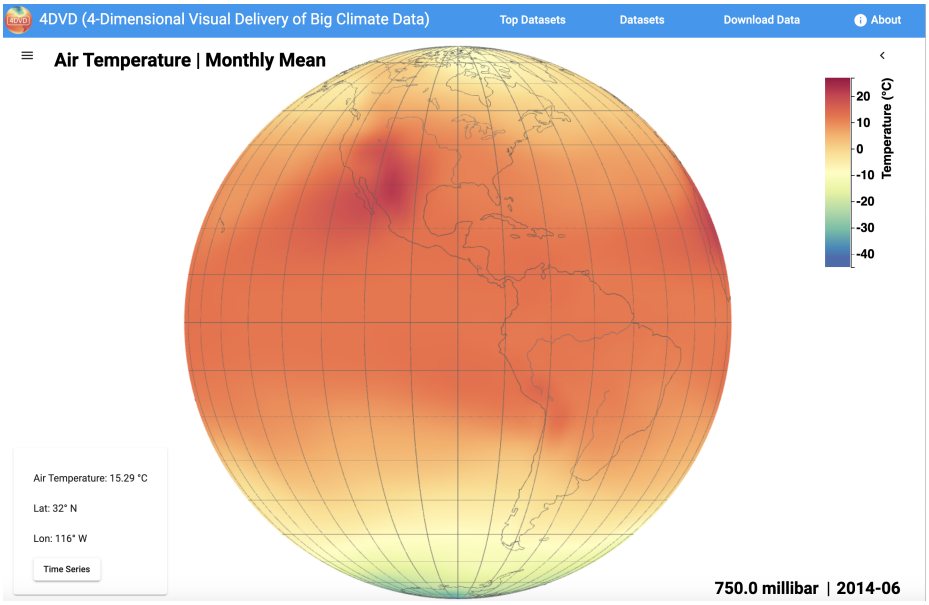


Fig. B.13 The 4DVD screenshot for the the NOAA-CIRES 20th Century Reanalysis temperature at 750 millibar height level for June 2014.

B.6.3 Other online climate data visualization tools

Besides R, Python, Panoply, ParaView, and 4DVD, there are many other data visualization and delivery software systems. A few popular and free ones are listed as follows.

Nullschool `https://nullschool.net` is a beautiful data visualizer of wind, ocean flows, and many climate parameters. It is supported by the data from a global numerical weather prediction system, named the Global Forecast System (GFS) run by the United States' National Weather Service (NWS).

Ventusky`www.ventusky.com` has both website and smartphone app. It has an attractive and user-friendly interface that allows users to get digital weather information instantly around the globe.

Climate Reanalyzer `climatereanalyzer.org` is a comprehensive tool for climate data plotting and download. It has a user-friendly interface for reanalysis and his-

torical station data. The data can be exported in either CSV (comma-separated values) format or JSON (JavaScript object notation) format.

Google Earth Engine `earthengine.google.com` provides visualization tools together with a huge multi-petabytes storage of climate data. Its modeling and satellite data sources are from multiple countries.

Giovanni `giovanni.gsfc.nasa.gov` is an online climate data plotting tool with an interface. It allows to download the plotted figures in different formats, such as png. It is supported by various kinds of NASA climate datasets.

Climate Engine `climateengine.org` is a web application for plotting climate and remote sensing data. Similar to Giovanni, it also has a tabular interface for a user to customized her data and maps.

NOAA Climate at a Glance `www.ncdc.noaa.gov/cag` is a data visualization tool mainly for visualizing the observed climate data over the United States. It has functions for both spatial maps and historical time series.

Web-based Reanalyses Intercomparison Tools (WRIT) `psl.noaa.gov/data/writ`. Similar to Giovanni, WRIT also has an interface table for a user to enter plot parameters. The plot (in postscript format) and its data (in netCDF format) can be downloaded. WRIT is designed for the data from climate reanalysis models.

B.6.4 Use ChatGPT as a study assistant

ChatGPT (Generative Pre-trained Transformer) is an artificial intelligence (AI) system that can help us develop computer codes, write essays, draft course syllabi, and more. For example, in your box of the ChatGPT interface, you can enter "Write a Python code to plot the cosine function as a blue thick dashed curve." ChatGPT will give you the following Python code:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi, np.pi, 100)
y = np.cos(x)

plt.plot(x, y, linestyle='--', linewidth=3, color='blue')
plt.xlabel('x')
plt.ylabel('cos(x)')
plt.title('Plot␣of␣the␣cosine␣function')
plt.show()
```

You can copy the above code and paste it to your Jupyter Notebook cell, then run the code to produce the curve of the cosine function in the interval $[-\pi, \pi]$. With this sample and according to your needs, you can modify the code for different functions, intervals, colors, thickness, title, axis labels, and more.

You can also ask ChatGPT to work on a dataset, such as "Write an R code to plot the NOAAGlobalTempts.csv data and its trend line." ChatGPT will give

you an R code. You copy and paste the code to your RStudio. Set your R code in the correct directory that has the NOAAGlobalTemp annual time series data file, `NOAAGlobalTempts.csv`. Run your R code to plot the data and the linear trend line. Of course, you can modify the code according to your requirements. The R code is as follows.

```
setwd('/Users/sshen/climmath/data')
library(tidyverse)

# Load the data
data <- read.csv("NOAAGlobalTempts.csv")

# Plot the data
ggplot(data, aes(x = Year, y = Tempts)) +
  geom_line() +
  geom_smooth(method = "lm", se = FALSE) +
  ggtitle("NOAA Global Temperature Data") +
  xlab("Year") +
  ylab("Temperature (Celsius)")
```

Another example is that you enter "Write an essay on global warming." ChatGPT will write an essay for you. Most likely, it does not reflect your idea, but it will give you a few tips, such as the definition of global warming, evidences and data, consequences, main causes, how to slow down the warming, and conclusion. Again, you can use this ChatGPT essay as a sample and modify the essay using your own data and ideas, such as green technologies, solar farms, microclimate resources for smart cities, trend of the oil business, lifestyles, AI in self-driving electric cars, and education.

ChatGPT was released by the OpenAI Lab in November 2022. You may use this tool as your assistant of learning and working. It is only your assistant. It is not you! You can efficiently use it to give you hints and inspire your ideas. You still have to produce your own work.

B.7 Chapter summary

This chapter has provided a brief introduction to useful statistical concepts and methods for climate science and has included the following materials.

(i) Formulas to compute the most commonly used statistical indices:

- Mean as a simple average, median as a datum whose value is in the middle of the sorted data sequence, i.e., the median is larger than 50% of the data and smaller than the remaining 50%,
- Standard deviation as a measure of the width of the probability distribution,
- Variance as the square of the standard deviation,
- Skewness as a measure of the degree of asymmetry in distribution, and

- Kurtosis as a measure of the peakedness of the data distribution compared to that of the normal distribution.

(ii) The commonly used statistical plots:

- Histogram for displaying the probability distribution of data,
- Linear regression line for providing a linear model for data,
- Box plot for quantifying the probability distribution of data, and
- Q-Q plot for checking whether the data are normally distributed.

(iii) Read and plot the netCDF file for plotting a 2D map by R and Python. Other data visualization tools, such as Panoply, Paraview, 4DVD, Plotly, and Nullschool, were briefly introduced. The online tools like 4DVD and Nullschool may be used for classroom teaching and learning.

Computer codes and climate data examples are given to demonstrate these concepts and the use of the relevant tools and formulas. With this background plus some R or Python programming skill, you will have sufficient knowledge to meet the needs of the basic statistical analysis for climate data.

References and Further Readings

[1] Hennemuth, B., Bender, S., K. Bulow, N. Dreier, E. Keup-Thiel, O. Kruger, C. Mudersbach, C. Radermacher, and R. Schoetter, 2013: *Statistical methods for the analysis of simulated and observed climate data, applied in projects and institutions dealing with climate change impact and adaptation.* CSC Report 13, Climate Service Center, Germany, 135pp. URL:

`http://www.climate-service-center.de/imperia/md/content/csc/projekte/csc-report13_englisch_final-mit_umschlag.pdf`

This free statistics recipe book outlines numerous methods for climate data analysis, which are collected and edited by climate science professionals and have examples of real climate data.

[2] IPCC, 2021: AR6 Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Masson-Delmotte, V., P. Zhai, A. Pirani, S. L. Connors, C. Pan, S. Berger, N. Caud, Y. Chen, L. Goldfarb, M. I. Gomis, M. Huang, K. Leitzell, E. Lonnoy, J. B. R. Matthews, T. K. Maycock, T. Waterfield, O. Yeleki, R. Yu and B. Zhou (eds.)]. Cambridge University Press.

This is the famous IPCC report for free at the IPCC website, `https://www.ipcc.ch/report/ar6/wg1/`. It includes many high quality figures plotted from climate data

[3] NCEI, 2021: *Anomalies vs. Temperature* Last accessed on 12 May 2021.

`https://www.ncdc.noaa.gov/monitoring-references/dyk/anomalies-vs-temperature`

This is an excellent site for education and gives an easy-to-understand justification of the use of anomalies. From this site, you can find many other educational resources for climate science, such as the concise description of climate extreme index and global precipitation percentile maps.

[4] Shen, S.S.P, and R.C.J. Somerville, 2019: Climate Mathematics: Theory and Applications. Cambridge University Press, 391pp.

This book attempts to modernize the mathematical education for undergraduate students majoring in atmospheric and oceanic sciences, or related fields. The book integrates six traditional mathematics courses (i.e., Calculus I, II, and III, Linear Algebra, Statistics, and Computer Programming) into a single course named Climate Mathematics. The course uses real climate data and can be taught in three semesters for freshmen and sophomores, or in one semester as an upper level or graduate class. Computer codes of R and Python and other learning resources are available at the book website `www.climatemathematics.org` .

[5] Smith, T.M., R.W. Reynolds, T.C. Peterson, and J. Lawrimore, 2008: Improvements to NOAA's historical merged landcean surface temperatures analysis (1880006). Journal of Climate, 21, 2283296, doi:10.1175/2007JCLI2100.1.

These authors are among the main contributors who reconstructed the sea surface temperature (SST). They began their endeavor in the early 1990s.

[6] Vose, R.S., D. Arndt, V.F. Banzon, D.R. Easterling, B. Gleason, B. Huang, E. Kearns, J.H. Lawrimore, M.J. Menne, T.C. Peterson, R.W. Reynolds, T.M. Smith, C.N. Williams, Jr., and D.L. Wuertz, 2012: NOAA's merged land-ocean surface temperature analysis. Bulletin of the American Meteorological Society, 93, 1677685.

These authors are experts on the reconstruction of both SST and the land surface air temperature, and have published many papers in data quality control and uncertainty quantifications.

Exercises

B.1 The NOAA Merged Land Ocean Global Surface Temperature Analysis (NOAAGlobalTemp) V5 includes the global land-spatial average annual mean temperature anomalies as shown below

```
1880   -0.843351   0.031336   0.009789   0.000850   0.020698
1881   -0.778600   0.031363   0.009789   0.000877   0.020698
1882   -0.802413   0.031384   0.009789   0.000897   0.020698
......
```

The first column is for time in years, and the second is the temperature anomalies in [Kelvin]. Columns 3-6 are data errors. This data file for the land temperature can be downloaded from the NOAAGlobalTemp website
`www.ncei.noaa.gov/data/noaa-global-surface-temperature/v5/access/timeseries`
The data file named `aravg.ann.land.90S.90N.v5.0.0.202104.asc.txt` is also included in `data.zip` that can be downloaded from the book website
`www.climatestatistics.org`
(a) Plot the anomalies against time from 1880 to 2020 using the point-line graph like Fig. B.1.
(b) Plot the anomalies against time from 1880 to 2020 using the staircase chart like Fig. B.2.
(c) Plot the anomalies against time from 1880 to 2020 using the color bar chart like Fig. B.3.

B.2 NOAAGlobalTemp V5 also has the global **ocean**-spatial average annual mean temperature anomalies contained in a data file named
`aravg.ann.locean.90S.90N.v5.0.0.202104.asc.txt`
For this dataset, please plot the anomaly data in the same three styles (a), (b) and (c) as in the previous problem. You may download the data file from the NOAAGlobalTemp V5 website or from `data.zip` for this book.

B.3 NOAAGlobalTemp V5 also has the global land-spatial average monthly mean temperature anomalies from January 1880 to April 2021. The data file is named
`aravg.mon.land.90S.90N.v5.0.0.202104.asc.txt`
For this dataset, please plot the January anomaly data from January 1880 to January 2021 in the same three styles (a) - (c) as in the previous problem.

B.4 For the monthly global land data of NOAAGlobalTemp V5 in the above problem,
(a) For the January data from 1880 to 2020, compute the following statistical indices: Mean, standard deviation, variance, skewness, kurtosis, max, 75%-percentile, median, 25%-percentile, and min.
(b) Do the same for February, March, $\cdots$, December.
(c) Aggregate all the results in (a) and (b) into a 10×12 matrix with the 10 statistical indices in rows and the 12 months in columns. Add proper column names, such as `Jan Feb Mar ...`, and also row names.
(d) Use 100-300 words to describe the differences of the statistical indices for different months.

B.5 For the monthly data CRUTEM4 (Climate Research Unit surface air temperature anomalies) in 4DVD, download the historical time series data for January of a location of your interest, and compute the following statistical indices: Mean, standard deviation, variance, skewness, kurtosis, max, 75%-percentile, median, 25%-percentile, and min.

B.6 Do the same as the previous problem but for July. Comment on the differences between the statistical indices in January and July.

B.7 (a) Plot a histogram of the January global land temperature anomalies using

the data file in the above problem, i.e.,
`aravg.mon.land.90S.90N.v5.0.0.202104.asc.txt`
(b) Do the same for July.
(c) Use 100-200 words to describe the comparison of the two histograms.

B.8 (a) Plot a box plot for the January global land temperature anomalies
`aravg.mon.land.90S.90N.v5.0.0.202104.asc.txt`
(b) Do the same but for the July data.
(c) Put the two box plots next to each other in the same figure.
(d) Use 100-200 words to describe the comparison of the two box plots.

B.9 (a) Use the monthly data in the above problem to generate twelve (12) Q-Q plots relative to the standard normal distribution for every month from January to December.
(b) From the 12 Q-Q plots, discuss which month's temperature anomalies are the closest to the normal distribution.

B.10 (a) Using the monthly data in the above problem
`aravg.mon.land.90S.90N.v5.0.0.202104.asc.txt`
compute the linear trend of January temperature anomalies from 1880 to 2020. Output your result in the unit: [°C/century].
(b) Repeat (a) for each of the other eleven months.
(c) Generate a list for the twelve linear trends.
(d) Use 100-200 words to discuss the numerical values of the list.

B.11 Plot the December 1997 surface temperature anomalies using the NOAAGlobalTemp V5 data on a $5° \times 5°$ grid in the netCDF format:

`NOAAGlobalTemp_v5.0.0_gridded_s188001_e202104_c20210509T133251.nc`

The data can be downloaded from NOAAGlobalTemp V5 or extracted from `data.zip` for this book at the website `www.climatestatistics.org`

B.12 Use Panoply to make the same December 1997 surface temperature anomalies plot as the above problem, but with the *Robinson* map projection.

B.13 Use Panoply to make the same December 1997 surface temperature anomalies plot as the above problem, but with the *Mercator* map projection.

B.14 Use Panoply to make the same December 1997 surface temperature anomalies plot as the above problem, but with the *Orthographic* map projection.

B.15 Use Panoply to make the same December 1997 surface temperature anomalies plot as the above problem, but with the *Stereographic (Two-Hemisphere)* map projection.

B.16 Plot a Hovmöller diagram like Fig. B.10 for the gridded NOAAGlobalTemp monthly anomalies at longitude 150°W and a latitude interval [50°S, 50°N] from January 1989 to December 2018 (i.e., 240 months).

B.17 Plot a Hovmöller diagram like Fig. B.10 for the gridded NOAAGlobalTemp monthly anomalies at longitude 140°W and a latitude interval [40°S, 40°N] from January 1971 to December 2000 (i.e., 360 months).

B.18 Plot a Hovmöller diagram like Fig. B.10 for the gridded NOAAGlobalTemp monthly anomalies on the equator with longitude from 160°E to 90°W] from January 1971 to December 2000 (i.e., 360 months).

B.19 Plot a Hovmöller diagram like Fig. B.10 for the gridded NOAAGlobalTemp monthly anomalies at latitude 5°S with longitude from 160°E to 90°W] from January 1971 to December 2000 (i.e., 360 months).

B.20 Use R or Python to plot four December 2015 potential temperature maps at the depth layers 5, 25, 105 and 195 meters based on the GODAS data. You can use the netCDF data file `godas2015.nc` in `data.zip` for this book or download the netCDF GODAS data from a NOAA website, such as `https://www.esrl.noaa.gov/psd/data/gridded/data.godas.html`

B.21 Use Panoply to plot the same maps as the previous problem but with the Robinson projection.

B.22 Use Plotly in R or Python to plot four December 2015 potential temperature maps at the depth layers 5, 25, 105 and 195 meters based on the GODAS data. Try to place the four maps together to make a 3D visualization.

B.23 Plot four June 2015 potential temperature maps at the depth layers 5, 25, 105 and 195 meters using the same netCDF GODAS data file as the above problem.

B.24 Use Plotly in R or Python to plot four June 2015 potential temperature maps at the depth layers 5, 25, 105 and 195 meters based on the GODAS data. Try to place the four maps together to make a 3D visualization.

B.25 Plot three cross-sectional maps of the December 2015 potential temperature at 10°S, equator, and 10°N using the same netCDF GODAS data file as the above problem. Each map is on a 40×360 grid, with 40 depth levels and 1-deg longitude resolution for the equator.

B.26 Do the same as the above but for the June 2015 GODAS potential temperature data.

B.27 Use R or Python a cross-sectional map of the December 2015 potential temperature along a meridional line at longitude at 170°E using the same netCDF GODAS data file as the above problem.

B.28 Use R or Python to plot a cross-sectional map of the June 2015 potential temperature along a meridional line at longitude at 170°E using the same netCDF GODAS data file as the above problem.

7015

Index