Use R!

Nina Golyandina · Anton Korobeynikov
Anatoly Zhigljavsky

Singular Spectrum Analysis with R

Springer

Use R!

Series editors
Robert Gentleman Kurt Hornik Giovanni Parmigiani

More information about this series at http://www.springer.com/series/6991

Nina Golyandina • Anton Korobeynikov •
Anatoly Zhigljavsky

Singular Spectrum Analysis with R

Springer

Nina Golyandina
Faculty of Mathematics and Mechanics
Saint Petersburg State University
Saint Petersburg, Russia

Anton Korobeynikov
Faculty of Mathematics and Mechanics
Saint Petersburg State University
Saint Petersburg, Russia

Anatoly Zhigljavsky
School of Mathematics
Cardiff University
Cardiff, United Kingdom

ISSN 2197-5736 ISSN 2197-5744 (electronic)
Use R!
ISBN 978-3-662-57378-5 ISBN 978-3-662-57380-8 (eBook)
https://doi.org/10.1007/978-3-662-57380-8

Library of Congress Control Number: 2018940189

Mathematics Subject Classification (2010): 37M10, 68U10

Printed on acid-free paper

This Springer imprint is published by the registered company Springer-Verlag GmbH, DE part of Springer Nature.
The registered company address is: Heidelberger Platz 3, 14197 Berlin, Germany

Preface

Singular spectrum analysis (SSA) is a well-known methodology for analysis and forecasting of time series. Since quite recently, SSA was also used to analyze digital images and other objects that are not necessarily of planar or rectangular form and may contain gaps. SSA is multipurpose and naturally combines both model-free and parametric techniques; this makes it a very special and attractive methodology for solving a wide range of problems arising in diverse areas, most notably those associated with time series and digital images. An effective, comfortable, and accessible implementation of SSA is provided by the R-package Rssa, which is available from CRAN and reviewed in this book.

For time series, SSA can be used for many different purposes, from non-parametric time series decomposition and filtration to parameter estimation, forecasting, gap filling, and change-point detection; many of these techniques can be extended to digital images. An essential difference between one-dimensional SSA and the majority of methods that analyze time series with trend and/or periodicities lies in the fact that SSA does not require any model for trend and no prior knowledge is needed about the number of periodical components and their frequencies. Also, periodicities can be modulated in different ways, and therefore, the type of model, additive or multiplicative, is not necessary to be assumed and taken into consideration for applying SSA. In the process of analysis, SSA constructs a number of the so-called subspace-based models; an example of a time series which satisfies such a model is provided by a series that follows a linear recurrence relation in the presence of noise. The validity of any of these models, however, is not a prerequisite for SSA, which makes SSA a very flexible tool that could be applied to virtually any data in the form of a time series or a digital image.

There are three books devoted to SSA[1] as well as many papers related to the methodological and theoretical aspects of SSA and comparison with ARIMA and other methods but especially to various applications. This present book expands the SSA methodology in many different directions and unifies various approaches and modifications within the SSA framework. Those mostly interested in applications may consider this book as an Rssa textbook.

Any method needs effective, comfortable, and accessible implementation. The R-package Rssa[2] provides such an implementation for SSA. Rssa is well documented and contains many standard and nonstandard tools for time series analysis and forecasting and image processing; it also has many visual tools which are useful for making proper choice of SSA parameters and examination of results. Rssa is the only SSA implementation available from CRAN and is almost certainly the most efficient implementation of SSA. Rssa is well theoretically and methodologically supported as its main routines are based on the books mentioned above. Most of the new SSA developments, which are systemized in this book, are also implemented in Rssa.

This book has the following goals: (a) to present the up-to-date SSA methodology, including multidimensional extensions, in the language accessible to a large circle of users; (b) to interconnect a variety of versions of SSA into a single tool, (c) to show the diverse tasks that SSA can be used for; (d) to formally describe the main SSA methods and algorithms; and (e) to provide tutorials on the Rssa package and the use of SSA. The companion website for the book is http://ssa-with-r-book.github.io.

The authors have vast experience with SSA. Nina Golyandina (NG) and Anatoly Zhigljavsky (AZ) have worked on SSA for more than 30 years and wrote about 20 papers each on the SSA-related topics. They authored the books Golyandina et al. (2001) and Golyandina and Zhigljavsky (2013) and in doing so they have spent huge efforts on trying to develop and tidy up the SSA methodology. NG is the head of the "Caterpillar-SSA" project http://gistatgroup.com/cat/ and a part of the team which has applied SSA to the analysis of gene expression; see Sect. 5.4.3. NG had supervised six PhD students (Eugene Osipov, Theodore Alexandrov, Konstantin Usevich, Alex Shlemov, Maxim Lomtev, and Nikita Zvonarev) whose projects were fully devoted to SSA. There is much material in this book which is based on recent papers of NG with coauthors; see Sect. 1.7.4 for details. AZ was behind the idea of using SSA for change-point detection and structural monitoring of time series, has participated in many projects on applications of SSA in economics, organized a number of conferences on SSA, and edited two recent SSA-oriented

[1] Elsner JB, Tsonis AA (1996) Singular spectrum analysis: a new tool in time series analysis. Plenum; Golyandina N, Zhigljavsky A (2013) Singular spectrum analysis for time series. Springer briefs in statistics. Springer; Golyandina N, Nekrutkin V, Zhigljavsky A (2001) Analysis of time series structure: SSA and related techniques. Chapman & Hall/CRC.

[2] Korobeynikov A, Shlemov A, Usevich K, Golyandina N (2017) Rssa: a collection of methods for singular spectrum analysis. R package version 1.0. http://CRAN.R-project.org/package=Rssa.

special issues of the *Statistics and Its Interface* journal. Both NG and AZ have taught SSA to different audiences which enormously helped them with the books. Anton Korobeynikov (AK) specializes in data analysis, computational statistics, and programming. He has a vast experience in programming in C, C++, and R and contributed to many open-source projects. He stood behind the initial ideas toward fast SSA implementation.[3] AK was the original author of RSSA; currently, he is its maintainer and one of the two major contributors to RSSA (the second major contributor is Alex Shlemov).

We believe that the book will be very useful to a very wide circle of readers including professional statisticians, specialists in signal and image processing, and specialists in numerous applied disciplines interested in using statistical methods of time series analysis, forecasting, and signal and image processing in their applications. The book is written on a level accessible to a very broad audience and contains a large number of examples; hence it can also be considered as a textbook for undergraduate and postgraduate courses on time series analysis and signal processing.

Acknowledgements The authors are very grateful to Vladimir Nekrutkin, their coauthor and one of the founders of the "Caterpillar-SSA" project, and to Alex Shlemov and Konstantin Usevich for their valuable contributions to RSSA and for their enormous contributions to the development of multidimensional extensions of SSA. Research of the third author was supported by the Russian Science Foundation, project No. 15-11-30022 "Global optimization, supercomputing computations, and application."

Saint Petersburg, Russia — Nina Golyandina
Saint Petersburg, Russia — Anton Korobeynikov
Cardiff, UK — Anatoly Zhigljavsky

[3] Korobeynikov A (2010) Computation- and space-efficient implementation of SSA. Stat Interface 3(3):357–368.

Contents

Common Symbols and Acronyms

SVD	Singular value decomposition
LRR	Linear recurrence relation
SSA	Singular spectrum analysis
$\mathbb{X}$ or $\mathbb{X}_N$	Time series or ordered collection of numbers
N	Length or size of $\mathbb{X}$
L	Window length or size
K	The number of L-lagged vectors obtained from $\mathbb{X}$; in the 1D case, $K = N - L + 1$
$\mathcal{T}$	Embedding operator
$\mathbf{X} = \mathcal{T}(\mathbb{X})$	The trajectory matrix of size $L \times K$ associated with $\mathbb{X}$
$\|\mathbf{X}\|_{\mathrm{F}}$	Frobenius matrix norm
$\operatorname{rank}\mathbf{X}$	Rank of $\mathbf{X}$
$\mathcal{X} = \mathcal{X}^{(L)}(\mathbb{X})$	L-trajectory space of $\mathbb{X}$
$\operatorname{rank}_L(\mathbb{X})$	L-rank of $\mathbb{X}$
$\Pi_{\mathcal{H}}$	Hankelization operator
λ_i	ith eigenvalue of the matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$
U_i	ith eigenvector of the matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$
$V_i = \mathbf{X}^{\mathrm{T}} U_i / \sqrt{\lambda_i}$	ith factor vector of the matrix $\mathbf{X}$
$(\sqrt{\lambda_i}, U_i, V_i)$	ith eigentriple of the SVD of the matrix $\mathbf{X}$
$\mathbf{I}_M$	Identity $M \times M$ matrix
R^L	Euclidean space of dimension L
$\mathsf{R}^{L \times K}$	Space of $L \times K$ matrices
$\mathcal{M}^{(\mathrm{H})}_{L,K}$	Set of trajectory matrices
$\operatorname{span}(P_1, \ldots, P_n)$	Linear subspace spanned by vectors $P_1, \ldots, P_n$
$\mathbf{A}^{\dagger}$	Pseudo-inverse of $\mathbf{A}$

Chapter 1
Introduction: Overview

The present book expands SSA methodology in many different directions and unifies different approaches and modifications within the SSA framework. This chapter is introductory; it outlines the main principles and ideas of SSA, presents a unified view on SSA, reviews its computer implementation in the form of the Rssa package, and gives references to all data sources used. The chapter contains eight sections serving different objectives.

Section 1.1 describes the generic structure of all methods from the SSA family and introduces the main concepts essential for understanding different versions of SSA and for making application of SSA in practice efficient.

Section 1.2 classifies different versions of SSA. As explained in that section, there are two complementary directions in which versions of SSA can be created: one is related to geometrical features of the object $\mathbb{X}$ and the other one is determined by the choice of the procedure of decomposition of the trajectory matrix into rank-one matrices. These two directions of variations of SSA are not related to each other so that any extension of SSA related to the geometry of $\mathbb{X}$ can be combined with any procedure of decomposition of the trajectory matrix.

Section 1.3 discusses the concept of separability, which is the most theoretically important concept of SSA. Achieving separability (for example, of a signal from noise) is the key task of SSA in most applications. Correct understanding of this concept is therefore imperative for making a particular application of SSA reliable and efficient. We will be returning to separability in many discussions within the book.

Section 1.4 briefly introduces the main underlying model used to apply SSA for common problems such as forecasting, imputation of missing data, and monitoring structural stability of time series. In the one-dimensional case, this model assumes that a part of the series can be described by a linear recurrent relation and in particular, by a sum of damped sinusoids. Estimation of parameters of this model often constitutes the main objective in signal processing.

N. Golyandina et al., *Singular Spectrum Analysis with R*, Use R!,
https://doi.org/10.1007/978-3-662-57380-8_1

In Sect. 1.5, we give information about most known implementations of SSA, describe the general structure of the Rssa package, and discuss efficiency of its implementation.

In Sect. 1.6 we briefly discuss the place of SSA among other methods of time series analysis, signal and image processing and provide a short overview of recent publications where a comparison of SSA with several traditional methods has been made.

In Sect. 1.7 we make a short historical survey of SSA, refer to recent applications of SSA and to papers which discuss combination of SSA with other methods; we also list the main papers, which a significant part of this book is based upon.

In Sect. 1.8 we make comments concerning installation of the Rssa package and describe the real-life data sets (taken from many different sources) which we have used in the book for illustrations. We provide the basic information about these data sets and specify their location. The corresponding references would help the reader to get more information about any of these data sets.

1.1 Generic Scheme of the SSA Family and the Main Concepts

We use SSA as a generic term for a family of methods based on a sequential application of the four steps schematically represented in Fig. 1.1 below and briefly described in the next section.

1.1.1 SSA Methods

We define an SSA method (or simply SSA) as any method performing the four steps depicted in Fig. 1.1 and briefly described below. The input object $\mathbb{X}$ is an ordered collection of N numbers (e.g., a time series or a digital image). We denote the set of such objects by $\mathcal{M}$. Unless stated otherwise, the entries of $\mathbb{X}$ are assumed to be real numbers although a straightforward generalization of the main SSA method to the case of complex numbers is available, see Sect. 4.1.

Input: $\mathbb{X}$, an ordered collection of N numbers.
Output: A decomposition of $\mathbb{X}$ into a sum of identifiable components: $\mathbb{X} = \widetilde{\mathbb{X}}_1 + \ldots + \widetilde{\mathbb{X}}_m$.

Step 1: Embedding The so-called *trajectory matrix* $\mathbf{X} = \mathcal{T}(\mathbb{X})$ is constructed, where $\mathcal{T}$ is a linear map transforming the object $\mathbb{X}$ into an $L \times K$ matrix of certain structure. Let us denote the set of all possible trajectory matrices by $\mathcal{M}^{(\mathrm{H})}_{L,K}$. The letter H is used to stress that these matrices have Hankel-related structure.

As an example, in 1D-SSA (that is, SSA for the analysis of one-dimensional real-valued time series), $\mathbb{X} = (x_1, \ldots, x_N)$ and $\mathcal{T} = \mathcal{T}_{\text{1D-SSA}}$ maps R^N to the space

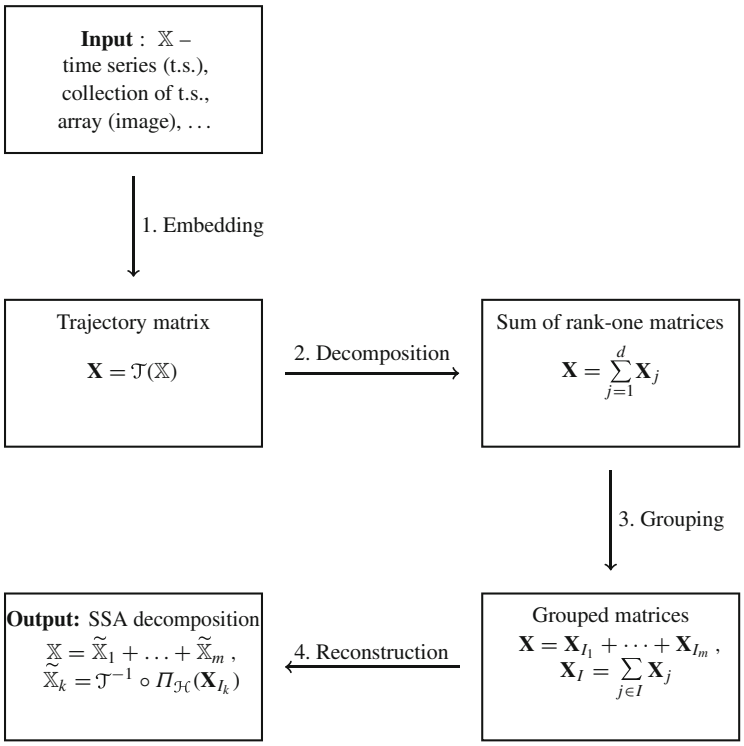

Fig. 1.1 SSA family: Generic scheme

of Hankel matrices $L \times K$ with equal values on the anti-diagonals, where N is the series length, L is the window length, which is a parameter, and $K = N - L + 1$:

$$\mathcal{T}_{\text{1D-SSA}}(\mathbb{X}) = \begin{pmatrix} x_1 & x_2 & x_3 & \ldots & x_K \\ x_2 & x_3 & x_4 & \ldots & x_{K+1} \\ x_3 & x_4 & x_5 & \ldots & x_{K+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_L & x_{L+1} & x_{L+2} & \ldots & x_N \end{pmatrix}. \tag{1.1}$$

Step 2: Decomposition of X into a Sum of Matrices of Rank 1 The result of this step is the decomposition

$$\mathbf{X} = \sum_i \mathbf{X}_i, \quad \mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}}, \tag{1.2}$$

where $U_i \in \mathsf{R}^L$ and $V_i \in \mathsf{R}^K$ are vectors such that $\|U_i\| = 1$ and $\|V_i\| = 1$ for all i and σ_i are non-negative numbers.

The main example of this decomposition is the conventional singular value decomposition (SVD) for real-valued matrices $\mathbf{X}$. If this conventional SVD is used, then we call the corresponding SSA method "Basic SSA" (Golyandina et al. 2001; Chapter 1). Let $\mathbf{S} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$, $\lambda_1 \ge \ldots \ge \lambda_L \ge 0$ be *eigenvalues* of the matrix $\mathbf{S}$, $d = \operatorname{rank}\mathbf{X} = \max\{j : \lambda_j > 0\}$, $U_1, \ldots, U_d$ be the corresponding *eigenvectors*, and $V_j = \mathbf{X}^{\mathrm{T}}U_j/\sqrt{\lambda_j}$, $j = 1, \ldots, d$, be *factor vectors*. Denote $\mathbf{X}_j = \sqrt{\lambda_j}U_jV_j^{\mathrm{T}}$. Then the SVD of the trajectory matrix $\mathbf{X}$ can be written as

$$\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d. \tag{1.3}$$

The triple $(\sqrt{\lambda_j}, U_j, V_j)$ consisting of the singular value $\sigma_j = \sqrt{\lambda_j}$, the left singular vector U_j and the right singular vector V_j of $\mathbf{X}$ is called *jth eigentriple*.

Step 3: Grouping The input in this step is the expansion (1.2) and a specification of how to group the components of (1.2).

Let $I = \{i_1, \ldots, i_p\} \subset \{1, \ldots, d\}$ be a set of indices. Then the resultant matrix $\mathbf{X}_I$ corresponding to the group I is defined as $\mathbf{X}_I = \mathbf{X}_{i_1} + \ldots + \mathbf{X}_{i_p}$.

Assume that a partition of the set of indices $\{1, \ldots, d\}$ into m disjoint subsets $I_1, \ldots, I_m$ is specified. Then the result of Grouping step is the *grouped matrix decomposition*

$$\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_m}. \tag{1.4}$$

If only one subset, I, of $\{1, \ldots, d\}$ is specified, then we still can assume that a partition of $\{1, \ldots, d\}$ is provided: this is the partition consisting of two subsets, I and $\bar{I} = \{1, \ldots, d\} \setminus I$. In this case, $\mathbf{X}_I$ is usually associated with the pattern of interest (for example, signal) and $\mathbf{X}_{\bar{I}} = \mathbf{X} - \mathbf{X}_I$ can be treated simply as a residual.

The grouping of the expansion (1.2), where $I_k = \{k\}$, is called *elementary*.

Step 4: Reconstruction At this step, each matrix $\mathbf{X}_{I_k}$ from the decomposition (1.4) is transferred back to the form of the input object $\mathbb{X}$. This transformation is performed optimally in the following sense: for a matrix $\mathbf{Y} \in \mathsf{R}^{L\times K}$, we seek for an object $\widetilde{\mathbb{Y}} \in \mathcal{M}$ that provides the minimum to $\|\mathbf{Y} - \mathcal{T}(\widetilde{\mathbb{Y}})\|_{\mathrm{F}}$, where $\|\mathbf{Z}\|_{\mathrm{F}} = \left(\sum_{ij} |z_{ij}|^2\right)^{1/2}$ is the Frobenius norm of $\mathbf{Z} = [z_{ij}] \in \mathsf{R}^{L\times K}$.

Let $\Pi_{\mathcal{H}} : \mathsf{R}^{L\times K} \to \mathcal{M}^{(\mathrm{H})}_{L,K}$ be the orthogonal projection from $\mathsf{R}^{L\times K}$ to $\mathcal{M}^{(\mathrm{H})}_{L,K}$ in the Frobenius norm. Then $\widetilde{\mathbb{Y}} = \mathcal{T}^{-1} \circ \Pi_{\mathcal{H}}(\mathbf{Y})$. The projection $\Pi_{\mathcal{H}}$ is simply the averaging of the entries corresponding to a given element of an object, see Sect. 1.1.2.6 for details. For example, in 1D-SSA the composite mapping $\mathcal{T}^{-1} \circ \Pi_{\mathcal{H}}$ uses the averaging along anti-diagonals so that $\widetilde{y}_k = \sum_{(i,j)\in\mathcal{A}_k}(\mathbf{Y})_{ij}/|\mathcal{A}_k|$, where $\mathcal{A}_k = \{(i, j) : i + j = k + 1, 1 \le i \le L, 1 \le j \le K\}$.

Let $\widehat{\mathbf{X}}_k = \mathbf{X}_{I_k}$ be the reconstructed matrices, $\widetilde{\mathbf{X}}_k = \Pi_{\mathcal{H}}\widehat{\mathbf{X}}_k$ be the corresponding trajectory matrices, and $\widetilde{\mathbb{X}}_k = \mathcal{T}^{-1}(\widetilde{\mathbf{X}}_k)$ be the reconstructed objects. Then the resulting decomposition of the initial object $\mathbb{X}$ is

$$\mathbb{X} = \widetilde{\mathbb{X}}_1 + \ldots + \widetilde{\mathbb{X}}_m. \tag{1.5}$$

If the grouping is elementary, then the reconstructed objects $\widetilde{\mathbb{X}}_k$ in (1.5) are called *elementary components*.

For convenience of referencing, Steps 1 and 2 of the generic SSA scheme are sometimes combined into the so-called "Decomposition stage" and Steps 3 and 4 are combined into "Reconstruction stage."

1.1.2 The Main Concepts

1.1.2.1 Parameters of the SSA Methods

Step 1: parameters of the linear map $\mathcal{T}$. For a given object $\mathbb{X}$, the trajectory matrix $\mathbf{X} = \mathcal{T}(\mathbb{X})$ can be computed in different ways. In 1D-SSA, there is only one parameter in Step 1, the window length L.

Step 2: no parameters if the conventional SVD is performed. Otherwise, if an alternative decomposition of $\mathbf{X}$ into a sum of rank-one matrices is used, there may be some parameters involved, see Sect. 1.2.1.

Step 3: the parameter (or parameters) that defines the grouping.

Step 4: no extra parameters.

1.1.2.2 Separability

A very important concept in the SSA methodology is separability. Let $\mathbb{X} = \mathbb{X}_1 + \mathbb{X}_2$. (Approximate) separability of $\mathbb{X}_1$ and $\mathbb{X}_2$ means that there exists a grouping such that the reconstructed object $\widetilde{\mathbb{X}}_1$ is (approximately) equal to $\mathbb{X}_1$. The representation $\mathbb{X} = \mathbb{X}_1 + \mathbb{X}_2$ can be associated with many different models such as "signal plus noise," "trend plus the rest," and "texture plus the main image."

If $\mathbb{X} = \mathbb{X}_1 + \mathbb{X}_2$ and $\mathbb{X}_1$ and $\mathbb{X}_2$ are approximately separable, then SSA can potentially separate $\mathbb{X}_1$ from $\mathbb{X}_2$; that is, it can find a decomposition $\mathbb{X} = \widetilde{\mathbb{X}}_1 + \widetilde{\mathbb{X}}_2$ so that $\widetilde{\mathbb{X}}_1 \approx \mathbb{X}_1$ and $\widetilde{\mathbb{X}}_2 \approx \mathbb{X}_2$.

Consider, as an example, Basic SSA. Properties of the SVD yield that the (approximate) orthogonality of columns and of rows of the trajectory matrices $\mathbf{X}_1$ and $\mathbf{X}_2$ of $\mathbb{X}_1$ and $\mathbb{X}_2$ can be considered as natural separability conditions.

There is a well-elaborated theory of separability for the one-dimensional time series (Golyandina et al. 2001; Sections 1.5 and 6.1). Many important decomposition problems, from noise reduction and smoothing to trend, periodicity and signal extraction, can be solved by SSA. The success of 1D-SSA in making separation between separable objects is related to the simplicity of the Hankel structure of the trajectory matrices and the optimality features of the SVD.

We will come back to the important concept of separability in Sect. 1.3 where we define the main characteristic which is used in SSA for separability checking.

1.1.2.3 Information for Grouping

The theory of SSA exhibits the ways of helping to detect the components (σ_i, U_i, V_i) in the decomposition (1.2) related to the object component with certain properties to perform proper grouping under the condition of separability. One of the rules is that U_i and V_i (eigenvectors and factor vectors in the case of Basic SSA) produced by an object component emulate the properties of this component. For example, in Basic 1D-SSA the eigenvectors produced by slowly-varying series components are slowly-varying, the eigenvectors produced by a sine wave are sine waves with the same frequency, and so on. These properties help to perform the grouping by visual inspection of eigenvectors and also by some automatic procedures, see Sect. 2.7.

1.1.2.4 Trajectory Spaces and Signal Subspaces

Let $\mathbf{X}$ be the trajectory matrix corresponding to some object $\mathbb{X}$. The *column (row) trajectory space* of $\mathbf{X}$ is the linear subspace spanned by the columns (correspondingly, rows) of $\mathbf{X}$. The term "trajectory space" usually means "column trajectory space." The column trajectory space is a subspace of R^L, while the row trajectory space is a subspace of R^K. In general, for real-world data the trajectory spaces coincide with the corresponding Euclidean spaces, since they are produced by a noisy data. However, in the "signal plus noise" model, when the signal has rank-deficient trajectory matrix, the signal trajectory space can be called "signal subspace." Both column and row signal subspaces can be considered; note that the dimensions of the row and column subspaces coincide.

1.1.2.5 Objects of Finite Rank

The class of objects that suit SSA are the so-called objects of finite rank. We say that the object (either time series or image) has L-rank r if the rank of its trajectory matrix is $r < \min(L, K)$; that is, the trajectory matrix is rank-deficient. If the L-rank r does not depend on the choice of L for any sufficiently large object and trajectory matrix sizes, then we say that the object is of finite rank and has rank r, see Sect. 2.1.2 for rigorous definitions.

Since the trajectory matrices considered in SSA methods are either pure Hankel or consist of Hankel blocks, then the rank-deficient trajectory matrices are closely related to the objects satisfying some linear relations. These linear relations can be used for building forecasting methods. In the one-dimensional case, under some non-restrictive conditions, rank-deficient Hankel matrices are in the one-to-one correspondence with the linear recurrence relations (LRRs) of the form

$$x_n = a_1 x_{n-1} + \ldots + a_r x_{n-r}$$

and therefore are related to the time series which can be expressed as sums of products of exponentials, polynomials, and sinusoids, see Sect. 2.1.2.2.

Each specific SSA extension produces a class of specific objects of finite rank. The knowledge of ranks of objects of finite rank can help to recognize the rank-one components for the component reconstruction. For example, in order to reconstruct the exponential trend in the one-dimensional case, we need to group only one rank-one component (the exponential function has rank 1), while to reconstruct a sine wave we generally need to group two SVD components (the rank of a sine wave equals 2).

The real-life time series or images are generally not of finite rank. However, if a given object $\mathbb{X}$ is a sum of a signal of finite rank and noise, then, in view of approximate separability, SSA may be able to approximately extract the signal and subsequently use the methods that are designed for series of finite rank.

1.1.2.6 Reconstruction (Averaging)

Let us formally describe the operation of reconstruction of a matrix used in Step 4 of the generic scheme described in Sect. 1.1.1. By analogy with the one-dimensional case this operation can also be called "averaging over diagonals" even if the averaging will be performed over more complicated patterns.

Assume that the entries x_τ of the object $\mathbb{X} = \{x_\tau\}$ are indexed by the index τ which can be simply a positive integer (for the one-dimensional series) or multi-index (for digital images).

A linear map $\mathcal{T}$ is making a one-to-one transformation of $\mathcal{M}$ to $\mathcal{M}^{(\mathrm{H})}_{L,K}$, the set of $L \times K$ matrices of a specified structure. It puts elements of $\mathbb{X}$ on certain places of the matrix $\mathcal{T}(\mathbb{X}) = \mathbf{X} = [(\mathbf{X})_{ij}]$.

Let $\mathbf{e}_\tau \in \mathcal{M}$ be the object with 1 as the τth entry with all other entries zero. Define the set of indices

$$\mathcal{A}_\tau = \{(i, j) \text{ such that } (\mathbf{E}_\tau)_{ij} = 1\},$$

where $\mathbf{E}_\tau$ is the matrix

$$\mathbf{E}_\tau = [(\mathbf{E}_\tau)_{ij}] = \mathcal{T}(\mathbf{e}_\tau) \in \mathcal{M}^{(\mathrm{H})}_{L,K}.$$

If τ is the place of an element $x_\tau \in \mathbb{X}$, then $(\mathbf{X})_{ij} = x_\tau$ for all $(i, j) \in \mathcal{A}_\tau$.

Assume now that $\widehat{\mathbf{X}} \in \mathsf{R}^{L\times K}$ is an arbitrary $L \times K$ matrix and we need to compute

$$\widetilde{\mathbb{X}} = (\widetilde{x}_\tau) = \mathcal{T}^{-1} \circ \Pi_{\mathcal{H}}(\widehat{\mathbf{X}})$$

by first making the orthogonal projection of $\widehat{\mathbf{X}}$ to the set $\mathcal{M}^{(\mathrm{H})}_{L,K}$ and then writing the result in the object space $\mathcal{M}$. This operation is the extension of the "diagonal

averaging" procedure applied in 1D-SSA: the elements $\widetilde{x}_\tau$ of $\widetilde{\mathbb{X}}$ are computed by the formula

$$\widetilde{x}_\tau = \sum_{(i,j)\in\mathcal{A}_\tau} (\widehat{\mathbf{X}})_{ij}/w_\tau = \langle \widehat{\mathbf{X}}, \mathbf{E}_\tau\rangle_{\mathrm{F}}/\|\mathbf{E}_\tau\|_{\mathrm{F}}^2 ,$$

where $\mathbf{E}_\tau = \mathcal{T}(\mathbf{e}_\tau)$, $w_\tau = |\mathcal{A}_\tau| = \|\mathbf{E}_\tau\|_{\mathrm{F}}^2$ is the number of elements in the set $\mathcal{A}_\tau$ and the Frobenius inner product $\langle\cdot,\cdot\rangle_{\mathrm{F}}$ is defined by

$$\langle \mathbf{Z}, \mathbf{Y}\rangle_{\mathrm{F}} = \sum_{i,j} z_{ij}\, y_{ij} . \tag{1.6}$$

1.2 Different Versions of SSA

Let us consider how the four steps of the generic scheme of SSA formulated in Sect. 1.1 can vary for different versions of SSA.

Step 1: the form of the object $\mathbb{X}$ and hence the specificity of the embedding operator $\mathcal{T}$ makes a big influence on how a particular version of SSA looks like.
Step 2: not only the conventional SVD but many other decompositions of $\mathbf{X}$ into rank-one matrices could be used.
Step 3: formally, this step is exactly the same for all versions of SSA although the tools used to perform the grouping may differ.
Step 4: the embedding operator $\mathcal{T}$ defined in Step 1 fully determines the operations performed at this step.

Therefore, we have two directions for creating different versions of SSA: the first direction is related to geometrical features of the object $\mathbb{X}$ and a form of the embedding operator $\mathcal{T}$, while the second direction is determined by the form of the decomposition at Step 2. Essentially, Step 1 is determined by the form of the object; therefore, its variations can be considered as extensions of 1D-SSA. If instead of SVD we use some other decomposition of $\mathbf{X}$ into rank-one matrices at Step 2, then we call the corresponding algorithm a modification of Basic SSA. These two directions of variations of SSA are not related to each other so that any extension of Step 1 can be combined with any modification of Step 2.

We start the discussion with outlining some modifications that can be offered for the use at Step 2.

1.2.1 Decomposition of $\mathbf{X}$ *into a Sum of Rank-One Matrices*

1.2.1.1 Variations of SSA Related to Methods of Decomposition

The conventional SVD formulated in the description of SSA in Sect. 1.1.1 is a decomposition of $\mathbf{X}$ into a sum of rank-one matrices, which has some optimality

properties, see Golyandina et al. (2001; Chapter 4). Therefore, Basic SSA, which is SSA with the conventional SVD used at Step 2, can be considered as the most fundamental version of SSA among all SSA methods.

Let us enumerate several variations of SSA, which could be useful for answering different questions within the framework of SSA.

A well-known modification of Basic SSA is *Toeplitz SSA* (Sect. 2.2), which was created for dealing with stationary time series. This modification is devised for the analysis of a natural estimate of the auto-covariance matrix of the original time series and assumes that this time series is stationary. However, if the time series $\mathbb{X}$ is non-stationary, then the reconstruction obtained by Toeplitz SSA can have a considerable bias.

An important variation of SSA is *SSA with projection* (Sect. 2.3). If we have a parametric model (which should be linear in parameters and agreeable with the finite-rank assumption) for one of the components of the series, such as trend of a one-dimensional series, then a projection on a suitable subspace is performed and is followed by a decomposition of the residual, e.g., by the SVD. The known methods of SSA with centering and SSA with double centering for extraction of constant and linear trends, respectively, are special cases of SSA with projection. More generally, an arbitrary polynomial trend can be extracted by a suitable version of SSA with projection. Another use of SSA with projection is to build a subspace from a supporting series and project the main series onto this subspace.

In some versions of SSA the intention is to improve separability properties of SVD. If we use an oblique version of the SVD, then the resulting SSA method becomes Oblique SSA. The following two versions of Oblique SSA seem to be useful in practice, namely *Iterative Oblique SSA* (Sect. 2.4) and *Filter-adjusted Oblique SSA* (Sect. 2.5). The latter is useful for separation of components with equal contribution.

1.2.1.2 Nested Application of Different Versions of SSA

Since Oblique SSA does not have good approximating features, it cannot replace Basic SSA which uses the conventional SVD. Therefore, Oblique SSA should be used in a nested manner so that Basic SSA is used first to extract several components without performing careful split of these components and then one of the proposed oblique methods is used for separating the mixed components.

If we use Basic SSA for denoising and then some other version of SSA (like Independent Component Analysis or Oblique SSA) for improvement of separability, then we can interpret this as if using Basic SSA for preprocessing and using another method for a more refined analysis. There is, however, a significant difference between this and some methods mentioned in Sect. 1.7.3, where Basic SSA is used for preprocessing and then ARIMA or other methods of different nature are employed. Indeed, when we use Basic SSA for denoising and some other SSA technique like Oblique SSA for improvement of separability, then we are using the signal subspace estimated by Basic SSA rather than the estimated signal itself (recall

that in the transition from the estimated signal subspace to the estimated signal we incur an additional error).

Let us schematically demonstrate the nested use of the methods as follows. Let $\mathbb{X} = \mathbb{X}^{(1)} + \mathbb{X}^{(2)} + \mathbb{X}^{(3)}$ be a decomposition of the time series and $\mathbf{X} = \mathbf{X}^{(1)} + \mathbf{X}^{(2)} + \mathbf{X}^{(3)}$ be the corresponding decomposition of the trajectory matrix of $\mathbb{X}$. Let Basic SSA return at Decomposition stage $\mathbf{X} = \widetilde{\mathbf{X}}^{(1,2)} + \widetilde{\mathbf{X}}^{(3)}$ and assume that a chosen nested method makes the decomposition $\widetilde{\mathbf{X}}^{(1,2)} = \widetilde{\mathbf{X}}^{(1)} + \widetilde{\mathbf{X}}^{(2)}$. Then the final result is $\mathbf{X} = \widetilde{\mathbf{X}}^{(1)} + \widetilde{\mathbf{X}}^{(2)} + \widetilde{\mathbf{X}}^{(3)}$ and, after the diagonal averaging, $\mathbb{X} = \widetilde{\mathbb{X}}^{(1)} + \widetilde{\mathbb{X}}^{(2)} + \widetilde{\mathbb{X}}^{(3)}$. There is no need for reconstruction of the signal by Basic SSA as only the estimated signal subspace is used for making a refined decomposition.

1.2.1.3 Features of Decompositions

The result of Decomposition step of SSA (Step 2) can be written in the form (1.2). The SVD is a particular case of (1.2) and corresponds to the orthonormal systems of $\{U_i\}$ and $\{V_i\}$. By analogy with the SVD, we will call (σ_i, U_i, V_i) eigentriples, σ_i singular values, U_i left singular vectors or eigenvectors, V_i right singular vectors or factor vectors. For most of SSA decompositions, each U_i belongs to the column space of $\mathbf{X}$ while each V_i belongs to the row space of $\mathbf{X}$. We shall call such decompositions *consistent*.

If the systems $\{U_i\}$ and $\{V_i\}$ are linearly independent, then the decomposition (1.2) is *minimal*; that is, it has smallest possible number of addends equal to $r = \operatorname{rank}\mathbf{X}$. If at least one of the systems $\{U_i\}$ or $\{V_i\}$ is not linearly independent, then the decomposition (1.2) is not minimal. If the decomposition (1.2) is not consistent, then it can be not minimal even if $\{U_i\}$ or $\{V_i\}$ are linearly independent, since their projections on the column (or row) space can be dependent.

If both vector systems $\{U_i\}$ and $\{V_i\}$ are orthogonal, then the decomposition (1.2) is called biorthogonal. If $\{U_i\}$ is orthogonal, then the decomposition is called left-orthogonal; if $\{V_i\}$ is orthogonal, then the decomposition is called right-orthogonal.

If $\mathbf{X}_i$ are F-orthogonal so that $\langle \mathbf{X}_i, \mathbf{X}_j \rangle_{\mathrm{F}} = 0$ for $i \neq j$, then we say that the corresponding decompositions are F-orthogonal. Either left or right orthogonality is sufficient for F-orthogonality. For F-orthogonal decompositions (1.2), $\|\mathbf{X}\|_{\mathrm{F}}^2 = \sum_i \|\mathbf{X}_i\|_{\mathrm{F}}^2$. In general, however, $\|\mathbf{X}\|_{\mathrm{F}}^2$ may differ from $\sum_i \|\mathbf{X}_i\|_{\mathrm{F}}^2$.

The contribution of kth matrix component $\mathbf{X}_k$ is defined as $\sigma_k^2 / \|\mathbf{X}\|_{\mathrm{F}}^2$, where $\sigma_k^2 = \|\mathbf{X}_k\|_{\mathrm{F}}^2$. For F-orthogonal decompositions, the sum of component contributions is equal to 1. Otherwise, this sum can considerably differ from 1 (e.g., the sum of component contributions can be 146%).

1.2.1.4 Decompositions in Different Versions of SSA

Let us gather several versions of 1D-SSA which are implemented in the Rssa package and based on different procedures used at Decomposition step, and indicate their features. Some of these variations are also implemented for multivariate and multidimensional versions of SSA.

Basic SSA: the conventional SVD, consistent, minimal, biorthogonal and therefore F-orthogonal decomposition. Implemented in `ssa` with `kind="1d-ssa"`.

Toeplitz SSA: generally, non-consistent, non-minimal F-orthogonal decomposition. Implemented in `ssa` with `kind="toeplitz-ssa"`.

SSA with projection: F-orthogonal but non-consistent decomposition if at least one basis vector used for the projection does not belong to the column (row) trajectory space. The components, which are obtained by projections, are located at the beginning of the decomposition and have indices $1, \dots, n_{\text{special}}$. Implemented in `ssa` with `kind="1d-ssa"` and non-`NULL` `row.projector` or `column.projector` arguments.

Oblique SSA with filter preprocessing (Filter-adjusted O-SSA): consistent, minimal F-orthogonal decomposition. The main particular case is DerivSSA. Implemented in `fossa`.

Iterative Oblique SSA (Iterative O-SSA): consistent, minimal oblique decomposition. Implemented in `iossa`.

Oblique versions of SSA are made to perform in a nested manner.

1.2.2 Versions of SSA Dealing with Different Forms of the Object

In this section, we briefly consider different versions of SSA which operate objects $\mathbb{X}$ of different forms. The main difference between different versions of SSA of this section is the form of the embedding operator $\mathcal{T}$.

As has been mentioned above, SSA can be applied to multivariate and multidimensional objects. SSA for a system of series is called Multivariate (or Multichannel) SSA, shortly MSSA (Sect. 4.2); SSA for digital gray-scale images is called 2D-SSA (Sect. 5.1).

Complex SSA (Sect. 4.1) is a special version of SSA for the analysis of two time series of equal length or a single one-dimensional complex-valued time series.

Shaped SSA (Sects. 2.6 and 5.2) can process data of complex structure and arbitrary shape; the dimension of the object $\mathbb{X}$ is irrelevant. Shaped SSA can be applied to many different kinds of data including time series, systems of time series, digital images of rectangular and non-rectangular shapes.

For both series and images, circular versions of SSA are available. For series, circular SSA works in the metric of a circle and therefore this version is suitable for series, which are indeed defined on a circle.

For images, circular versions of SSA provide a possibility to decompose images given on a cylinder (for example, obtained as a cylindrical projection of a sphere or an ellipsoid) or given on a torus. Circular versions allow to eliminate the edge effects, which are unavoidable in the case of, e.g., planar unfolding of a cylinder.

Table 1.1 contains a list of the extensions considered in this book.

Table 1.1 Classification of different versions of SSA based on different geometrical features of the object $\mathbb{X}$

Method	Data	Notation	Trajectory matrix	Section
1D-SSA	Time series (t.s.)	$\mathbb{X} = (x_1, \dots, x_N)$ of length N	Hankel	2.1
CSSA	Complex t.s.	$\mathbb{X}^{(1)} + \mathrm{i}\,\mathbb{X}^{(2)}$	Complex Hankel	4.1
MSSA	System of t.s.	$\mathbb{X}^{(p)}$ of length N_p, $p = 1, \dots, s$	Stacked Hankel	4.2
2D-SSA	Rectangular image	$\mathbb{X} = (x_{ij})_{i,j=1}^{N_x,N_y}$	Hankel-Block-Hankel	5.1
ShSSA	Shaped objects	$\mathbb{X} = (x_{(i,j)})_{(i,j)\in\mathfrak{N}}$	Quasi-Hankel	5.2
Circular SSA	Circular/cylindrical	Objects on circle or cylinder	Quasi-Hankel	5.2.1.3

1.3 Separability in SSA

In this section, we discuss the SSA separability in more detail; see also Sects. 2.1.3, 2.3.3, 2.4.2, 2.5.2 for special cases.

Let us assume that we observe a sum of two objects $\mathbb{X} = \mathbb{X}^{(1)} + \mathbb{X}^{(2)}$. We say that SSA separates these two objects if a grouping at Grouping step (Step 3) can be found such that $\widetilde{\mathbb{X}}^{(k)} = \mathbb{X}^{(k)}$ for $k = 1, 2$. If these equalities hold approximately, then this defines approximate separability. Asymptotic separability can be introduced if the series length $N \to \infty$. In this case, approximate separability takes place for large enough series lengths. The separability property is very important for SSA as it means that the method potentially works; that is, it is able to extract the object components.

In Basic SSA and its multidimensional extensions, (approximate) separability means (almost) orthogonality of the object components, since the biorthogonal SVD decomposition is used. In other versions of SSA, different conditions of separability can be formulated.

If the decomposition (1.2) at Decomposition step is not unique, then two variants of separability are introduced. Weak separability means that there exists a decomposition such that after grouping we obtain $\widetilde{\mathbb{X}}^{(k)} = \mathbb{X}^{(k)}$. Strong separability means that this equality is achievable for any admissible decomposition.

Conditions of exact separability are very restrictive whereas asymptotic separability takes place for a wide range of object components. For example, slowly varying smooth components are asymptotically separable from regular oscillations and they both are asymptotically separable from noise.

In order to verify separability of the reconstructed components $\widetilde{\mathbb{X}}_1$ and $\widetilde{\mathbb{X}}_2$, we should check orthogonality of their reconstructed trajectory matrices $\widetilde{\mathbf{X}}_1$ and $\widetilde{\mathbf{X}}_2$. A convenient measure of their orthogonality is the Frobenius inner product $\langle \widetilde{\mathbf{X}}_1, \widetilde{\mathbf{X}}_2 \rangle_{\mathrm{F}}$ defined in (1.6).

The normalized measure of orthogonality is

$$\rho(\widetilde{\mathbf{X}}_1, \widetilde{\mathbf{X}}_2) = \langle \widetilde{\mathbf{X}}_1, \widetilde{\mathbf{X}}_2 \rangle_{\mathrm{F}} / (\|\widetilde{\mathbf{X}}_1\|_{\mathrm{F}} \|\widetilde{\mathbf{X}}_2\|_{\mathrm{F}}) .$$

Since the trajectory matrix consists of $w_\tau = |\mathcal{A}_\tau| = \|\mathcal{T}(\mathbf{e}_\tau)\|_{\mathrm{F}}^2$ entries of the τth element x_τ of the initial ordered object, we can introduce the weighted inner product in the space $\mathcal{M}$: $(\mathbb{Y}, \mathbb{Z})_{\mathbf{w}} = \sum_\tau w_\tau y_\tau z_\tau$. Then the quantity

$$\rho_{\mathbf{w}}(\widetilde{\mathbb{X}}_1, \widetilde{\mathbb{X}}_2) = \rho(\widetilde{\mathbf{X}}_1, \widetilde{\mathbf{X}}_2) = \frac{(\widetilde{\mathbb{X}}_1, \widetilde{\mathbb{X}}_2)_{\mathbf{w}}}{\|\widetilde{\mathbb{X}}_1\|_{\mathbf{w}} \|\widetilde{\mathbb{X}}_2\|_{\mathbf{w}}} \tag{1.7}$$

will be called **w**-*correlation* by statistical analogy. Note however that in this definition the means are not subtracted.

Let $\widetilde{\mathbb{X}}_j$ be the elementary reconstructed components produced by the elementary grouping $I_j = \{j\}$. Then the matrix of $\rho_{ij}^{(\mathbf{w})} = \rho_{\mathbf{w}}(\widetilde{\mathbb{X}}_i, \widetilde{\mathbb{X}}_j)$ is called **w**-*correlation matrix*.

The norm $\|\cdot\|_{\mathbf{w}}$ is called the *weighted norm* and serves as a measure of contribution of the components in the decomposition (1.5): the contribution of $\widetilde{\mathbb{X}}_k$ is defined as $\|\widetilde{\mathbb{X}}_k\|_{\mathbf{w}}^2 \big/ \|\mathbb{X}\|_{\mathbf{w}}^2$.

If the weighted correlation between a pair of elementary components is large, then this suggests that these two components are highly correlated and should perhaps be included into the same group.

1.4 Forecasting, Interpolation, Low-Rank Approximation, and Parameter Estimation in SSA

There is a class of objects, which is special for SSA. This is the class of objects satisfying linear recurrence relations. Trajectory matrices of these objects are rank-deficient and, moreover, for these objects the number of non-zero terms in the SVD (1.3) does not depend on the window length if this length is sufficiently large; we will say in such cases that the objects are of finite rank. The class of objects satisfying linear recurrence relations provides a natural model of the signal for SSA, which is a fundamentally important concept for forecasting of time series.

Linear recurrence relation (LRR) for a time series $\mathbb{S}_N = (s_i)_{i=1}^N$ is a relation of the form

$$s_{i+t} = \sum_{k=1}^{t} a_k s_{i+t-k}, \ 1 \le i \le N-t, \ a_t \ne 0, \ t < N. \tag{1.8}$$

It is well known (see, e.g., Hall (1998; Theorem 3.1.1)) that a sequence $\mathbb{S}_\infty = (s_1, \ldots, s_n, \ldots)$ satisfies the LRR (1.8) for all $i \ge 0$ if and only if for some integer p we have for all n

$$s_n = \sum_{k=1}^{p} P_k(n) \mu_k^n, \tag{1.9}$$

where $P_k(n)$ are polynomials in n and μ_k are some complex numbers. For real-valued time series, (1.9) implies that the class of time series governed by the LRRs consists of sums of products of polynomials, exponentials, and sinusoids.

A simplified model of (1.9) is $s_n = \sum_{k=1}^{p} c_k \mu_k^n$. Estimation of complex numbers $\mu_k = \rho_k e^{\mathrm{i}2\pi\omega_k}$ is equivalent to estimating the frequencies ω_k and the rates $\ln \rho_k$.

For images $\mathbb{S} = [s_{mn}]$, LRRs are two-dimensional and the common term of the signal (which can be called pattern) under the model has the form

$$s_{mn} = \sum_{k=1}^{p} P_k(m,n)\mu_k^m \nu_k^n. \tag{1.10}$$

This important fact is well known starting from Kurakin et al. (1995; §2.20).

In many real-life problems, a noisy signal (or noisy pattern for images) is observed and the problem is to forecast the signal, impute gaps in the signal, estimate signal parameters, find change-points in the signal, and so on. Note that it is not compulsory to assume that the noise is random. In a very general sense, noise is a residual which does not require further investigation.

SSA may provide estimates of the signal space, which is the subspace spanned by the chosen basis during Grouping step in SSA. The estimation of the signal subspace can be performed by iterations (so-called Cadzow iterations), which consist of iterative SSA processing, see Sect. 3.4.

In the process of estimation of the signal subspace we also obtain a parametric estimate of the signal; that is, of the set of $\{\mu_k\}$ in the 1D case and the set of $\{\mu_k, \nu_k\}$ in the 2D case. One of the most common methods is called ESPRIT for series and 2D-ESPRIT for images (see Sects. 3.1.1.2 and 5.3).

For series, sequential on-line estimation of the signal structure produces natural algorithms of subspace tracking (e.g., for monitoring structural stability of an object changing in time) and change-point detection. Also, we can forecast the found structure, e.g., by application of the constructed LRR. By performing interpolation (which is forecasting inside the series) missing data can be filled in.

1.5 The Package

1.5.1 SSA Packages

There are many implementations of SSA. They differ by potential application areas, implemented methods, interactive or non-interactive form, free or commercial use,

computer system (Windows, Unix, Mac), level of reliability and support. The most known supported software packages implementing SSA are the following:

1. http://gistatgroup.com:
   general-purpose interactive "Caterpillar"-SSA software (Windows) following the methodology described in Golyandina et al. (2001), Golyandina and Zhigljavsky (2013);
2. http://www.atmos.ucla.edu/tcd/ssa:
   oriented mainly on climatic applications SSA-MTM Toolkit for spectral analysis (Ghil et al. 2002) (Unix) and its commercial extension kSpectra Toolkit (Mac), interactive;
3. The commercial statistical software, SAS, has an econometric extension called SAS/ETS®, which includes SSA in its rather basic form; this version of SSA is based on the methodology of Golyandina et al. (2001).
4. http://cran.r-project.org/web/packages/Rssa:
   R package Rssa (Korobeynikov et al. 2017), an implementation of the main SSA procedures for major platforms, extensively developed.

We consider the Rssa package as an efficient implementation of a large variety of the main SSA algorithms. This package also contains many visual tools which are useful for making proper choice of SSA parameters and examination of results.

Rssa is the only SSA package available from CRAN and we believe it is the fastest implementation of SSA. Another important feature of the package is its very close relation to the SSA methodology thoroughly described in Golyandina et al. (2001), Golyandina and Zhigljavsky (2013), Golyandina et al. (2015). As a result of this, the use of the package is well theoretically and methodologically supported.

1.5.2 Tools for Visual Control and Choice of Parameters

SSA needs tools to help choosing the method parameters and controlling the results. To a great extent, SSA is an exploratory technique and hence visual tools are vital and they are extensively used in Rssa. For example, in order to help to choose the groups in (1.4), Rssa allows plotting measures of separability of series components in the obtained decompositions.

The tools for accuracy control are divided into two groups. First, stability of results with respect to parameter changes can be checked. Second, the bootstrap procedure could be used when the model (not necessarily parametric) of either the series or other object is built on the base of signal reconstruction and then the accuracy of this model is assessed by simulation according to the estimated model. Shortly, Rssa allows the user to enjoy a large variety of graphical tools and bootstrap procedures.

1.5.3 *Short Introduction to* RSSA

The RSSA package implements all methods and tools mentioned above.

The main function is `ssa`, which initializes an `ssa` object and by default performs the decomposition by different methods. Together with `reconstruct`, they implement the SSA method. For nested versions, `iossa` and `fossa` serve for refined decompositions.

An `ssa` object `s` contains, among others, elements of the decomposition (1.2), which can be accessed as `s$sigma`, `s$U`, and `s$V`. Features of the decompositions differ for different versions of SSA (see Sect. 1.2.1.4). For Basic SSA, `s$sigma` are called singular values; squares of `s$sigma` are called eigenvalues; `s$U` are called eigenvectors. (We keep these names for other versions of SSA as well.) The relative contributions of components to the decomposition can be obtained as `contributions(s)`; see Sect. 1.2.1.3, where formulas for their calculation are given and explained.

A variety of functions `plot` help to visualize the results and additional information. Functionality of SSA-related methods is supplemented by the functions `forecast`, `parestimate`, and some others.

All essential versions of SSA are implemented in RSSA but not all further actions like forecasting and gap filling are consistent with all implemented versions of SSA. The user can check the `ssa` object, which is returned by the main function `ssa`, for compatibility by the function `ssa.capabilities`. This function returns a list of capabilities with information `TRUE` or `FALSE`, respectively.

A general scheme of investigation by means of RSSA is as follows:

1. perform decomposition by `ssa`;
2. visualize the result by `plot`;
3. if necessary, refine decomposition by `iossa` or `fossa`;
4. again, visualize the result by `plot`;
5. perform grouping based on the obtained visual and numerical information; in particular, choose the group of signal components;
6. then perform one of the following actions: reconstruction of series components by `reconstruct`, forecasting by `forecast`, parameter estimation by `parestimate`;
7. visualize the result by `plot`.

Note that RSSA contains more algorithms than this book formulates. However, the book has enough information to understand how to extend the algorithms, such as parameter estimation, filling-in missing data and O-SSA, to different dimensions and geometries. Many of these versions are implemented in RSSA but not described in the book explicitly: there are infinitely many dimensions and geometries and the algorithms are formulated in the book in such a manner that they can be easily generalized if needed.

1.5.4 Implementation Efficiency

The user does not need to know the specifics of the internal implementation of the Rssa functions. However, understanding of the general principles of implementation can help to use the package more effectively.

The fast implementation of SSA-related methods, which was suggested in Korobeynikov (2010), extended in Golyandina et al. (2015) and is used in the Rssa package (Korobeynikov et al. 2017), relies on the following techniques (see Shlemov and Golyandina (2014) for a more thorough discussion).

1. The truncated SVD calculated by the Lanczos methods (Golub and Van Loan 1996; Ch. 9) is used. In most SSA applications, only a number of leading SVD components correspond to the signal and therefore are used at Grouping step of the SSA algorithm. Thus, a truncated SVD rather than the full SVD is usually required by SSA.
2. Lanczos methods do not use the explicit representation of the decomposed matrix $\mathbf{A}$. They need only the results of multiplication of $\mathbf{A}$ and $\mathbf{A}^{\mathrm{T}}$ on some vectors. In view of the special Hankel-type structure of $\mathbf{A}$ in the SSA algorithms, multiplication by a vector can be implemented very efficiently with the help of the Fast Fourier Transform (FFT). Fast SVD algorithms are implemented in the R-package svd (Korobeynikov et al. 2016) in such a way that their input is the function of a vector which yields the result of fast multiplication of the vector by the trajectory matrix. Therefore, the use of svd in Rssa allows a fast and space-efficient implementation of the SSA algorithms.
3. At Reconstruction step, hankelization or quasi-hankelization of a matrix of rank 1, stored as $\sigma U V^{\mathrm{T}}$, can be written by means of the convolution operator and therefore can also be effectively implemented; this is also done in Rssa.

The overall complexity of the computations is $O(kN\log(N) + k^2N)$, where N is the number of elements in a shaped object and k is the number of considered eigentriples, see details in Korobeynikov (2010) and in Golyandina et al. (2015). This makes the computations feasible for large data sets and large window sizes. For example, the case of an image of size 299×299 and a window size 100×100 can be handled rather easily, whereas the conventional algorithms (e.g., the full SVD (Golub and Van Loan 1996)) are impractical, because the matrix that needs to be decomposed has size $10^4 \times 4 \cdot 10^4$. Using larger window sizes is often advantageous, since, for example, separability of signal from noise (in the "signal+noise" scenario) can be significantly improved.

1.6 Comparison of SSA with Other Methods

In this section, we provide some notes and a short bibliographical overview concerning comparison of SSA with several traditional methods.

1.6.1 Fourier Transform, Filtering, Noise Reduction

- Fourier Transform uses a basis given in advance, while SSA uses an adaptive basis, which is not restricted to a frequency grid with resolution $1/N$. The wavelet transform also uses fixed bases; the advantage of the wavelet transform is that a change in the frequencies can be detected by the used time-space basis. In the framework of SSA, the analysis of time series with changing frequency structure can be performed by using moving procedures, e.g., by subspace tracking.
- One of the state-of-the-art methods of frequency estimation is the high-resolution method ESPRIT, which is a subspace-based method. This method can be considered as an SSA-related method and indeed it is frequently used in the present book and in Rssa.
- Fourier Transform is very inefficient for series with modulations in amplitudes and frequencies. SSA can easily deal with amplitude modulation but cannot efficiently deal with frequency modulation.
- SSA decomposition can sometimes be viewed as an application of a set of linear filters (Bozzo et al. 2010; Harris and Yan 2010; Golyandina and Zhigljavsky 2013) with an interpretation depending on the window length L. For small L, each decomposition component on the interval $[L, K]$, where $K = N - L + 1$, can be obtained by a linear filter. Therefore, the viewpoint of filtering on the decomposition result can be adequate. For example, the reconstruction by the leading components is close to application of the triangle filter.
  If $L \simeq K$ and hence the interval $[L, K]$ is small, then it is not so. In this case, the separability approach, which is based on orthogonality of separated components, is more appropriate. Note that oblique versions of SSA can weaken the condition of orthogonality, see Sect. 2.4.
- There is a big difference between the moving averaging and SSA for noise reduction. Consider an example of a noisy sinusoid. The moving averaging will add a bias in estimation caused by the second derivative of the signal, while SSA with large L will provide an unbiased estimate of the signal.
  Note that even for small L, when the reconstruction by the leading component is a weighted moving average with positive weights and therefore has the same drawbacks as the moving averaging, the user can add additional components to remove the possible bias.
- Filtering by SSA to obtain noise reduction can be considered from the viewpoint of the low-rank approximation. The good approximation properties yield appropriate noise suppression. Empirical-mode decomposition (EMD), in turn, starts Intrinsic Mode Functions (IMF) with high frequencies, while the trend is contained in the last IMFs.

As an example of comparison of SSA, Fourier transform and wavelet transform see, e.g., Kumar et al. (2017). The authors conclusion states: “the SSA-based filtering technique is robust for regional gravity anomaly separation and could be effectively exploited for filtering other geophysical data.”

In Barrios-Muriel et al. (2016), an SSA-based de-noising technique for removal of electrocardiogram interference in Electromyography signals is compared with the high-pass Butterworth filter, wavelets, and EMD. The authors of this paper state: "the proposed SSA approach is a valid method to remove the ECG artifact from the contaminated EMG signals without using an ECG reference signal."

In Watson (2016), many different methods for trend extraction are compared for synthetic data simulating sea level behavior; SSA is compared against moving average, wavelets, regression, EMD. The author writes: "the optimum performing analytic is most likely to be SSA whereby interactive visual inspection (VI) techniques are used by experienced practitioners to optimize window length and component separability."

Comparison of SSA filtering and Kalman filters (KF) can be found in Chen et al. (2016), where it is shown that "both SSA and KF obtain promising results from the stations with strong seasonal signals, while for the stations dominated by the long-term variations, SSA seems to be superior."

1.6.2 Parametric Regression

Parametric regression naturally assumes a parametric model. There is a big difference between parametric and non-parametric models: if the assumed model is true, then the related parametric methods are the most appropriate methods (if there are no outliers in the data). If the assumed parametric model is not true, then the results of parametric methods are biased and may be very misleading. Drawbacks of non-parametric methods are also clear: there are problems with forecasting, testing the model, confidence interval construction, and so on. Frequently, non-parametric methods serve as preprocessing tools for parametric methods. As discussed in Sect. 1.7.3, this is often the case for SSA.

For comparison of SSA with double centering and linear regression see, for example, Sect. 2.3 and Golyandina and Shlemov (2017). It appears that SSA with double centering as preprocessing method considerably improves the accuracy of linear trend estimation.

SSA has a very rare advantageous property: it can be a non-parametric method for preliminary analysis and can also be parametric for modeling the series governed by LRRs. Moreover, the forecast by an LRR uses the parametric model in implicit manner; therefore, it is more robust to deviations from the model than the forecast based on explicit parameter estimation.

One of the subspace-based method for constructing the model of the signal, which is governed by an LRR, is Hankel low-rank approximation (HLRA). HLRA can be considered as a method of parameter estimation in a parametric model, where only the rank of the signal is given rather than exact parametric form, see Sect. 3.4.

1.6.3 ARIMA and ETS

First, the (Seasonal) ARIMA and Exponential smoothing models (ETS, which means Error, Trend, Seasonal) totally differ from the model of SSA (for a comprehensive introduction to ARIMA and ETS, see Hyndman and Athanasopoulos (2013)). In particular, in ARIMA the noise is added at each recurrence step, while for SSA the noise is added after the signal is formed. Also, trends/seasonality in SSA are deterministic, while in ARIMA/ETS the trends/seasonality are random. As in many classical methods, ARIMA and ETS need the period values to be specified for the periodic components.

However, if one considers the analysis/forecast of real-life time series, then these time series do not exactly follow any model. Therefore, the problem of comparison of methods of different nature is not easy.

As a rule, confidence intervals for ARIMA forecasts are too large but the mean forecast can often be adequate. Advantage of Seasonal ARIMA and ETS is that the model and its parameters can be fitted automatically on the base of information criteria.

Rigorously substantiated information criteria are not constructed for SSA. One of the reasons for this is the fact that SSA is a non-parametric method. The most standard approach for the choice of parameters, when there are no given models, is the minimization of the forecasting error on the validation period. In the most frequent case, when the forecast is constructed on the base of r leading eigentriples, SSA has only two parameters (L and r), which can be estimated by the minimization of the forecasting error for several forecasts performed within the validation period, see Sect. 3.5.7.

Comparison of SSA and ARIMA/ETS was performed in many papers. Some examples are as follows.

- It is demonstrated in Hassani et al. (2015) that SSA has topped several other methods in an example involving forecasting of tourist arrivals,
- It is exhibited in Vile et al. (2012) that for predicting ambulance demands "SSA produces superior longer-term forecasts (which are especially helpful for EMS planning), and comparable shorter-term forecasts to well established methods."
- The author of Iqelan (2017) concludes: "The forecasting results are compared with the results of exponential smoothing state space (ETS) and ARIMA models. The three techniques do similarly well in forecasting process. However, SSA outperforms the ETS and ARIMA techniques according to forecasting error accuracy measures."
- In Hassani et al. (2009, 2013), the univariate and multivariate SSA were favorable in a comparison with ARIMA and VAR for forecasting of several series of European industrial production.

1.7 Bibliographical Notes

1.7.1 Short History

Commencement of SSA is usually associated with publication in 1986 of the papers Broomhead and King (1986) and Broomhead and King (1986b). However, some ideas, which later became parts of SSA, have been formulated very long ago (de Prony 1795). Arguably, the most influential papers on SSA published in the 1980 and 1990s, in addition to Broomhead and King (1986,b), are Fraedrich (1986), Vautard and Ghil (1989), Vautard et al. (1992), Allen and Smith (1996). In view of many successful applications of SSA, the number of publications considering SSA methodology grows exponentially and has surely reached few hundred.

A parallel development of SSA (under the name "Caterpillar") has been conducted in the former USSR, especially, in St.Petersburg (known at that time as Leningrad), see, e.g., Danilov and Zhigljavsky (1997). The authors of this book continue the traditions of the St.Petersburg school of SSA.

The monograph (Golyandina et al. 2001) contains a comprehensive description of the theoretical and methodological foundations of SSA for one-dimensional (1D) time series; the authors of that monograph tried to summarize all the knowledge about 1D-SSA available at that time. A short book (Golyandina and Zhigljavsky 2013) developed further the methodology of 1D-SSA. It reflected the authors' new understandings as well as new SSA insights including subspace-based methods, filtering and rotations in the signal space for improving separability. A substantial paper (Golyandina et al. 2015) supplements the above books by expanding SSA for processing multivariate time series and digital images.

1.7.2 Some Recent Applications of SSA

The number of publications devoted to applications of SSA is steadily increasing. In addition to the standard applications areas such as climatology, meteorology, and geophysics, there are now many papers devoted to applications in engineering, economics, finance, biomedicine, and other areas. One can find many references to recent publications in Zhigljavsky (2010) and many papers in the two special issues of *Statistics and Its Interface* (2010, v.3, No.3 and 2017, v.10, No.1), which are either fully or partly devoted to SSA. In this short section we briefly mention some recent applications of SSA. In most of these papers, only the simplest versions of SSA (that is, Basic SSA of Sect. 2.1 and Toeplitz SSA of Sect. 2.2) have been used.

Advantages of 2D-SSA (described in Sect. 5.1) over some other methods of image processing are demonstrated in Zabalza et al. (2014, 2015) in application to hyperspectral imaging. Application of 2D-SSA to gap-filling is considered in von Buttlar et al. (2014). Application of Multivariate SSA for detecting oscillator clusters in multivariate datasets is proposed in Groth and Ghil (2015).

It is not easy to find applied areas related to analysis of temporal data, where 1D-SSA was not applied. Let us give some examples. In Salgado et al. (2013) and several other papers of the same authors, SSA has been used as the main technique in the development of a tool-wear monitoring system. Security of mobile devices is considered in Genkin et al. (2016), where SSA is used for preprocessing. In Sella et al. (2016), SSA was used for extraction of economic cycles. Filho and Lima (2016) use SSA for gap filling of precipitation data. Some recent applications in climatology were considered in Mudersbach et al. (2013), Monselesan et al. (2015) and in Pepelyshev and Zhigljavsky (2017). In Karnjana et al. (2017), SSA helps to solve the problem of unauthorized modification in speech signals. In Barrios-Muriel et al. (2016), SSA is used for de-noising in the problem of removal of electrocardiogram interference in electromyography signals. The paper (Hudson and Keatley 2017) is related to the decomposition and reconstruction of long-term flowering records of eight eucalypt species. In Wang et al. (2017), SSA was used as a preprocessing tool prior to making a classification of a medical data; the authors wrote: "the results have demonstrated the robustness of the approach when testing on large scale datasets with clinically acceptable sensitivity and specificity."

1.7.3 SSA for Preprocessing/Combination of Methods

For many different methods, SSA provides improvement if it is used as a preprocessing tool. There are dozens of papers, where hybrid methods incorporating SSA are considered. In most of the applications, SSA serves for either denoising or feature extraction. Let us give some examples of papers considering hybrids of SSA and other methods.

SSA is used as a preprocessing step for ARIMA in Zhang et al. (2011). A cooperative hybrid of SSA, ARIMA, and Holt-Winters is suggested in Xin et al. (2015). In Lakshmi et al. (2016) it is shown that the hybrid SSA + ARMAX is better than ARMAX alone for detection of structural damages for problems of Structural Health Monitoring.

In machine leaning, SSA is frequently used to obtain new characteristics of time series for a subsequent use of them in other models and methods. This is called feature extraction. The paper (Sivapragasam et al. 2001) is considered as a one of the first papers, where SSA is used together with Support Vector Machines (SVM). A hybrid of SSA with Neural Networks was suggested in Lisi et al. (1995).

In Wang et al. (2016), support vector machine regression (SVR) is applied separately to the trend and fluctuations, which are extracted by SSA. The constructed method is applied to forecast a time series data of failures gathered at the maintenance stage of the Boeing 737 aircraft. It is shown that the suggested hybrid SSA+SVR outperforms Holt-Winters, autoregressive integrated moving average, multiple linear regression, group method of data handling, SSA, and SVR used separately. Similar techniques are considered in Xiao et al. (2014), where SSA is employed for extraction of the trend and seasonality and then Neural Networks and

fuzzy logic are applied to them separately with consequent combination. In Wu and Chau (2011), SSA is successfully used for noise removal before Neural Networks are applied. This work contains a review of different approaches to Rainfall-runoff modeling by means of SSA used in combination with other methods.

In Zabalza et al. (2014), SSA has been applied in hyperspectral imaging for effective feature extraction (noise removal), and then SVM was used for classification. It appeared that SSA performed preprocessing better than Empirical Mode Decomposition (EMD). Note that SSA and EMD do not only compete; they can be successful as hybrids. For example, in Yang et al. (2012) EMD is used for trend extraction and then SSA is applied to forecast changes in the trend.

1.7.4 Materials Used in This Book

In writing this book we have used much material from different sources. Many sections contain the material which is entirely new but other sections are based on our previous publications. Let us briefly describe the main references we have used in writing the theoretical and methodological material of the book.

Chapter 1 (Introduction: Overview) contains an original approach to the SSA modifications from a general viewpoint. The generic scheme of SSA-family methods from Sect. 1.1 was suggested in Golyandina et al. (2015).

1D-SSA is well elaborated and therefore Chap. 2 (SSA analysis of one-dimensional time series), in addition to some new material (this especially concerns new examples and the discussions concerning RSSA), revises standard SSA techniques. Sections 2.1 (Basic SSA) and Sect. 2.2 (Toeplitz SSA) contain standard material partially taken from Golyandina et al. (2001; Chapter 1). Ideas of Sect. 2.3 (SSA with projection) were firstly suggested in Golyandina et al. (2001; Section 1.7.1) (centering) and then extended in Golyandina and Shlemov (2017). Methods described in Sect. 2.4 (Iterative Oblique SSA) and Sect. 2.5 (Filter-adjusted O-SSA and SSA with derivatives) were suggested in Golyandina and Shlemov (2015); these two sections closely follow this paper. Section 2.6 (Shaped 1D-SSA) contains a particular case of Shaped SSA, which was suggested in Golyandina et al. (2015) for multidimensional case and is described in Sect. 5.2. Section 2.7 (Automatic grouping in SSA) follows Alexandrov (2009) and Golyandina and Zhigljavsky (2013; Section 1.4.5).

Much of the theoretical material of Chap. 3 (Parameter estimation, forecasting, gap filling) is standard for the methodologies of the subspace-based methods. In writing Sect. 3.1 (Parameter estimation) we have extensively used Golyandina and Zhigljavsky (2013; Sections 2.2, 2.8). Section 3.2 (Forecasting) includes the algorithms from Golyandina et al. (2001; Chapter 2). Gap filling in Sect. 3.3 contains two methods, iterative method taken from Kondrashov and Ghil (2006) and the subspace-based method taken from Golyandina and Osipov (2007). Both methods are described in accordance with Golyandina and Zhigljavsky (2013). Section 3.4 devoted to structured low-rank approximation (briefly, SLRA) describes

Cadzow-like (Cadzow 1988) iterative algorithms for finding low-rank approximations. SLRA is a very standard approach, which was extended to weighted Cadzow iterations in Zhigljavsky et al. (2016a) and Zvonarev and Golyandina (2017).

In writing Chap. 4 (SSA for multivariate time series) we use Golyandina and Stepanov (2005) and Golyandina et al. (2015). In Chap. 5 (Image processing), we mainly follow Golyandina et al. (2015). Moreover, in Sect. 5.1 (2D-SSA) we use material from Golyandina and Usevich (2010) and in Sect. 5.2 (Shaped 2D-SSA) and Sect. 5.3 (2D ESPRIT) we incorporate the ideas developed in Golyandina et al. (2015) and Shlemov and Golyandina (2014).

Some material in the algorithmic and RSSA sections is based on the papers (Golyandina and Korobeynikov 2013; Golyandina et al. 2015).

1.8 Installation of RSSA and Description of the Data Used in the Book

1.8.1 Installation of RSSA *and Usage Comments*

The package RSSA is available from CRAN on http://CRAN.R-project.org/package=Rssa and can be installed via the standard `install.packages` routine and therefore all the dependencies are installed automatically.

There is a special library, FFTW (Frigo and Johnson 2005), which is not mandatory for the installation of RSSA; however, possibilities of RSSA would be considerably lower if FFTW is not installed. The library FFTW should be installed prior to RSSA by the standard tools of the used operating system. For example, FFTW can be installed by running `apt-get install libfftw3-bin libfftw3-dev` (Ubuntu Linux) or `brew install fftw` (MacOS, homebrew). Windows pre-built packages from CRAN already use FFTW.

Sources of RSSA can be found at https://github.com/asl/rssa, where the user can ask questions about installation and usage problems. In addition, the current development version of RSSA could be installed straight from github repository using `install_github` from DEVTOOLS (Wickham and Chang 2017).

In this book, RSSA v1.0 has been used for all illustrative examples. All SSA computations can be reproduced by the reader and should run correctly by any later version of RSSA. The sets of data used in these examples are included into the R-package SSABOOK, unless a particular set is contained in one of the following three R-packages: built-in DATASETS, FMA, and RSSA. The description of the datasets is contained in Tables 1.2 and 1.3. In order to reproduce the examples from the book, SSABOOK should be installed as well as LATTICE, LATTICEEXTRA, PLYR, and FMA. Source codes for all examples as well as the R-package SSABOOK can be downloaded from the web-site devoted to the book https://ssa-with-r-book.github.io.

In the following chapters there are quite a few sections named "Description of functions." In these sections, we describe the main RSSA functions and their basic

Table 1.2 Description of data and R-packages

Data	Short description	Timing	R package	Dataframe name
AustralianWine	Australian wine sales: thousands of liters. By wine makers in bottles $\leq$ 1 l	Monthly	Rssa	AustralianWine
CO2	Atmospheric concentrations of CO2 in parts per million (ppm)	Monthly	datasets	co2
White dwarf	Time variation of the intensity of the variable white dwarf star PG1159-035	Each 10 s	ssabook	dwarfst
Production	Crude oil and natural gas plant liquids production	Monthly	ssabook	oilproduction
Tree rings	Tree ring indices, Douglas fir, Snake river basin, USA	Annual	ssabook	dftreerings
MotorVehicle	Total domestic and foreign car sales in the USA in thousands	Monthly	Rssa	MotorVehicle
US unemployment	U.S. male and female unemployment figures in thousands (16–19 years and from 20 years)	Monthly	Rssa	USUnemployment
Hotel	Hotel occupied room av.	Monthly	ssabook	hotel
PayNSA	All employees: Total nonfarm payrolls	Monthly	ssabook	paynsa
Elec	Australian electricity production	Monthly	fma	elec
Cowtemp	Daily morning temperature of a cow, measured at 6.30am	Daily	fma	cowtemp
Glonass	Glonass time corrections	Each 5 min	ssabook	g15
Sunspots	Mean total sunspot number	Annual	ssabook	sunspot2
Bookings	Numbers of hotel bookings through a particular web-site	Hourly	ssabook	bookings
EuStockMarkets	Closing prices of major European stock indices	Daily	datasets	EuStockMarkets
Mars	An image of Mars performed by Pierre Thierry	Image	Rssa	Mars
Brecon Beacons	The test DTM of a region in South Wales, UK	Image	ssabook	brecon
Kruppel	Regularized data of gene expression for the “Krüppel” gene (a drosophila embryo)	image	ssabook	kruppel
Monet	“A preview of the painting of Claude Monet called “Study of a Figure Outdoors: Woman with a Parasol, facing left,” 1886	Image	ssabook	monet

Table 1.3 Description of data and sources

Data	Length	Time range	Source
AustralianWine	176	Jan 1980–Jul 1995	Hyndman (2013)
CO2	468	Jan 1959–Jan 1997	Cleveland (1993)
White dwarf	618	during March 1989	Weigend and Gershenfeld (1993)
Production	300	Jan 1973–Dec 1997	Source: U.S. Energy Information Administration (Jan 2016) http://www.eia.gov/totalenergy/data/monthly/#summary
Tree rings	669	1282–1950	Hyndman (2013)
MotorVehicle	541	Jan 1967–Jan 2012	U.S. Bureau of Economic Analysis, 2015. Table 7.2.5s https://www.bea.gov/histdata/Releases/GDP_and_PI/2015/Q1/Third_June-24-2015/UND/Section7ALL_xls.xls
US unemployment	408	Jan 1948–Jan 1981	Andrews and Herzberg (1985)
Hotel	168	Jan 1963–Dec 1976	Hyndman (2013)
PayNSA	913	Jan 1939–Jan 2015	Hyndman (2013)
Elec	476	Jan 1956–Aug 1995	Makridakis et al. (1998)
Cowtemp	75	Unknown	Makridakis et al. (1998)
Glonass	104832	02/01/2014 to 31/12/2014	https://www.glonass-iac.ru/en/index.php
Sunspots	316	1700–2015	Hyndman (2013)
Bookings	4344	23/09/2016 to 22/03/2017	Provided by Crimtan, UK, to the authors of the R-package SSABOOK
EuStockMarkets	1860	During 1994–1998	Provided by Erste Bank AG, Vienna, Austria, to the authors of the R-package DATASETS
Mars	258×275	NA	http://www.astrosurf.com/buil/iris/tutorial8/doc23_us.htm
Brecon Beacons	80×100	NA	The data are obtained by means of the function `getData` of the R-package RASTER
Kruppel	200×200	NA	http://bdtnp.lbl.gov/Fly-Net/
Monet	$400 \times 263 \times 3$	NA	https://commons.wikimedia.org/wiki/File:Monet.012.sonnenschirm.jpg

arguments. For more information about the RSSA functions and their arguments, we refer the reader to the help information in the RSSA package.

1.8.2 Data Description

Tables 1.2 and 1.3 present the data used in the book for examples. All these sets of data can be found in the R-packages indicated in the fourth column of Table 1.2. Table 1.3 contains one of possible references; detailed descriptions and references can be found in the corresponding R-packages.

References

Alexandrov T (2009) A method of trend extraction using singular spectrum analysis. RevStat 7(1):1–22

Allen M, Smith L (1996) Monte Carlo SSA: Detecting irregular oscillations in the presence of colored noise. J Clim 9(12):3373–3404

Andrews D, Herzberg A (1985) Data. A collection of problems from many fields for the student and research worker. Springer, New York

Barrios-Muriel J, Romero F, Alonso FJ, Gianikellis K (2016) A simple SSA-based de-noising technique to remove ECG interference in EMG signals. Biomed Signal Process Control 30:117–126

Bozzo E, Carniel R, Fasino D (2010) Relationship between singular spectrum analysis and Fourier analysis: Theory and application to the monitoring of volcanic activity. Comput Math Appl 60(3):812–820

Broomhead D, King G (1986) Extracting qualitative dynamics from experimental data. Physica D 20:217–236

Broomhead D, King G (1986b) On the qualitative analysis of experimental dynamical systems. In: Sarkar S (ed) Nonlinear phenomena and chaos. Adam Hilger, Bristol, pp 113–144

von Buttlar J, Zscheischler J, Mahecha MD (2014) An extended approach for spatiotemporal gapfilling: dealing with large and systematic gaps in geoscientific datasets. Nonlinear Process Geoph 21:203–215

Cadzow JA (1988) Signal enhancement: a composite property mapping algorithm. IEEE Trans Acoust 36(1):49–62

Chen Q, Weigelt M, Sneeuw N, van Dam T (2016) On time-variable seasonal signals: Comparison of SSA and Kalman filtering based approach. Springer International Publishing, Cham, pp 75–80

Cleveland WS (1993) Visualizing data. Hobart Press

Danilov D, Zhigljavsky A (eds) (1997) Principal components of time series: the "Caterpillar" method. St. Petersburg Press (in Russian)

Filho ASF, Lima GAR (2016) Gap filling of precipitation data by SSA - singular spectrum analysis. J Phys Conf Ser 759(1):012,085

Fraedrich K (1986) Estimating dimensions of weather and climate attractors. J Atmos Sci 43: 419–432

Frigo M, Johnson SG (2005) The design and implementation of FFTW3. Proc IEEE 93(2):216–231

Genkin D, Pachmanov L, Pipman I, Tromer E, Yarom Y (2016) Ecdsa key extraction from mobile devices via nonintrusive physical side channels. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. ACM, New York, NY, USA, CCS '16, pp 1626–1638

Ghil M, Allen RM, Dettinger MD, Ide K, Kondrashov D, Mann ME, Robertson A, Saunders A, Tian Y, Varadi F, Yiou P (2002) Advanced spectral methods for climatic time series. Rev Geophys 40(1):1–41

Golub GH, Van Loan CF (1996) Matrix computations, 3rd edn. Johns Hopkins University Press, Baltimore, MD, USA

Golyandina N, Korobeynikov A (2013) Basic singular spectrum analysis and forecasting with R. Comput Stat Data Anal 71:943–954

Golyandina N, Osipov E (2007) The "Caterpillar"-SSA method for analysis of time series with missing values. J Stat Plan Inference 137(8):2642–2653

Golyandina N, Shlemov A (2015) Variations of singular spectrum analysis for separability improvement: Non-orthogonal decompositions of time series. Stat Interface 8(3):277–294

Golyandina N, Shlemov A (2017) Semi-nonparametric singular spectrum analysis with projection. Stat Interface 10(1):47–57

Golyandina N, Stepanov D (2005) SSA-based approaches to analysis and forecast of multidimensional time series. In: Proceedings of the 5th St.Petersburg workshop on simulation, June 26–July 2, 2005. St. Petersburg State University, St. Petersburg, pp 293–298

Golyandina N, Usevich K (2010) 2D-extension of singular spectrum analysis: algorithm and elements of theory. In: Olshevsky V, Tyrtyshnikov E (eds) Matrix methods: Theory, algorithms and applications. World Scientific Publishing, pp 449–473

Golyandina N, Zhigljavsky A (2013) Singular apectrum analysis for time series. Springer briefs in statistics. Springer

Golyandina N, Nekrutkin V, Zhigljavsky A (2001) Analysis of time series structure: SSA and related techniques. Chapman&Hall/CRC

Golyandina N, Korobeynikov A, Shlemov A, Usevich K (2015) Multivariate and 2D extensions of singular spectrum analysis with the Rssa package. J Stat Softw 67(2):1–78

Groth A, Ghil M (2015) Monte Carlo singular spectrum analysis (SSA) revisited: Detecting oscillator clusters in multivariate datasets. J Climate 28(19):7873–7893

Hall MJ (1998) Combinatorial theory. Wiley, New York

Harris T, Yan H (2010) Filtering and frequency interpretations of singular spectrum analysis. Physica D 239:1958–1967

Hassani H, Heravi S, Zhigljavsky A (2009) Forecasting European industrial production with singular spectrum analysis. Int J Forecast 25(1):103–118

Hassani H, Heravi S, Zhigljavsky A (2013) Forecasting UK industrial production with multivariate singular spectrum analysis. J Forecast 32(5):395–408

Hassani H, Webster A, Silva ES, Heravi S (2015) Forecasting U.S. tourist arrivals using optimal singular spectrum analysis. Tourism Manag 46:322–335

Hudson IL, Keatley MR (2017) Singular spectrum analytic (ssa) decomposition and reconstruction of flowering: Signatures of climatic impacts. Environ Model Assess 22(1):37–52

Hyndman R, Athanasopoulos G (2013) Forecasting: principles and practice. OTexts, URL http://otexts.org/fpp/, accessed on 28.07.2017

Hyndman RJ (2013) Time Series Data Library. URL http://data.is/TSDLdemo, accessed on 10/08/2013

Iqelan BM (2017) A singular spectrum analysis technique to electricity consumption forecasting. Int J Eng Res Appl 7(3):92–100

Karnjana J, Unoki M, Aimmanee P, Wutiwiwatchai C (2017) Tampering detection in speech signals by semi-fragile watermarking based on singular-spectrum analysis. Springer International Publishing, Cham, pp 131–140

Kondrashov D, Ghil M (2006) Spatio-temporal filling of missing points in geophysical data sets. Nonlinear Process Geophys 13(2):151–159

Korobeynikov A (2010) Computation- and space-efficient implementation of SSA. Stat Interface 3(3):357–368

Korobeynikov A, Larsen RM, Wu KJ, Yamazaki I (2016) SVD: Interfaces to various state-of-art SVD and eigensolvers. URL http://CRAN.R-project.org/package=svd, R package version 0.4

Korobeynikov A, Shlemov A, Usevich K, Golyandina N (2017) Rssa: A collection of methods for singular spectrum analysis. URL http://CRAN.R-project.org/package=Rssa, R package version 1.0

Kumar KS, Rajesh R, Tiwari RK (2017) Regional and residual gravity anomaly separation using the singular spectrum analysis-based low pass filtering: a case study from Nagpur, Maharashtra, India. Explor Geophys.

Kurakin V, Kuzmin A, Mikhalev A, Nechaev A (1995) Linear recurring sequences over rings and modules. J Math Sci 76(6):2793–2915

Lakshmi K, Rao ARM, Gopalakrishnan N (2016) Singular spectrum analysis combined with ARMAX model for structural damage detection. Struct Control Health Monit. https://doi.org/10.1002/stc.1960

Lisi F, Nicolis O, Sandri M (1995) Combining singular-spectrum analysis and neural networks for time series forecasting. Neural Process Lett 2(4):6–10

Makridakis S, Wheelwright S, Hyndman R (1998) Forecasting: Methods and applications, 3rd edn. Wiley, New York

Monselesan DP, O'Kane TJ, Risbey JS, Church J (2015) Internal climate memory in observations and models. Geophys Res Lett 42(4):1232–1242

Mudersbach C, Wahl T, Haigh ID, Jensen J (2013) Trends in high sea levels of German North Sea gauges compared to regional mean sea level changes. Cont Shelf Res 65:111–120

Pepelyshev A, Zhigljavsky A (2017) Ssa analysis and forecasting of records for earth temperature and ice extents. Stat Interface 10(1):151–163

de Prony G (1795) Essai expérimental et analytique sur les lois de la dilatabilité des fluides élastiques et sur celles de la force expansive de la vapeur de l'eau et la vapeur de l'alkool à différentes températures. J de l'Ecole Polytechnique 1(2):24–76

Salgado D, Cambero I, Olivenza JH, Sanz-Calcedo JG, López PN, Plaza EG (2013) Tool wear estimation for different workpiece materials using the same monitoring system. Procedia Eng 63:608–615

Sella L, Vivaldo G, Groth A, Ghil M (2016) Economic cycles and their synchronization: A comparison of cyclic modes in three European countries. J Bus Cycle Res 12(1):25–48

Shlemov A, Golyandina N (2014) Shaped extensions of singular spectrum analysis. In: 21st international symposium on mathematical theory of networks and systems, July 7–11, 2014. Groningen, The Netherlands, pp 1813–1820

Sivapragasam C, Liong SY, Pasha M (2001) Rainfall and runoff forecasting with SSA–SVM approach. J Hydroinform 3(3):141–152

Vautard M, Ghil M (1989) Singular spectrum analysis in nonlinear dynamics, with applications to paleoclimatic time series. Physica D 35:395–424

Vautard R, Yiou P, Ghil M (1992) Singular-spectrum analysis: A toolkit for short, noisy chaotic signals. Physica D 58:95–126

Vile JW J L nnd Gillard, Harper PR, Knight VA (2012) Predicting ambulance demand using singular spectrum analysis. J Oper Res Soc 63(11):1556–1565

Wang S, Tang HL, turk LIA, Hu Y, Sanei S, Saleh GM, Peto T (2017) Localizing microaneurysms in fundus images through singular spectrum analysis. IEEE Trans Biomed Eng 64(5):990–1002

Wang X, Wu J, Liu C, Wang S, Niu W (2016) A hybrid model based on singular spectrum analysis and support vector machines regression for failure time series prediction. Qual Reliab Eng Int 32(8):2717–2738, qRE-16-0186.R2

Watson PJ (2016) Identifying the best performing time series analytics for sea level research. Springer International Publishing, Cham, pp 261–278

Weigend A, Gershenfeld N (eds) (1993) Time series prediction: Forecasting the future and understanding the past. Addison-Wesley, Reading

Wickham H, Chang W (2017) DEVTOOLS: Tools to Make Developing R Packages Easier. URL http://CRAN.R-project.org/package=devtools, R package version 1.13.2

Wu C, Chau K (2011) Rainfall–runoff modeling using artificial neural network coupled with singular spectrum analysis. J Hydrol 399(3):394–409

Xiao Y, Liu JJ, Hu Y, Wang Y, Lai KK, Wang S (2014) A neuro-fuzzy combination model based on singular spectrum analysis for air transport demand forecasting. J Air Transp Manag 39:1–11

Xin W, Chao L, Weiren X, Ying L (2015) A failure time series prediction method based on UML model. In: 2015 4th international conference on computer science and network technology (ICCSNT), vol 01, pp 62–70

Yang Z, Bingham C, Ling WK, Zhang Y, Gallimore M, Stewart J (2012) Unit operational pattern analysis and forecasting using EMD and SSA for industrial systems. Springer, Berlin, Heidelberg, pp 416–423

Zabalza J, Ren J, Wang Z, Marshall S, Wang J (2014) Singular spectrum analysis for effective feature extraction in hyperspectral imaging. IEEE Geosci Remote Sens Lett 11(11):1886–1890

Zabalza J, Ren J, Zheng J, Han J, Zhao H, Li S, Marshall S (2015) Novel two-dimensional singular spectrum analysis for effective feature extraction and data classification in hyperspectral imaging. IEEE Trans Geosci Remote Sens 53(8):4418–4433

Zhang Q, Wang BD, He B, Peng Y, Ren ML (2011) Singular spectrum analysis and ARIMA hybrid model for annual runoff forecasting. Water Resour Manag 25(11):2683–2703

Zhigljavsky A (2010) Singular spectrum analysis for time series: Introduction to this special issue. Stat Interface 3(3):255–258

Zhigljavsky A, Golyandina N, Gillard J (2016a) Analysis and design in the problem of vector deconvolution. In: Kunert J, Müller HC, Atkinson CA (eds) mODa 11 - Advances in model-oriented design and analysis. Springer International Publishing, pp 243–251

Zvonarev N, Golyandina N (2017) Iterative algorithms for weighted and unweighted finite-rank time-series approximations. Stat Interface 10(1):5–18

Chapter 2
SSA Analysis of One-Dimensional Time Series

In the present chapter and Chap. 3, we thoroughly examine the use of SSA for one-dimensional data. This chapter is fully devoted to the SSA analysis of such data. Consideration of SSA forecasting, gap filling, and estimation of parameters of the signal is delayed until Chap. 3. The main difference between the materials of these two chapters is the use of the models. In the present chapter, the use of models is minimal; on the contrary, the methodologies of Chap. 3 are model-based.

In the terminology of Chap. 1, SSA for one-dimensional data should be referred to as 1D-SSA. However, but for the sake of brevity, in this chapter we will refer to it simply as SSA. The SSA input for all algorithms of this chapter is a collection $\mathbb{X}_N = (x_1, \ldots, x_N)$ of N real numbers, which can be thought of as a time series.

Let us start with the common parts of all versions of SSA algorithms considered in this chapter. These common parts are the embedding procedure at Step 1 of SSA and the diagonal averaging which makes the reconstruction at Step 4 (see Fig. 1.1).

Let L $(1 < L < N)$ be some integer called *window length* and set $K = N - L + 1$. Construct L-lagged vectors of size L as

$$X_i = (x_i, \ldots, x_{i+L-1})^{\mathrm{T}}, \quad i = 1 \ldots, K.$$

Define the embedding operator $\mathcal{T} = \mathcal{T}_{\mathrm{SSA}}$ by

$$\mathcal{T}_{\mathrm{SSA}}(\mathbb{X}) = \mathbf{X} = [X_1 : \ldots : X_K] = \begin{pmatrix} x_1 & x_2 & x_3 & \ldots & x_K \\ x_2 & x_3 & x_4 & \ldots & x_{K+1} \\ x_3 & x_4 & x_5 & \ldots & x_{K+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_L & x_{L+1} & x_{L+2} & \ldots & x_N \end{pmatrix}. \tag{2.1}$$

N. Golyandina et al., *Singular Spectrum Analysis with R*, Use R!,
https://doi.org/10.1007/978-3-662-57380-8_2

Matrix $\mathbf{X}$ of (2.1) is called trajectory (or L-trajectory) matrix. There are two important properties of this matrix:

(a) both the rows and columns of $\mathbf{X}$ are subseries of the original series, and
(b) $\mathbf{X}$ has equal elements on its anti-diagonals, which is equivalent to saying that $\mathbf{X}$ is a Hankel matrix.

The operator $\mathcal{T} = \mathcal{T}_{\mathrm{SSA}} : \mathsf{R}^N \to \mathcal{M}^{(\mathrm{H})}_{L,K}$ makes a correspondence between time series (collections of N numbers) and $\mathcal{M}^{(\mathrm{H})}_{L,K}$, the set of Hankel matrices of size $L \times K$. Since the correspondence defined by $\mathcal{T}$ is one-to-one, there exists the inverse $\mathcal{T}^{-1}$, which transfers any Hankel matrix of size $L \times K$ to a series of length N.

Let us also introduce the projector $\Pi_{\mathcal{H}} : \mathsf{R}^{L\times K} \to \mathcal{M}^{(\mathrm{H})}_{L,K}$ into the space of Hankel matrices as the operator of hankelization

$$(\Pi_{\mathcal{H}}\mathbf{Y})_{ij} = \sum_{(l,k)\in A_s} y_{lk} \big/ w_s, \tag{2.2}$$

where $s = i + j - 1$, $A_s = \{(l,k) : l + k = s + 1, 1 \le l \le L, 1 \le k \le K\}$ and $w_s = |A_s|$ denotes the number of elements in the set A_s. This corresponds to averaging the matrix elements over the "anti-diagonals." The weights w_s are equal to the number of series elements x_s in the trajectory matrix (2.1) and has a trapezoidal shape, decreasing towards both ends of the series.

On the base of (2.2), any matrix $\mathbf{Y} \in \mathsf{R}^{L\times K}$ can be transferred to a series of length N by applying $\mathcal{T}^{-1} \circ \Pi_{\mathcal{H}}$. Note that this is done in an optimal way.

2.1 Basic SSA

Basic SSA is the SSA for the analysis of one-dimensional series such that the decomposition into rank-one matrices at Step 2 (see the general scheme in Fig. 1.1) is done by the SVD.

2.1.1 Method

2.1.1.1 Step 1: Embedding

The series $\mathbb{X}$ is mapped to a sequence of L-lagged vectors, which form the trajectory matrix $\mathbf{X} = \mathcal{T}_{\mathrm{SSA}}(\mathbb{X})$, as shown in (2.1).

2.1.1.2 Step 2: Decomposition

Set $\mathbf{S} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$ and denote by $\lambda_1, \ldots, \lambda_d$ the positive *eigenvalues* of $\mathbf{S}$ taken in the decreasing order of magnitude ($\lambda_1 \geq \ldots \geq \lambda_d > 0$) and by $U_1, \ldots, U_d$ an orthonormal system of the *eigenvectors* of the matrix $\mathbf{S}$ corresponding to these eigenvalues; $V_i = \mathbf{X}^{\mathrm{T}} U_i / \sqrt{\lambda_i}$ are called factor vectors. At this step, we perform the singular value decomposition (SVD) of the trajectory matrix:

$$\mathbf{X} = \sum_{i=1}^{d} \sqrt{\lambda_i} U_i V_i^{\mathrm{T}} = \mathbf{X}_1 + \ldots + \mathbf{X}_d. \tag{2.3}$$

The matrices $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$ in (2.3) have rank 1; such matrices are called *elementary matrices*. The collection $(\sqrt{\lambda_i}, U_i, V_i)$ consisting of the singular value $\sqrt{\lambda_i}$, the left singular vector U_i and the right singular vector V_i will be called ith *eigentriple* (abbreviated as ET). Note that $\lambda_i = \|\mathbf{X}_i\|_{\mathrm{F}}^2$ and $\|\mathbf{X}\|_{\mathrm{F}}^2 = \|\mathbf{X}_1\|_{\mathrm{F}}^2 + \ldots + \|\mathbf{X}_d\|_{\mathrm{F}}^2$. The contribution of ith component $\mathbf{X}_i$ can thus be measured by $\lambda_i / \sum_j \lambda_j$.

For real-world time series, $d = \operatorname{rank} \mathbf{X}$ is typically equal to $\min(L, K)$; that is, the trajectory matrix is of full rank.

2.1.1.3 Step 3: Eigentriple Grouping

The input in this step is the expansion (2.3) and the specification of how to group the components of (2.3).

Let $I = \{i_1, \ldots, i_p\} \subset \{1, \ldots, d\}$ be a set of indices. Then the resultant matrix $\mathbf{X}_I$ corresponding to the group I is defined as $\mathbf{X}_I = \mathbf{X}_{i_1} + \ldots + \mathbf{X}_{i_p}$.

Assume that a partition of the set of indices $\{1, \ldots, d\}$ into m disjoint subsets $I_1, \ldots, I_m$ is specified. Then the expansion (2.3) leads to the decomposition

$$\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_m}. \tag{2.4}$$

The procedure of choosing the sets $I_1, \ldots, I_m$ is called *eigentriple grouping*. If $m = d$ and $I_j = \{j\}$, $j = 1, \ldots, d$, then the corresponding grouping is called *elementary*.

The grouping is performed by analyzing the eigentriples so that each group corresponds to an identifiable series component. The choice of several leading eigentriples corresponds to an optimal approximation of the time series, in accordance with the well-known optimality property of the SVD.

2.1.1.4 Step 4: Reconstruction (Diagonal Averaging)

The diagonal averaging (2.2) applied to a resultant matrix $\mathbf{X}_{I_k}$ produces a *reconstructed series* $\widetilde{\mathbb{X}}^{(k)} = (\widetilde{x}_1^{(k)}, \ldots, \widetilde{x}_N^{(k)}) = \mathfrak{T}^{-1} \circ \Pi_{\mathcal{H}}(\mathbb{X}^{(k)})$. This way, the initial

series $(x_1, \ldots, x_N)$ is decomposed into a sum of m reconstructed series:

$$x_n = \sum_{k=1}^{m} \widetilde{x}_n^{(k)}, \quad n = 1, \ldots, N. \tag{2.5}$$

The reconstructed series produced by the elementary grouping will be called *elementary reconstructed series*.

If grouping is sensible, then we obtain a reasonable decomposition into identifiable series components. Typical resultant decompositions are signal plus noise or trend plus seasonality plus noise.

As well as in the generic scheme, Steps 1 and 2 of Basic SSA are sometimes combined into the so-called "Decomposition stage" and Steps 3 and 4 are combined into "Reconstruction stage."

2.1.2 Appropriate Time Series

2.1.2.1 Time Series of Finite Rank

Although the SSA method is model-free and therefore SSA can be considered as an exploratory method, there is a model that perfectly suits SSA.

We say that a series has L-rank r if its L-trajectory matrix has rank r. Series $\mathbb{S}$ is called a series of finite rank r ($\operatorname{rank}\mathbb{S} = r$) if $\operatorname{rank}_L \mathbb{S} = r$ for any sufficiently large series length N and window length L. The term "finite rank" also has a meaning for the case of infinite series. For a generic infinite times series (cut at some N), $L = L(N)$ can tend to infinity and in this case the rank of the trajectory matrix would typically tend to infinity too. For a time series of finite rank, the rank r of the trajectory matrix is finite and fixed for large enough N and L such that $r \le \min(L, N - L + 1)$.

It is useful to know rank of a time series and the form of the singular vectors of its trajectory matrix, since knowing rank means knowing the number of the SVD components, which we should group for extraction of the corresponding series component, while the form of the singular vectors along with properties of eigenvalues helps in finding these SVD components.

We refer to Golyandina et al. (2001; Chapter 5) for details and full description. Here we mention that an exponential series $s_n = Ae^{\alpha n}$, $n = 1, 2, \ldots$, has rank 1, a linear function $s_n = an+b$, $a \neq 0$, has rank 2, a sinusoid with $s_n = A\sin(2\pi\omega n+\phi)$ has rank 2 for $0 < \omega < 0.5$ and rank 1 for $\omega = 0.5$. Singular vectors of trajectory matrices of time series have the same form as the series itself, which follows from the fact that rows and columns of the trajectory matrices are subseries of the original series. This information helps in choosing the groups at Grouping step.

2.1.2.2 Linear Recurrence Relations, Characteristic Polynomials and Roots

Time series of finite rank are closely related to the series governed by linear recurrence relations (LRRs). In particular, for infinite time series, the class of time series governed by LRRs coincides with the class of time series of finite rank.

Definition 2.1 A time series $\mathbb{S}_N = (s_i)_{i=1}^N$ is governed by an LRR, if there exist $a_1, \ldots, a_t$ such that

$$s_{i+t} = \sum_{k=1}^{t} a_k s_{i+t-k}, \ 1 \le i \le N-t, \ a_t \ne 0, \ t < N-1. \tag{2.6}$$

The number t is called the order of the LRR and $a_1, \ldots, a_t$ are the coefficients of the LRR. If $t = r$ is the minimal order of an LRR that governs the time series S_N, then the corresponding LRR is called minimal.

The minimal LRR is unique and its order is equal to the series rank.

As was mentioned in Sect. 1.4, it is well known that the time series $\mathbb{S}_\infty = (s_1, \ldots, s_n, \ldots)$ satisfies the LRR (2.6) for all $i \ge 0$ if and only if

$$s_n = \sum_{m=1}^{p} \left(\sum_{j=0}^{k_m-1} c_{mj} n^j \right) \mu_m^n, \tag{2.7}$$

where the complex coefficients c_{mj} depend on the first t points $s_1, \ldots, s_t$.

For real-valued time series, (2.7) implies that the class of time series governed by the LRRs consists of sums of products of polynomials, exponentials, and sinusoids

$$s_n = \sum_{m=1}^{\widetilde{p}} \left(\sum_{j=0}^{k_m-1} \widetilde{c}_{mj} n^j \right) e^{\alpha_m n} \cos(2\pi\omega_m n + \phi_m), \quad 0 \le \omega_m \le 0.5. \tag{2.8}$$

The minimal LRR determines all, except c_{mj}, parameters in (2.7) and all, except $\widetilde{c}_{mj}$ and ϕ_m, parameters in (2.8).

Definition 2.2 The polynomial $P_t(\mu) = \mu^t - \sum_{k=1}^{t} a_k \mu^{t-k}$ is called *characteristic polynomial* of the LRR (2.6).

Roots of the characteristic polynomial are called *characteristic roots* of the corresponding LRR. The roots of the characteristic polynomial of the minimal LRR governing the series, which can be called *signal roots of the LRR* or *characteristic roots of the series*, determine the values of parameters μ_m and k_m in (2.7) as follows. Let the time series $\mathbb{S}_\infty = (s_1, \ldots, s_n, \ldots)$ satisfy the LRR (2.6) with $a_t \ne 0$ and $i \ge 1$. Consider the characteristic polynomial of the LRR (2.6) and denote its different (complex) roots by $\mu_1, \ldots, \mu_p$, where $p \le t$. All these roots are

non-zero as $a_t \neq 0$ with k_m being the multiplicity of the root μ_m ($1 \leq m \leq p$, $k_1 + \ldots + k_p = t$).

Let $c_{k_p-1,j} \neq 0$ for all j; this corresponds to the case of the minimal LRR. Then the rank of time series $\mathbb{S}_\infty$ given in (2.7) is equal to $r = \sum_{m=1}^{p} k_m$. In the real-valued case, if $\widetilde{c}_{k_{\widetilde{p}}-1,j} \neq 0$ for all j, then the rank of time series $\mathbb{S}_\infty$ given in (2.8) is equal to $r = \sum_{m=1}^{\widetilde{p}} \delta_m k_m$, where $\delta_m = 1$ for ω_m equal 0 or 0.5 and $\delta_m = 2$ for $0 < \omega_m < 0.5$.

If we find the signal roots $\mu_m = \rho_m e^{\pm \mathrm{i} 2\pi \omega_m}$ of the characteristic polynomial of the LRR governing the signal, then we can estimate the signal parameters. For example, the frequency ω_m of an exponentially-modulated sinusoid can be found using the argument of the corresponding conjugate roots, whereas the root modulus ρ_m gives the exponential rate $\alpha_m = \ln \rho_m$.

2.1.3 Separability and Choice of Parameters

Understanding separability is very important for understanding how SSA works. Recall that if two time series $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$ are separable, then $\mathbb{X}_N^{(1)}$ can be extracted from the observed series $\mathbb{X}_N = \mathbb{X}_N^{(1)} + \mathbb{X}_N^{(2)}$. This means that there exists a partition into groups at Grouping step such that $\widetilde{\mathbb{X}}_N^{(m)} = \mathbb{X}_N^{(m)}$.

Let us define the separability formally. Let $\mathbf{X}^{(m)}$ be the trajectory matrices of the considered series components, $\mathbf{X}^{(m)} = \sum_{i=1}^{d_m} \sqrt{\lambda_{m,i}} U_{m,i} V_{m,i}^{\mathrm{T}}$, $m = 1, 2$, be their SVDs. The column and row spaces of the trajectory matrices are called *column* and *row trajectory spaces* correspondingly.

Definition 2.3 Let L be fixed. Two series $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$ are called *weakly separable* (or simply *separable*) if their column trajectory spaces are orthogonal and the same is valid for their row trajectory spaces; that is, $(\mathbf{X}^{(1)})^{\mathrm{T}} \mathbf{X}^{(2)} = \mathbf{0}_{K,K}$ and $\mathbf{X}^{(1)} (\mathbf{X}^{(2)})^{\mathrm{T}} = \mathbf{0}_{L,L}$.

Definition 2.4 Two series $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$ are called *strongly separable*, if they are weakly separable and the sets of singular values of their L-trajectory matrices are disjoint; that is, $\lambda_{1,i} \neq \lambda_{2,j}$ for any i and j.

Weak separability means that at Decomposition step there exists such an SVD that allows the proper grouping. A possibility of a non-separating SVD expansion which does not allow a proper grouping is related to the non-uniqueness of the SVD in the case of equal singular values. Strong separability means that any SVD of the trajectory matrix admits the proper grouping. Therefore, in order to be sure that SSA makes an accurate separation we have to require strong (approximate) separability.

By the definition, weak separability means orthogonality of the column and row spaces of the trajectory matrices of the series components $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$. For approximate (asymptotic) separability with $\widetilde{\mathbb{X}}_N^{(m)} \approx \mathbb{X}_N^{(m)}$, we need the condition of

approximate (asymptotic) orthogonality of subseries of the considered components. Asymptotic separability is considered if L and/or K tend to infinity.

For sufficiently long time series, SSA can approximately separate, for example, signal and noise, sine waves with different frequencies, trend and seasonality (Golyandina et al. 2001; Chapter 6, Section 1.5; Golyandina and Zhigljavsky 2013; Section 2.3.3).

Let us demonstrate the separability of two sinusoids with different frequencies ω_1 and ω_2: $x_n^{(i)} = A_i \cos(2\pi\omega_i n + \phi_i)$. These sinusoids are asymptotically weakly separable; that is, their subseries are asymptotically orthogonal as their lengths tend to infinity. However, the rate of convergence depends on the difference between the frequencies. If the frequencies are close and the time series length is not long enough, the two series can be far from orthogonal and therefore not separable. Note that two sinusoids with equal amplitudes are asymptotically weakly separable, but not strongly asymptotically separable and therefore are mixed in the SSA decompositions.

2.1.3.1 Separability Measure

The so-called **w**-correlation matrix contains very helpful information that can be used for detection of separability and identification of groups. This matrix consists of weighted cosines of angles between the reconstructed time series components.

Let w_n ($n = 1, 2, \ldots, N$) be the weights defined in (2.2): w_n is equal to the number of times the series element x_n appears in the trajectory matrix. Define the **w**-scalar product of time series of length N as $(\mathbb{Y}_N, \mathbb{Z}_N)_{\mathbf{w}} = \sum_{n=1}^{N} w_n y_n z_n = \langle \mathbf{Y}, \mathbf{Z} \rangle_{\mathrm{F}}$, where $\mathbf{Y}$ and $\mathbf{Z}$ are the L-trajectory matrices of the series $\mathbb{Y}_N$ and $\mathbb{Z}_N$, respectively. Define the so-called **w**-correlation between $\mathbb{Y}_N$ and $\mathbb{Z}_N$ as

$$\rho_{\mathbf{w}}(\mathbb{Y}_N, \mathbb{Z}_N) = (\mathbb{Y}_N, \mathbb{Z}_N)_{\mathbf{w}} / (\|\mathbb{Y}_N\|_{\mathbf{w}} \|\mathbb{Z}_N\|_{\mathbf{w}}).$$

Well-separated components in (2.5) have weak (or zero) correlation whereas poorly separated components typically have high correlation. Therefore, looking at the matrix of **w**-correlations between elementary reconstructed series $\widetilde{\mathbb{X}}_N^{(i)}$ and $\widetilde{\mathbb{X}}_N^{(j)}$ one can find groups of correlated series components and use this information for the subsequent grouping. One of the main rules is: "do not include highly correlated components into different groups." The **w**-correlations can also be used for checking the grouped decomposition.

It is very instructive to depict the matrix of absolute values of **w**-correlations between the series components graphically in the white-black scale, where small correlations are shown in white and correlations with their absolute values close to 1 are shown in black; see, for example, Figs. 2.4 and 2.15.

2.1.3.2 Choice of Parameters

The conditions of (approximate) separability yield recommendations for the choice of the window length L: it should be large enough ($L \sim N/2$) and if we want to extract a periodic component with known period, then the window lengths, which are divisible by the period, provide better separability. Choice of parameters is discussed in Golyandina et al. (2001; Section 1.6) and Golyandina (2010). If we choose a few leading eigentriples, then SSA with small L performs smoothing of the series as a filter of order $2L-1$, see Golyandina and Zhigljavsky (2013; Section 3.9). Generally, the choice of the window length is important but the result is usually stable with respect to small changes in the values of L.

If the time series has a complex structure, then the so-called Sequential SSA (Golyandina et al. 2012a; Section 2.5.5) is recommended. Sequential SSA consists of two stages; at the first stage, trend is extracted with a small window length and then periodic components are detected and extracted from the residual with $L \sim N/2$.

2.1.3.3 Justification

If we use SSA as a model-free and exploratory technique, then the justification of the decomposition cannot be formal; it must be based on the separability theory and the interpretability of the results. Real-time or batch processing by SSA is possible if the class of series is not too broad and well-determined so that one can fix the rule for choosing proper parameters. For performing statistical testing, a model of the time series should be specified.

2.1.4 Algorithm

In Sect. 2.1.1 we described the Basic SSA method. Here we formally present the algorithm of Basic SSA. Note that the Rssa package implements the algorithms efficiently (see Sect. 1.5.4 for a brief discussion). Since effective implementation is complicated and hides the sense of algorithm steps, we write down the algorithms in the original form.

Input data for the whole algorithm of Basic SSA are the window length and the way of grouping of the elementary components $\mathbf{X}_i$ in (2.3). However, the rule for grouping is made after the decomposition (2.3) is made. Therefore, the grouping becomes the input data for Reconstruction stage. For this reason, we split the algorithm into two parts. Note that modifications of Basic SSA mostly differ by Decomposition step only; Reconstruction stage is the same for virtually all SSA versions.

Algorithm 2.1 Basic SSA: decomposition

Input: Time series $\mathbb{X}$ of length N, window length L.
Output: Decomposition of the trajectory matrix on elementary matrices $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where $d = \operatorname{rank} \mathbf{X}$ and $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$ $(i = 1, \ldots, d)$.
1: Construct the trajectory matrix $\mathbf{X} = \mathfrak{T}_{\mathrm{SSA}}(\mathbb{X})$.
2: Compute the SVD $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.

Reconstruction algorithms are almost the same for different versions of SSA; their inputs have a decomposition of the trajectory matrix into a sum of rank-one matrices and the split of the rank-one components into groups. We therefore formulate a general algorithm of reconstruction and will make comments concerning specific features of modifications in the corresponding sections. The specific feature of Basic SSA is: the input decomposition is the SVD and hence the biorthogonal decomposition into the rank-one components is ordered according to component contribution $\sigma_i^2 = \lambda_i$ so that $\sigma_1 \geq \ldots \geq \sigma_d$.

Algorithm 2.2 Reconstruction

Input: Decomposition $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where $\mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}}$ and $\|U_i\| = \|V_i\| = 1$; partition of indices: $\{1, \ldots, d\} = \bigsqcup_{j=1}^{m} I_j$.
Output: Decomposition of the time series $\mathbb{X}$ into identifiable components $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$.
1: Construct the grouped matrix decomposition $\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_m}$, where $\mathbf{X}_I = \sum_{i \in I} \mathbf{X}_i$.
2: Compute $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$, where $\mathbb{X}_i = \mathfrak{T}_{\mathrm{SSA}}^{-1} \circ \Pi_{\mathcal{H}}(\mathbf{X}_{I_i})$.

2.1.5 Basic SSA in RSSA

2.1.5.1 Description of Functions

The main function of RSSA is `ssa` which constructs the so-called `ssa` object holding the decomposition and various auxiliary information necessary for performing a particular implementation. The function has many arguments; some of them are common for different types of SSA, some are specific. Below we will outline the main arguments of `ssa` in a typical function call:

```
s <- ssa(x, L = (N + 1) %/% 2, neig = NULL,
         kind = "1d-ssa", svd.method = "auto")
```

where `N` is the series length.

Arguments:

`x` is an object to be decomposed. For Basic SSA it is assumed to be a simple vector or vector-like object (e.g., univariate `ts` or `zooreg` object). Everything else is coerced to a vector.

`L` is a window length. By default it is fixed to half of the series length.
`neig` is the number of desired eigentriples. If `neig = NULL`, a default value, which depends on L and N, will be used.
`kind` specifies the version of SSA to be used; it can be omitted in non-ambiguous cases (e.g., when `x` is a vector or a `ts` object).
`svd.method` selects the SVD method to use. Full description is given in Sect. 2.1.5.2.

In addition to constructing an `ssa` object `s`, by default the `ssa` function also performs Decomposition step and thus corresponds to Algorithm 2.1. If necessary, Decomposition stage can be skipped setting the argument `force.decompose` to `FALSE`.

The function returns an `ssa` object. The precise layout of the object is hidden and can be different in different versions of the package. However, there are several fields that are available to users and can be extracted with the help of `$` operator, namely:

`s$sigma` is a vector of singular values;
`s$U` is a matrix of eigenvectors;
`s$V` is a matrix of factor vectors. Note that it may not be calculated for particular selections of the SVD method.

The number of the calculated singular values, eigenvectors, and factor vectors can be obtained by means of the functions `nsigma(s)`, `nu(s)`, and `nv(s)` correspondingly. Call of `summary(s)` provides a consolidated information about the `ssa` object.

The next function is `reconstruct` which implements Reconstruction stage (Algorithm 2.2). The basic signature of the function call is

```
r <- reconstruct(s, groups = list(trend = 1:2, c(3:6,9)))
```

Arguments:

`s` is an `ssa` object holding the decomposition.
`groups` is a list of numeric vectors consisting of indices of the elementary components used for reconstruction; the entries of the list can be named.
`drop` acts only if one group is chosen; `TRUE` value means that the result is transformed from the list of the reconstructed series to the reconstructed series itself (`FALSE` is default).

The function returns a list of reconstructed objects. Elements of the list have the same names as elements of `groups` (e.g., `r$trend`). If a group is unnamed, then the corresponding component will obtain the name `Fn`, where `n` is its index in the `groups` list (e.g. `r$F2`).

By default, the routine tries to preserve all the attributes of the input object. In this way, for example, the reconstruction result of the `ts` object is the `ts` object with the same time scale. This feature can be disabled by setting the argument `drop.attributes` to `TRUE`.

2.1.5.2 SVD Methods

In many cases only few leading eigentriples are of interest for the SSA analysis. Thus the full SVD of the trajectory matrix can yield large computational and memory space burdens. Instead, the so-called Truncated SVD can be used and only a number of desired leading eigentriples can be computed. Four different SVD implementations are available in RSSA and can be specified via the argument `svd.method` of the function `ssa`:

- `"auto"`—Automatic method of selection depending on the series length, the window length, and the number of desired eigentriples.
- `"nutrlan"`—Truncated SVD via thick-restart Lanczos bidiagonalization algorithm (Yamazaki et al. 2008). The method internally calculates the eigenvalues and eigenvectors of the matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$. Factor vectors are calculated on-fly during Reconstruction stage when necessary.
- `"propack"`—Implicitly restarted Lanczos bidiagonalization with partial reorthogonalization (Larsen 1998). The method calculates the truncated SVD of the trajectory matrix $\mathbf{X}$ (and hence calculates the factor vectors as well).
- `"eigen"` and `"svd"`—Full decomposition of the trajectory matrix using either eigendecomposition or SVD routines from LAPACK (Anderson et al. 1999). Using `ssa` with these `svd.methods` yields the straightforward implementations of Basic SSA algorithm without computational and space complexity reductions via additional sophisticated algorithms. Note that both methods perform full decompositions and thus the argument `neig` (which allows one to request a desired number of eigentriples) is silently ignored for these methods.

Selecting the best method for performing the SVD is not easy. However, there are several simple rules of thumb which work well in most situations.

First of all, it is unwise to use the Lanczos-based truncated SVD methods if the trajectory matrix is small or "wide." This corresponds to small series lengths (say, $N < 100$) or small window lengths ($L < 50$). Also, it is unwise to ask for too many eigentriples: when more than $L/2$ eigentriples are needed then it is better to use the full SVD instead of a truncated one. The SVD method `eigen` works best for small L.

Usually the method `propack` tends to be slightly faster and more numerically stable than `nutrlan`; however, it may yield considerable memory consumption when factor vectors are large. For example, for a time series of length 87000 and window length 43500, the decomposition with the method `nutrlan` took 16 s while with `propack` it took only 13 s (we are not aware of any other implementation of SVD, besides RSSA implementations, which can perform the decomposition with such a large window length at all). The memory consumption for the latter method is twice higher than the consumption of the former. This difference is more important for multivariate versions of SSA and should not be a problem in the 1D case.

A specific feature of the Lanczos-based truncated SVD methods is their possible non-convergence in the case of coinciding eigenvalues. In real-life time series, the exact coincidence of eigenvalues happens very rarely and hence we can often enjoy

the outstanding effectiveness of these SVD methods. For a time series of finite rank r, zero eigenvalue has the multiplicity $L - r$; therefore, the number of the truncated components should be chosen appropriately, e.g. $r + 1$ components can be requested for calculation. Since for a time series of finite rank r even a small window length $L \simeq r + 1$ can be sufficient for the analysis and forecasting, the use of the `eigen` method is recommended.

By default, the method `nutrlan` is selected. However, the function `ssa` tries to correct the selection, when the chosen method is clearly not the most suitable. In particular, for short series, small window lengths or large number of desired eigentriples, the method `eigen` is automatically selected.

It should be noted that the truncated SVD implementations were extracted from the RSSA package into a separate package SVD (Korobeynikov et al. 2016) and thus can be used independently.

2.1.5.3 Typical Code

For demonstration, we consider the series of sales of fortified wines (shortly "FORT") taken from the dataset "Australian Wines" (monthly wine sales in thousands of liters). The full dataset contains sales from January, 1980, to July, 1995 (187 points). However, the data after June, 1994 have missing values. Therefore, we analyze the first 174 points.

Fragment 2.1.1 contains the standard code for loading the package RSSA and for the input of the data included into the package.

Fragment 2.1.1 ("Australian Wines": Input)

```
> library("Rssa")
> data("AustralianWine", package = "Rssa")
> wine <- window(AustralianWine, end = time(AustralianWine)[174])
```

Fragment 2.1.2 contains a typical code for extraction of the trend and seasonality. The resultant decomposition is depicted in Fig. 2.1.

Fragment 2.1.2 ("FORT": Reconstruction)

```
> fort <- wine[, "Fortified"]
> s.fort <- ssa(fort, L = 84, kind = "1d-ssa")
> r.fort <- reconstruct(s.fort,
+                       groups = list(Trend = 1,
+                                     Seasonality = 2:11))
> plot(r.fort, add.residuals = TRUE, add.original = TRUE,
+      plot.method = "xyplot",
+      superpose = TRUE, auto.key = list(columns = 2))
```

Roughly speaking (see details in Golyandina and Korobeynikov (2013)), `ssa` performs Steps 1 and 2 of the algorithmic scheme described in Sect. 1.1.1, while `reconstruct` performs steps 3 and 4 of the algorithm. The argument values `kind = "1d-ssa"` and `svd.method = "auto"` are default and can be omitted. The

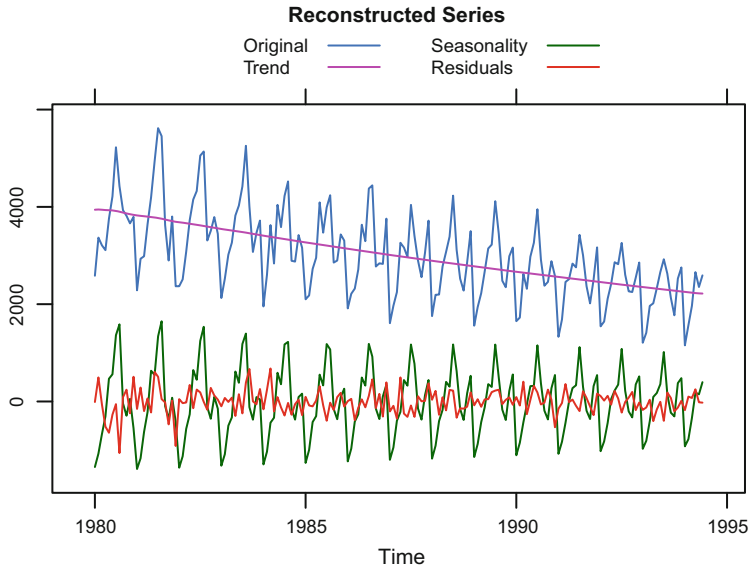

Fig. 2.1 "FORT": Decomposition

function `plot` applied to the reconstruction object performs different special kinds of plotting. In addition to specific parameters, this function can include parameters of the function `xyplot` from the standard package LATTICE (see the last two parameters of `plot` in Fragment 2.1.2).

The choice of groups for reconstruction was made on the base of the following information obtained from the `ssa` object:

1. one-dimensional (1D) figures of the eigenvectors U_i (Fig. 2.2),
2. two-dimensional (2D) figures of the eigenvectors (U_i, U_{i+1}) (Fig. 2.3), and
3. matrix of **w**-correlations $\rho_{\mathbf{w}}$ between the elementary reconstructed series (functions `wcor` and `plot`, Fig. 2.4).

The following fragment shows the code that reproduces Figs. 2.2–2.5.

Fragment 2.1.3 ("FORT": Identification)

```
> plot(s.fort, type = "vectors", idx = 1:8)
> plot(s.fort, type = "paired", idx = 2:11, plot.contrib = FALSE)
> print(parestimate(s.fort, groups = list(2:3, 4:5),
+                   method = "pairs"))
$F1
   period      rate   |     Mod     Arg   |      Re         Im
   11.971   0.000000  |  1.00000    0.52  |  0.86540    0.50109
$F2
   period      rate   |     Mod     Arg   |      Re         Im
    4.005   0.000000  |  1.00000    1.57  |  0.00177    1.00000
> plot(wcor(s.fort, groups = 1:30),
```

```
+            scales = list(at = c(10, 20, 30)))
> plot(reconstruct(s.fort, groups = list(G12 = 2:3, G4 = 4:5,
+                                        G6 = 6:7, G2.4 = 8:9)),
+      plot.method = "xyplot", layout = c(2, 2),
+      add.residuals = FALSE, add.original = FALSE)
```

Let us explain how the figures obtained by means of Fragment 2.1.3 can help to perform the grouping. Figure 2.2 shows that the first eigenvector is slowly-varying and therefore the eigentriple ET1 should be included into the trend group. Figure 2.3 shows that the pairs 2–3, 4–5, 6–7, 8–9, 10–11 are produced by modulated sine-waves, since the corresponding 2D-scatterplots of eigenvectors resemble regular

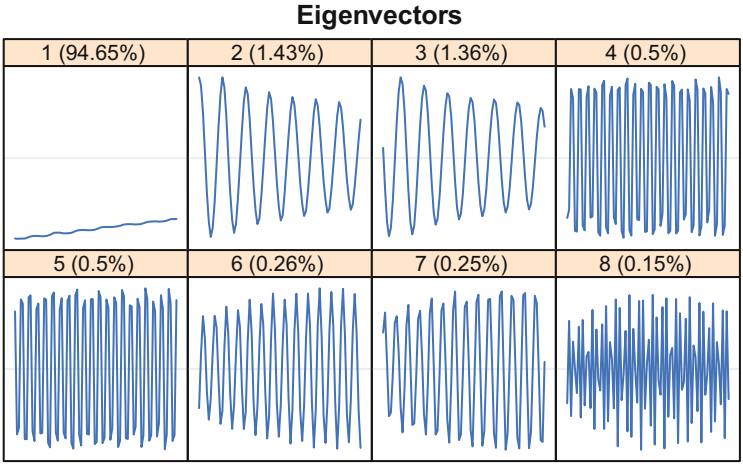

Fig. 2.2 "FORT": 1D graphs of eigenvectors

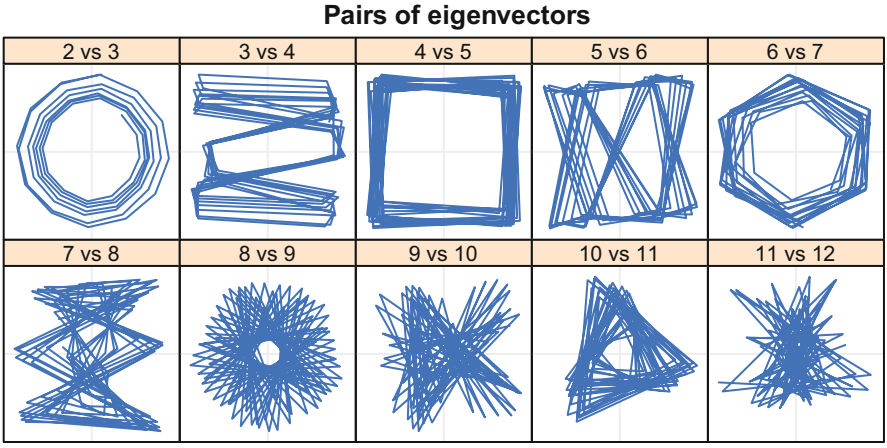

Fig. 2.3 "FORT": 2D scatterplots of eigenvectors

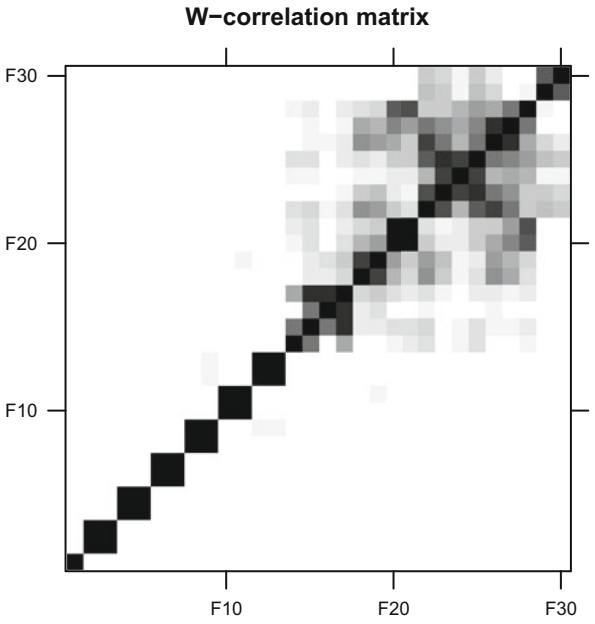

Fig. 2.4 "FORT": Weighted correlations

polygons. This way of identification is based on the following properties: a sine wave has rank 2 and produces two eigentriples, which are sine waves with the same frequency and have a phase shift exactly or approximately equal to $\pi/2$, due to the orthogonality of eigenvectors.

By counting the numbers of polygon vertices in Fig. 2.3, the periods of the sine-waves can be determined as 12, 4, 6, 2.4, 3. Alternatively, automatic methods of frequency calculation can be employed, such as LS-ESPRIT and TLS-ESPRIT methods (Roy and Kailath 1989). These methods are implemented in RSSA in the function `parestimate`, see Sect. 3.1, and are described in Golyandina et al. (2001; Sections 2.4.2.4. and 3.8.2) and Golyandina and Korobeynikov (2013) for one-dimensional time series. The periods, calculated by the automatic `parestimate` method in Fragment 2.1.3, agree with the numbers of vertices in Fig. 2.3 for the five pairs listed.

The matrix of absolute values of **w**-correlations in Fig. 2.4 is depicted in grayscale (white color corresponds to zero and black color corresponds to the absolute values equal to 1). Figure 2.4 confirms that the indicated pairs are separated between themselves and also from the trend component, since the **w**-correlations between the pairs are small, while **w**-correlations between the components from the same pair are very large. The block of 12–84 components is "gray," therefore we can expect that these components are mixed and are largely produced by noise.

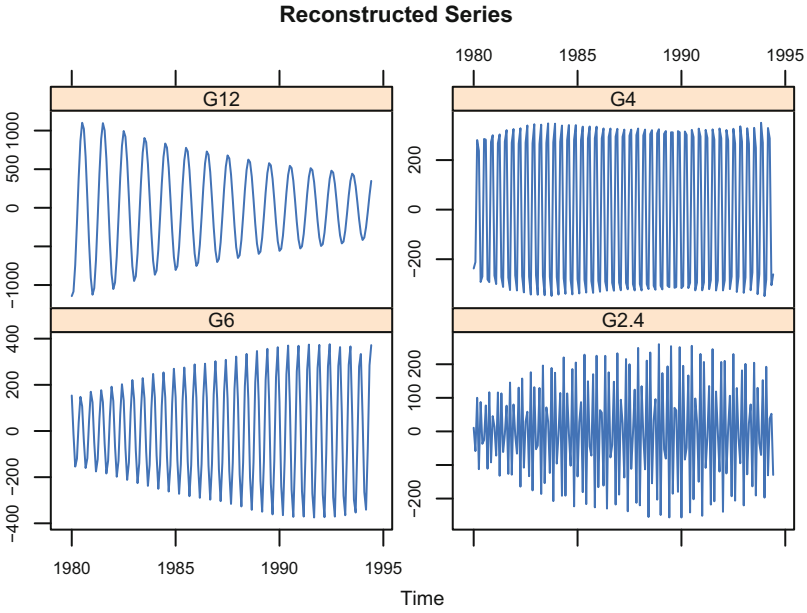

Fig. 2.5 "FORT": Reconstructed sine waves

Figure 2.5 contains four reconstructed modulated sine waves and shows that several sine waves have increasing amplitudes, while others are decreasing; the same can be seen in Fig. 2.2. In Fig. 2.1, we grouped the modulated sine waves and obtained the seasonal component with varying annual behavior.

2.2 Toeplitz SSA

2.2.1 Method

Toeplitz SSA differs from Basic SSA only in Step 2 of the generic scheme presented in Fig. 1.1; that is, in the decomposition of $\mathbf{X}$ into rank-one matrices. Basic SSA uses the SVD at this step with U_i calculated as eigenvectors of $\mathbf{S} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$. If the length N of the series $\mathbb{X}$ is not large and the series is assumed to be stationary, then the usual recommendation is to replace the matrix $\mathbf{S}$ by some other matrix, which is constructed under the assumption of stationarity.

Note first that in Basic SSA we can consider the *lag-covariance matrix* $\mathbf{C} = \mathbf{S}/K$ instead of $\mathbf{S}$ for obtaining the SVD of the trajectory matrix $\mathbf{X}$. Indeed, the eigenvectors of the matrices $\mathbf{S}$ and $\mathbf{C}$ are the same and the eigenvalues differ only by the factor $1/K$.

Denote by $c_{ij} = c_{ij}(N)$ the elements of the lag-covariance matrix $\mathbf{C}$. If the time series is stationary with zero mean, L is fixed and $K \to \infty$, then $\lim c_{ij}(N) = R_{\mathbb{X}}(|i-j|)$ as $N \to \infty$, where $R_{\mathbb{X}}(k)$ stands for the lag-k term of the time series autocovariance function. We can therefore define a Toeplitz version of the lag-covariance matrix by putting equal values $\widetilde{c}_{ij}$ at each matrix auxiliary diagonal $|i-j| = k$. The most natural way for defining the values $\widetilde{c}_{ij}$ and the corresponding matrix $\widetilde{\mathbf{C}}$ is to compute

$$\widetilde{c}_{ij} = \frac{1}{N-|i-j|} \sum_{m=1}^{N-|i-j|} x_m x_{m+|i-j|}, \quad 1 \le i, j \le L. \tag{2.9}$$

While using this formula it is usually assumed that the time series $\mathbb{X}$ is centered so that the mean $\bar{x} = \sum_{i=1}^{N} x_i/N$ is subtracted from all $x_i \in \mathbb{X}$.

Let $L \le K$ and denote by P_i $(i = 1, \dots, L)$ the eigenvectors of $\widetilde{\mathbf{C}}$; these vectors form an orthonormal basis of R^L. Then the decomposition on elementary matrices can be written as $\mathbf{X} = \sum_{i=1}^{L} P_i(\mathbf{X}^{\mathrm{T}} P_i)^{\mathrm{T}}$. Ordering of addends is performed by the magnitudes of $\sigma_i = \|\mathbf{X}^{\mathrm{T}} P_i\|$. Note that this ordering generally differs from the ordering of eigenvalues of the matrix $\widetilde{\mathbf{C}}$ corresponding to the eigenvectors P_i. Some of these eigenvalues could even be negative as the matrix $\widetilde{\mathbf{C}}$ is not necessarily positive definite.

If the original series is stationary with zero mean, then the use of *Toeplitz lag-covariance matrix* $\widetilde{\mathbf{C}}$ can be more appropriate than the use of the lag-covariance matrix $\mathbf{C}$. On the other hand, Toeplitz SSA is not suitable for nonstationary series; if the original series has an influential nonstationary component, then Basic SSA works better than Toeplitz SSA. For example, if we are dealing with a pure exponential series, then it is described by a single eigentriple for any window length, while Toeplitz SSA produces L eigentriples for the window length L; moreover, the eigenvectors in Toeplitz SSA have some special properties (Andrew 1973), which do not depend on the series. The same effect takes place for the linear series, exponential-cosine series, etc.

A number of papers devoted to SSA analysis of climatic time series (e.g., Ghil et al. (2002), where Toeplitz SSA is often referred to as a VG method) consider Toeplitz SSA as the main version of SSA and state that the difference between the Basic and Toeplitz versions of SSA is marginal. However, using the Toeplitz version of SSA is unsafe if the series contains a trend or oscillations with increasing or decreasing amplitude. Examples of effects observed when Toeplitz SSA is applied to non-stationary time series are presented in Golyandina (2010). For the study of theoretical properties of Toeplitz SSA, see, for example, Harris and Yan (2010).

2.2.2 Algorithm

Algorithm 2.3 Toeplitz SSA: decomposition

Input: Time series $\mathbb{X}$ of length N, window length L.
Output: Decomposition of the trajectory matrix on elementary matrices $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_L$, where $\mathbf{X}_i = \sigma_i P_i Q_i^{\mathrm{T}}$, $\|P_i\| = \|Q_i\| = 1$.
1: Construct the trajectory matrix $\mathbf{X} = \mathcal{T}_{\mathrm{SSA}}(\mathbb{X})$.
2: Obtain the decomposition

$$\mathbf{X} = \sum_{i=1}^{L} \sigma_i P_i Q_i^{\mathrm{T}} = \mathbf{X}_1 + \ldots + \mathbf{X}_L, \tag{2.10}$$

where $\{P_i\}_{i=1}^{L}$ are eigenvectors of the matrix $\widetilde{\mathbf{C}}$ with entries computed by (2.9), $S_i = \mathbf{X}^{\mathrm{T}} P_i$, $Q_i = S_i / \|S_i\|$ and $\sigma_i = \|\mathbf{X}_i\|_{\mathrm{F}} = \|S_i\|$. Components are ordered by the magnitudes of σ_i: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_L$.

The reconstruction algorithm is exactly the same as Algorithm 2.2 with vectors (P_i, Q_i) substituted for (U_i, V_i).

2.2.3 *Toeplitz SSA in* Rssa

2.2.3.1 Description of Functions

In Rssa, Toeplitz SSA is implemented via the same `ssa` function as Basic SSA. One should use `kind="toeplitz-ssa"` to enable the Toeplitz version. All other arguments have the meaning as described in Sect. 2.1.5:

```
s <- ssa(x, L = (N + 1) %/% 2, neig = NULL,
         kind = "toeplitz-ssa", svd.method = "auto")
```

where `N` is the series length.

Note that the triples (σ_i, P_i, Q_i), which are generated by the decomposition (2.10), are also called eigentriples in Rssa and the access to $\{P_i\}$ and $\{Q_i\}$ is provided by the codes `s$U` and `s$V`.

2.2.3.2 Typical Code

To our mind, Toeplitz SSA has a limited range of applications, since it requires stationarity for both signal and noise, which is first unnatural and second impossible to verify. Hence, in the example below we use simulated data (Fragment 2.2.1).

As mentioned above, before using Toeplitz SSA it is recommended to center the series. Then Toeplitz SSA can be used in exactly the same way as Basic SSA.

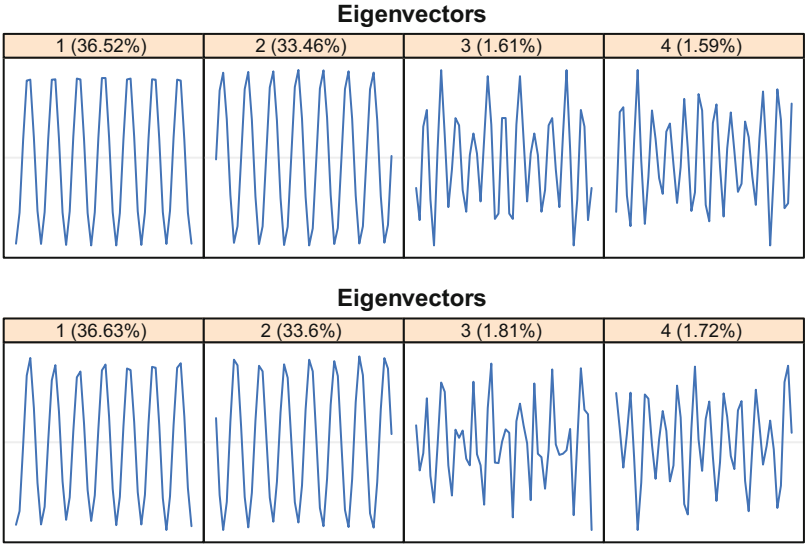

Fig. 2.6 Noisy sinusoid: 1D graphs of eigenvectors (top: Toeplitz SSA, bottom: Basic SSA)

Fragment 2.2.1 (Noisy Sinusoid: Toeplitz SSA)

```
> N <- 100
> sigma <- 0.5
> set.seed(1)
> F <- sin (2 * pi * (1:N) / 7) + sigma * rnorm(N)
> Fcenter <- F - mean(F)
> st <- ssa(Fcenter, L = 50, kind = "toeplitz-ssa")
> s <- ssa(F, L = 50, kind = "1d-ssa")
> p <- plot(s, type = "vectors", idx = 1:4, layout = c(4, 1))
> pt <- plot(st, type = "vectors", idx = 1:4, layout = c(4, 1))
> plot(pt, split = c(1, 1, 1, 2), more = TRUE)
> plot(p, split = c(1, 2, 1, 2), more = FALSE)
> pt <- plot(reconstruct(st, groups = list(1:2)),
+            plot.method = "xyplot", layout = c(3, 1))
> p <- plot(reconstruct(s, groups = list(1:2)),
+           plot.method = "xyplot", layout = c(3, 1))
> plot(pt, split = c(1, 1, 1, 2), more = TRUE)
> plot(p, split = c(1, 2, 1, 2), more = FALSE)
```

Here we see that for Toeplitz SSA the amplitude of the sinusoid reconstruction is closer to a constant than that for Basic SSA (Figs. 2.6 and 2.7). Generally, eigenvectors for Toeplitz SSA are more regular, even for the noise decomposition. This is due to the properties of eigenvectors of Toeplitz matrices (Fig. 2.7).

Note that typically the window length for Toeplitz SSA should be rather small, since the used estimate of auto-covariance matrix of the series tends to the true auto-covariance matrix only if L is fixed and K tends to infinity.

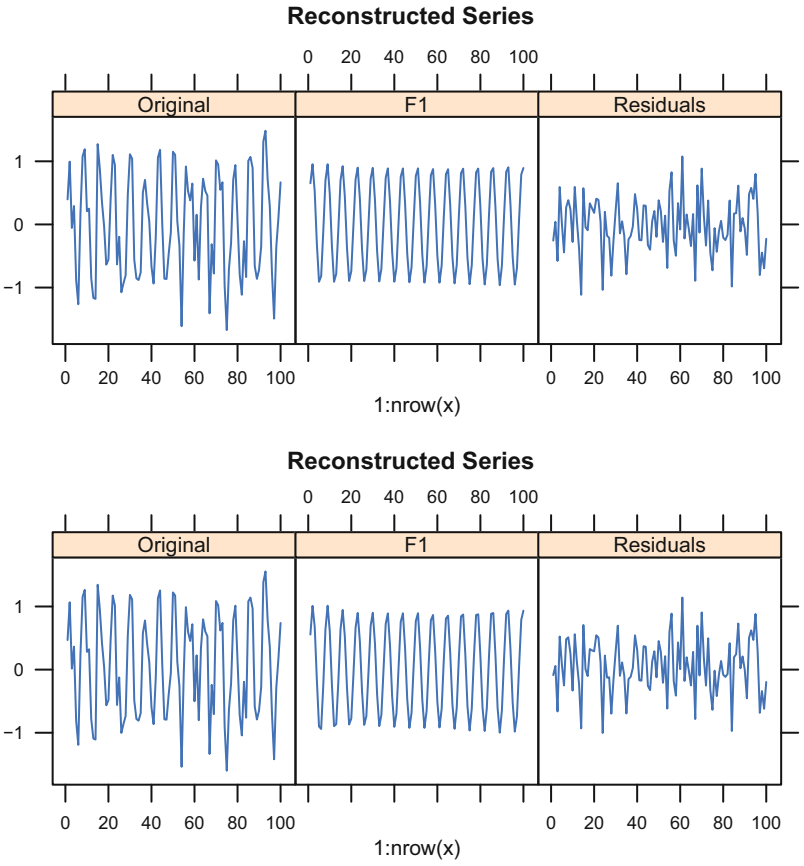

Fig. 2.7 Noisy sinusoid: Reconstruction (top: Toeplitz SSA, bottom: Basic SSA)

2.2.3.3 Simulated Example

As was mentioned above, for stationary time series the use of Toeplitz SSA is appropriate, while it makes no sense to apply Toeplitz SSA for trend extraction. Also, if a periodic component (e.g., a seasonal behavior) is changing in time, the accuracy of signal reconstruction is worse than that for Basic SSA.

Let us demonstrate this by means of simulation. We consider the signal in the form $s_n = \exp(\alpha n)\sin(2\pi n/7), n = 1, \ldots, 100$, and the noisy series $x_n = s_n + \sigma\varepsilon_n$, where $\sigma = 0.5$, ε_n is white Gaussian noise.

For $\alpha = 0$ this series can be considered as stationary with stationary deterministic signal (see definition in Golyandina et al. (2001; Sections 1.7.2 and 6.4)). For non-zero α, this series is not stationary. Thus, let us consider α from [0, 0.01].

Fragment 2.2.2 (Simulation: Comparison of Toeplitz and Basic SSA)

```
> SIMUL <- FALSE
> N <- 100
> sigma <- 0.5
> set.seed(1)
> alpha <- seq(0.0, 0.01, 0.001)
> L <- 50
> Q <- 1000
> if (SIMUL) {
+   RMSE <-
+     sapply(alpha,
+            function(a) {
+              sqrt(rowMeans(replicate(Q, {
+                S <- exp(a * (1:N)) * sin(2 * pi * (1:N) / 7)
+                F <- S + sigma * rnorm(N)
+                Fcenter <- F - mean(F)
+                st <- ssa(Fcenter, L = L, kind = "toeplitz-ssa")
+                s <- ssa(F, L = L, kind = "1d-ssa")
+                rec <- reconstruct(s, groups = list(1:2))$F1
+                rec.t <- reconstruct(st, groups = list(1:2))$F1
+                c("1d-ssa" = mean((rec - S)^2),
+                  "toeplitz" = mean((rec.t - S)^2))
+              })))
+            })
+
+   toeplitz.sim <- as.data.frame(t(RMSE))
+ } else {
+   data("toeplitz.sim", package = "ssabook")
+ }
> matplot(alpha, toeplitz.sim, type = "l", ylim = c(0, 0.25))
```

Figure 2.8 shows the dependence of the reconstruction accuracy on the exponential rate α constructed with the help of the code from Fragment 2.2.2. The window length $L = 50$ was chosen and RMSE was taken as a measure of accuracy. One can see that the accuracy of Basic SSA reconstruction does not depend on α, while the error of Toeplitz SSA increases as α increases. If a series is very close to a stationary series, Toeplitz SSA has a slightly smaller error than Basic SSA. However, already for $\alpha = 0.01$ Toeplitz SSA makes considerably larger errors than Basic SSA.

2.3 SSA with Projection

2.3.1 Method

As was mentioned in Sect. 1.2.1.1, the goal of SSA with projection is an efficient use of a known information about series components. The well-known methods of SSA with centering and SSA with double centering for extraction of constant and linear trends, respectively, are special cases of SSA with projection.

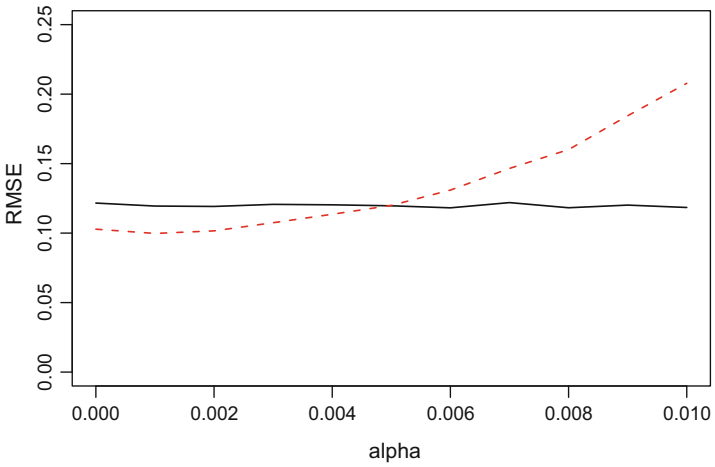

Fig. 2.8 Simulation: Reconstruction error of Toeplitz (dash red line) and Basic SSA (solid black line)

Let $\mathbb{X}$ be a time series of length N, L be the window length, $K = N-L+1$, and $\mathbf{X}$ be the trajectory matrix. The general form of centering can be expressed as follows:

1. Calculation of a special matrix $\mathbf{C}^{(\mathrm{center})} = \mathbf{C}(\mathbf{X})$ based on a priori information.
2. Computation of $\mathbf{X}^\star = \mathbf{X} - \mathbf{C}^{(\mathrm{center})}$.
3. Construction of the SVD: $\mathbf{X}^\star = \sum_{i=1}^{d^\star} \sqrt{\lambda_i^\star} U_i^\star (V_i^\star)^{\mathrm{T}}$.

As a result, we obtain the decomposition $\mathbf{X} = \mathbf{C}^{(\mathrm{center})} + \sum_{i=1}^{d^\star} \sqrt{\lambda_i^\star} U_i^\star (V_i^\star)^{\mathrm{T}}$.

Denote $E_M = (1, \ldots, 1)^{\mathrm{T}} \in \mathsf{R}^M$ the M-vector of ones. The following three types of centering can be considered (Golyandina et al. 2001; Sections 1.7.1 and 6.3):

- *Single row centering* when $\mathbf{C}_{\mathrm{row}}^{(\mathrm{center})}(\mathbf{X}) = (\mathbf{X}E_K/K)E_K^{\mathrm{T}}$ corresponds to averaging by rows; that is, each element of a row of $\mathbf{C}_{\mathrm{row}}^{(\mathrm{center})}$ consists of the average of the corresponding row of the trajectory matrix.
- *Single column centering* when $\mathbf{C}_{\mathrm{col}}^{(\mathrm{center})}(\mathbf{X}) = E_L(\mathbf{X}^{\mathrm{T}}E_L/L)^{\mathrm{T}}$ corresponds to averaging by columns.
- *Double centering* when $\mathbf{C}_{\mathrm{both}}^{(\mathrm{center})} = \mathbf{C}_{\mathrm{row}}^{(\mathrm{center})} + \mathbf{C}_{\mathrm{col}}^{(\mathrm{center})}\big(\mathbf{X} - \mathbf{C}_{\mathrm{row}}^{(\mathrm{center})}(\mathbf{X})\big)$ corresponds to averaging by both rows and columns.

Note that the single centering can be considered as a projection of rows or columns of $\mathbf{X}$ on $\mathrm{span}(E_K)$ or $\mathrm{span}(E_L)$, respectively, since $E_K E_K^{\mathrm{T}}$ and $E_L E_L^{\mathrm{T}}$ are exactly the matrices of the projection operators. Therefore, centering in SSA can be considered as a preliminary projection of the trajectory matrix on a given subspace; the residual matrix will be subsequently expanded by the SVD or any other decomposition.

Let us generalize this approach to projections to arbitrary spaces following Golyandina and Shlemov (2017). Let $\Pi_{\mathrm{col}} : \mathsf{R}^L \to \mathcal{L}_{\mathrm{col}}$ and $\Pi_{\mathrm{row}} : \mathsf{R}^K \to \mathcal{L}_{\mathrm{row}}$ be orthogonal projectors, where $\mathcal{L}_{\mathrm{col}} \in \mathsf{R}^L$ is called the column projection space and $\mathcal{L}_{\mathrm{row}} \in \mathsf{R}^K$ is called the row projection space. For any $\mathbf{Y} \in \mathsf{R}^{L\times t}$, denote $\Pi_{\mathrm{col}}(\mathbf{Y})$ the matrix consisting of the columns, which result from projections of the columns of $\mathbf{Y}$. Similarly, for any $\mathbf{Y} \in \mathsf{R}^{t\times K}$, denote $\Pi_{\mathrm{row}}(\mathbf{Y})$ the matrix consisting of the rows, which result from projections of the rows of $\mathbf{Y}$.

Denote a basis of the column projection space $\{P_i, i = 1, \dots, p\}$ and a basis of the row projection space $\{Q_i, i = 1, \dots, q\}$. Let $\mathbf{P} = [P_1 : \dots : P_p]$ and $\mathbf{Q} = [Q_1 : \dots : Q_q]$. Without loss of generality we assume that $\{P_i, i = 1, \dots, p\}$ and $\{Q_i, i = 1, \dots, q\}$ are orthonormal bases of $\mathcal{L}_{\mathrm{col}}$ and $\mathcal{L}_{\mathrm{row}}$ (otherwise, we can perform ortho-normalization).

In SSA with projection, the scheme of SSA with centering is extended to arbitrary projections; that is, $\mathbf{C} = \Pi_{\mathrm{col}}(\mathbf{X})$ for the column projection, $\mathbf{C} = \Pi_{\mathrm{row}}(\mathbf{X})$ for the row projection and $\mathbf{C} = \Pi_{\mathrm{both}}(\mathbf{X})$ for the double projection, where

$$\begin{aligned}\Pi_{\mathrm{both}}(\mathbf{X}) &= \Pi_{\mathrm{row}}(\mathbf{X}) + \Pi_{\mathrm{col}}(\mathbf{X} - \Pi_{\mathrm{row}}(\mathbf{X})) \\ &= \Pi_{\mathrm{col}}(\mathbf{X}) + \Pi_{\mathrm{row}}(\mathbf{X} - \Pi_{\mathrm{col}}(\mathbf{X})) \\ &= \Pi_{\mathrm{row}}(\mathbf{X}) + \Pi_{\mathrm{row}}(\mathbf{X}) - (\Pi_{\mathrm{col}} \circ \Pi_{\mathrm{row}})(\mathbf{X}). \end{aligned} \tag{2.11}$$

If either the column or row basis is absent (that is, the space for column or row projection consists of zero), then we formally set the corresponding projector to be zero operator implying $\mathbf{C} = \Pi_{\mathrm{both}}(\mathbf{X})$ in any mode.

A general form of the decomposition provided by SSA with projection is

$$\mathbf{X} = \mathbf{C} + \sum_{i=1}^{d^\star} \sqrt{\lambda_i^\star} U_i^\star (V_i^\star)^{\mathrm{T}}, \tag{2.12}$$

where $\mathbf{C} = \Pi_{\mathrm{both}}(\mathbf{X})$ and $\sum_{i=1}^{d^\star} \sqrt{\lambda_i^\star} U_i^\star (V_i^\star)^{\mathrm{T}}$ is the SVD of $\mathbf{X}^\star = \mathbf{X} - \mathbf{C}$.

It is shown in Golyandina and Shlemov (2017) that (2.12) can be represented as a sum of elementary matrices of rank 1. The matrix $\mathbf{C}$ can be considered as a sum of $q + p$ elementary matrices of the forms $\sigma_i^{(r)} \widetilde{P}_i Q_i^{\mathrm{T}}$, $i = 1, \dots, q$, and $\sigma_i^{(c)} P_i \widetilde{Q}_i^{\mathrm{T}}$, $i = 1, \dots, p$. The triples $(\sigma_i^{(r)}, \widetilde{P}_i, Q_i)$ and $(\sigma_i^{(c)}, P_i, \widetilde{Q}_i)$ have the same meaning as eigentriples. For double projection, this representation depends on the order of projections; we will apply the row projector first. Therefore, the decomposition (2.12) can be transformed into a decomposition of $\mathbf{X}$ into a sum of $q + p + d^\star$ elementary rank-one matrices, which are orthogonal with respect to the Frobenius norm $\|\cdot\|$, by construction. As a consequence, the contribution of the projection term $\mathbf{C}$ into the decomposition is measured by $\|\mathbf{C}\|^2/\|\mathbf{X}\|^2$.

Thus, (2.12) is a decomposition of $\mathbf{X}$ on elementary matrix components unambiguously defined. Reconstruction stage is exactly the same as in the Basic SSA method. Note that it has little sense to include the eigentriples produced by

projections to different groups, since the projections are performed on the subspaces as a whole. It follows from Golyandina and Shlemov (2017; Lemma 1) that the rank $d^\star$ of the matrix $\mathbf{X}^\star$ obtained after projection cannot be larger than the rank of the original matrix $\mathbf{X}$ and can not be smaller than $\operatorname{rank}\mathbf{X} - (q + p)$.

When we use projections, we should expect some specific form for one of the series component. For example, to extract a sine wave using projections, we should know its period, and to extract exponential trend, we should know its rate. These conditions are often too restrictive. A clear exception is extraction of the polynomial trends, when we should assume only the degree of the polynomial to define its trajectory space.

Note finally that SSA with projection can be applied in the shaped version, when the series has gaps.

2.3.2 Appropriate Time Series

For SSA with projection, a known series component with a trajectory matrix $\mathbf{Y}$ should be in agreement with projections so that $\Pi_{\mathrm{col}}(\mathbf{Y}) = \mathbf{Y}$ for the column projection, $\Pi_{\mathrm{row}}(\mathbf{Y}) = \mathbf{Y}$ for the row projection, and $\Pi_{\mathrm{both}}(\mathbf{Y}) = \mathbf{Y}$ for the double projection.

Clearly, for column and row projections, this is true if the corresponding projection is performed on the column or row trajectory space of the known series component. For example, the trajectory space of an exponential component $s_n = \mu^n$ spans $(1, \mu, \dots, \mu^L)^{\mathrm{T}}$, while the trajectory space of the linear function $s_n = an + b$ spans $(1, 1, \dots, 1)^{\mathrm{T}}$ and $(1, 2, \dots, L)^{\mathrm{T}}$ for any a and b.

Let us introduce a condition sufficient for $\Pi_{\mathrm{both}}(\mathbf{X}) = \mathbf{X}$ to hold for the general case of the double projection.

Recall that a series governed by an LRR, whose characteristic polynomial has the given set of roots called characteristic roots, is of the form (1.9).

Theorem 2.1 (Golyandina and Shlemov (2017)) *Let series $\mathbb{Y}^{(m)}$, $m = 1, 2$, be governed by minimal LRRs of orders r_m, $\mathbf{Y}^{(m)}$ be their trajectory matrices. Denote $\{\mu_j;\ j = 1, \dots, s\}$ the set containing the characteristic roots of both series. Assume that $\mathbb{Y}^{(m)}$, $m = 1, 2$, have the signal roots μ_j, $j = 1, \dots, s$, with multiplicities $d_j^{(m)} \ge 0$, $\sum_{j=1}^{s} d_j^{(m)} = r_m$. Let Π_{col} be the projector on the column space of $\mathbf{Y}^{(1)}$, Π_{row} be the projector on the row space of $\mathbf{Y}^{(2)}$, Π_{both} be given in (2.11). Then $\Pi_{\mathrm{both}}(\mathbf{X}) = \mathbf{X}$ if and only if the set of characteristic roots of the series $\mathbb{X}$ consists of the roots μ_j, $j = 1, \dots, s$, of multiplicities $d_j \le d_j^{(1)} + d_j^{(2)}$.*

Corollary 2.1 *Let $\mathbb{Y}$ be a series of dimension r, $\mathbf{Y}$ be its trajectory matrix, Π_{row} be the projector on its row trajectory space, Π_{col} be the projector on its column trajectory space. Consider the series $\mathbb{X}$ with $x_n = (an + b)y_n$. Then $\Pi_{\mathrm{both}}(\mathbf{X}) = \mathbf{X}$.*

Remark 2.1 Note that multiplication by $an + b$ means that the multiplicities of the characteristic roots increase by 1.

Since for polynomial trends of a degree k there is the unique characteristic root equal to 1 of multiplicity $k+1$ (Golyandina et al. 2001; Example 5.3) and we should assume only the degree of the polynomial trend to obtain its trajectory space, this case is of particular interest.

Corollary 2.2 *Let Π_{row} be the projection on the row trajectory space of the polynomial of order m, Π_{col} be the projection on the column trajectory space of the polynomial of order k. Then for the polynomial $\mathbb{X} = P_{m+k+1}$ of order $m+k+1$ we have $\Pi_{\mathrm{both}}(\mathbf{X}) = \mathbf{X}$.*

It immediately follows from the projection definition that in the conditions of Corollary 2.2, for any polynomial $\mathbb{X} = P_m$ of degree m we have $\Pi_{\mathrm{row}}(\mathbf{X}) = \mathbf{X}$ and for any polynomial $\mathbb{X} = P_k$ of degree k we have $\Pi_{\mathrm{col}}(\mathbf{X}) = \mathbf{X}$.

2.3.3 Separability

We can expect that if the bases of the spaces to be projected to are chosen properly (for example, if an LRR governing a time series component is known), then SSA with projection improves the resultant decomposition, in comparison with Basic SSA.

Using the notion of separability, we can formulate this improvement as follows. Let $\mathbb{X} = \mathbb{X}^{(1)} + \mathbb{X}^{(2)}$. We will say that a time series component $\mathbb{X}^{(1)}$ is separated by SSA with projection if its trajectory matrix $\mathbf{X}^{(1)}$ coincides with $\mathbf{C}$, where $\mathbf{C}$ is as in (2.12). Therefore, separability by SSA with projection means that the series component $\mathbb{X}^{(1)}$ can be reconstructed by projecting components of the matrix decomposition (2.12) only.

Let $\mathbb{X}^{(1)}$ be a series of finite rank, $\mathbb{X} = \mathbb{X}^{(1)} + \mathbb{X}^{(2)}$. Similar to Golyandina et al. (2001; Sections 1.7.1 and 6.3), where conditions for separability by SSA with centering are considered, the following conditions of separability can be obtained:

1. Basic SSA: $\mathbb{X}^{(1)}$ and $\mathbb{X}^{(2)}$ are separable if (if and only if, by the definition) their row and column spaces are orthogonal.
2. SSA with row projection on the row space of $\mathbb{X}^{(1)}$: $\mathbb{X}^{(1)}$ and $\mathbb{X}^{(2)}$ are separable if their row spaces are orthogonal.
3. SSA with column projection on the column space of $\mathbb{X}^{(1)}$: $\mathbb{X}^{(1)}$ and $\mathbb{X}^{(2)}$ are separable if their column spaces are orthogonal.
4. SSA with double projection on the row and column space of $\mathbb{Y}$, where $\mathbb{X}^{(1)}$ and $\mathbb{Y}$ are such that $x_n^{(1)} = (an+b)y_n$, $a \neq 0$: $\mathbb{X}^{(1)}$ and $\mathbb{X}^{(2)}$ are separable by SSA with double projection if $\mathbb{Y}$ and $\mathbb{X}^{(2)}$ are separable by Basic SSA.

For an approximate separability $\mathbf{X}^{(1)} \approx \mathbf{C}$, we need an approximate orthogonality. Also, an asymptotic separability and the rate of convergence can be considered by analogy with the conventional separability for Basic SSA and SSA with centering.

Recall that the usual double centering in SSA corresponds to a constant series $\mathbb{Y}$ and therefore a linear series $\mathbb{X}^{(1)}$. Orthogonality to a constant series is a much weaker condition than that to a linear series. Therefore, for extraction of a linear trend the use of double centering is recommended.

We can expect that in the case of a polynomial trend, SSA with double centering can work better than SSA with row or column centering and also than Basic SSA.

Also, the separability conditions imply that for extraction of polynomial trends double projection can be used for better separability and therefore the result can be more accurate than the ordinary least-squares polynomial regression provides; see Golyandina et al. (2001; Section 1.7.1) and Golyandina and Shlemov (2017) containing comparison with least-squares regression estimates.

It is important that separability of SSA with projection, if it takes place, is always strong, since the elementary components, which are produced by the projection, precede the SVD components, by construction of the decomposition.

2.3.4 Algorithm

Algorithm 2.4 SSA with projection: decomposition

Input: Time series $\mathbb{X}$ of length N, window length L, orthonormal basis of the column projection space $\{P_i, i = 1, \dots, p\}$ and orthonormal basis of the row projection space $\{Q_i, i = 1, \dots, q\}$. Either p or q can be zero.

Output: Decomposition of the trajectory matrix on elementary matrices $\mathbf{X} = \mathbf{X}_1 + \dots + \mathbf{X}_d$, where $\mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}}$ are either zero or rank-one matrices.

1: Construct the trajectory matrix $\mathbf{X} = \mathfrak{T}_{\mathrm{SSA}}(\mathbb{X})$.

2: Subtract the row projection: $\mathbf{X}' = \mathbf{X} - \mathbf{C}_{\mathrm{row}}$, where

$$\mathbf{C}_{\mathrm{row}} = \Pi_{\mathrm{row}}(\mathbf{X}) = \sum_{i=1}^{q} \sigma_i^{(r)} \widetilde{P}_i Q_i^{\mathrm{T}},$$

$\sigma_i^{(r)} = \|\mathbf{X} Q_i\|$, $\widetilde{P}_i = \mathbf{X} Q_i / \sigma_i^{(r)}$ if $\sigma_i^{(r)} > 0$; otherwise, $\widetilde{P}_i$ is the zero vector.

3: Subtract the column projection: $\mathbf{X}^\star = \mathbf{X}' - \mathbf{C}_{\mathrm{col}}$, where

$$\mathbf{C}_{\mathrm{col}} = \Pi_{\mathrm{col}}(\mathbf{X}') = \sum_{i=1}^{p} \sigma_i^{(c)} P_i \widetilde{Q}_i^{\mathrm{T}},$$

$\sigma_i^{(c)} = \|\mathbf{X}'^{\mathrm{T}} P_i\|$, $\widetilde{Q}_i = \mathbf{X}'^{\mathrm{T}} P_i / \sigma_i^{(c)}$ if $\sigma_i^{(c)} > 0$; otherwise, $\widetilde{Q}_i$ is the zero vector.

4: Construct an SVD of the matrix $\mathbf{X}^\star$: $\mathbf{X}^\star = \sum_{i=1}^{d^\star} \mathbf{X}_i^\star$, where $\mathbf{X}_i^\star = \sqrt{\lambda_i^\star} U_i^\star (V_i^\star)^{\mathrm{T}}$.

5: As a result, $\mathbf{X} = \sum_{i=1}^{d} \mathbf{X}_i$, where $d = q + p + d^\star$, $\mathbf{X}_i = \sigma_i^{(r)} \widetilde{P}_i Q_i^{\mathrm{T}}$ for $i = 1, \dots, q$, $\mathbf{X}_{i+q} = \sigma_i^{(c)} P_i \widetilde{Q}_i^{\mathrm{T}}$ for $i = 1, \dots, p$, and $\mathbf{X}_{i+q+p} = \sqrt{\lambda_i^\star} U_i^\star (V_i^\star)^{\mathrm{T}}$ for $i = 1, \dots, d^\star$.

Algorithm 2.5 SSA with projection: reconstruction

Input: Decomposition $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, $\mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}}$, number q of row-projection components, number p of column-projection components, grouping $\{1, \ldots, d\} = \bigsqcup_{j=1}^{m} I_j$, which does not split the first $q + p$ projection components.

Output: Decomposition of the time series on identifiable components $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$.

1: Construct the grouped matrix decomposition $\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_m}$, where $\mathbf{X}_I = \sum_{i \in I} \mathbf{X}_i$.

2: Compute $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$, where $\mathbb{X}_i = \mathfrak{T}_{\mathrm{SSA}}^{-1} \circ \Pi_{\mathcal{H}}(\mathbf{X}_{I_i})$.

The only essential difference with the reconstruction by Basic SSA is that the set of the matrices $\mathbf{X}_i$, $i = 1, \ldots, q + p$, produced by projections, should be included to the same group.

2.3.5 SSA with Projection in RSSA

2.3.5.1 Description of Functions

In RSSA, Basic SSA with projection is a special case of Basic SSA, hence `ssa` function should be used with additional arguments that would specify column and row projection bases. The meaning of all other arguments is the same as described in Sect. 2.1.5. A typical call is as follows:

```
s <- ssa(x, L = (N + 1) %/% 2, neig = NULL,
         kind = "1d-ssa", svd.method = "auto",
         column.projector = "centering",
         row.projector = "centering")
```

where `N` is the series length.

Arguments:

`column.projector`, `row.projector` Each may be a matrix of orthonormal basis of the projection subspace, or a single integer, which will be interpreted as the dimension of the orthogonal polynomial basis (note that the dimension equals to the degree of the basis plus 1, e.g. the quadratic basis has dimension 3), or one of following character strings: `"none"`, `"constant"` (or `"centering"`), `"linear"`, `"quadratic,"` or `"cubic"` for orthonormal bases of the corresponding polynomial series.

The `ssa` call when both projectors are set to `"none"` corresponds to ordinary Basic SSA, `column.projector = "centering"` (or, the same, `column.projector=1`) corresponds to Basic SSA with centering, `column.projector = "centering"` and `row.projector = "centering"` corresponds to Basic SSA with double centering. The mode `kind = "toeplitz-ssa"` is unavailable for any choice of projections.

Note that the special triples generated by projections are included into the whole set of triples produced by the adaptive decomposition used. In RSSA, all the triples

are named eigentriples for uniformity. The first `nspecial(s)` triples correspond to projections. Hence, reconstruction by `groups = list(1:nspecial(n))` corresponds to reconstruction of the projection term. For example, for double centering mode, one obtains a linear trend estimation.

2.3.5.2 Typical Code

Let us consider the example "CO2" (Mauna Loa Atmospheric CO_2 Concentration). It seems that for such kind of time series, many methods can yield very similar results. Basic SSA provides very natural way for trend extraction, it is demonstrated in Golyandina and Korobeynikov (2013), where the choice $L = 120$, $ET1, 4$ was considered.

Fragment 2.3.1 demonstrates the code, which allows to perform reconstruction by means of SSA with projection.

Fragment 2.3.1 ("CO2": SSA with Projection)

```
> s2 <- ssa(co2, column.projector = "centering",
+           row.projector = "centering")
> plot(reconstruct(s2, groups =
+                    list(Linear.trend = seq_len(nspecial(s2)))),
+      add.residuals = FALSE, plot.method = "matplot")
> s4 <- ssa(co2, column.projector = 2, row.projector = 2)
> plot(reconstruct(s4, groups =
+                    list(Trend = seq_len(nspecial(s4)))),
+      add.residuals = FALSE, plot.method = "matplot")
> plot(s4, type = "vectors", idx = 1:12)
> r <- reconstruct(s4,
+                  groups =
+                     list(Signal = c(seq_len(nspecial(s4)), 5:8)))
> plot(r, plot.method = "xyplot")
```

We start with extraction of linear trend and therefore choose `column.projector = "centering"`, `row.projector = "centering"` to perform SSA with double centering. Note that the same choice of projectors can be achieved by setting `column.projector` and `row.projector` equal to 1, where 1 is the dimension of the trajectory space of a polynomial series of degree 0; that is, of a constant series. Recall that the choice `column.projector = p`, `row.projector = q` corresponds to extraction of a polynomial trend of degree $p + q - 1$. To select all the projection components in the decomposition `s2` for extraction, we set the trend group consisting of the first `nspecial(s2)` components. The extracted trend is close to linear, see Fig. 2.9. Certainly, the accurate trend of "CO2" series is not linear.

To extract a more accurate trend, let us choose other subspaces for projections, `column.projector = 2` and `row.projector = 2`, to extract a trend, which is close to a cubic polynomial. Figure 2.10 shows that the extracted trend is quite accurate. This trend is very similar to that in Golyandina and Korobeynikov (2013), which was extracted by Basic SSA. Note that in this example the trend

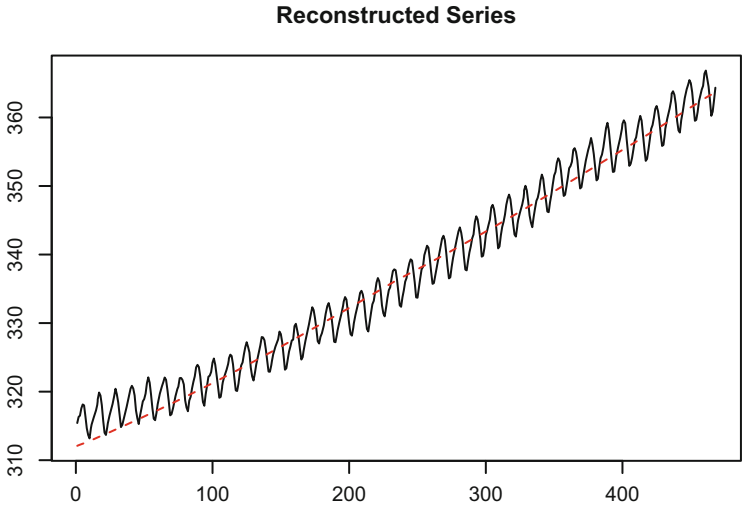

Fig. 2.9 "CO2": Reconstruction of linear trend

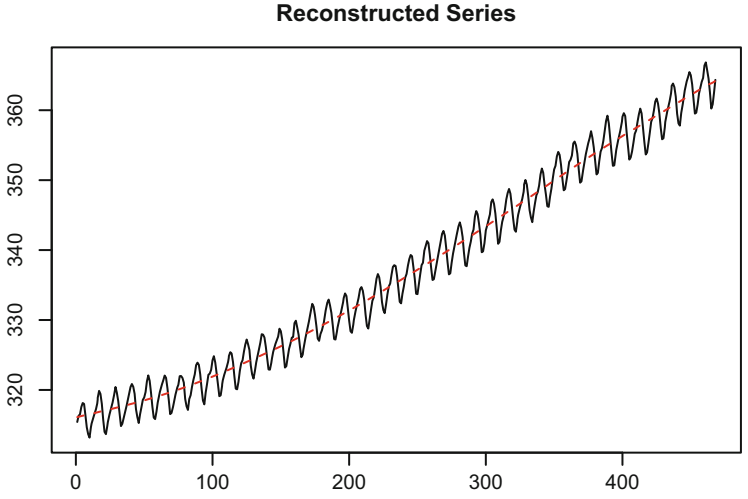

Fig. 2.10 "CO2": Reconstruction of the cubic trend

is approximated by a sum of two exponentials in Basic SSA (ET1,4 for $L = 120$) and a polynomial of order 3 in SSA with projection; both approximations have four parameters and achieve similar accuracy.

Note that SSA with projection allows us to extract not only a trend but also other kinds of series components, similar to what Basic SSA does. Figure 2.11 presents graphs of eigenvectors for the "CO2" example. The first two components contain two vectors produced by projecting rows on the row projection space and the next

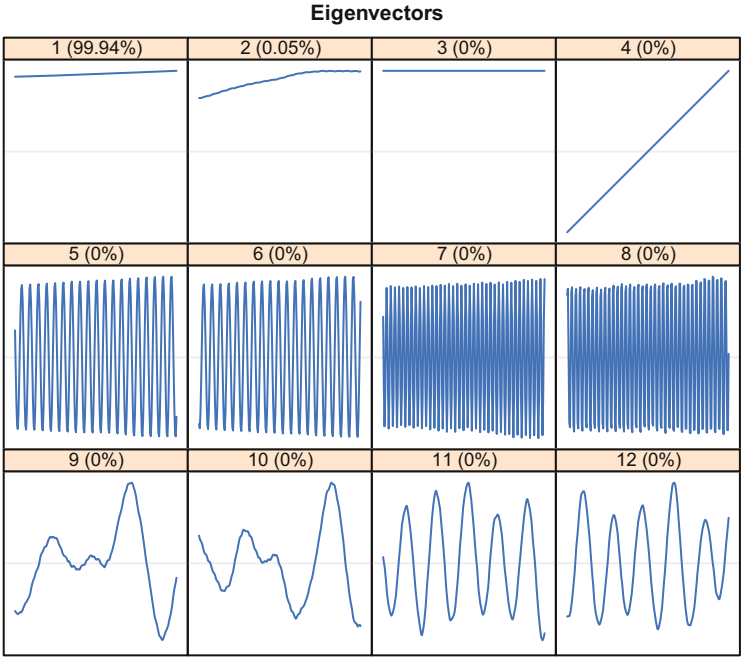

Fig. 2.11 "CO2": 1D graphs of eigenvectors

two components contain a basis of the column space (two linear functions here). Other graphs show singular vectors of the matrix $\mathbf{X}''$ obtained by subtraction of the projection matrices. Choice of the trend components ET1–4 and the seasonality components ET5–8 leads to the signal extraction depicted in Fig. 2.12.

2.3.5.3 Simulated Examples: Polynomial Regression

Here we consider an example showing the difference between SSA with projection and the least-squares parametric regression for polynomial trend extraction. Let us take a polynomial trend $t_n = 10(n/N - 0.5)^5$ of order 5, $x_n = t_n + \sin(2\pi n/10)$, where $N = 199$ and $n = 1, \ldots, N$.

Projections, which keep a polynomial of degree 5, can be composed in different ways. It can be purely either column or row projection on the 6-dimensional polynomial trajectory space. Also, a double projection can be considered. For example, we can take both row and column projections on the row and column trajectory spaces of a polynomial of degree 2 (and of dimension 3). By Corollary 2.2, this double projection with $k = m = 2$ keeps polynomial of degree 5.

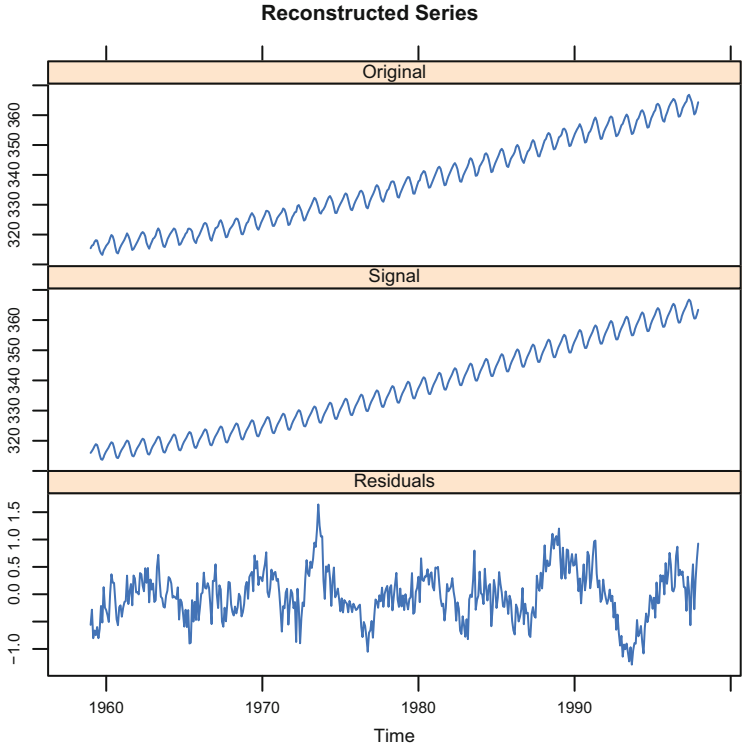

Fig. 2.12 "CO2": Reconstruction of signal

SSA with projection for neither choice provides approximate separability of the polynomial trend from a sinusoid. However, in view of the separability conditions we can expect that the choice $k = m = 2$ probably corresponds to the best accuracy.

Fragment 2.3.2 (Polynomial Trend: SSA with Projection)

```
> N <- 199
> tt <- (1:N) / N
> r <- 5
> F0 <- 10 * (tt - 0.5)^r
> F <- F0 + sin(2 * pi * (1:N) / 10)
> L <- 100
> dec <- ssa(F, L = L, column.projector = 3, row.projector = 3)
> rec1 <- reconstruct(dec, groups =
+                       list(Trend = seq_len(nspecial(dec))))
> fit1 <- rec1$Trend
> fit1_3b <- lm(fit1 ~ poly((1:N), r, raw = TRUE))
> fit3b <- lm(F ~ poly((1:N), r, raw = TRUE))
> li <- 1:199
> d <- data.frame(Initial = F[li],
+                 dproj = fit1[li],
```

```
+                   dproj_reg = predict(fit1_3b)[li],
+                   regr = predict(fit3b)[li], trend = F0[li])
> xyplot(as.formula(paste(paste(colnames(d), collapse = "+"),
+                           "~", "1:nrow(d)")),
+        data = d,
+        type = "l", ylab = "", xlab = "",
+        lty = c(1, 1, 1, 1, 1), lwd = c(1, 2, 2, 2, 2),
+        auto.key = list(columns = 3,
+                        lines = TRUE, points = FALSE))
```

We compare the following trend estimations (see Fragment 2.3.2). First, we set $L = 100$ and consider the trend obtained by double projection with $k = m = 2$ (this is called "dproj"). Then, we find the least-squares polynomial regression of order 5 for the initial series ("regr") and for "dproj" ("dproj_regr").

If L and K are divisible by the period, then the separability accuracy is better and the result is in a sense unbiased. Least-squares polynomial regression of order 5 does not estimate the polynomial trend in the meaning considered in this example as it minimizes the prediction mean square error, by the definition.

The results are presented in Fig. 2.13. SSA with double projection extracts the trend approximately with visible mixture with the sine-wave component. However, these oscillations are around the proper trend. Least-squares polynomial regression of order 5 applied to the result of double projection confirms it.

Least-squares parametric regression provides a poor estimator of trend in the considered example. For longer time series the difference is not so dramatic and the trend estimates are closer.

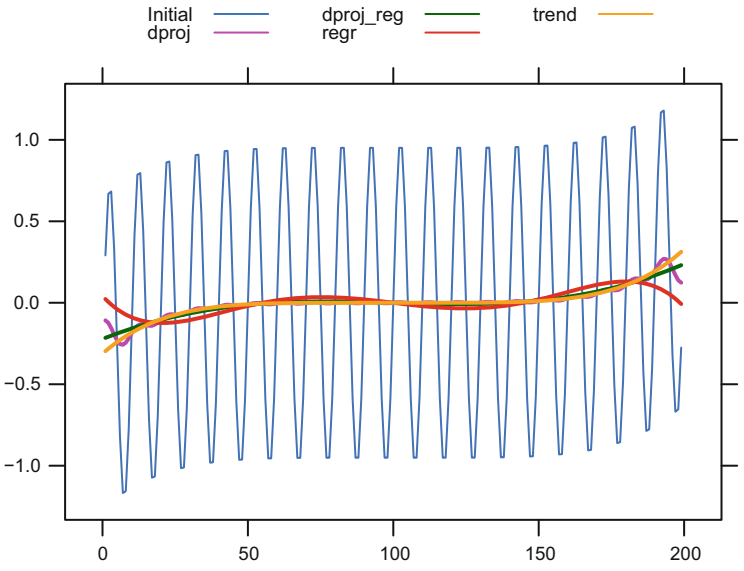

Fig. 2.13 Polynomial trend: Comparison of trend reconstructions

2.4 Iterative Oblique SSA

For reasonably long time series lengths and moderate noise levels, interpretable components such as trends, oscillations, and noise are approximately separable by Basic SSA (Golyandina et al. 2001; Sections 1.5 and 6.1). However, the conditions of approximate separability can be restrictive, especially, for short time series.

Orthogonality of subseries, which is the main condition for separability in Basic SSA, see Sect. 2.1.3, can be a strong limitation on the series which we want to separate. However, if we consider orthogonality with respect to a non-standard Euclidean inner product, conditions of separability are considerably weaker. This approach yields the method called Oblique SSA (O-SSA) with the SVD performed in a non-orthogonal coordinate system at Decomposition step. The idea of Oblique SSA is similar to that of prewhitening which is frequently used in statistics as preprocessing: if we know covariances between components, then we can perform linear transformation and obtain uncorrelated components. Since the "covariances" of the components are not known in advance, an iterative method called Iterative Oblique SSA can be used. Also, the method is able to change contributions of the components in a specific way so that their strong separability will most likely to be improved.

2.4.1 Method

2.4.1.1 Use of Oblique SVD

Although many interpretable series components like trend (a slowly varying component) and seasonality are asymptotically orthogonal, for a given time series length the orthogonality can be unreachable even approximately. Therefore, it would be helpful to weaken the orthogonality condition. Oblique SSA uses different orthogonality, which still means the equality of an inner product to 0, but this time a non-standard inner product is used; this inner product is adapted to the time series components, which we want to separate.

It is well-known that any inner product in the Euclidean space is associated with a symmetric positive-definite matrix $\mathbf{A}$ and is defined as $\langle X_1, X_2\rangle_{\mathbf{A}} = (\mathbf{A}X_1, X_2)$. The standard inner product corresponds to the use of the identity matrix as $\mathbf{A}$. The notion of inner product implies the notion of $\mathbf{A}$-orthogonality: two vectors X_1 and X_2 are $\mathbf{A}$-orthogonal if $\langle X_1, X_2\rangle_{\mathbf{A}} = 0$. If the matrix $\mathbf{A}$ is semi-definite, then it defines the inner product in its column space (also, in the row space which is the same in view of symmetry). While considering $\langle X_1, X_2\rangle_{\mathbf{A}}$, we will always assume that the vectors X_1 and X_2 belong to the column space of $\mathbf{A}$.

The non-standard Euclidean inner products induce such notions as oblique coordinate systems, orthogonality of vectors, which are oblique in the ordinary sense, and so on.

Let us consider an elementary example. Let $X = (1, 2)^{\mathrm{T}}$ and $Y = (1, 1)^{\mathrm{T}}$. These two vectors are not orthogonal in the usual sense as $(X, Y) = 3$. However, if we define

$$\mathbf{A} = \begin{pmatrix} 5 & -3 \\ -3 & 2 \end{pmatrix}, \tag{2.13}$$

then $\langle X, Y\rangle_{\mathbf{A}} = (\mathbf{A}X, Y) = 0$ and $(\mathbf{O_A}X, \mathbf{O_A}Y) = 0$ for any $\mathbf{O_A}$ such that $\mathbf{O_A}^{\mathrm{T}}\mathbf{O_A} = \mathbf{A}$, e.g.,

$$\mathbf{O_A} = \begin{pmatrix} 1 & -1 \\ -2 & 1 \end{pmatrix}.$$

This means that $\{X, Y\}$ is an orthogonal basis with respect to the $\mathbf{A}$-inner product $\langle\cdot,\cdot\rangle_{\mathbf{A}}$ and the matrix $\mathbf{O_A}$ defines the orthogonalizing map. The matrix $\mathbf{A}$ can be chosen in such a way that X and Y have any given $\mathbf{A}$-norms. The choice (2.13) corresponds to $\mathbf{A}$-orthonormality.

To describe Oblique SSA, let us introduce the SVD of a matrix $\mathbf{X}$ produced by two oblique bases, $\mathbf{L}$-orthonormal and $\mathbf{R}$-orthonormal correspondingly, in the row and column spaces.

Definition 2.5 We say that

$$\mathbf{X} = \sum_{i=1}^{d} \sigma_i P_i Q_i^{\mathrm{T}} \tag{2.14}$$

is the $(\mathbf{L}, \mathbf{R})$-SVD, if $\{P_i\}_{i=1}^{d}$ is an $\mathbf{L}$-orthonormal system and $\{Q_i\}_{i=1}^{d}$ is an $\mathbf{R}$-orthonormal system; that is, the decomposition is $(\mathbf{L}, \mathbf{R})$-biorthogonal.

This kind of SVD is called Restricted SVD (RSVD) given by the triple $(\mathbf{X}, \mathbf{L}, \mathbf{R})$, see De Moor and Golub (1991) for details. The mathematics related to inner products $\langle\cdot,\cdot\rangle_{\mathbf{A}}$ with positive-semidefinite matrix $\mathbf{A}$ and the corresponding RSVD is shortly described in Golyandina and Shlemov (2015; Appendix A) from the viewpoint of decompositions into a sum of elementary matrices.

Oblique SSA (O-SSA) is a modification of the Basic SSA method described in Sect. 2.1, where the standard SVD at Decomposition step is replaced by the $(\mathbf{L}, \mathbf{R})$-SVD for some matrices $\mathbf{L}$ and $\mathbf{R}$. We will use all the notions related to Basic SSA for this oblique modification.

If $\mathbf{L}$ and $\mathbf{R}$ are the identity matrices, then Oblique SSA coincides with Basic SSA, $\sigma_i = \sqrt{\lambda_i}$, $P_i = U_i$, and $Q_i = V_i$.

Computationally, oblique SVD is straightforwardly reduced to the ordinary SVD (see Golyandina and Shlemov (2015; Proposition 4)) and therefore its calculation does not require special numerical techniques, see Algorithm 2.6.

2.4.1.2 Nested Oblique SSA

Unlike the ordinary SVD, the SVD with respect to a non-orthogonal coordinate system provides a matrix approximation which does not have obvious approximation properties. This implies that Oblique SSA is not a good tool for extraction of the leading components, in particular, for extraction of the signal and for denoising.

Therefore, we suggest to use Oblique SSA in the nested way. The approach is somewhat similar to factor analysis, where a factor space can be estimated by principal component analysis and then interpretable factors are extracted from the factor space.

Suppose that in a particular application Basic SSA is able to extract the signal but is not able to separate the signal components. For example, let the time series consist of a noisy sum of two sinusoids with close frequencies. Then Basic SSA can perform the denoising but it is unlikely that it will be able to separate these sinusoids. Hence Basic SSA should only be used for estimation of the subspace of the sum of sinusoids and then some other method is advised to be employed for separating the sinusoids. Note that the nested approach is similar to the refined grouping used in Golyandina and Zhigljavsky (2013; Section 2.5.4) for the SSA-ICA and Golyandina and Lomtev (2016) for the SSA-AMUSE algorithms, which use ideas taken from independent component analysis.

Thus, let us apply Basic SSA with proper parameters and let a matrix decomposition $\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_p}$ be obtained at Grouping step of Basic SSA; each group corresponds to a separated time series component. Let the sth group $I = I_s$ be chosen for a refined decomposition. Denote $\mathbf{Y} = \mathbf{X}_I, r = \operatorname{rank} \mathbf{Y}$, $\mathbb{Y} = \mathfrak{T}^{-1} \circ \Pi_{\mathcal{H}}(\mathbf{Y})$ the series obtained from $\mathbf{Y}$ by diagonal averaging.

Let us describe the scheme of Nested Oblique SSA. The aim of the nested scheme is obtaining a refined decomposition of $\mathbf{Y} = \mathbf{X}_I$ in the matrix form $\mathbf{Y} = \mathbf{Y}^{(1)} + \ldots + \mathbf{Y}^{(l)}$, using the $(\mathbf{L}, \mathbf{R})$-SVD and therefore getting a decomposition of the corresponding time series $\mathbb{Y} = \widetilde{\mathbb{Y}}^{(1)} + \ldots + \widetilde{\mathbb{Y}}^{(l)}$.

For correctness of the scheme, we should assume that the matrices $\mathbf{L}$ and $\mathbf{R}$ are consistent with $\mathbf{Y}$; that is, the column space of $\mathbf{Y}$ is a subset of the column space of $\mathbf{L}$ and the row space of $\mathbf{Y}$ is a subset of the column space of $\mathbf{R}$.

Nested O-SSA is very similar to Basic SSA; the difference is that there is no Embedding step and that the matrix $\mathbf{X}_I$, which is not necessarily a Hankel matrix, is used instead of the conventional trajectory matrix.

At Decomposition step, we construct the $(\mathbf{L}, \mathbf{R})$-SVD of $\mathbf{Y}$ in the form

$$\mathbf{Y} = \sum_{i=1}^{r} \sigma_i P_i Q_i^{\mathrm{T}}, \tag{2.15}$$

see Algorithm 2.6.

The rest of the method coincides with Reconstruction stage of Basic SSA. After Grouping step, we obtain the decomposition $\mathbf{Y} = \mathbf{Y}_{J_1} + \ldots + \mathbf{Y}_{J_l}$ and then, as

a result, the refined series decomposition $\mathbb{Y} = \widetilde{\mathbb{Y}}^{(1)} + \ldots + \widetilde{\mathbb{Y}}^{(l)}$, where $\widetilde{\mathbb{Y}}^{(m)} = \mathfrak{T}^{-1} \circ \Pi_{\mathcal{H}}(\mathbf{Y}_{J_m})$.

Therefore, after application of Nested O-SSA to the group I_s, we obtain the following decomposition of the original series $\mathbb{X}$:

$$\mathbb{X} = \widetilde{\mathbb{X}}^{(1)} + \ldots + \widetilde{\mathbb{X}}^{(p)}, \text{ where } \widetilde{\mathbb{X}}^{(s)} = \widetilde{\mathbb{Y}}^{(1)} + \ldots + \widetilde{\mathbb{Y}}^{(l)}.$$

For simplicity, below we will consider the case $l = 2$.

2.4.1.3 Iterative Approach to O-SSA

Let us describe an iterative version of Nested O-SSA; that is, an iterative algorithm for obtaining appropriate matrices $\mathbf{L}$ and $\mathbf{R}$ for the $(\mathbf{L}, \mathbf{R})$-SVD of $\mathbf{X}_I$. For proper use of nested decompositions, we should expect that the matrix $\mathbf{X}_I$ is close to a rank-deficient trajectory matrix of rank r.

To explain the main principle of the method, assume that $\mathbf{X}_I = \mathbf{Y}$ is the trajectory matrix of $\mathbb{Y}$. Let $\mathbb{Y} = \mathbb{Y}^{(1)} + \mathbb{Y}^{(2)}$ and the trajectory matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$ be of ranks r_1 and r_2, $r_1 + r_2 = r$. Then by Golyandina and Shlemov (2015; Theorem 1) there exist r-rank separating matrices $\mathbf{L}^*$, $\mathbf{R}^*$ of sizes $L \times L$ and $K \times K$ correspondingly and a partition $\{1, \ldots, r\} = J_1 \sqcup J_2$ such that we can perform the proper grouping in the $(\mathbf{L}^*, \mathbf{R}^*)$-SVD and thereby obtain $\mathbf{Y}_{J_1} = \mathbf{Y}_1$ and $\mathbf{Y}_{J_2} = \mathbf{Y}_2$.

The separating matrices $\mathbf{L}^*$ and $\mathbf{R}^*$ are unknown as they are determined by unknown trajectory spaces of $\mathbb{Y}^{(1)}$ and $\mathbb{Y}^{(2)}$. Therefore, we want to construct a sequence of $(\mathbf{L}, \mathbf{R})$-SVD decompositions (2.14), which in the limit gives the required separating decomposition.

Let us have an initial $(\mathbf{L}^{(0)}, \mathbf{R}^{(0)})$-SVD decomposition of $\mathbf{Y}$ and group its components to obtain some initial estimates $\widetilde{\mathbb{Y}}^{(1,0)}$ and $\widetilde{\mathbb{Y}}^{(2,0)}$ of $\mathbb{Y}^{(1)}$ and $\mathbb{Y}^{(2)}$. Then we can use the trajectory spaces of $\widetilde{\mathbb{Y}}^{(1,0)}$ and $\widetilde{\mathbb{Y}}^{(2,0)}$ to construct the new inner product which is expected to be closer to the separating one. Therefore, we can expect that $\widetilde{\mathbb{Y}}^{(1,1)}$ and $\widetilde{\mathbb{Y}}^{(2,1)}$ will be closer to $\mathbb{Y}^{(1)}$ and $\mathbb{Y}^{(2)}$ and therefore we take their trajectory spaces to construct a new inner product; and so on. Of course, if the initial decomposition is strongly separating, then we obtain $\widetilde{\mathbb{Y}}^{(m,1)} = \widetilde{\mathbb{Y}}^{(m,0)} = \mathbb{Y}^{(m)}$, $m = 1, 2$.

2.4.1.4 Basic Iterative Algorithm

We call the iterative version of Nested Oblique SSA *Iterative Oblique SSA* or *Iterative O-SSA*.

As before, we consider a nested O-SSA whose input is the matrix $\mathbf{Y} = \mathbf{X}_I$ of rank r. For Basic SSA and for nested O-SSA, a partition of eigentriple numbers for grouping is made after Decompositions stage. For Iterative O-SSA, a partition $I = \widetilde{J}_1 \sqcup \widetilde{J}_2$, $r_m = |\widetilde{J}_m|$, should be specified in advance, since iterations are performed

in an automatic mode. Certainly, the choice of partition is not made in dark since before the use of a nested version, we have a full decomposition which we use for choosing the group I and its partition.

Iterative O-SSA is made of repeated application of nested O-SSA with recalculation of $(\mathbf{L}^{(k)}, \mathbf{R}^{(k)})$. As any iterative algorithm, Iterative O-SSA should have initial data $(\mathbf{L}^{(0)}, \mathbf{R}^{(0)})$ and a stopping rule. Standard stopping rule includes the maximum number of iterations M and the precision threshold ε. In Iterative O-SSA, the algorithm stops if the reconstructed series components $\widetilde{\mathbb{Y}}^{(m,k)}$, $m = 1, 2$, change very little. For a function $\rho(\cdot)$ defining a vector norm, the iterations stop under the condition $\max\left(\rho(\widetilde{\mathbb{Y}}^{(m,k)} - \widetilde{\mathbb{Y}}^{(m,k-1)}), m = 1, 2\right) < \varepsilon$.

Note that since the consistence of $(\mathbf{L}, \mathbf{R})$ with $\mathbf{Y}$ is needed for a correct $(\mathbf{L}, \mathbf{R})$-SVD, the initial data $(\mathbf{L}^{(0)}, \mathbf{R}^{(0)})$ should also be consistent.

Remark 2.2 The initial matrices $(\mathbf{L}^{(0)}, \mathbf{R}^{(0)})$ together with grouping can be specified so that the initial decomposition is a part of the SVD (2.3) given by the set of indices I and $I = \widetilde{J}_1 \sqcup \widetilde{J}_2$, where $\widetilde{J}_1$ and $\widetilde{J}_2$ are chosen on the base of analysis of the components in (2.3). Since (2.3) is biorthogonal, $\mathbf{L}^{(0)}$ and $\mathbf{R}^{(0)}$ are the identity matrices. It is convenient to denote by J_1 and J_2 the sets consisting of ordinal indices of the elements of $\widetilde{J}_1$ and $\widetilde{J}_2$ in I. Thereby, $\{1, \dots, r\} = J_1 \sqcup J_2$. For example, if $\widetilde{J}_1 = \{11, 14\}$ and $\widetilde{J}_2 = \{12, 18\}$, then $I = \{11, 12, 14, 18\}$, $J_1 = \{1, 3\}$ and $J_1 = \{2, 4\}$.

To finalize the Iterative O-SSA method, we present a formal description of iterations. The separating decomposition $\mathbf{Y} = \mathbf{Y}_1 + \mathbf{Y}_2$ should satisfy the following properties:

(a) $\mathbf{Y}_1$ and $\mathbf{Y}_2$ are Hankel;
(b) $\operatorname{rank}\mathbf{Y}_1 = r_1$, $\operatorname{rank}\mathbf{Y}_2 = r_2$;
(c) the column and row spaces of $\mathbf{Y}_1$ and $\mathbf{Y}_2$ lie in the column and row spaces of $\mathbf{Y}$;
(d) $\mathbf{Y}_1$ and $\mathbf{Y}_2$ are $(\mathbf{L}, \mathbf{R})$-biorthogonal for $\mathbf{L} = \mathbf{L}^*$ and $\mathbf{R} = \mathbf{R}^*$.

Define Π_{col} the orthogonal projection operator (for the Euclidean norm) on the column space of $\mathbf{Y}$, Π_{row} the projection operator on the row space of $\mathbf{Y}$. The nested group is the ordered union $I = \widetilde{J}_1 \sqcup \widetilde{J}_2$, $r_m = |\widetilde{J}_m|$, J_1 and J_2 are defined in Remark 2.2; the pair of matrices $(\mathbf{L}^{(k-1)}, \mathbf{R}^{(k-1)})$ is the input for the kth iteration.

Let us formulate the kth iteration steps.

(A) To obtain Hankel matrices, we perform hankelization of the input decomposition $\widetilde{\mathbf{Y}}_m = \Pi_{\mathcal{H}}\mathbf{Y}_{J_m}^{(k-1)}$, $m = 1, 2$.
(B) Then, to obtain a low-rank approximation of ranks r_1 and r_2 correspondingly, we construct the ordinary SVDs $\widetilde{\mathbf{Y}}_m = \sum_{i=1}^{d_m} \sqrt{\lambda_i^{(m)}} U_i^{(m)} (V_i^{(m)})^{\mathrm{T}}$, $m = 1, 2$, and take the leading r_m terms.
(C) Since we should not fall outside the column space of the input matrix $\mathbf{Y}$ (we consider a nested decomposition), we find the projections $\widehat{U}_i^{(m)} = \Pi_{\mathrm{col}} U_i^{(m)}$ and $\widehat{V}_i^{(m)} = \Pi_{\mathrm{row}} V_i^{(m)}$ for $i = 1, \dots, r_m$, $m = 1, 2$. Denote

$$\widehat{\mathbf{U}}^{(m)} = [\widehat{U}_1^{(m)} : \dots : \widehat{U}_{r_m}^{(m)}], \ \widehat{\mathbf{V}}^{(m)} = [\widehat{V}_1^{(m)} : \dots : \widehat{V}_{r_m}^{(m)}].$$

For the algorithm correctness, we assume that the matrices $\widehat{\mathbf{U}}^{(m)}$ and $\widehat{\mathbf{V}}^{(m)}$ are of full rank; otherwise, the algorithm may not work.

(D) Finally, calculate $\mathbf{L}^{(k)} = (\widehat{\mathbf{U}}^{\dagger})^{\mathrm{T}}\widehat{\mathbf{U}}^{\dagger}$ and $\mathbf{R}^{(k)} = (\widehat{\mathbf{V}}^{\dagger})^{\mathrm{T}}\widehat{\mathbf{V}}^{\dagger}$, where $\widehat{\mathbf{U}} = [\widehat{\mathbf{U}}^{(1)} : \widehat{\mathbf{U}}^{(2)}]$ and $\widehat{\mathbf{V}} = [\widehat{\mathbf{V}}^{(1)} : \widehat{\mathbf{V}}^{(2)}]$, to achieve the $(\mathbf{L}^{(k)}, \mathbf{R}^{(k)})$-biorthogonality.

The convergence of $\widetilde{\mathbb{Y}}^{(1,k)}$ and $\widetilde{\mathbb{Y}}^{(2,k)}$ to a proper decomposition is not proved theoretically. However, looking at the construction scheme, which resembles the alternating projections, we do expect this convergence, at least if the case chosen is not too unusual. Numerical experiments confirm the convergence in the majority of examples. Note also that Iterative O-SSA does not change the separating decomposition; that is, the separating decomposition is a fixed point of the algorithm.

2.4.1.5 Modification with Sigma-Correction

If the initial point $(\mathbf{L}^{(0)}, \mathbf{R}^{(0)})$ for iterations is not far from the separating pair $(\mathbf{L}^*, \mathbf{R}^*)$, then we can expect that the convergence in the algorithm above will take place, since we are close to the fixed-point value and we can expect that $\sigma_i^{(k)}$ in the $(\mathbf{L}^{(k)}, \mathbf{R}^{(k)})$-SVDs $\mathbf{Y} = \sum_{i=1}^{r} \sigma_i^{(k)} P_i^{(k)} (Q_i^{(k)})^{\mathrm{T}}$ are just slightly changed during iterations. In general, however, a possible reordering of the decomposition components between iterations of Iterative O-SSA can interfere with convergence. The case of $J_1 = \{1, \ldots, r_1\}$, when the minimal singular value σ_{r_1} of the first series is kept significantly larger than the maximal singular value σ_{r_1+1} of the second series, would prevent the component reordering and hence improve the convergence.

Let us describe a modification of Iterative O-SSA that provides reordering of the components, moves them apart and thereby relaxes the problem of component mixing. In this modification, an adjustment is made for calculation of $\widehat{\mathbf{U}}^{(2)}$ and $\widehat{\mathbf{V}}^{(2)}$ at Step (C) of iterations.

Let us choose a parameter $\varkappa > 1$. If $\lambda_{r_1}^{(1)} < \varkappa^2 \lambda_1^{(2)}$ at Step (C), then define $\mu = \varkappa \sqrt{\lambda_1^{(2)}/\lambda_{r_1}^{(1)}}$ and change $\widehat{\mathbf{U}}^{(2)} \leftarrow \sqrt{\mu}\widehat{\mathbf{U}}^{(2)}$, $\widehat{\mathbf{V}}^{(2)} \leftarrow \sqrt{\mu}\widehat{\mathbf{V}}^{(2)}$. To be consistent with the reordering, set $J_1 = \{1, \ldots, r_1\}$, $J_2 = \{r_1 + 1, \ldots, r\}$.

Note that if $\lambda_{r_1}^{(1)} < \varkappa^2 \lambda_1^{(2)}$, then the adjustment above makes a change in the order of the matrix components in (2.18), since they are ordered by $\sigma_i^{(k)}$. Hence we force an increase of the matrix components related to the first series component. For explanation of how this sigma-correction works, see Golyandina and Shlemov (2015; Proposition 5).

Remark 2.3 The reordering procedure is made by sequential adjustment of the component weights and therefore depends on the component enumeration.

Summarizing, the described correction can help to improve convergence and to provide strong separability of components in the case when only weak separability takes place.

2.4.2 Separability

The notion of weak and strong (**L**, **R**)-separability, which is similar to the conventional separability described in Sect. 2.1.3, can be introduced. Let $\mathbb{X} = \mathbb{X}^{(1)} + \mathbb{X}^{(2)}$, **X** be the trajectory matrix of $\mathbb{X}$, $\mathbf{X}^{(m)}$ be the trajectory matrices of the series components, $\mathbf{X}^{(m)} = \sum_{i=1}^{r_m} \sigma_{m,i} P_{m,i} Q_{m,i}^{\mathrm{T}}$ be their (**L**, **R**)-SVDs, $m = 1, 2$. We assume that **L** and **R** are consistent with **X**, $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$.

Definition 2.6 Let L be fixed. Two series $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$ are called *weakly* (**L**, **R**)-*separable*, if their column trajectory spaces are **L**-orthogonal and their row trajectory spaces are **R**-orthogonal; that is, $(\mathbf{X}^{(1)})^{\mathrm{T}}\mathbf{L}\mathbf{X}^{(2)} = \mathbf{0}_{K,K}$ and $\mathbf{X}^{(1)}\mathbf{R}(\mathbf{X}^{(2)})^{\mathrm{T}} = \mathbf{0}_{L,L}$.

Definition 2.7 Two series $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$ are called *strongly* (**L**, **R**)-*separable*, if they are weakly (**L**, **R**)-separable and $\sigma_{1,i} \neq \sigma_{2,j}$ for any i and j.

The (**L**, **R**)-separability of two series components means **L**-orthogonality of their subseries of length L and **R**-orthogonality of the subseries of length $K = N - L + 1$. For suitably chosen L and R, the (**L**, **R**)-separability is much less restrictive than the ordinary one. Indeed, Theorem 1 from Golyandina and Shlemov (2015) states that for series $\mathbb{X} = \mathbb{X}^{(1)} + \mathbb{X}^{(2)}$ of rank r, where $\mathbb{X}^{(m)}$ is the series of rank r_m, $m = 1, 2$, and $r_1 + r_2 = r$, there exist separating matrices $\mathbf{L} \in \mathsf{R}^{L\times L}$ and $\mathbf{R} \in \mathsf{R}^{K\times K}$ of rank r such that the series $\mathbb{X}^{(1)}$ and $\mathbb{X}^{(2)}$ are strongly (**L**, **R**)-separable. Moreover, the separating matrices **L** and **R** can be explicitly written down.

Denote by $\{P_i^{(m)}\}_{i=1}^{r_m}$ a basis of the column space of $\mathbf{X}^{(m)}$ and by $\{Q_i^{(m)}\}_{i=1}^{r_m}$ a basis of the row space of $\mathbf{X}^{(m)}$, $m = 1, 2$; e.g., $P_i^{(m)} = P_{m,i} \in \mathsf{R}^L$, $Q_i^{(m)} = Q_{m,i} \in \mathsf{R}^K$. Define

$$\mathbf{P} = [P_1^{(1)} : \ldots : P_{r_1}^{(1)} : P_1^{(2)} : \ldots : P_{r_2}^{(2)}],$$

$$\mathbf{Q} = [Q_1^{(1)} : \ldots : Q_{r_1}^{(1)} : Q_1^{(2)} : \ldots : Q_{r_2}^{(2)}].$$

Then the separating matrices have the form $\mathbf{L} = (\mathbf{P}^{\dagger})^{\mathrm{T}}\mathbf{P}^{\dagger}$ and $\mathbf{R} = (\mathbf{Q}^{\dagger})^{\mathrm{T}}\mathbf{Q}^{\dagger}$ (compare with Step (D) of the algorithm scheme in Sect. 2.4.1.4).

The condition $r_1 + r_2 = r$ can be expressed in terms of the characteristic roots. This condition is satisfied if the sets of the characteristic roots of the series are disjoint.

Thus, any two times series governed by LRRs with different characteristic roots can be separated by some (**L**, **R**)-SVD for sufficiently large series and window lengths. This statement is not constructive, since the trajectory spaces of the separated series should be known for exact separation. However, we can try to estimate these spaces and therefore improve the separability.

We have already explained how to achieve weak separability. Proposition 5 from Golyandina and Shlemov (2015) shows how to correct the decomposition to get strong separability. Denote by I the set of decomposition components corresponding

to $\mathbb{X}^{(1)}$ in a separating $(\mathbf{L}, \mathbf{R})$-SVD

$$\mathbf{Y} = \sum_i \sigma_i P_i Q_i^{\mathrm{T}} = \sum_{i \in I} \sigma_i P_i Q_i^{\mathrm{T}} + \sum_{i \notin I} \sigma_i P_i Q_i^{\mathrm{T}}. \tag{2.16}$$

If in the group I there is a σ_i, which coincides with a σ_j from the residual group, then the SVD decomposition is not unique and therefore the calculated SVD can differ from the separating SVD (2.16). This situation can be easily avoided as follows: let us take $\widetilde{P}_i = \mu_i P_i$ and $\widetilde{Q}_i = \nu_i Q_i$ for some μ_i and ν_i, then the $(\widetilde{\mathbf{L}}, \widetilde{\mathbf{R}})$-SVD

$$\mathbf{Y} = \sum_i \widetilde{\sigma}_i \widetilde{P}_i \widetilde{Q}_i^{\mathrm{T}}$$

for $\widetilde{\mathbf{L}} = (\widetilde{\mathbf{P}}^{\dagger})^{\mathrm{T}}\widetilde{\mathbf{P}}^{\dagger}$ and $\widetilde{\mathbf{R}} = (\widetilde{\mathbf{Q}}^{\dagger})^{\mathrm{T}}\widetilde{\mathbf{Q}}^{\dagger}$ will still be separating; however, $\widetilde{\sigma}_i = \sigma_i/(\mu_i \nu_i)$ can be made equal to any given numbers by choosing appropriate μ_i and ν_i.

Measures of oblique separability. If Oblique SSA does not separate the components exactly, a measure of separability is necessary. As stated in Sect. 1.3, the main measure of separability in Basic SSA is the **w**-correlation between two time series: $\rho_{\mathbf{w}}(\mathbb{X}, \mathbb{Y}) = \langle \mathbf{X}, \mathbf{Y} \rangle_{\mathrm{F}} / (\|\mathbf{X}\|_{\mathrm{F}} \|\mathbf{Y}\|_{\mathrm{F}})$, where $\mathbf{X}$ and $\mathbf{Y}$ are the trajectory matrices of the series.

In Oblique SSA with $(\mathbf{L}, \mathbf{R})$-SVD we then naturally consider $\rho_{\mathbf{L},\mathbf{R}}$, which is similar to $\rho_{\mathbf{w}}$ and defines the inner product by

$$\langle \mathbf{X}, \mathbf{Y} \rangle_{(\mathbf{L},\mathbf{R})} = \mathrm{trace}(\mathbf{L}\mathbf{X}\mathbf{R}\mathbf{Y}^{\mathrm{T}}).$$

Note that if the matrices $\mathbf{L}$ and $\mathbf{R}$ are not consistent with $\mathbf{X}$ and $\mathbf{Y}$, then $\rho_{\mathbf{L},\mathbf{R}}$ takes into consideration only projections of their columns and rows on the column spaces of $\mathbf{L}$ and $\mathbf{R}$. This means that $\rho_{\mathbf{L},\mathbf{R}}$ can underestimate the separability inaccuracy.

For Oblique SSA, when only one of two coordinate systems (left or right) is oblique, the conventional **w**-correlations between series are more appropriate measures of separability, since in the case of exact oblique separability we have both $\rho_{\mathbf{w}}$ and $\rho_{\mathbf{L},\mathbf{R}}$ equal to zero.

Another important measure of separability is the closeness of the reconstructed series components to set of time series of finite rank. This can be measured by the contribution of the leading $r_m = |I_m|$ eigentriples into the SVD of the trajectory matrix $\widetilde{\mathbf{X}}^{(m)}$ of the mth reconstructed series component $\widetilde{\mathbb{X}}^{(m)}$. If we denote $\widetilde{\lambda}_{m,i}$ the squared singular values in the ordinary SVD of $\widetilde{\mathbf{X}}^{(m)}$, then

$$\tau_{r_m}(\widetilde{\mathbb{X}}^{(m)}) = 1 - \sum_{i=1}^{r_m} \widetilde{\lambda}_{m,i} / \|\widetilde{\mathbf{X}}^{(m)}\|^2 \tag{2.17}$$

can be considered as a characteristic of closeness of the mth series to the set of series of rank r_m.

2.4.3 Algorithms

Let us present the method described in Sect. 2.4 in the form of algorithms. The method consists of different parts and therefore we describe it as several algorithms.

Let us start with a general algorithm demonstrating how Oblique SVD of a matrix $\mathbf{Z}$ can be reduced to an ordinary SVD.

Algorithm 2.6 ($\mathbf{L}$, $\mathbf{R}$)-SVD

Input: Matrix $\mathbf{Z} \in \mathsf{R}^{L\times K}$ to decompose and matrices $\mathbf{L} \in \mathsf{R}^{L\times L}$, and $\mathbf{R} \in \mathsf{R}^{K\times K}$ of rank r, where $(\mathbf{L}, \mathbf{R})$ is consistent with $\mathbf{Z}$.

Output: The $(\mathbf{L}, \mathbf{R})$-SVD in the form $\mathbf{Z} = \sum_{i=1}^{r} \sigma_i P_i Q_i^{\mathrm{T}}$.

1: Find $\mathbf{O_L} \in \mathsf{R}^{r\times L}$ and $\mathbf{O_R} \in \mathsf{R}^{r\times K}$ such that $\mathbf{O_L^{\mathrm{T}} O_L} = \mathbf{L}$ and $\mathbf{O_R^{\mathrm{T}} O_R} = \mathbf{R}$.
2: Calculate $\mathbf{O_L Z O_R^{\mathrm{T}}}$.
3: Compute the ordinary SVD decomposition $\mathbf{O_L Z O_R^{\mathrm{T}}} = \sum_{i=1}^{r} \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.
4: Set $\sigma_i = \sqrt{\lambda_i}$, $P_i = \mathbf{O_L^{\dagger}} U_i$ and $Q_i = \mathbf{O_R^{\dagger}} V_i$, where $\dagger$ denotes pseudo-inverse.

Now we formulate the Iterative O-SSA algorithm. Denote $\mathbf{Y} = \mathbf{X}_I$, $r = \operatorname{rank} \mathbf{Y}$, $\mathbb{Y} = \mathcal{T}^{-1} \circ \Pi_{\mathcal{H}}(\mathbf{Y})$ the series obtained from $\mathbf{Y}$ by the diagonal averaging.

We separate the whole algorithm into two parts. Algorithm 2.7 shows a general scheme of Iterative O-SSA, but it does not show how to calculate the pair of matrices $(\mathbf{L}^{(k)}, \mathbf{R}^{(k)})$ at each iteration. Algorithm 2.8 covers this gap.

Algorithm 2.7 Iterative O-SSA

Input: Decomposition of the L-trajectory matrix $\mathbf{X} = \sum_{i=1}^{d} \sigma_i P_i Q_i^{\mathrm{T}}$ of the series $\mathbb{X}$; disjoint sets of indices $\widetilde{J}_1$ and $\widetilde{J}_2$ from $\{1, \dots, d\}$; the accuracy tolerance ε; function ρ for calculating the accuracy; the maximal number of iterations M; initial matrices $(\mathbf{L}^{(0)}, \mathbf{R}^{(0)})$ consistent with $\mathbf{Y} = \mathbf{X}_I$. The set $I = \{i_1, \dots, i_r\}$ is defined as $I = \widetilde{J}_1 \sqcup \widetilde{J}_2$, $r_m = |\widetilde{J}_m|$, $r = |I| = r_1 + r_2$, the sets J_1 and J_2 are defined in Remark 2.2. This partition produces the decompositions for the matrices and series: $\mathbf{Y} = \mathbf{Y}^{(0)}_{J_1} + \mathbf{Y}^{(0)}_{J_2}$ and $\mathbb{Y} = \widetilde{\mathbb{Y}}^{(1,0)} + \widetilde{\mathbb{Y}}^{(2,0)}$.

Output: $\mathbb{Y} = \widetilde{\mathbb{Y}}^{(1)} + \widetilde{\mathbb{Y}}^{(2)}$.

1: Set $k = 1$.
2: Call Algorithm 2.8 for calculation of $(\mathbf{L}^{(k)}, \mathbf{R}^{(k)})$ consistent with $\mathbf{Y}$.
3: Compute the $(\mathbf{L}^{(k)}, \mathbf{R}^{(k)})$-SVD of $\mathbf{Y}$ by Algorithm 2.6:

$$\mathbf{Y} = \sum_{i=1}^{r} \sigma_i^{(k)} P_i^{(k)} (Q_i^{(k)})^{\mathrm{T}} = \mathbf{Y}^{(k)}_{J_1} + \mathbf{Y}^{(k)}_{J_2}. \tag{2.18}$$

4: Obtain the decomposition of the series $\mathbb{Y} = \widetilde{\mathbb{Y}}^{(1,k)} + \widetilde{\mathbb{Y}}^{(2,k)}$, where $\widetilde{\mathbb{Y}}^{(m,k)} = \mathcal{T}^{-1} \circ \Pi_{\mathcal{H}}(\mathbf{Y}^{(k)}_{J_m})$, $m = 1, 2$.
5: If $k \geq M$ or $\max\left(\rho(\widetilde{\mathbb{Y}}^{(m,k)} - \widetilde{\mathbb{Y}}^{(m,k-1)}), m = 1, 2\right) < \varepsilon$, then $\widetilde{\mathbb{Y}}^{(m)} \leftarrow \widetilde{\mathbb{Y}}^{(m,k)}$, $m = 1, 2$, and STOP; else $k \leftarrow k + 1$ and go to Step 2.

Algorithm 2.8 presents the iteration itself, including the sigma-correction, which may be useful for achieving the strong separability.

Algorithm 2.8 Calculation of $(\mathbf{L}^{(k)}, \mathbf{R}^{(k)})$

Input: Partition $\{1, \dots, r\} = J_1 \sqcup J_2$; $r_m = |J_m|$; pair of matrices $(\mathbf{L}^{(k-1)}, \mathbf{R}^{(k-1)})$; parameter for sigma-correction $\varkappa > 1$ (if $\varkappa = 0$, then the sigma-correction is not performed).

Output: Pair of matrices $(\mathbf{L}^{(k)}, \mathbf{R}^{(k)})$ for kth iteration.

1: Calculate $\widetilde{\mathbf{Y}}_m = \Pi_{\mathcal{H}} \mathbf{Y}_{J_m}^{(k-1)}$, $m = 1, 2$.

2: Construct the ordinary SVDs:

$$\widetilde{\mathbf{Y}}_m = \sum_{i=1}^{d_m} \sqrt{\lambda_i^{(m)}} U_i^{(m)} (V_i^{(m)})^{\mathrm{T}}, \ m = 1, 2,$$

(we need the first r_m terms only).

3: Sigma-correction (if $\varkappa \neq 0$): If $\lambda_{r_1}^{(1)} < \varkappa^2 \lambda_1^{(2)}$, then define $\mu = \varkappa \sqrt{\lambda_1^{(2)} / \lambda_{r_1}^{(1)}}$ and change $\widehat{\mathbf{U}}^{(2)} \leftarrow \sqrt{\mu} \widehat{\mathbf{U}}^{(2)}$, $\widehat{\mathbf{V}}^{(2)} \leftarrow \sqrt{\mu} \widehat{\mathbf{V}}^{(2)}$. In view of reordering, set $J_1 = \{1, \dots, r_1\}$, $J_2 = \{r_1 + 1, \dots, r\}$.

4: Find the projections $\widehat{U}_i^{(m)} = \Pi_{\mathrm{col}} U_i^{(m)}$ and $\widehat{V}_i^{(m)} = \Pi_{\mathrm{row}} V_i^{(m)}$ for $i = 1, \dots, r_m$, $m = 1, 2$. Denote

$$\widehat{\mathbf{U}}^{(m)} = [\widehat{U}_1^{(m)} : \dots : \widehat{U}_{r_m}^{(m)}], \ \widehat{\mathbf{V}}^{(m)} = [\widehat{V}_1^{(m)} : \dots : \widehat{V}_{r_m}^{(m)}].$$

5: Calculate $\mathbf{L}^{(k)} = (\widehat{\mathbf{U}}^{\dagger})^{\mathrm{T}} \widehat{\mathbf{U}}^{\dagger}$ and $\mathbf{R}^{(k)} = (\widehat{\mathbf{V}}^{\dagger})^{\mathrm{T}} \widehat{\mathbf{V}}^{\dagger}$, where $\widehat{\mathbf{U}} = [\widehat{\mathbf{U}}^{(1)} : \widehat{\mathbf{U}}^{(2)}]$ and $\widehat{\mathbf{V}} = [\widehat{\mathbf{V}}^{(1)} : \widehat{\mathbf{V}}^{(2)}]$.

Remark 2.4 Algorithm 2.7, which uses the sigma-correction, may change the groups of indices. The new groups in Algorithm 2.8 are constructed in such a way that J_1 and J_2 partition the set $\{1, \dots, r\}$. The new partition of I is obtained as $\widetilde{J}_1 = \{i_k \in I : k \in J_1\}$ and $\widetilde{J}_2 = \{i_k \in I : k \in J_2\}$.

Remark 2.5 Algorithm 2.7 describes a refined decomposition of the matrix $\mathbf{X}_I$. However, we can consider Iterative O-SSA as an algorithm, where the full decomposition of the trajectory matrix $\mathbf{X}$ of an original series $\mathbb{X}$ is used (which changes components from the group I). The result would also be a full decomposition.

2.4.4 *Iterative O-SSA in* RSSA

2.4.4.1 Description of Functions

Since Iterative O-SSA is a nested method, the `ssa` function can be called for obtaining an `ssa` object `s`, see "Description of Function" in Sects. 2.1–2.3, 2.6. For Iterative O-SSA itself, the function `iossa` is used. Since the result of `iossa` is

also an ssa object, which contains the full decomposition, the iossa function can be applied to the result of another application of iossa.

Let us outline the main arguments of iossa in a typical function call:

```
ios <- iossa(s, nested.groups = list(c(1,4),7:10), trace = FALSE,
             tol = 1e-5, maxiter = 100,
             norm = function(x) sqrt(mean(x^2)),
             kappa = 2)
```

Arguments:

s is an ssa object holding the full one-dimensional SSA (Basic SSA, Toeplitz SSA, Basic SSA with projections, Shaped SSA) decomposition.

nested.groups is a list of groups of eigentriples from the full decomposition s; the list gives the initial grouping for Iterative O-SSA iterations.

tol, maxiter, norm are related to the convergence of iterations: tolerance with respect to the indicated norm and the number of iterations. Function norm calculates a norm of a vector; this norm is applied to the difference between the reconstructed series at sequential iterations and is used for convergence detection. If this norm is smaller than tol, then iterations stop.

trace indicates whether the convergence process should be traced.

kappa is the "kappa"-parameter for the sigma-correction procedure. If kappa = NULL, the sigma-correction is not applied.

Note that only s and nested.groups should be set if the default values are appropriate.

The returning value of the function is an object of ossa class. In addition to usual ssa object fields, it also contains the following fields:

iossa.result is an object of iossa.result class, a list which contains algorithm parameters, condition numbers, separability measures, the number of iterations, and the convergence status.

iossa.groups is a list of groups within the nested decomposition; indices of components correspond to their indices in the full decomposition.

iossa.groups.all is a list, which describes the cumulative grouping after sequential Iterative O-SSA decompositions in the case of non-intersecting groups given by nested.groups. Otherwise, the list iossa.groups.all coincides with iossa.groups.

ossa.set is a vector of indices of elementary components used in Iterative O-SSA (that is, used in nested.groups).

To look at weighted oblique correlations of the obtained elementary components, one can call owcor(ios, groups = ios$ossa.set).

The reconstruct function performs reconstruction as usual. A possible question is how to set the groups, which are a part of the result of iossa. A typical call is

```
r.ios <- reconstruct(ios, groups = ios$iossa.groups)
```

2.4.4.2 Typical Code

Fragment 2.4.1 demonstrates the method on a simulated example. Since Iterative O-SSA is designed to separate non-orthogonal series components, let us consider a noisy sum of three sine waves with two of them having close frequencies, see Fig. 2.14. For achieving separability from noise we assume that the level of noise is low.

First, we apply Basic SSA. In Fig. 2.15 we see that the signal is contained in components 1–6 and is separated from noise. Weighted correlations do not show any

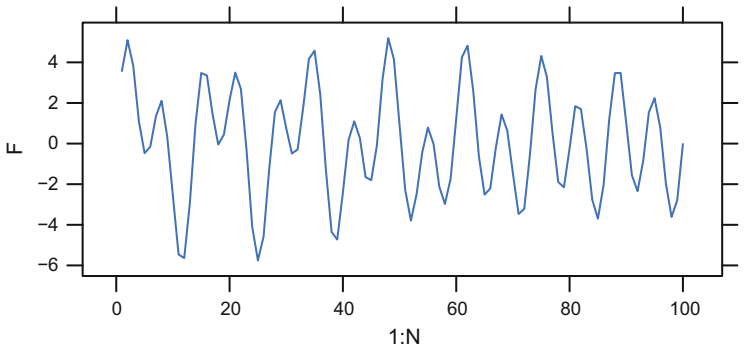

Fig. 2.14 Noisy sum of three sinusoids: The original series

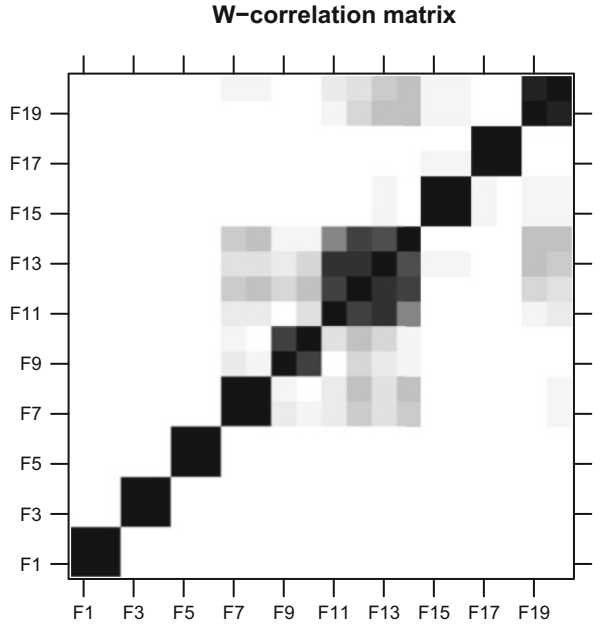

Fig. 2.15 Noisy sum of three sinusoids, Basic SSA: **w**-Correlation matrix

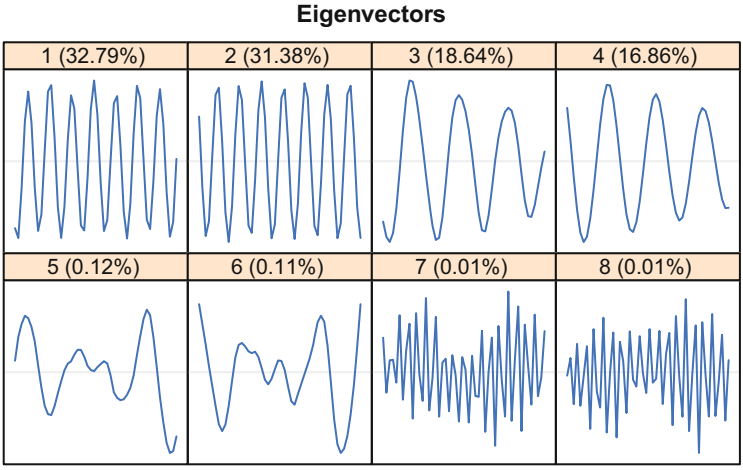

Fig. 2.16 Noisy sum of three sinusoids, Basic SSA: Eigenvectors

problem with separability of three groups of components, 1–2, 3–4, 5–6. However the graph with eigenvectors (Fig. 2.16) shows that the pairs 3–4 and 5–6 are most likely mixed within the signal. This means that Iterative O-SSA may help.

Fragment 2.4.1 (Noisy Sum of Three Sinusoids: Iterative O-SSA)

```
> N <- 100
> L <- 50
> omega1 <- 0.07
> omega2 <- 0.065
> omega3 <- 0.15
> sigma <- 0.1
> set.seed(3)
> F <- 2 * sin(2 * pi * omega1 * (1:N)) +
+   sin(2 * pi * omega2 * (1:N)) +
+   3 * sin(2 * pi * omega3 * (1:N)) + sigma * rnorm(N)
> xyplot(F ~ 1:N, type = "l")
> s <- ssa(F, L)
> plot(s, type = "vectors", idx = 1:8, layout = c(4, 2))
> plot(wcor(s, groups = 1:20), scales = list(at = seq(1,20,2)))
> ios <- iossa(s, nested.groups = list(3:4, 5:6), maxiter = 1000)
> plot(ios, type = "vectors", idx = 1:8, layout = c(4, 2))
> ior <- reconstruct(ios, groups = c(list(1:2), ios$iossa.groups))
> plot(ior, plot.method = "xyplot", add.original = FALSE,
+      add.residuals = FALSE)
```

Indeed, the reconstruction by Basic SSA has failed, while the nested reconstruction of the signal components by Iterative O-SSA is successful, see Figs. 2.17 and 2.18. We can apply Iterative O-SSA to the whole signal but the separability is better if we take only the mixed components 3–6. We choose initial grouping

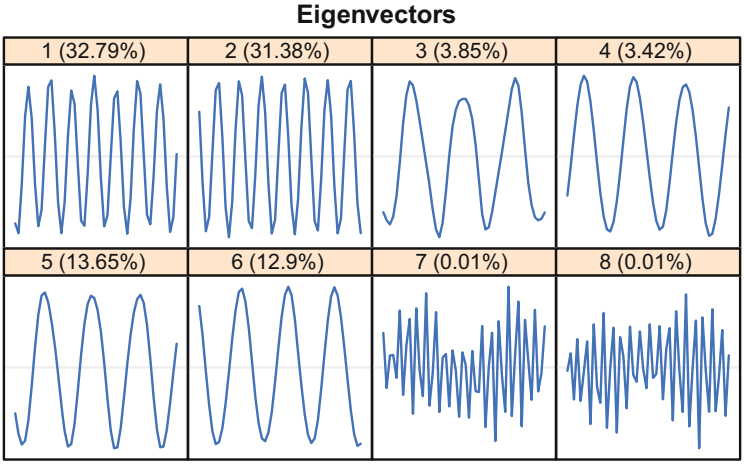

Fig. 2.17 Noisy sum of three sinusoids, Iterative O-SSA: Eigenvectors

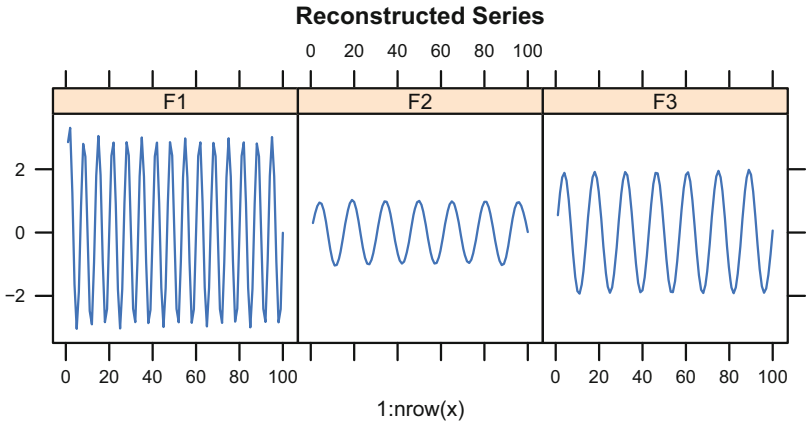

Fig. 2.18 Noisy sum of three sinusoids, Iterative O-SSA: Reconstruction

`list(3:4, 5:6)` since these groups look like corrupted pairs; we could have also chosen `list(3:6, 4:5)`.

For Iterative O-SSA, it is convenient to use automatically generated groups `ios$iossa.groups` for grouping within the refined decomposition. The groups returned by `ios$iossa.groups` can differ from the groups which were set initially, especially if the sigma-correction is used (`kappa` is not zero). In the considered example, the resulting groups after sigma-correction will be `list(3:4, 5:6)`.

If the problem of lack of conventional weak separability is supplemented by the problem of lack of strong separability (in the considered example, when the amplitudes of sinusoids coincide or are close to each other), the use of `kappa` can still allow us to achieve the right decomposition.

Fragment 2.4.2 contains characteristics of the resultant decomposition by Iterative O-SSA. The summary contains measures of quality of the initial and the final decompositions. One can see that the iterations converged according to a default tolerance (`tol`) in 243 iterations. The measure τ defined in (2.17) as a measure of rank-deficiency of trajectory matrices of the found components has decreased considerably. Standard **w**-correlations appear inappropriate as measures of separability.

Fragment 2.4.2 (Noisy Sum of Three Sinusoids: Iterative O-SSA, Summary)

```
> print(ios$iossa.groups)
[[1]]
[1] 3 4
[[2]]
[1] 5 6
> summary(ios)
Call:
iossa.ssa(x = s, nested.groups = list(3:4, 5:6), maxiter = 1000)
Series length: 100,Window length: 50,        SVD method: eigen
Special triples:  0
Computed:
Eigenvalues: 50,        Eigenvectors: 50,     Factor vectors: 6
Precached: 0 elementary series (0 MiB)
Overall memory consumption (estimate): 0.0352 MiB
Iterative O-SSA result:
        Converged:              yes
        Iterations:             243
        Initial mean(tau):      0.1032
        Initial tau:            0.0007976, 0.2055299
        I. O-SSA mean(tau):     0.0004452
        I. O-SSA tau:           0.0006709, 0.0002196
        Initial max wcor:       0.02442
        I. O-SSA max wcor:      0.06986
        I. O-SSA max owcor:     0.0732
```

2.4.4.3 Simulated Example: Separability of Sine Waves

Let us add noise to the sum of two sinusoids and take

$$x_n = \sin(2\pi\omega_1 n) + A\sin(2\pi\omega_2 n) + \delta\varepsilon_n$$

with close frequencies $\omega_1 = 0.07$ and $\omega_2 = 0.06$ and unequal amplitudes, 1 and $A = 1.2$. Here ε_n is white Gaussian noise with variance 1, $\delta = 1$. Let $N = 150$, $L = 70$.

Basic SSA separates well the sinusoids from noise, but cannot separate these sinusoids from each other. Thus, Iterative O-SSA, applied to the estimated signal subspace, should be used. We use the sigma-correction with $\varkappa = 2$, since the difference between amplitudes, 1 and 1.2, appears to be small for achieving strong separability in the presence of noise. We set the initial grouping ET1–2 and ET3–4.

Let us investigate the dependence of number of iterations on ω_1 with fixed $\omega_2 = 0.06$. We change ω_1 from 0.03 to 0.059 and from 0.061 to 0.1. Fragment 2.4.3 depicts the results.

Fragment 2.4.3 (Dependence of `iossa` Error on Difference Between Frequencies)

```
> rowMeansQuantile <- function(x, level = 0.05) {
+   apply(x, 1,
+         function(x) {
+           q <- quantile(x, c(level / 2, 1 - level / 2))
+           x[x < q[1]] <- q[1]
+           x[x > q[2]] <- q[2]
+
+           mean(x)
+         })
+ }
> data("iossa.err", package = "ssabook")
> lseq <- c(seq(0.03, 0.058, 0.002), seq(0.062, 0.1, 0.002))
> iter.real <- rowMeansQuantile(iossa.err$iter.real)
> iter.est <- iossa.err$iter.est
> err1 <- sqrt(rowMeansQuantile(iossa.err$err1))
> err2 <- sqrt(rowMeansQuantile(iossa.err$err2))
> xlab <- expression(omega[1])
> ylab1 <- expression(N[plain(iter)])
> ylab2 <- expression(RMSE)
> p1 <- xyplot(iter.real + iter.est ~ lseq,
+              type = "l", ylab = ylab1, xlab = xlab)
> p2 <- xyplot(err1 + err2 ~ lseq,
+              type = "l", ylab = ylab2, xlab = xlab)
> print(p1, split = c(1, 1, 1, 2), more = TRUE)
> print(p2, split = c(1, 2, 1, 2), more = FALSE)
```

Figure 2.19 (top) shows the number of iterations for noiseless signal (blue line) and the estimated mean number of iterations for the noisy signal (red line); the number of repetitions equals 1000, 5% winsorized estimates of means were calculated. Note that the number of iterations was limited by 200, although for the pure signal the convergence was achieved for each ω_1 from the considered set. A surprisingly small number of iterations for the noisy signal and close frequencies is explained by the convergence to a wrong limit, see Fig. 2.19 (bottom) with root mean square errors of LS-ESPRIT estimates for ω_1 and ω_2 based on the subspaces spanned by eigenvectors from ET1–2 and ET3–4 (see Algorithm 3.3 for the ESPRIT algorithms). Since we use the nested decomposition, the noise slightly influences the reconstruction accuracy for the frequencies that are quite different (ω_1 smaller than 0.048 and larger than 0.072).

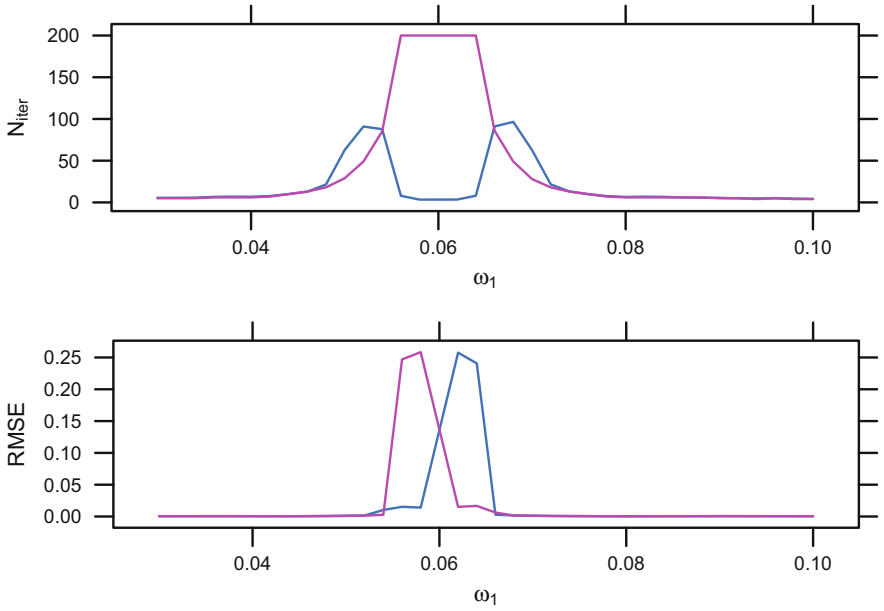

Fig. 2.19 Dependence of number of iterations (top) and RMSE errors of frequency estimations (bottom) on ω_1 for $\omega_2 = 0.6$

2.5 Filter-Adjusted O-SSA and SSA with Derivatives

2.5.1 Method

In this section we describe further variations of SSA that help to overcome the problem of lack of strong separability of components if weak separability holds.

Recall that the lack of strong separability of two series components is caused by equal singular values in the sets of the singular values generated by the two components. In turn, the singular values depend on the coefficients A_1 and A_2 in the representation of the signal as the sum $s_n = A_1 s_n^{(1)} + A_2 s_n^{(2)}$. The question is how to change the coefficients A_1 and A_2 with unknown $s_n^{(1)}$ and $s_n^{(2)}$ to make the singular values different.

It appears that it could be advantageous to use the derivative of the time series in order to change the coefficients without changing the component subspaces. For example, if $x_n = A\sin(2\pi\omega n + \phi)$, then $x'_n = 2\pi\omega A\cos(2\pi\omega n + \phi)$; that is, the new coefficient is $A' = 2\pi\omega A$. For two sinusoids with different frequencies, derivatives change their amplitudes differently. The derivative of $x_n = Ae^{\alpha n}$ also changes the coefficient before the exponential, since $x'_n = \alpha A e^{\alpha n}$, preserving the rate. For most of the series of finite rank, the derivative subspace coincides with

the series subspace. The exception is the polynomial series, when the derivative subspace is a subset of the initial subspace.

As we deal with discrete time, we consider the differences $\varphi_n(\mathbb{X}) = x_{n+1} - x_n$ instead of derivatives, but this is still an approach that seems to be working well. For example, for the series $\mathbb{X} = \mathbb{X}_N$ of length N with $x_n = A\sin(2\pi\omega n + \phi)$, the differences give us the series $\Phi_{N-1}(\mathbb{X}) = (\varphi_1(\mathbb{X}), \ldots, \varphi_{N-1}(\mathbb{X}))$ of length $N-1$ with $\varphi_n(\mathbb{X}) = 2\sin(\pi\omega)A\cos(2\pi\omega n + \pi\omega + \phi)$; for $x_n = Ae^{\alpha n}$, we obtain $\varphi_n(\mathbb{X}) = (e^{\alpha} - 1)Ae^{\alpha n}$.

Thus, we can combine the initial series and the series of its differences to change the balance for the component contributions and therefore to reach strong separability. For sinusoids with small periods, an increase of the sinusoid amplitudes is large. Therefore, taking derivatives (or differences) increases the contribution of high frequencies. This can also increase the level of the noise component, if the series is corrupted by a high-frequency noise. Hence, the nested version of the method implementation should be employed; in particular, the noise component should be removed by Basic SSA first.

The approach involving derivatives (that is, sequential differences) can be naturally extended to considering an arbitrary linear filtration φ instead of taking simple (sequential) differences. We start with the version with derivatives (that is, differences), since this particular case is simple and has very useful applications.

2.5.1.1 Nested O-SSA with Derivatives (DerivSSA)

Taking the sequential differences changes contributions of the components. Therefore, the method is inappropriate as an approximation method for signal extraction. Thus, the suggested method should be applied in a nested manner (see Sect. 2.4.1.2).

Let us formulate the nested version of O-SSA with derivatives called DerivSSA (Golyandina and Shlemov 2015). As well as in Sect. 2.4.1.2, let L be the window length, $K = N - L + 1$, and $\mathbf{Y} = \mathbf{X}_I \in \mathsf{R}^{L\times K}$ be one of the matrices in the decomposition $\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_p}$ obtained at Grouping step of Basic SSA; each group corresponds to a separated time series component and we want to construct a refined decomposition of $\mathbf{Y}$. As before, denote $r = \operatorname{rank}\mathbf{Y}$, $\mathbb{Y} = \mathfrak{T}^{-1} \circ \Pi_{\mathfrak{H}}(\mathbf{Y})$.

DerivSSA is similar to Basic SSA. Since DerivSSA is applied in a nested manner, the window length is already chosen. Therefore, DerivSSA adds only one additional parameter γ, which regulates the contribution of derivatives.

The DerivSSA method consists of decomposition and reconstruction. First we take sequential differences for each row of $\mathbf{Y} = [Y_1 : \ldots : Y_K]$ and hence compute the matrices $\Phi(\mathbf{Y}) = [Y_2 - Y_1 : \ldots : Y_K - Y_{K-1}] \in \mathsf{R}^{L\times(K-1)}$ and $\mathbf{Z} = [\mathbf{Y} : \gamma\Phi(\mathbf{Y})]$. Then DerivSSA works almost exactly as Basic SSA but it uses $\mathbf{Z}$ instead of the conventional trajectory matrix.

After Decomposition step, we obtain the SVD of $\mathbf{Z}$ in the form $\mathbf{Z} = \sum_{i=1}^{r}\sqrt{\lambda_i}U_iV_i^{\mathrm{T}}$. We are interested only in the first K columns in this matrix decomposition. Since the column space of $\mathbf{Z}$ coincides with the column space of

$\mathbf{Y}$ and therefore $\{U_i\}_{i=1}^r$ is a basis of the column space of $\mathbf{Y}$, we can rewrite the decomposition as $\mathbf{Y} = \sum_{i=1}^r U_i U_i^{\mathrm{T}} \mathbf{Y}$.

The rest of the method coincides with Reconstruction step of Basic SSA. After Grouping step, we obtain the decomposition $\mathbf{Y} = \mathbf{Y}_{J_1} + \ldots + \mathbf{Y}_{J_l}$ and then, as a result, the refined series decomposition $\mathbb{Y} = \widetilde{\mathbb{Y}}^{(1)} + \ldots + \widetilde{\mathbb{Y}}^{(l)}$, where $\widetilde{\mathbb{Y}}^{(m)} = \mathfrak{T}^{-1} \circ \Pi_{\mathcal{H}}(\mathbf{Y}_{J_m})$, $m = 1, \ldots, l$.

The following proposition shows that DerivSSA is a version of Oblique SSA with a specific pair of matrices $(\mathbf{L}, \mathbf{R})$, where $P_i = U_i$ and Q_i are normalized vectors $\mathbf{Y}^{\mathrm{T}} U_i$ in (2.15). Denote $\mathbf{I}_M$ the $M \times M$ identity matrix.

Proposition 2.1 (Golyandina and Shlemov (2015)) *The left singular vectors of the ordinary SVD of* $\mathbf{Z}$ *coincide with the left singular vectors of the* $(\mathbf{I}_L, \mathbf{R})$*-SVD of the input matrix* $\mathbf{Y}$*, where* $\mathbf{R}$ *is defined by the equality* $\mathbf{R} = \mathbf{I}_K + \gamma^2 \mathbf{F}^{\mathrm{T}} \mathbf{F}$ *and*

$$\mathbf{F} = \begin{pmatrix} -1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 1 & 0 \\ 0 & \cdots & 0 & 0 & -1 & 1 \end{pmatrix} \in \mathsf{R}^{(K-1)\times K}.$$

2.5.1.2 Filter-Adjusted O-SSA

Note that sequential differences, which are taken for each row of the matrix $\mathbf{Y}$, can be extended to an arbitrary linear filter of the rows. That is, we can choose coefficients of a linear filter $A = (a_1, \ldots, a_t)^{\mathrm{T}}$ and define $\Phi(\mathbf{Y}) = [Y_1^*, \ldots, Y_{K-t+1}^*]$, where $Y_i^* = a_1 Y_i + \ldots + a_t Y_{i+t-1}$. The rest of the respective version of SSA is the same as DerivSSA. DerivSSA corresponds to $t = 2$ and $A = (-1, 1)^{\mathrm{T}}$.

2.5.2 Separability

Let $\mathbb{X}_N = \mathbb{X}_N^{(1)} + \mathbb{X}_N^{(2)}$ and $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$ be of finite rank and approximately weakly separable which implies that their row and column trajectory spaces are approximately orthogonal. The same is then true for $\Phi_{N-1}(\mathbb{X}^{(1)})$ and $\Phi_{N-1}(\mathbb{X}^{(2)})$, due to the fact that their column spaces belong to the column spaces of $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$, while their row spaces are spanned by the vectors of the same structure that the vectors constituting bases of the row spaces of $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$, except that these basis vectors have length $K - 1$ and not K. Therefore, approximate orthogonality still holds. Since $\Phi_{N-1}(\mathbb{X}) = \Phi_{N-1}(\mathbb{X}^{(1)}) + \Phi_{N-1}(\mathbb{X}^{(2)})$, DerivSSA applied to $(\mathbb{X}_N, \gamma \Phi_{N-1}(\mathbb{X}))$ will approximately separate the time series $\mathbb{X}_N^{(1)}$ and $\mathbb{X}_N^{(2)}$.

Thus, DerivSSA does not worsen weak separability and can achieve strong separability. It is important to always keep in mind that DerivSSA increases the contribution of high-frequency components and decreases that for low-frequency components.

2.5.3 Algorithm

The general algorithm of Filter-adjusted O-SSA is described in two equivalent forms in Algorithms 2.9 and 2.10. Algorithm 2.9 directly follows the description of the method given in Sect. 2.5.1, while Algorithm 2.10 is more appropriate for an effective implementation and for a modification implemented in Algorithm 2.11.

Algorithm 2.9 Filter-adjusted O-SSA: decomposition

Input: Decomposition of the L-trajectory matrix $\mathbf{X} = \sum \sigma_i P_i Q_i^{\mathrm{T}}$, group of components I, $|I| = r$, filter coefficients $(a_1, \ldots, a_t)$, weight $\gamma > 0$.

Output: Decomposition of $\mathbf{Y} = \mathbf{X}_I$ on elementary matrices $\mathbf{Y} = \mathbf{Y}_1 + \ldots + \mathbf{Y}_r$, where $\mathbf{Y}_i = \sigma'_i P'_i (Q'_i)^{\mathrm{T}}$.

1: Form the matrix $\mathbf{Y} = \mathbf{X}_I = \sum_{i \in I} \sigma_i P_i Q_i^{\mathrm{T}}$.
2: Denote $\Phi(\mathbf{Y}) = [Y_1^*, \ldots, Y_{K-t+1}^*]$, where $Y_i^* = a_1 Y_i + \ldots + a_t Y_{i+t-1}$. Construct the matrix $\mathbf{Z} = [\mathbf{Y} : \gamma \Phi(\mathbf{Y})]$.
3: Compute the SVD of $\mathbf{Z}$: $\mathbf{Z} = \sum_{i=1}^{r} \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.
4: Construct the following decomposition of $\mathbf{Y} = \mathbf{X}_I$ into a sum of elementary matrices: $\mathbf{Y} = \sum_{i=1}^{r} U_i U_i^{\mathrm{T}} \mathbf{Y}$.
5: Obtain the decomposition $\mathbf{Y} = \sum_{i=1}^{r} \sigma'_i P'_i (Q'_i)^{\mathrm{T}}$, where $\sigma'_i = \|U_i^{\mathrm{T}} \mathbf{Y}\|$, $P'_i = U_i$, $Q'_i = U_i^{\mathrm{T}} \mathbf{Y} / \sigma'_i$.

Algorithm 2.10 Filter-adjusted O-SSA: decomposition (equivalent)

Input: Decomposition of the trajectory matrix $\mathbf{X} = \sum \sigma_i P_i Q_i^{\mathrm{T}}$, group of components I, $|I| = r$, filter coefficients $(a_1, \ldots, a_t)$, weight γ.

Output: Decomposition of $\mathbf{Y} = \mathbf{X}_I$ on elementary matrices $\mathbf{Y} = \mathbf{Y}_1 + \ldots + \mathbf{Y}_r$, where $\mathbf{Y}_i = \sigma'_i P'_i (Q'_i)^{\mathrm{T}}$.

1: Form the matrix $\mathbf{Y} = \mathbf{X}_I = \sum_{i \in I} \sigma_i P_i Q_i^{\mathrm{T}}$ and compute its thin SVD $\mathbf{Y} = \mathbf{U}_r \Lambda_r^{1/2} \mathbf{V}_r^{\mathrm{T}}$. Set $\mathbf{S} = [S_1 : \ldots : S_K] = \Lambda_r^{1/2} \mathbf{V}_r^{\mathrm{T}} \in \mathsf{R}^{r \times K}$.
2: Denote $\Phi(\mathbf{S}) = [S_1^*, \ldots, S_{K-t+1}^*]$, where $S_i^* = a_1 S_i + \ldots + a_t S_{i+t-1}$. Construct the matrix $\mathbf{Z} = [\mathbf{S} : \gamma \Phi(\mathbf{S})] \in \mathsf{R}^{r \times (2K-t+1)}$.
3: Calculate the rotation matrix $\widetilde{\mathbf{U}} \in \mathsf{R}^{r \times r}$ consisting of the eigenvectors of $\mathbf{Z}\mathbf{Z}^{\mathrm{T}}$ as columns.
4: Set $\widetilde{\mathbf{P}} = [\widetilde{P}_1 : \ldots : \widetilde{P}_r] = \mathbf{U}_r \widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{Q}} = [\widetilde{Q}_1 : \ldots : \widetilde{Q}_r] = \mathbf{S}^{\mathrm{T}} \widetilde{\mathbf{U}}$.
5: Obtain the decomposition $\mathbf{Y} = \sum_{i=1}^{r} \sigma'_i P'_i (Q'_i)^{\mathrm{T}}$, where $\sigma'_i = \|\widetilde{Q}_i\|$, $P'_i = \widetilde{P}_i$, $Q'_i = \widetilde{Q}_i / \sigma'_i$.

The method introduced in Sect. 2.5.1 has a modification implemented in Rssa, which can slightly worsen the separability but has an advantage that it orders the eigentriples corresponding to sine-waves exactly by the decrease of their frequencies, independently of the values of the sine-wave amplitudes. We will call

this modification "Filter-adjusted O-SSA with normalization." The main difference between Algorithms 2.10 and 2.11 is in the construction of the matrix $\mathbf{S}$ at step 1.

Algorithm 2.11 Filter-adjusted O-SSA with normalization: decomposition

Input: Decomposition of the trajectory matrix $\mathbf{X} = \sum \sigma_i P_i Q_i^{\mathrm{T}}$, group of components I, $|I| = r$, filter coefficients $(a_1, \ldots, a_t)$, weight γ.

Output: Decomposition of $\mathbf{Y} = \mathbf{X}_I$ on elementary matrices $\mathbf{Y} = \mathbf{Y}_1 + \ldots + \mathbf{Y}_r$, where $\mathbf{Y}_i = \sigma'_i P'_i (Q'_i)^{\mathrm{T}}$.

1: Form the matrix $\mathbf{Y} = \mathbf{X}_I = \sum_{i \in I} \sigma_i P_i Q_i^{\mathrm{T}}$ and construct its thin SVD $\mathbf{Y} = \mathbf{U}_r \Lambda_r^{1/2} \mathbf{V}_r^{\mathrm{T}}$. Set $\mathbf{S} = [S_1 : \ldots : S_K] = \mathbf{V}_r^{\mathrm{T}} \in \mathsf{R}^{r \times K}$.

2: Denote $\Phi(\mathbf{S}) = [S_1^*, \ldots, S_{K-t+1}^*]$, where $S_i^* = a_1 S_i + \ldots + a_t S_{i+t-1}$. Construct the matrix $\mathbf{Z} = [\mathbf{S} : \gamma \Phi(\mathbf{S})] \in \mathsf{R}^{r \times (2K-t+1)}$.

3: Calculate the rotation matrix $\widetilde{\mathbf{U}} \in \mathsf{R}^{r \times r}$ consisting of the eigenvectors of $\mathbf{Z}\mathbf{Z}^{\mathrm{T}}$ as columns.

4: Set $\widetilde{\mathbf{P}} = [\widetilde{P}_1 : \ldots : \widetilde{P}_r] = (\mathbf{U}_r \Lambda_r^{1/2}) \widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{Q}} = [\widetilde{Q}_1 : \ldots : \widetilde{Q}_r] = \mathbf{S}^{\mathrm{T}} \widetilde{\mathbf{U}}$.

5: Obtain the decomposition $\mathbf{Y} = \sum_{i=1}^r \sigma'_i P'_i (Q'_i)^{\mathrm{T}}$, where $\sigma'_i = \|\widetilde{P}_i\|$, $P'_i = \widetilde{P}_i / \sigma'_i$, $Q'_i = \widetilde{Q}_i$.

Remark 2.6 Algorithms 2.9–2.11 can be extended to the case when several filters in a stacked manner are applied. For example, if filters Φ_1 and Φ_2 are given, then the matrix $\mathbf{Z}$ at Step 2 has the forms $\mathbf{Z} = [\mathbf{Y} : \gamma \Phi_1(\mathbf{Y}) : \gamma \Phi_2(\mathbf{Y})]$ and $\mathbf{Z} = [\mathbf{S} : \gamma \Phi_1(\mathbf{S}) : \gamma \Phi_2(\mathbf{S})]$, respectively.

The reconstruction algorithm is the same as for most versions of SSA. Since Filter-adjusted O-SSA (as well as DerivSSA) is a nested method, the result is a decomposition of a chosen series component rather than a decomposition of the original series.

Algorithm 2.12 Filter-adjusted O-SSA: reconstruction

Input: Decomposition $\mathbf{Y} = \sum_{i=1}^r \mathbf{Y}_i$, where $\mathbf{Y}_i = \sigma'_i P'_i (Q'_i)^{\mathrm{T}}$, grouping $I = \bigsqcup_{k=1}^l J_k$.

Output: Refined series decomposition $\mathbb{Y} = \widetilde{\mathbb{Y}}^{(1)} + \ldots + \widetilde{\mathbb{Y}}^{(l)}$.

1: Obtain the grouped matrix decomposition $\mathbf{Y} = \mathbf{Y}_{J_1} + \ldots + \mathbf{Y}_{J_l}$, where $\mathbf{Y}_J = \sum_{j \in J} \mathbf{Y}_j$.

2: Obtain a refined series decomposition $\mathbb{Y} = \widetilde{\mathbb{Y}}^{(1)} + \ldots + \widetilde{\mathbb{Y}}^{(l)}$, where $\widetilde{\mathbb{Y}}^{(m)} = \mathfrak{T}^{-1} \circ \Pi_{\mathfrak{H}}(\mathbf{Y}_{J_m})$, $m = 1, \ldots, l$.

Remark 2.7 Algorithm 2.12 describes a refined decomposition of the matrix $\mathbf{X}_I$. However, we can consider Filter-adjusted O-SSA as an algorithm, which we apply to the full decomposition of the trajectory matrix $\mathbf{X}$ of an original series $\mathbb{X}$, which changes components from the group I. Then the result is also a full decomposition of $\mathbf{X}$.

2.5.4 *Filter-Adjusted O-SSA in* Rssa

2.5.4.1 Description of Functions

As in the case of Iterative O-SSA, since Filter-adjusted O-SSA is a nested method, the `ssa` function must be called prior to this in order to obtain an `ssa` object `s`, see "Description of function" in Sects. 2.1–2.3, 2.6. For Filter-adjusted O-SSA itself, the function `fossa` is used. Since the result of `fossa` is also an `ssa` object, which contains the full decomposition, the `fossa` and `iossa` functions can be applied to the result of another application of `fossa`.

Let us outline the main arguments of `fossa` in a typical function call:

```
fos <- fossa(s, nested.groups = list(2:3, 6:7),
             filter = c(-1,1), gamma = 1, normalize = TRUE)
```

Arguments:

`s` is an `ssa` object holding the full one-dimensional SSA (Basic SSA, Toeplitz SSA, Basic SSA with projections, Shaped SSA) decomposition.

`nested.groups` is a vector of indices of eigentriples from the full decomposition for the nested decomposition. The argument is coerced to a vector, if necessary.

`filter` is a list of numeric vectors of reverse impulse response coefficients for filter adjustment. Value by default `c(-1,1)` corresponds to DerivSSA, which is the most common case.

`gamma` is the weight of filter adjustment; the value `Inf` corresponds to the removal of the first part of the matrix $\mathbf{Z}$: $\mathbf{Z} = \Phi(\mathbf{S})$.

`normalize` indicates if the modification with normalization is used (see Algorithm 2.11).

Note that for Filter-adjusted SSA, only a group I should be given; the partition is performed later at Reconstruction step. Therefore, the list of groups, which are given for the parameter `nested.groups`, is transformed to a single vector of indices composing I (compare with `iossa`). That is, in the considered function call, $I = \{2, 3, 5, 6\}$.

The return value is an object of class `ossa`. The field `ossa.set` contains the vector of indices of elementary components used in Filter-adjusted O-SSA (that is, used in `nested.groups`, which is in fact just I). For example, to look at weighted oblique correlations of the obtained elementary components, one can call `owcor(fos, groups = fos$ossa.set)`.

2.5.4.2 Typical Code

This example demonstrates the difference between Filter-adjusted O-SSA and Iterative O-SSA with sigma-correction. Let us consider a noisy sum of two sinusoids with different and not close frequencies (see Fragment 2.5.1). These sinusoids are approximately separable. However, since the sinusoid amplitudes are equal, there

is no strong separability and therefore after application of Basic SSA we obtain an unsatisfactory decomposition, an arbitrary mixture of the sinusoids (the top pictures of Fig. 2.21 with eigenvectors).

Fragment 2.5.1 (Separation of Two Sine Waves with Equal Amplitudes)

```
> N <- 100
> L <- 50
> omega1 <- 0.03
> omega2 <- 0.06
> sigma <- 0.1
> set.seed(3)
> F <- sin(2 * pi * omega1 * (1:N)) +
+   sin(2 * pi * omega2 * (1:N)) +
+   sigma * rnorm(N)
> s <- ssa(F, L = L, neig = min(L, N - L + 1)) #full decomposition
> plot(s)
> p1 <- plot(s, type = "vectors", idx = 1:4, layout = c(4, 1),
+            main = "Eigenvectors, Basic SSA")
> fos <- fossa(s, nested.groups = list(1:2, 3:4), gamma = 10,
+              normalize = FALSE)
> # The total percent is equal to 100%
> print(sum(fos$sigma^2) / sum(s$sigma^2) * 100)
[1] 100
> p2 <- plot(fos, type = "vectors", idx = 1:4, layout = c(4, 1),
+            main = "Eigenvectors, SSA with derivatives")
> ios1 <- iossa(s, nested.groups = list(1:2, 3:4), maxiter = 1)
> # The total percent is not equal to 100%
> print(sum(ios1$sigma^2) / sum(s$sigma^2) * 100)
[1] 99.62939
> p3 <- plot(ios1, type = "vectors", idx = 1:4, layout = c(4, 1),
+            main = "Eigenvectors, Iterative O-SSA, 1 iter")
> ios2 <- iossa(ios1, nested.groups = list(1:2, 3:4), maxiter = 1)
> # The total percent is not equal to 100%
> print(sum(ios2$sigma^2) / sum(s$sigma^2) * 100)
[1] 101.7544
> p4 <- plot(ios2, type = "vectors", idx = 1:4, layout = c(4, 1),
+            main = "Eigenvectors, Iterative O-SSA, 2 iter")
> plot(p1, split = c(1, 1, 1, 4), more = TRUE)
> plot(p2, split = c(1, 2, 1, 4), more = TRUE)
> plot(p3, split = c(1, 3, 1, 4), more = TRUE)
> plot(p4, split = c(1, 4, 1, 4), more = FALSE)
> fo.rec <- reconstruct(fos, groups = list(1:2, 3:4))
> pr1 <- plot(fo.rec, plot.method = "xyplot",
+             main = "SSA with derivatives", xlab = "")
> io.rec <- reconstruct(ios2, groups = ios2$iossa.groups)
> pr2 <- plot(io.rec, plot.method = "xyplot",
+             main = "Iterative O-SSA", xlab = "")
> plot(pr1, split = c(1, 1, 1, 2), more = TRUE)
> plot(pr2, split = c(1, 2, 1, 2), more = FALSE)
```

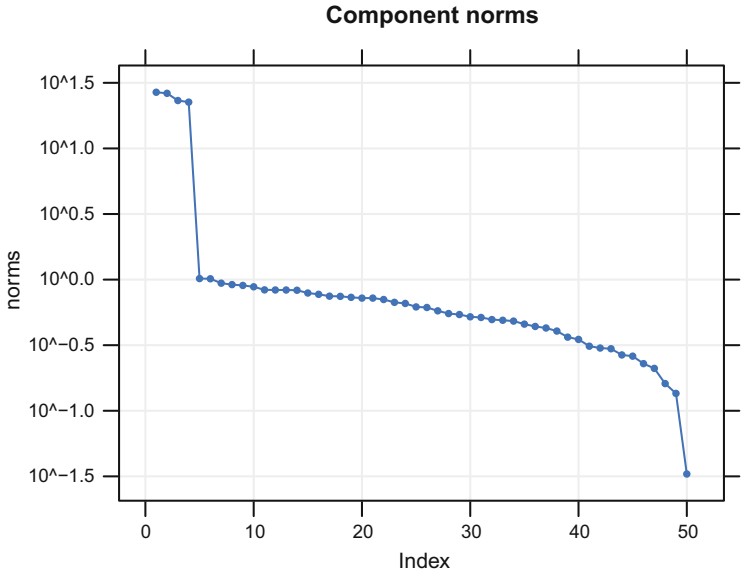

Fig. 2.20 Noisy sum of sinusoids: Graph of eigenvalues for Basic SSA

To apply a nested version of Oblique SSA, we should start with the signal subspace extraction. Figure 2.20 confirms that the signal is contained in eigentriples 1–4, since the corresponding eigenvalues are considerably larger than the eigenvalues of the residual components.

Thus, we apply SSA with derivatives to the group ET1–4, taking a large enough $\gamma = 10$. The eigenvectors become regular (Fig. 2.21, top) and the reconstruction is accurate (Fig. 2.22, top).

The Iterative O-SSA algorithm is designed to improve weak separability. Let us also use its ability to change component contribution during iterations by means of the sigma-correction. One can see that one iteration slightly improves the decomposition, while the second iteration provides almost ideal decomposition (see Fig. 2.21 with eigenvectors and Fig. 2.22 with reconstruction).

Comparing the results in this example, we see that among the versions of the two methods with the same computational cost, DerivSSA is better; however, one more iteration in Iterative O-SSA makes Iterative O-SSA advantageous to DerivSSA.

Let us draw attention to different order of the sinusoids in the reconstructions (Fig. 2.22). The order of components produced by DerivSSA is explained by an increase of contribution of high frequencies due to taking the differences, while the order of components in Iterative O-SSA is more or less random in this example.

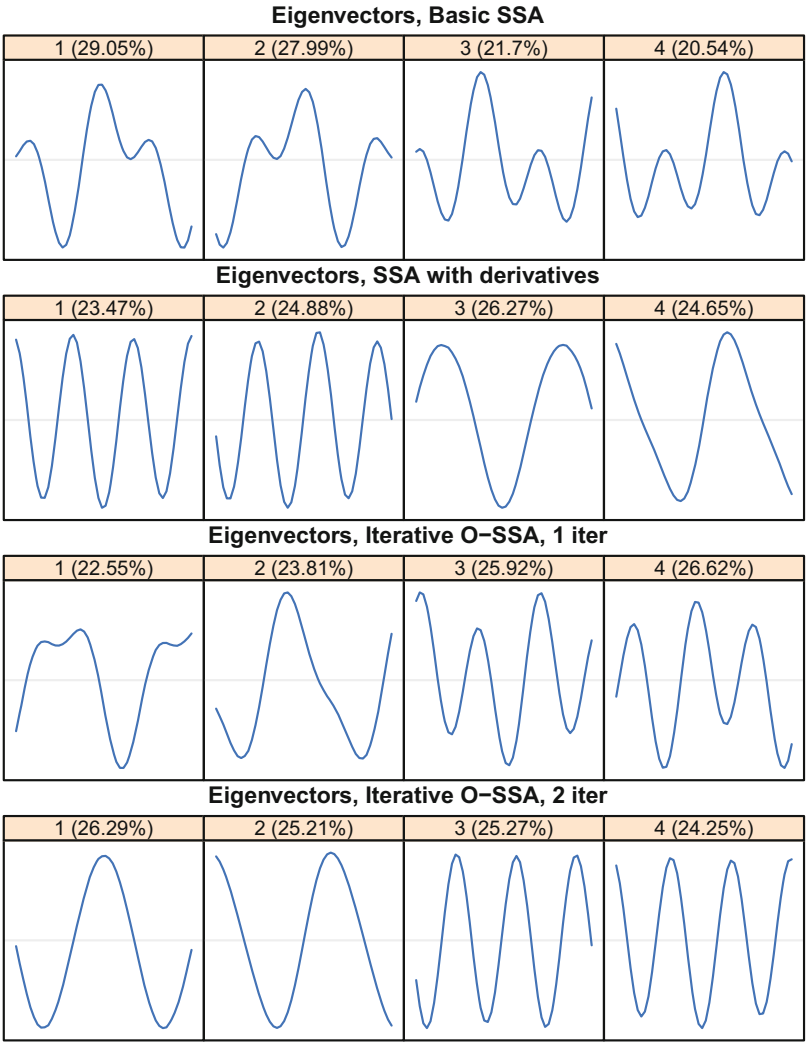

Fig. 2.21 Noisy sum of sinusoids: 1D graphs of eigenvectors for Basic SSA (top), DerivSSA (middle) and Iterative O-SSA, 1 iteration (second from bottom) and 2 iterations (bottom)

For DerivSSA, the decomposition is F-orthogonal and therefore the contributions of series components are correct. Iterative O-SSA may have the sum of contributions different from 100%. This is exactly the case in our example, see the last warning message:

```
  In .contribution(x, idx, ...): Elementary matrices are not
F-orthogonal (max F-cor is -0.016). Contributions can be
irrelevant.
```

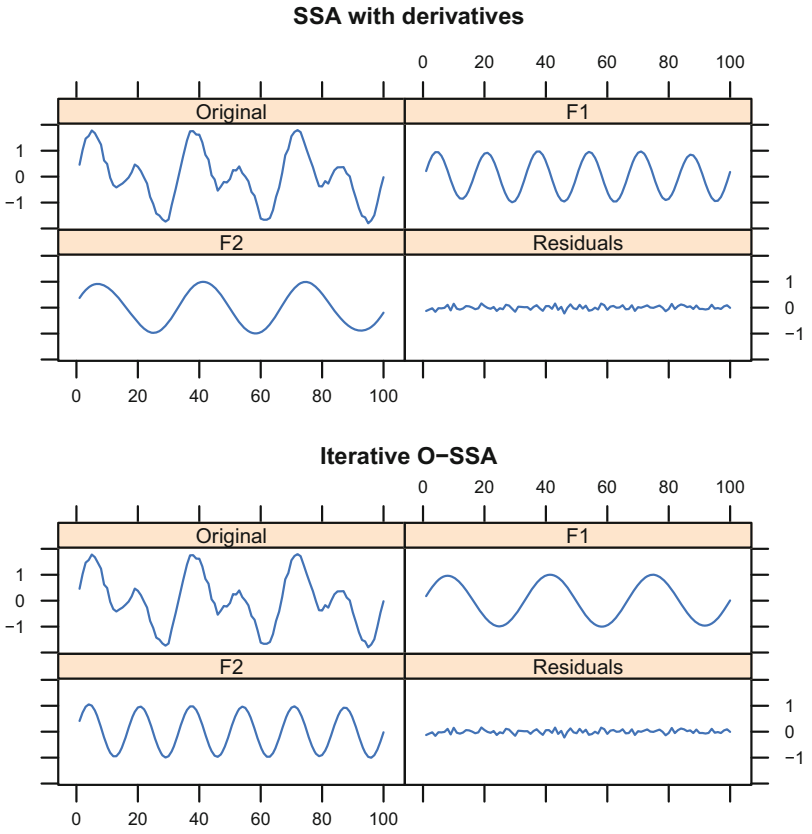

Fig. 2.22 Noisy sum of sinusoids: Reconstructions for DerivSSA (top) and Iterative O-SSA, 2 iterations (bottom)

2.6 Shaped 1D-SSA

2.6.1 Method

Shaped SSA is a very general SSA method. Formally, any other SSA method can be considered as a particular case of Shaped SSA. For multivariate extensions, the versions of Shaped SSA, which cannot be reduced to conventional SSA, have many different applications. For one-dimensional time series, Shaped SSA is particularly useful when the time series contains gaps as in this case the versions of 1D-SSA considered above are not directly applicable.

The specificity of Shaped SSA is in the construction of the L-trajectory matrix, which we denote $\widetilde{\mathbf{X}} = \mathcal{T}_{\mathrm{shSSA}}(\mathbb{X})$. This matrix is constructed so that its columns are the complete L-lagged vectors. Any incomplete lagged vectors containing missing values are not included into $\mathcal{T}_{\mathrm{shSSA}}(\mathbb{X})$.

Denote the set of series elements, which are presented in the trajectory matrix $\widetilde{\mathbf{X}}$, as $\mathfrak{N}$; that is, $\mathfrak{N}$ is the set of non-missed elements of $\mathbb{X}$, which are covered by windows of length L. The operator $\mathfrak{T}_{\mathrm{shSSA}}$ makes a one-to-one correspondence between a restriction of the series to $\mathfrak{N}$ and the set of trajectory matrices, if the location of the missing data is fixed.

The SSA decomposition is performed by any technique (excluding Toeplitz SSA) described in Sects. 2.1–2.5 including nested Iterative O-SSA and Filter-adjusted O-SSA. All these SSA decompositions, except for Toeplitz SSA, are eligible tools since construction of the trajectory matrix and a tool for SSA decomposition do not affect one another. Thus, we can naturally define shaped Basic SSA, shaped SSA with projection, and so on.

For Toeplitz SSA, the decomposition is performed in a very specific way, which is not directly based on the trajectory matrix; thereby the shaped version of Toeplitz SSA does not make much sense and hence it is not implemented in the current version of Rssa.

After decomposition of the trajectory matrix into a sum of elementary matrices and then into a sum of grouped matrices, we need to obtain the series decomposition. Generally speaking, the trajectory matrix is not Hankel in view of the gaps. Therefore, we need a more general procedure than the hankelization and diagonal averaging. This procedure is determined by the operator $\Pi_{\mathcal{H},\mathrm{sh}}(\cdot)$ which is defined as follows.

For ith term of the series, where $i \in \mathfrak{N}$, denote $\mathbb{E}_i$ the series with zeros everywhere except ith term, which is equal to 1, $\mathbf{B}_i = \mathfrak{T}_{\mathrm{shSSA}}(\mathbb{E}_i)$. For a given matrix $\mathbf{Y}$, define a series $\widetilde{\mathbb{Y}}$ by $\widetilde{y}_i = \langle \mathbf{Y}, \mathbf{B}_i \rangle_{\mathrm{F}} / \|\mathbf{B}_i\|^2$ which gives

$$\Pi_{\mathcal{H},\mathrm{sh}}(\mathbf{Y}) = \mathfrak{T}_{\mathrm{shSSA}}(\widetilde{\mathbb{Y}}).$$

2.6.2 Separability

Definition and conditions of separability for Shaped SSA are the same as for underlying modifications of SSA (see, e.g., Sect. 2.1.3). However, the separability accuracy is naturally worse when there are gaps in the series. Moreover, there are extreme cases where ranks of series can be corrupted by gaps and where the conditions of separability cannot be satisfied.

2.6.3 Algorithm

The SSA modifications described in Sects. 2.2–2.5 differ from Basic SSA (Sect. 2.1) by Decomposition step, while Shaped SSA differs from these modifications by Embedding step, which depends on the window shape and the shape of the analyzed object. Therefore, we consider Shaped SSA as an extension of SSA. Let us describe the Shaped SSA algorithm for the analysis of time series with gaps.

Algorithm 2.13 Shaped SSA: decomposition

Input: Time series $\mathbb{X}$ of length N with missing data, window length L, an SSA modification for making a decomposition.
Output: Decomposition of the trajectory matrix on elementary matrices $\widetilde{\mathbf{X}} = \widetilde{\mathbf{X}}_1 + \ldots + \widetilde{\mathbf{X}}_d$, where $\widetilde{\mathbf{X}}_i = \sigma_i P_i Q_i^{\mathrm{T}}$.
1: Construct the trajectory matrix $\widetilde{\mathbf{X}} = \mathfrak{T}_{\mathrm{shSSA}}(\mathbb{X})$.
2: Obtain the decomposition $\widetilde{\mathbf{X}} = \sum_{i=1}^{d} \widetilde{\mathbf{X}}_i$, where $\widetilde{\mathbf{X}}_i = \sigma_i P_i Q_i^{\mathrm{T}}$, by means of Decomposition step of the chosen SSA modification.

Since Decomposition stage differs by Embedding step, which transfers a time series with gaps into a trajectory matrix, Reconstruction stage differs by the way a matrix is transferred to a time series with gaps.

Algorithm 2.14 Shaped SSA: reconstruction

Input: Decomposition $\widetilde{\mathbf{X}} = \widetilde{\mathbf{X}}_1 + \ldots + \widetilde{\mathbf{X}}_d$, $\widetilde{\mathbf{X}}_i = \sigma_i P_i Q_i^{\mathrm{T}}$, grouping $\{1, \ldots, d\} = \bigsqcup_{j=1}^{m} I_j$.
Output: Decomposition of the time series on identifiable components $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$.
1: Construct the grouped matrix decomposition $\widetilde{\mathbf{X}} = \widetilde{\mathbf{X}}_{I_1} + \ldots + \widetilde{\mathbf{X}}_{I_m}$, where $\widetilde{\mathbf{X}}_I = \sum_{i \in I} \widetilde{\mathbf{X}}_i$.
2: $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$, where $\mathbb{X}_j = \mathfrak{T}_{\mathrm{shSSA}}^{-1} \circ \Pi_{\mathcal{H},\mathrm{sh}}(\widetilde{\mathbf{X}}_{I_j})$, $j = 1, \ldots, m$.

We have assumed here that all series points can be covered by the window of the chosen length. If it is not so, then we obtain the reconstruction given only on the covered points.

2.6.4 Shaped SSA in RSSA

2.6.4.1 Description of Functions

There is no specific form of the `ssa` function for Shaped SSA. If the series has gaps (that is, it contains `NA` values), then the shaped version of Basic SSA is applied by default.

In the 1D case as well as in the case of 2D-SSA, specific masks for the window shape can be done. For example, the window can be not a whole interval but an interval with a gap. Since windows with gaps do not have much sense in the one-dimensional case we do not discuss general shaped window in this chapter; see Sect. 5.2 on how to set shaped windows for 2D-data.

2.6.4.2 Typical Code

Let us consider the time series “CO2” and set up several artificial missing values.

Fragment 2.6.1 (Decomposition for Series with a Gap)

```
> F <- co2; F[100:200] <- NA
> # Prompt for the choice of window length
> clplot(F)
> # Perform shaped SSA
> s1 <- ssa(F, L = 72)
> plot(s1, type = "vectors", idx = 1:12)
> plot(s1, type = "series", groups = 1:6, layout = c(2, 3))
> plot(wcor(s1, groups = 1:20), scales = list(at = seq(1,20,2)))
> plot(reconstruct(s1, groups = list(c(1, 4, 7))),
+      add.residuals = FALSE,
+      plot.method = "xyplot", superpose = TRUE)
```

Fragment 2.6.1 demonstrates that the code looks similar to the code for Basic SSA. However, there is a specificity. First, the choice of the window length should take into consideration the number of complete vectors. The function `clplot` (Fig. 2.23) shows the proportions of complete vectors; this proportion is equal to 1 only if there are no gaps. Eigenvectors serve for component identification in the same way as for the series with no gaps (Fig. 2.24). Note that plotting the factor vectors can be misleading, since factor vectors are depicted point-by-point and therefore the gaps are not visible. The reconstructed series, however, are drawn correctly, with the gaps (Figs. 2.25, 2.27). Weighted correlations, as usual, help for component identification (Fig. 2.26).

Since in Basic SSA large window lengths can provide better accuracy, Fragment 2.6.2 performs reconstruction with window length $L = 120$. This window

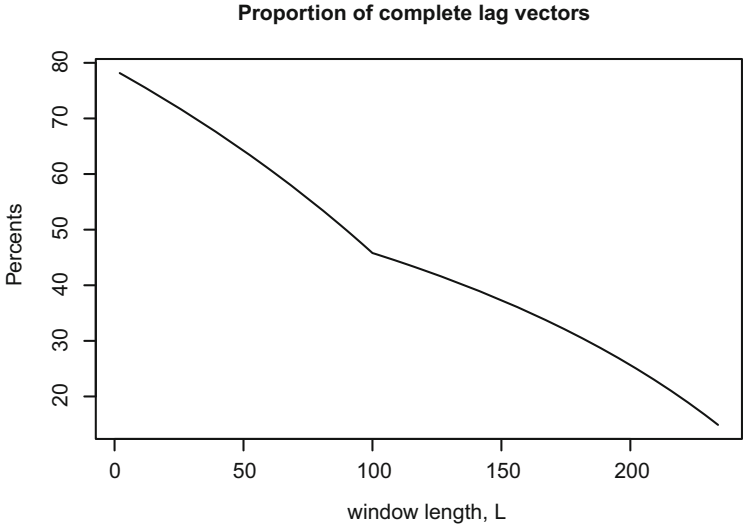

Fig. 2.23 "CO2" with gaps, Shaped SSA: Dependence of proportion of complete vectors on window length

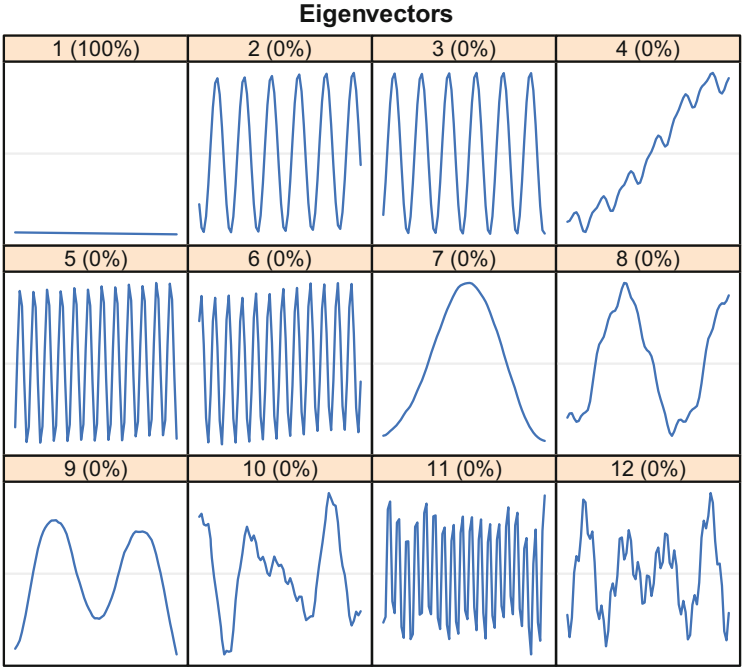

Fig. 2.24 "CO2" with gaps, Shaped SSA: Eigenvectors, $L = 72$

cannot cover the series points on the left from the gap and therefore the left part of the series cannot be reconstructed (Fig. 2.28).

Fragment 2.6.2 (Incomplete Decomposition for a Series with a Gap)

```
> s2 <- ssa(F, L = 120)
> # plot(s2, type = "vectors")
> # plot(wcor(s2, groups = 1:20))
> # plot of reconstruction
> plot(reconstruct(s2, groups = list(c(1, 6, 7))),
+      add.residuals = FALSE,
+      plot.method = "xyplot", superpose = TRUE)
```

2.7 Automatic Grouping in SSA

2.7.1 Methods

While the choice of the window length is well supported by the SSA theory, the procedure for choosing the eigentriples for grouping is much less formal. Several methods for component identification and automatic grouping are described in

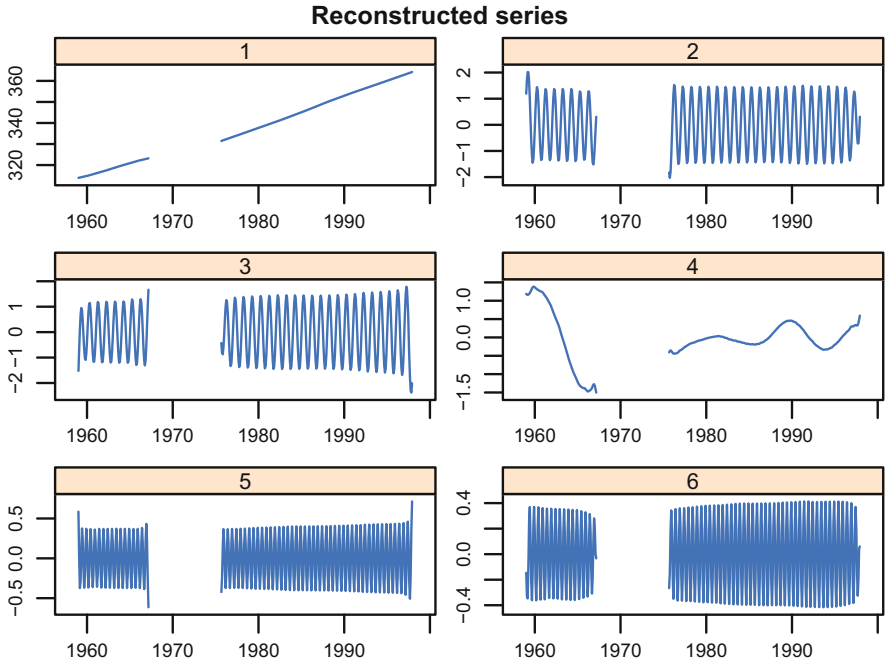

Fig. 2.25 "CO2" with gaps, Shaped SSA: Elementary reconstructed series, $L = 72$

Golyandina and Zhigljavsky (2013; Section 2.4.5). Let us shortly discuss these methods and the basic principles of automatic grouping.

Automatic grouping assumes that the components to be identified are (approximately) separated between themselves and from the residual. Grouping is based on finding common features in the components. The first method measures the communality of components by means of the **w**-correlations $\rho_{ij}^{(\mathbf{w})}$, see (1.7), between them: if a weighted correlation is large, then the corresponding components have similar behavior and should be included into the same group. This approach is very similar to the so-called correlation clustering of variables in multivariate statistics. The input dissimilarity matrix for clustering methods contains values $1 - |\rho_{ij}^{(\mathbf{w})}|$.

The second method is based on finding components with similar frequency characteristics. Contribution of frequencies is defined through the periodogram

$$\Pi_y^M(k/M) = \begin{cases} c_0^2 & \text{for } k = 0, \\ (c_k^2 + s_k^2)/2 & \text{for } 0 < k < M/2, \\ c_{M/2}^2 & \text{for } k = M/2 \text{ if } M \text{ is even}, \end{cases} \tag{2.19}$$

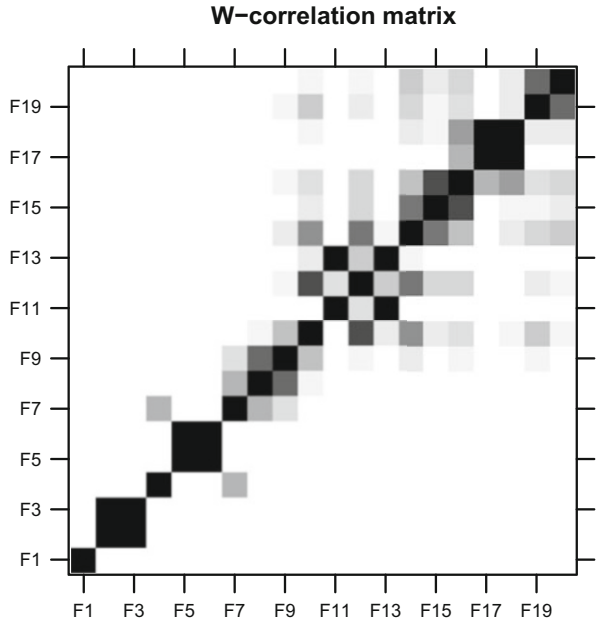

Fig. 2.26 "CO2" with gaps, Shaped SSA: **w**-Correlation matrix, $L = 72$

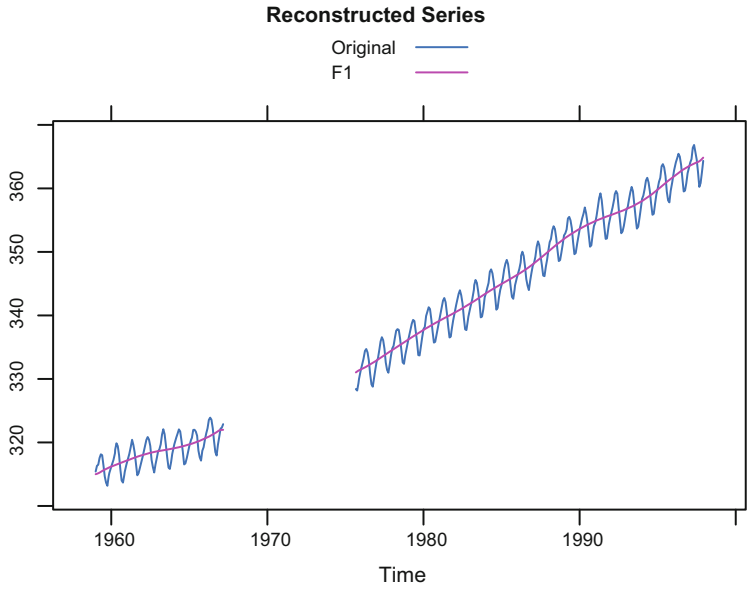

Fig. 2.27 "CO2" with gaps, Shaped SSA: Trend reconstruction, $L = 72$

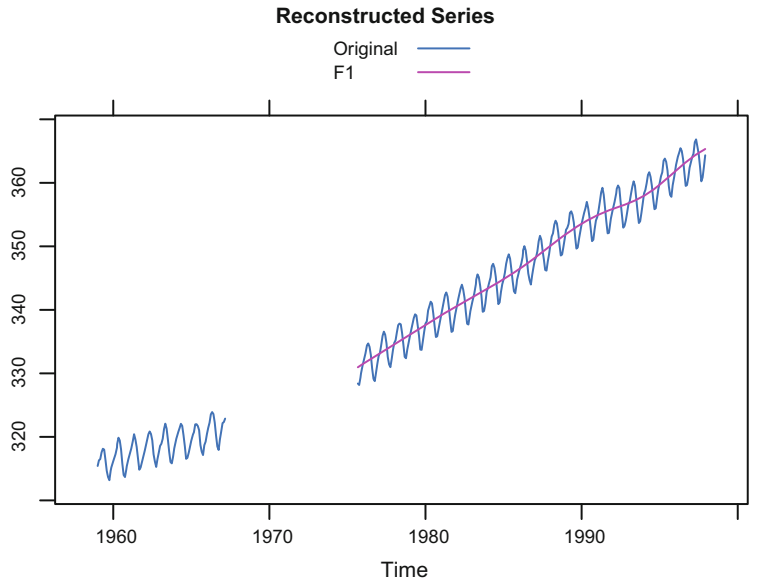

Fig. 2.28 "CO2" with gaps, Shaped SSA: Incomplete trend reconstruction, $L = 120$

where the coefficients c_k and s_k are taken from the Fourier decomposition of $\mathbb{Y} = (y_1, \ldots, y_M)$:

$$y_n = c_0 + \sum_{k=1}^{\lfloor M/2 \rfloor} \Big(c_k \cos(2\pi n\, k/M) + s_k \sin(2\pi n\, k/M) \Big),$$

For a series $\mathbb{Y}$ of length M and for $0 \le \omega_1 \le \omega_2 \le 0.5$, we define

$$T(\mathbb{Y}; \omega_1, \omega_2) = \sum_{k:\omega_1 \le k/M < \omega_2} I_y^M(k/M), \tag{2.20}$$

where $I_y^M(k/M) = M\, \Pi_y^M(k/M)/\|\mathbb{Y}\|^2$, Π_y^M is defined in (2.19). Since $\|\mathbb{Y}\|^2 = M \sum_{k=1}^{[M/2]} \Pi_y^M(k/M)$, the measure $T(\mathbb{Y}; \omega_1, \omega_2)$ can be considered as a proportion of frequencies contained in the frequency bin $[\omega_1, \omega_2)$.

One of the aims in performing grouping is the extraction of a series component with frequency range mostly from the chosen frequency bin. Therefore, it is natural to calculate the value of T for elementary reconstructed components. Moreover, SSA reconstruction can be considered as a linear filter. It appears that the frequency response of the filter generated by the ith eigentriple is almost the same as the periodogram of the corresponding singular vector, see Golyandina and Zhigljavsky (2013; Proposition 3.13). Therefore, it is reasonable to apply T also to singular vectors to reconstruct the series components with the given frequency ranges.

Since the trend of a series can be defined as its slowly varying series component, for extracting a trend a frequency bin in the form $[0, \omega)$ should be chosen. The value of ω reflects the frequency range, which we associated with a trend. For example, if the series has monthly seasonality, ω should be notably smaller than $1/12$. Note that the grouping method does not answer the question whether the extracted component is indeed a deterministic trend or simply a result of smoothing.

We can also consider several frequency bins, perhaps overlapping; in this case, the described method can be applied to each bin separately. A modification of grouping with several bins can be suggested.

Let the whole frequency range be divided into disjoint bins. Then we can refer a component to a frequency bin with the largest proportions of the corresponding frequency range; that is, with the maximal value of T. In this modification, all the considered bins participate simultaneously and are hence dependent. This modification can be used for splitting the series into a set of the components according to the specified frequency ranges.

Values of T for each elementary decomposition component can be used for devising the grouping. To perform an automatic grouping, a threshold T_0, $0 \le T_0 \le 1$, should be given. For example, if the value $T(\mathbb{Y}_i; 0, \omega)$ is larger than T_0 for some small ω, where $\mathbb{Y}_i$ is the ith elementary series or ith left/right singular vector, then the corresponding eigentriple can be automatically considered as a part of trend.

2.7.2 Algorithm

The algorithm of auto-grouping, which uses the **w**-correlation matrix, supplements an algorithm which performs clustering based on the (dis)similarity matrix.

Algorithm 2.15 Auto-grouping: Clustering

Input: **w**-Correlation matrix $[\rho_{ij}^{(\mathbf{w})}]$ between reconstructed components, number of groups.
Output: Groups of components.

1: Use a method of cluster analysis to the dissimilarity measure defined by $1 - |\rho_{ij}^{(\mathbf{w})}|$.
2: Obtain the given number of groups from the results of cluster analysis.

The algorithm of component identification based on frequency characteristics of the components is implemented in two versions. In the first version, each frequency interval is considered separately and the desired components are selected by comparing the values of (2.20) to a given threshold. For simplicity, we formulate Algorithm 2.16 for one frequency interval only. The second version, Algorithm 2.17, uses the set of frequency intervals simultaneously.

Algorithm 2.16 Auto-grouping: Frequency ranges, by the threshold

Input: Frequency range $[\omega_1, \omega_2)$, threshold T_0, group I, type of series: eigenvectors, factor vectors or reconstructed series.

Output: A group of components $J \subset I$.

1: For each series $\mathbb{Y}_i, i \in I$, the measure $T(\mathbb{Y}_i; \omega_1, \omega_2)$ given in (2.20) is calculated.
2: The resultant group J consists of indices $i \in I$ such that $T(\mathbb{Y}_i; \omega_1, \omega_2) \geq T_0$.

Algorithm 2.17 Auto-grouping: Frequency ranges, by the maximal contribution

Input: Set of frequency ranges $[\omega_1^{(m)}, \omega_2^{(m)})$, $m = 1, \ldots, k$ (if the separating points $0 = \omega_0 < \omega_1 < \omega_2 < \ldots < \omega_k$ for frequencies are given, then $[\omega_1^{(m)}, \omega_2^{(m)}) = [\omega_{m-1}, \omega_m)$), group I, type of series $\mathbb{Y}_i$: eigenvectors, factor vectors or reconstructed series.

Output: Set of k groups $J_m \subset I$, $I = \bigsqcup_m J_m$.

1: For each series $\mathbb{Y}_i, i \in I$, and each frequency interval $[\omega_1^{(m)}, \omega_2^{(m)})$, $m = 1, \ldots, k$, the measure $T(\mathbb{Y}_i; \omega_1^{(m)}, \omega_2^{(m)})$ given in (2.20) is calculated.
2: Each index i, $i \in I$, is referred to a group J_{m_0} with the maximal value of the measures $T(\mathbb{Y}_i; \omega_1^{(m)}, \omega_2^{(m)})$, $m \in \{1, \ldots, k\}$. As a result, k groups are formed.

2.7.3 Automatic Grouping in RSSA

2.7.3.1 Description of Functions

Let us outline the main arguments of `grouping.auto` for Algorithm 2.15 in typical function calls (the two grouping calls below are equivalent):

```
g <- grouping.auto(s, nclust = 2, groups = 1:20,
                   method = "complete", grouping.method = "wcor")
g <- grouping.auto.wcor(s, nclust = 2,
                        groups = 1:20, method = "complete")
```

Arguments

`s` is an `ssa` object holding the full one-dimensional SSA (Basic SSA, Toeplitz SSA, SSA with projection, Shaped SSA, DerivSSA, Iterative O-SSA) decomposition.

`grouping.method` is a method for automatic grouping.

`groups` is a list of groups, which is coerced to a vector with component numbers to obtain the elementary reconstructed components and calculate the **w**-correlation matrix for them.

`nclust` is a number of clusters.

`method` determines the way of cluster amalgamation; `method` is a parameter of the `R` function `hclust` from the `STATS` package, which performs the hierarchical cluster analysis.

The result of the function `grouping.auto.wcor` can be depicted by the function call `plot(g)` in the form of a hierarchical tree.

For application of Algorithms 2.16 and 2.17, the typical call is

```
gp <- grouping.auto(s, groups = 1:20, base = "series",
                    freq.bins = list(0.01,0.02),
                    threshold = 0.8,
                    grouping.method = "pgram")
```

Arguments

`s` is an `ssa` object holding the full one-dimensional SSA (Basic SSA, Toeplitz SSA, SSA with projection, Shaped SSA, DerivSSA, Iterative O-SSA) decomposition.

`grouping.method` is a method for automatic grouping.

`groups` is a list of indices of elementary components for grouping, which is coerced to a vector.

`base` is an input for the periodogram analysis: elementary reconstructed series (`"series"`), eigenvectors (`"eigen"`), or factor vectors (`"factor"`).

`freq.bins` could be: a single integer larger than 1, which defines the number of intervals of equal length dividing the frequency range $[0, 1/2]$; a vector of frequency separating points (of length ≥ 2); a list of frequency ranges. For each range, if only one frequency is indicated, then it will be used as the upper bound, while the lower bound will be zero. If the frequency intervals, given by the parameter `freq.bins`, are named, then the resultant groups will take these names.

`threshold` is a threshold for frequency contributions. The value `threshold`=0 indicates that Algorithm 2.17 will be used.

`method` is a method of interpolation (`"const"` or `"linear"`) of the periodogram values, which are initially given on the regular grid.

The result of `grouping.auto.pgram` (that is, of `grouping.auto`, where the parameter `grouping.method = "pgram"`) can be depicted in the form of component contributions T by the call

```
plot(gp, superpose = TRUE, order = TRUE)
```

Here `superpose` is logical and indicates whether to plot contributions for all intervals on one panel. If the parameter `order` is `TRUE`, then the depicted component contributions are ordered by their values.

2.7.3.2 Typical Code

Let us demonstrate how to replicate the examples of automatic grouping taken from Golyandina and Zhigljavsky (2013; Section 2.4.5) by means of the RSSA package.

Grouping Based on **w**-Correlations

Let us consider the "White dwarf" data and apply clustering to the corresponding **w**-correlation matrix for window length $L = 100$. To do that we use Fragment 2.7.1.

Fragment 2.7.1 ("White dwarf": Auto Grouping by Clustering)

```
> data("dwarfst", package = "ssabook")
> s <- ssa(dwarfst, L = 100)
> g <- grouping.auto(s, grouping.method = "wcor",
+                    method = "average", nclust = 2)
> print(g[[1]])
 [1]  1  2  3  4  5  6  7  8  9 10 11
> plot(wcor(s, groups = 1:30), scales = list(at = c(1, 11, 30)))
> plot(reconstruct(s, groups = g),
+      add.residuals = FALSE,
+      plot.method = "xyplot", superpose = FALSE)
```

The **w**-correlations between the 30 leading elementary reconstructed components are depicted in Fig. 2.29. We can deduce from this figure that the components can be partitioned into two groups, signal (ET1–11) and noise (ET12–100). Hierarchical clustering with average linkage into two groups provides a proper split into two clusters with the first cluster consisting exactly of ET1–11. Reconstruction with automatic grouping is presented in Fig. 2.30.

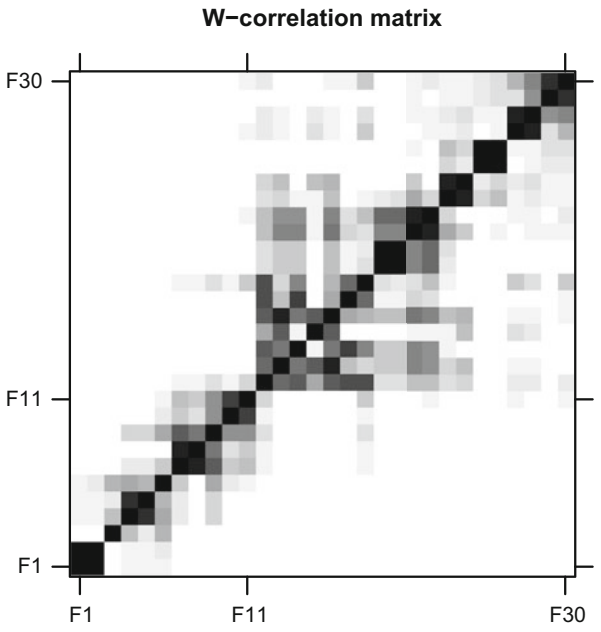

Fig. 2.29 "White dwarf": **w**-Correlation matrix, $L = 100$

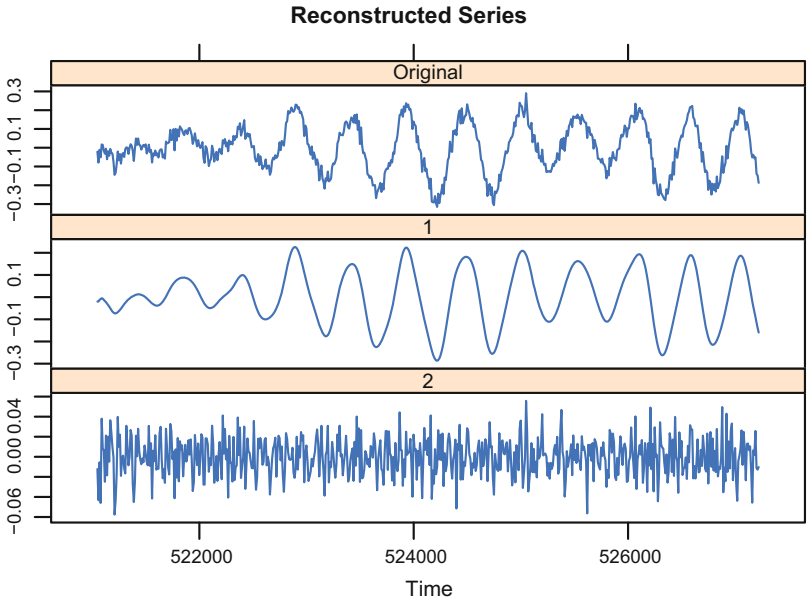

Fig. 2.30 "White dwarf": Decomposition with automatic grouping performed by clustering

Identification of Trend

Let us consider the "Production" example. Fragment 2.7.2 demonstrates how to choose the threshold and how to extract trends of different forms by means of the frequency approach. We consider two frequency ranges, $[0, 1/240]$ and $[0, 1/24]$. To understand what is a reasonable value of the threshold, we first choose an arbitrary small threshold to draw the plot of component contributions in the chosen frequency ranges; we reorder the components by their contributions. Figure 2.31 shows that the threshold should be between the contributions of the 9th and 10th components. We choose the contribution of the 9-th component, which is approximately equal to 0.89, as a new threshold.

Fragment 2.7.2 ("Production": Auto Grouping by Frequency Analysis)

```
> data("oilproduction", package = "ssabook")
> s <- ssa(oilproduction, L = 120)
> plot(s, type = "vectors", vectors = "factor", idx = 1:12)
> g0 <- grouping.auto(s, base = "series",
+                     freq.bins = list(Tendency = 1/240,
+                                      Trend = 1/24),
+                     threshold = 0.1)
> plot(g0, order = TRUE, type = "b")
> contrib <- attr(g0, "contributions")[, 2]
> print(thr <- sort(contrib, decreasing = TRUE)[9])
       8
```

```
0.861955
> g <- grouping.auto(s, base = "series",
+                    freq.bins = list(Tendency = 1/240,
+                                     Trend = 1/24),
+                    threshold = thr)
> print(g[[1]])
[1] 1 2
> print(g[[2]])
[1]  1  2  3  6  8 11 12 17 18
> plot(reconstruct(s, groups = g),
+      add.residuals = FALSE,
+      plot.method = "xyplot", superpose = TRUE)
```

If we choose $\omega_0 = 1/24$ and $T_0 = 0.89$, then the described procedure identifies ET1–3,6,8,11,12,17,18; a rough trend is thus identified accurately enough, see Fig. 2.33, red line. Figure 2.32 with factor vectors explains the result. Indeed, the detected factor vectors are slowly varying. The function `grouping.auto` allows to consider several frequency intervals. In this case, one should set a threshold for each frequency bin. In the code of Fragment 2.7.2, by rules of `R`, `threshold = thr` is equivalent to `threshold = list(thr, thr)`. For convenience, the implementation of `grouping.auto` allows to write `freq.bins = list(1/240,1/24)` instead of `freq.bins = list(c(0,1/240),c(0,1/24))`. Figure 2.33 shows two trends of different forms obtained by means of different frequency intervals.

If we were interested in the general tendency only, then the measure T with $\omega_0 = 1/240$ and the threshold $T_0 = 0.89$ identifying one leading component would be sufficient.

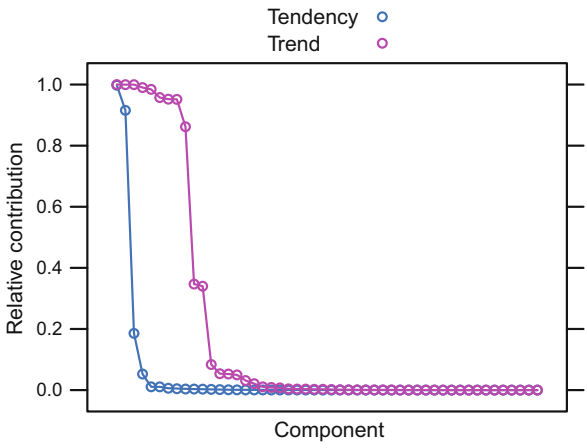

Fig. 2.31 "Production": Ordered frequency contributions of factor vectors, $L = 120$

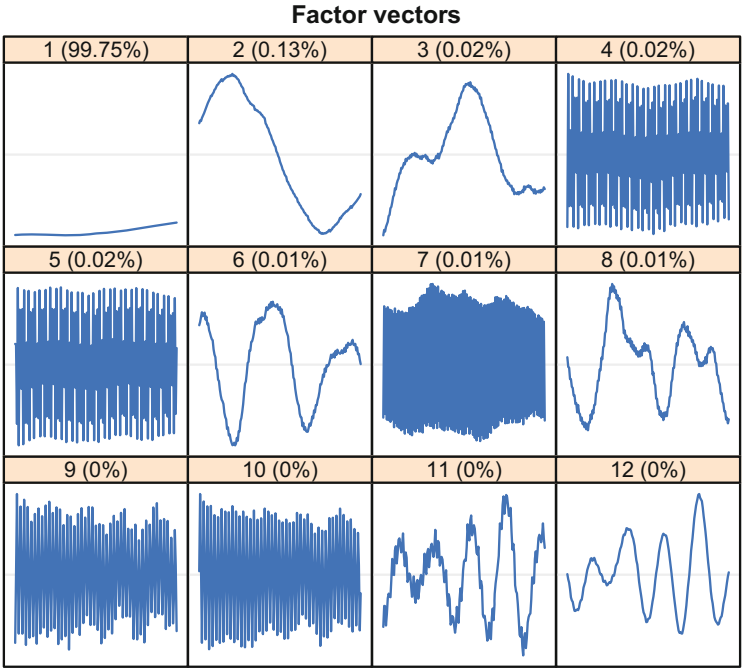

Fig. 2.32 "Production": Factor vectors, $L = 120$

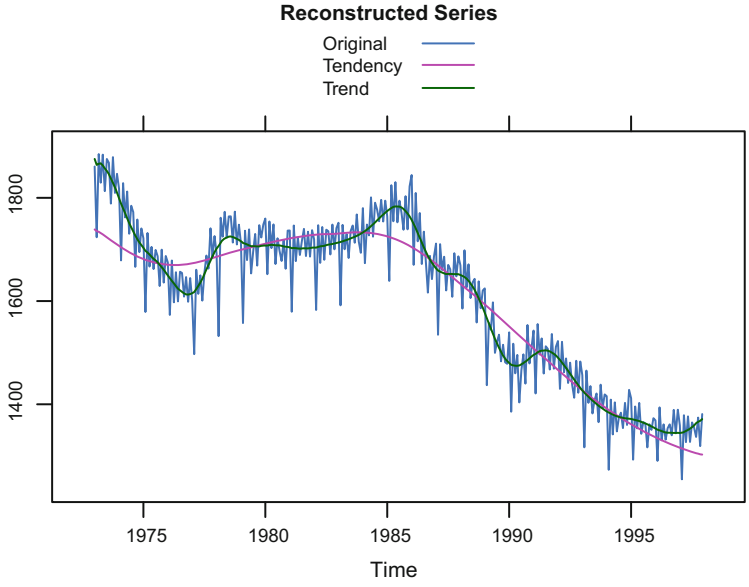

Fig. 2.33 "Production": Two extracted trends of different resolution, automatic grouping by frequencies

2.8 Case Studies

2.8.1 *Extraction of Trend and Oscillations by Frequency Ranges*

A decomposition on interpretable series components may differ from a decomposition on components with different frequency ranges but sometimes these decompositions can be similar. For example, extraction of a trend can be considered as a smoothing, i.e., extraction of a slowly-varying series component with a frequency range close to zero.

In Sect. 2.1.5.3, a typical decomposition of the series of sales of fortified wines in Australia into a sum of a trend, a seasonal component and a noise is shown.

Let us introduce an example of frequency decomposition. Consider the series "Tree rings" (tree ring width, annual, 1282–1950).

Fragment 2.8.1 makes a decomposition of the series "Tree rings" into components from the following frequency ranges: [0, 0.1), [0.1, 0.2), [0.2, 0.3), [0.3, 0.4), and [0.4, 0.5]. In the code, the last frequency is depicted as `+Inf` but in fact the upper bound is 0.5.

Fragment 2.8.1 ("Tree rings": Frequency Decomposition)

```
> data("dftreering", package = "ssabook")
> L <- 300
> s.tree <- ssa(dftreering, L = L, neig = L)
> g.tree <- grouping.auto(s.tree, base = "series",
+                         freq.bins = c(0.1, 0.2, 0.3, 0.4, +Inf))
> r.tree <- reconstruct(s.tree, groups = g.tree)
> plot(r.tree, add.residuals = FALSE, add.original = TRUE,
+      plot.method = "xyplot")
> specs <-
+   lapply(r.tree, function(x) spectrum(x, plot = FALSE)$spec)
> w.tree <- seq(0, length.out = length(specs$F1),
+               by = 1/length(dftreering))
> xyplot(F1 + F2 + F3 + F4 + F5 ~ w.tree, data = specs,
+        superpose = FALSE, type = "l", xlab = NULL, ylab = NULL,
+        auto.key = list(lines = TRUE, points = FALSE,
+                        column = 5))
```

The resultant decomposition is depicted in Fig. 2.34. Since the given frequency ranges split the whole range [0, 0.5], we obtain a full decomposition of the original series.

Figure 2.35 shows spectrums of the series components depicted in Fig. 2.34. It can be seen that the frequency ranges of the series components are almost disjoint. Since the window length L makes an influence on the resolution of the method (see, e.g., Golyandina and Zhigljavsky (2013; Section 2.9)), the intersection of frequency ranges increases for small window lengths.

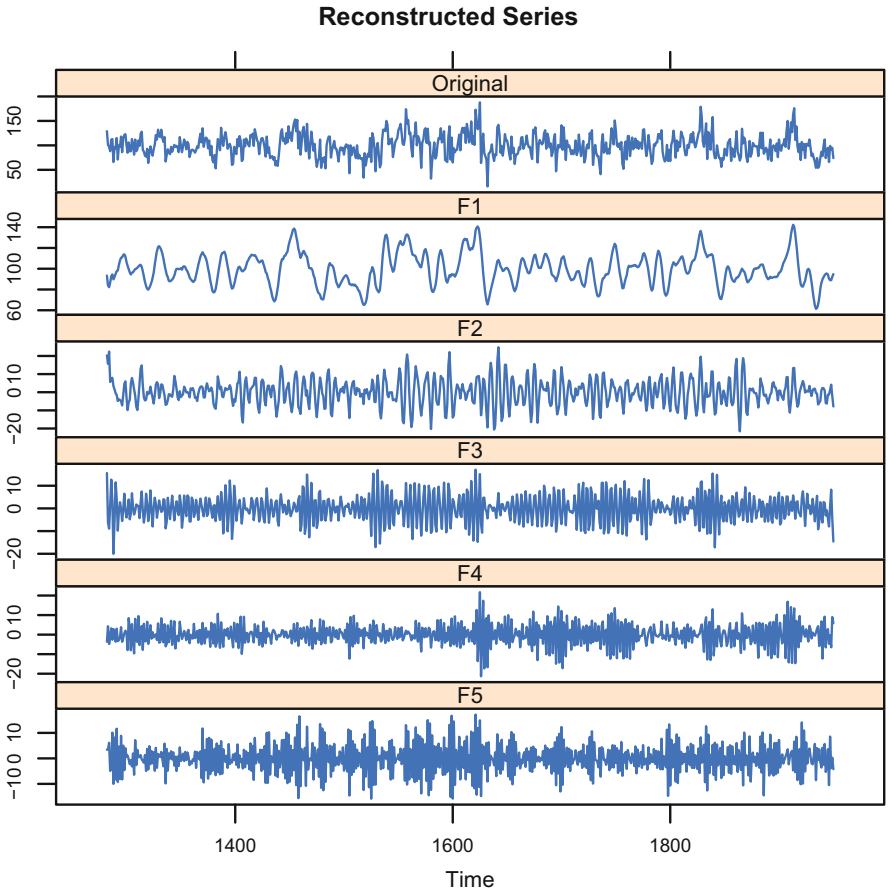

Fig. 2.34 "Tree rings": Frequency decomposition

2.8.2 Trends in Short Series

Let us consider the series "FORT" from the dataset "Australian Wines" with monthly sales. The first 120 points of the series are depicted in Fig. 2.36.

The series length is long enough to obtain weak separability; therefore, we will consider short subseries to demonstrate the ability of Iterative O-SSA to improve separability.

We choose the window length $L = 18$ to make the difference between Basic SSA and Iterative O-SSA clearly visible on the figures, although the relation between accuracies of the considered methods is very similar for other choices of the window length. Let us consider the subseries consisting of the points from 30th to 72th.

Let us start with Basic SSA. ET1 is identified as corresponding to trend; other components are produced by seasonality and noise (we do not include their plots).

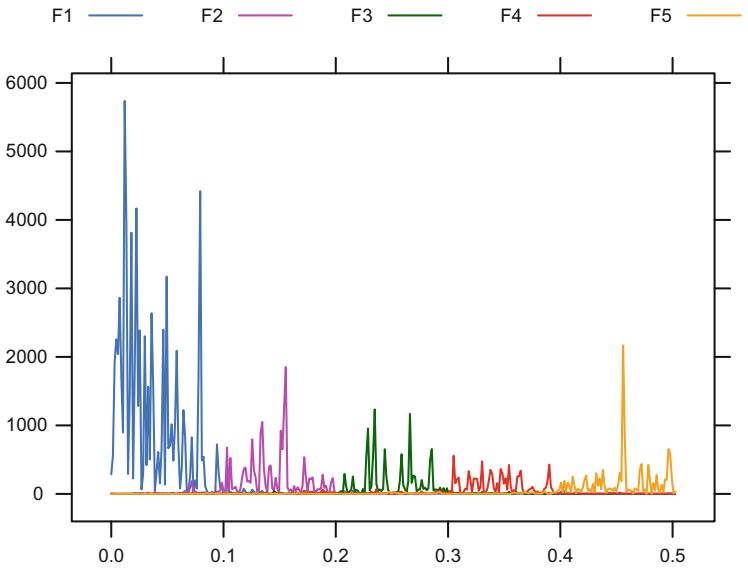

Fig. 2.35 "Tree rings": Periodograms of the series components

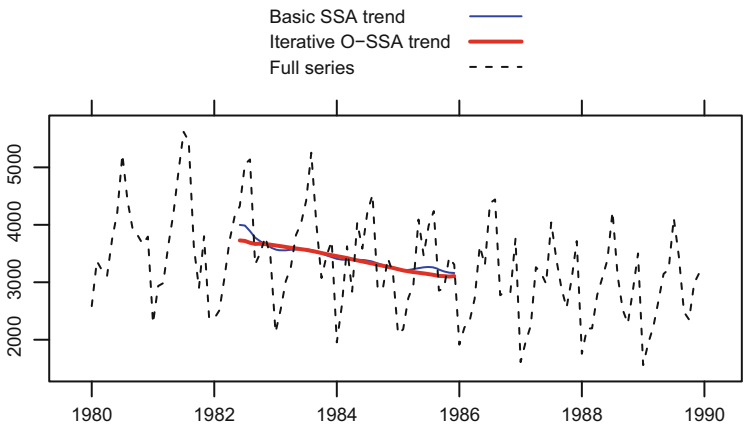

Fig. 2.36 "FORT": Trend reconstruction by Iterative O-SSA for the subseries consisting of points 30–72

One can see in Fig. 2.36 that the reconstructed trend is slightly mixed with the seasonality and follows the seasonal component at the ends of the series.

To apply Iterative O-SSA, we should choose a group of elementary components containing the trend components and approximately separated from the residual. Let it be ET1–7. Thus, we apply one iteration of O-SSA to the refined groups ET1 and ET2–7. Since the trend has much larger contribution than the residual, we consider Iterative O-SSA with no sigma-correction. The result of reconstruction

is much more relevant, see Fig. 2.36. This reconstruction is obtained by means of the code of Fragment 2.8.2.

Fragment 2.8.2 ("FORT": Basic SSA and Iterative O-SSA Trends)

```
> data("AustralianWine", package = "Rssa")
> Nfull <- 120
> wine <- window(AustralianWine,
+                end = time(AustralianWine)[Nfull])
> fort_sh <- window(wine[, "Fortified"],
+                   start = c(1982, 6), end = c(1985, 12))
> ss_sh <- ssa(fort_sh, L = 18)
> res_ssa_sh <- reconstruct(ss_sh, groups = list(1, 2:7))
> iss_sh <- iossa(ss_sh, nested.groups = list(1, 2:7),
+                 kappa = 0, maxiter = 1, tol = 1e-5)
> res_issa_sh <- reconstruct(iss_sh, groups = iss_sh$iossa.groups)
> theme <- simpleTheme(col = c("blue", "red", "black"),
+                      lwd = c(1, 2, 1),
+                      lty = c("solid", "solid", "dashed"))
> xyplot(cbind(res_ssa_sh$F1, res_issa_sh$F1, wine[, "Fortified"]),
+        superpose = TRUE,
+        xlab = "", ylab = "", type = "l", lwd = c(1, 2, 1),
+        col = c("blue", "red", "black"),
+        auto.key = list(text = c("Basic SSA trend",
+                                 "Iterative O-SSA trend",
+                                 "Full series"),
+                        type = c("l", "l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme)
```

2.8.3 Trend and Seasonality of Complex Form

Let us analyze the time series "MotorVehicle" which contains monthly data of total domestic and foreign car sales in the USA, from 1967 to 2012, January. The total series length is 541. This time series was investigated in Golyandina and Korobeynikov (2013) by means of Sequential SSA.

Figure 2.37 shows that the shape of the trend is complex. From the viewpoint of SSA, complexity of a trend means that it can only be approximated by a time series of a large rank and therefore it is decomposed into a large number of elementary components, if a large enough window length was chosen. Therefore, there are high odds that the trend decomposition components are going to be mixed with seasonal components in Basic SSA.

Since seasonal components are approximately orthogonal to slowly-varying components, we can consider the problem of mixing as a problem of lack of strong separability.

Rssa offers several options helping to avoid mixing. The first option is to use Sequential SSA as done in Golyandina and Korobeynikov (2013) (see Sect. 2.1.3.2, where the idea of Sequential SSA is briefly described).

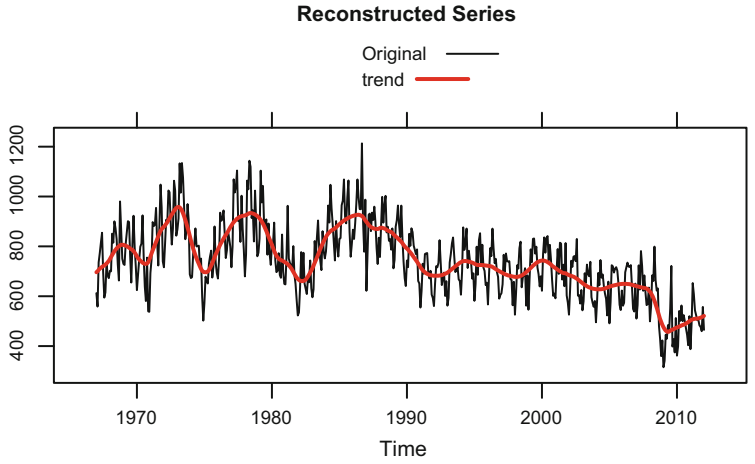

Fig. 2.37 "MotorVehicle": Trend extracted by Basic SSA with small window length $L = 12$

First, let us extract a trend. Since for a trend of such a difficult shape its extraction is similar to smoothing, we start with choosing a minimally possible window length, which in this case is $L = 12$. The reason for this choice of window length is similar to that in the moving averaging procedure: for smoothing a time series containing a periodic component, the window length should be divisible by the period. Then, the residual is decomposed with a large window length 264 to extract the seasonality, since the seasonal component can be considered as a sum of exponentially-modulated harmonics and therefore has a rank not exceeding 11.

Fragment 2.8.3 shows how Sequential SSA can be performed (see the explanation of the component choice in Golyandina and Korobeynikov (2013)). Figure 2.37 shows the resultant decomposition.

Fragment 2.8.3 ("MotorVehicle": Decomposition by Sequential SSA)

```
> data("MotorVehicle", package = "Rssa")
> s1 <- ssa(MotorVehicle, L = 12)
> res1 <- reconstruct(s1, groups = list(trend = 1))
> trend <- res1$trend
> plot(res1, add.residuals = FALSE, plot.type = "single",
+      col = c("black", "red"), lwd = c(1, 2),
+      plot.method = "xyplot", superpose = TRUE)
> res.trend <- residuals(res1)
> s2 <- ssa(res.trend, L = 264)
> res2 <- reconstruct(s2, groups = list(seasonality = 1:10))
> seasonality <- res2$seasonality
> res <- residuals(res2)
> # The resultant decomposition consists of
> # trend, seasonality and residual
```

Sequential SSA consists of a repeated application of one of the SSA methods, for example, Basic SSA. Let us demonstrate the use of another two-step approach, DerivSSA of Sect. 2.5, using a nested decomposition. In DerivSSA, a signal subspace should be estimated at the first step and then an additional rotation is performed in the signal subspace to avoid a mixture. Fragment 2.8.4 shows how DerivSSA can be applied. The signal subspace was detected by the analysis of eigenvectors and the **w**-correlation matrix. Identification of components of the refined decomposition, which was obtained by means of DerivSSA, is performed in the same way as it is done in Basic SSA.

Fragment 2.8.4 ("MotorVehicle": Decomposition by DerivSSA)

```
> data("MotorVehicle", package = "Rssa")
> s <- ssa(MotorVehicle, L = 264)
> sf <- fossa(s, nested.groups = 1:19)
> rf <- reconstruct(sf, groups =
+                     list(seasonality = 1:10, trend = 11:19))
> plot(rf, plot.method = "xyplot", superpose = TRUE,
+      add.residuals = FALSE,
+      col = c("black", "darkgreen", "red"), lwd = c(1, 1, 2))
> p<- parestimate(sf, groups = list(1:10),
+                 method = "esprit")
> print(p$period[seq(1, 10, 2)], digits = 3)
[1]  3.00 12.01  2.40  5.99  4.02
```

The decomposition results are similar. We present the results of full DerivSSA decomposition (we have used the version with normalization, see Algorithm 2.11) into trend, seasonal component and a noise in Fig. 2.38.

2.8.4 Finding Noise Envelope

Here we demonstrate how to estimate the variance of a heterogeneous noise. The procedure is based on the following two observations: first, the variance of noise is equal to the expectation of the squared noise values, and second, for a stochastic process the trend is its expectation. Therefore, the variance can be estimated as the trend of squared residuals. This trend can be extracted by SSA with a small window length and reconstructed by the leading eigentriple. The choice of the window length influences the level of detail with which we see the extracted trend. For the "MotorVehicle" data, window length $L = 30$ provides an appropriate trend. The result of Fragment 2.8.5 is depicted in Fig. 2.39 which shows the residuals and the standard deviation bounds.

Fragment 2.8.5 ("MotorVehicle": Finding Noise Envelope)

```
> resf <- residuals(rf)
> s.env <- ssa(resf^2, L = 30)
> rsd <- sqrt(reconstruct(s.env, groups = list(1))$F1)
> xyplot(resf + rsd + (-rsd) ~ time(resf), type = "l")
```

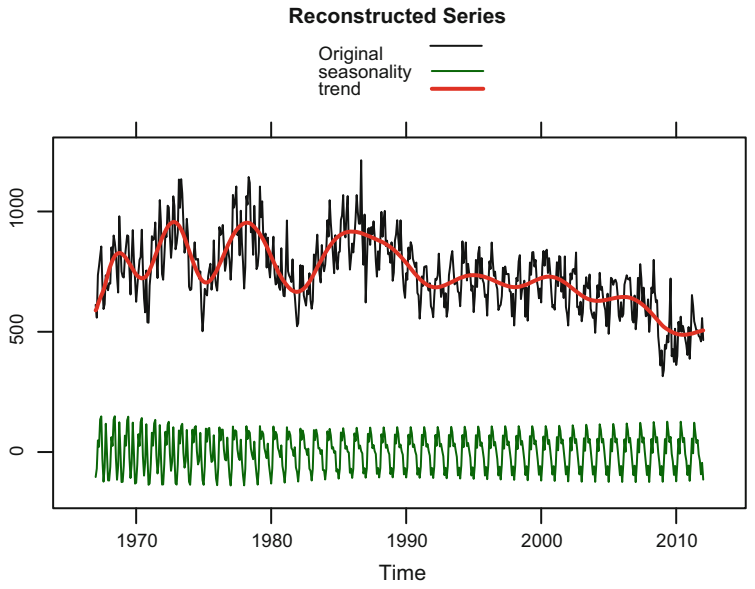

Fig. 2.38 "MotorVehicle": Decomposition by DerivSSA with $L = 264$

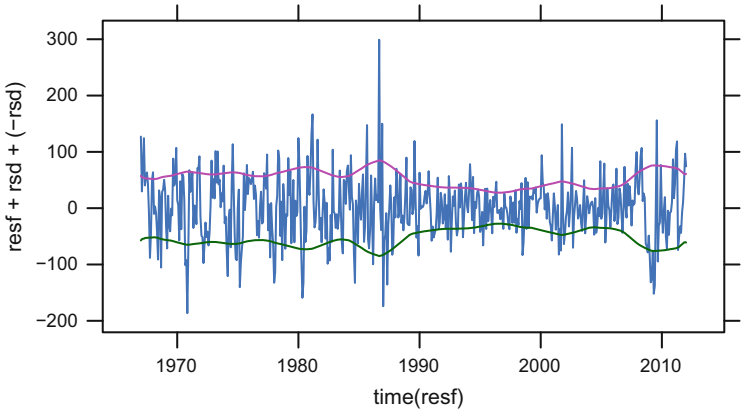

Fig. 2.39 "MotorVehicle": Residuals with envelopes

2.8.5 Elimination of Edge Effects

Let us consider "US Unemployment" data (monthly, 1948–1981, thousands) for male (20 years and over). The series length is $N = 408$. Since the series is long, we can expect weak separability between the trend and seasonality. For better weak separability we choose the window length equal to $L = N/2 = 204$, which is divisible by 12. Fragment 2.8.6 demonstrates how DerivSSA allows to improve the decomposition.

Fragment 2.8.6 ("US unemployment": Improvement by DerivSSA)

```
> data("USUnemployment", package = "Rssa")
> ser <- USUnemployment[, "MALE"]
> Time <- time(ser)
> L = 204
> ss <- ssa(ser, L = L, svd.method = "eigen")
> res<- reconstruct(ss, groups =
+                     list(c(1:4, 7:11), c(5, 6, 12, 13)))
> trend <- res$F1
> seasonality <- res$F2
> w1 <- wcor(ss, groups = 1:30)
> fss <- fossa(ss, nested.groups =
+                list(c(1:4, 7:11), c(5, 6, 12, 13)),
+              gamma = Inf)
> fres <- reconstruct(fss, groups = list(5:13, 1:4))
> ftrend <- fres$F1
> fseasonality <- fres$F2
> theme1 <- simpleTheme(col = c("grey", "blue","red"),
+                       lwd = c(2, 1, 1),
+                       lty = c("solid", "solid", "solid"))
> theme2 <- simpleTheme(col = c("blue", "red"), lwd = c(1, 1),
+                       lty = c("solid", "solid"))
> p1 <- xyplot(ser + trend + ftrend ~ Time,
+        xlab = "", ylab = "", type = "l", lwd = c(2, 1, 1),
+        col = c("grey", "blue","red"),
+        auto.key = list(text = c("Full series",
+                                 "Basic SSA trend",
+                                 "DerivSSA trend"),
+                        type = c("l", "l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme1)
> p2 <- xyplot(seasonality + fseasonality ~ Time,
+        xlab = "", ylab = "", type = "l", lwd = c(2, 1),
+        col = c("blue", "red"),
+        auto.key = list(text = c("Basic SSA seasonality",
+                                 "DerivSSA seasonality"),
+                        type = c("l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme2)
> plot(p1, split = c(1, 1, 1, 2), more = TRUE)
> plot(p2, split = c(1, 2, 1, 2), more = FALSE)
```

Basic SSA does not separate the trend and seasonality for this time series. It is another very typical situation that if trend has a complex form, then trend components are mixed with the seasonality components and therefore the so-called Sequential SSA was recommended (Golyandina et al. 2001; Section 1.7.3). However, this is also the case when DerivSSA is able to help.

We apply DerivSSA (the version with normalization) to the group ET1–13 that can be related to the signal. DerivSSA separates different frequencies so that components with higher frequencies become the leading ones. Since the low-frequency components in the considered series have large contribution, the weight

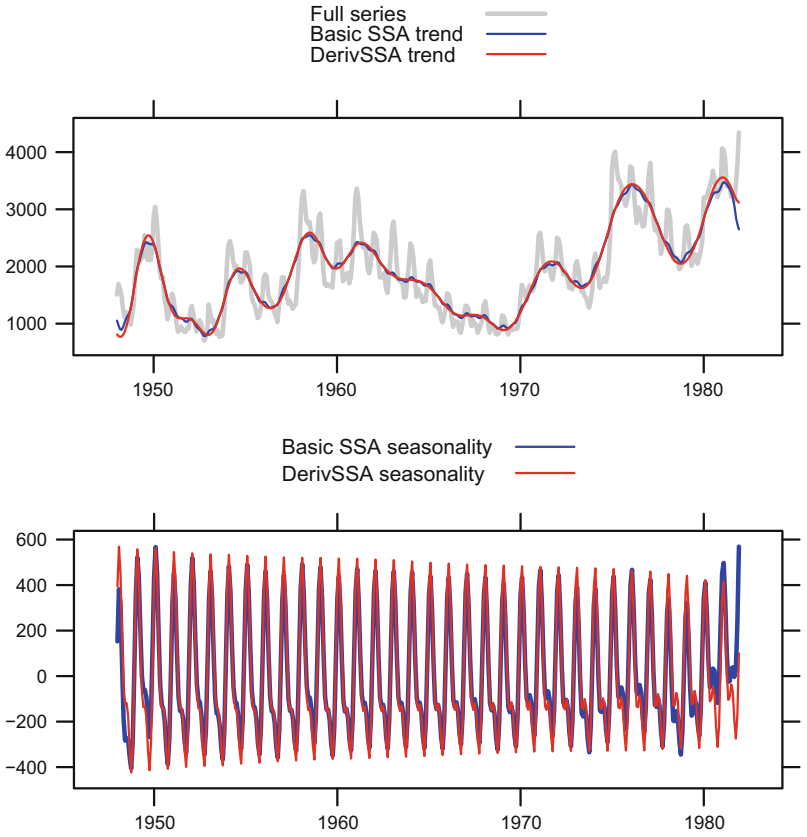

Fig. 2.40 "US unemployment": Decompositions by Basic SSA and DerivSSA

of derivatives should be large in order to make the seasonal components leading; we take `gamma = Inf` to exclude non-derivative part from the decomposed matrix.

Figure 2.40 depicting the DerivSSA reconstructions of the trend and the seasonality confirms that DerivSSA visibly improves the reconstruction accuracy, especially at both ends of the series. It is shown in Golyandina and Shlemov (2015) how to obtain a similar effect by means of Iterative O-SSA.

2.8.6 Extraction of Linear Trends

Here we consider the example "Hotel" following Golyandina et al. (2001; Section 1.7.1). We extract trend from a short subseries of length n. Then we compare predictions by linear regressions constructed from the series itself and constructed from the trends extracted by SSA. More detailed comparison of SSA and regression for simulated examples can be found in Golyandina and Shlemov (2017).

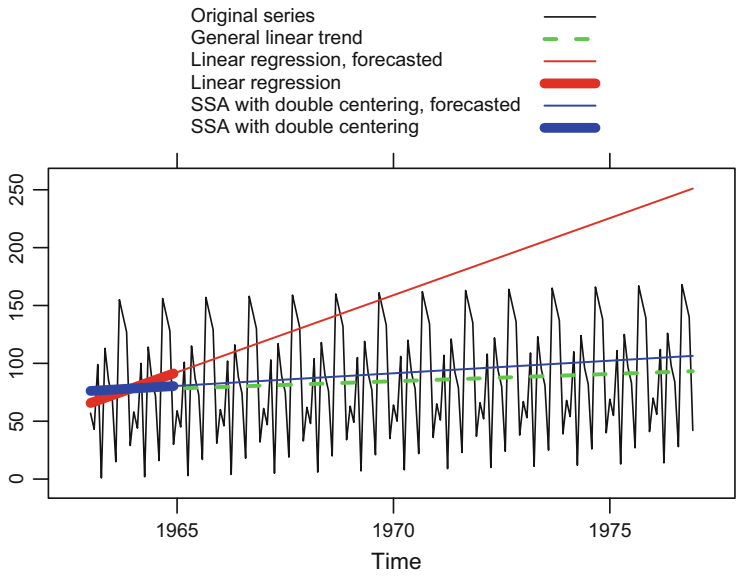

Fig. 2.41 "Hotel": SSA with projection, linear trend detection

Consider two cases: $n = 24$ (Fragment 2.8.7) and $n = 30$ (Fragment 2.8.8). For $n = 24$, the best separability of the linear trend from the residual is achieved by SSA with double centering (a particular case of SSA with projection). This is because we can choose L and K approximately divisible by the period 12. One can see in Fig. 2.41 that SSA linear trend (blue) is very close to a linear trend constructed by the whole long time series (green). Linear regression line (red) gives an approximation of the trend which is much worse than for SSA.

However, for $n = 30$, there is no appropriate window length providing desired orthogonalities. Therefore, direct SSA with double centering is not so successful in this case but to improve separability we can apply Iterative O-SSA. The result shown in Fig. 2.42 is rather good.

Fragment 2.8.7 ("Hotel": SSA with Projection, Linear Trend Detection)

```
> data("hotel", package = "ssabook")
> len <- length(hotel)
> n <- 24
> hotel.2years <- window(hotel, end = time(hotel)[n])
> sp <- ssa(hotel.2years, L = 12,
+           row.projector = "center",
+           column.projector = "center")
> r <- reconstruct(sp, groups = list(trend = 1:2))
> hotel.2years.data <- data.frame(x = 1:n, y = hotel.2years)
> fit.2years <- lm(y ~ x, data = hotel.2years.data)
> fit.2years.continued <- predict(fit.2years,
+                                 newdata = data.frame(x = 1:len))
```

```
> hotel.data <- data.frame(x = 1:len, y = hotel)
> fit <- lm(y ~ x, data = hotel.data)
> fit.rec <- lm(r$trend ~ x, data = hotel.2years.data)
> fit.rec.continued <- predict(fit.rec,
+                              newdata = data.frame(x = 1:len))
> xyplot(cbind(hotel,
+              predict(fit),
+              fit.2years.continued,
+              ts(predict(fit.2years),
+                 start = c(1963, 1), freq = 12),
+              fit.rec.continued,
+              ts(predict(fit.rec),
+                 start = c(1963, 1), freq = 12)),
+       superpose = TRUE,
+       type = "l", ylab = "",
+       lty = c(1, 2, 1, 1, 1, 1),
+       lwd = c(1, 2, 1, 5, 1, 5),
+       col = c("black", "green", "red", "red",
+               "blue", "blue"),
+       auto.key =
+          list(text = c("Original series",
+                        "General linear trend",
+                        "Linear regression, forecasted",
+                        "Linear regression",
+                        "SSA with double centering, forecasted",
+                        "SSA with double centering"),
+               type = c("l", "l", "l", "l", "l", "l"),
+               lines = TRUE, points = FALSE))
```

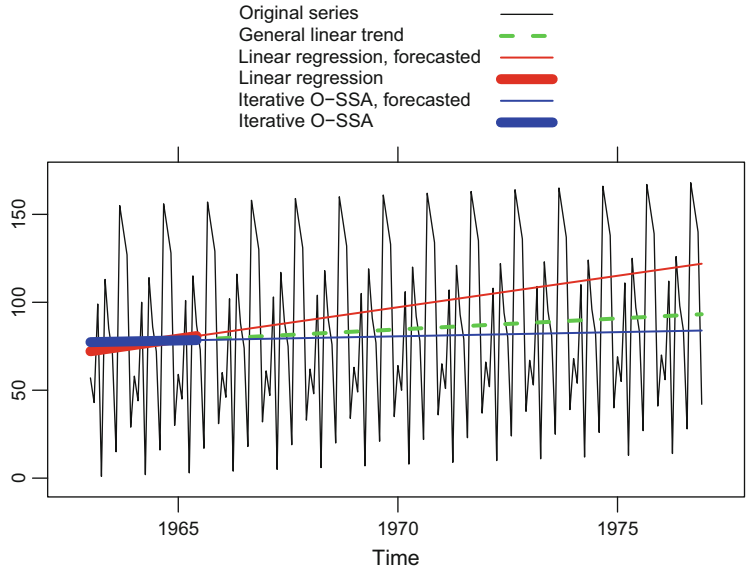

Fig. 2.42 "Hotel": Iterative O-SSA, linear trend detection

Fragment 2.8.8 ("Hotel": Iterative O-SSA, Linear Trend Detection)

```
> n <- 30
> hotel.2years <- window(hotel, end = time(hotel)[n])
> s <- ssa(hotel.2years, L = 12)
> ios <- iossa(s, nested.groups = list(1, 2:5))
> r <- reconstruct(ios, groups = list(trend = 1))
> hotel.2years.data <- data.frame(x = 1:n, y = hotel.2years)
> fit.2years <- lm(y ~ x, data = hotel.2years.data)
> fit.2years.continued <- predict(fit.2years,
+                                 newdata = data.frame(x = 1:len))
> hotel.data <- data.frame(x = 1:len, y = hotel)
> fit <- lm(y ~ x, data = hotel.data)
> fit.rec <- lm(r$trend ~ x, data = hotel.2years.data)
> fit.rec.continued <- predict(fit.rec,
+                              newdata = data.frame(x = 1:len))
> xyplot(cbind(hotel,
+              predict(fit),
+              fit.2years.continued,
+              ts(predict(fit.2years),
+                 start = c(1963, 1), freq = 12),
+              fit.rec.continued,
+              ts(predict(fit.rec),
+                 start = c(1963, 1), freq = 12)),
+        superpose = TRUE,
+        type = "l", ylab = "",
+        lty = c(1, 2, 1, 1, 1, 1),
+        lwd = c(1, 2, 1, 5, 1, 5),
+        col = c("black", "green", "red", "red",
+                "blue", "blue"),
+        auto.key =
+            list(text = c("Original series",
+                          "General linear trend",
+                          "Linear regression, forecasted",
+                          "Linear regression",
+                          "Iterative O-SSA, forecasted",
+                          "Iterative O-SSA",
+                 type = c("l", "l", "l", "l", "l", "l"),
+                 lines = TRUE, points = FALSE))
```

2.8.7 Automatic Decomposition

Let us consider an automatic identification based on frequency characteristics of the elementary series components (Fragment 2.8.9). This way of identification is very well suited for the problem of trend extraction. For the extraction of periodicities, we would recommend a more sophisticated approach described in Alexandrov and Golyandina (2005) and based on an idea from Vautard et al. (1992).

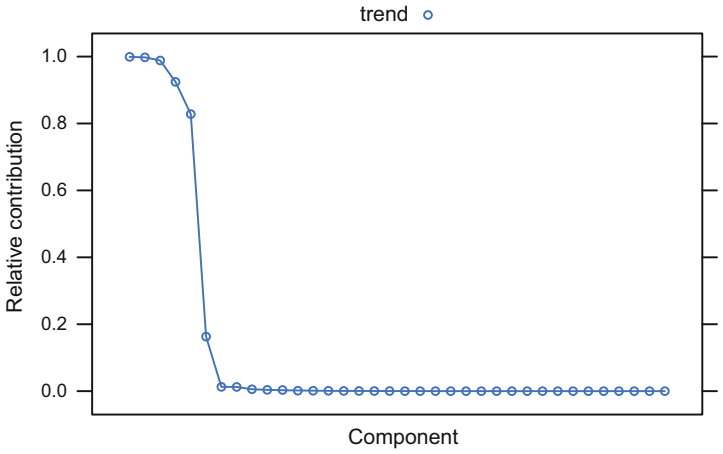

Fig. 2.43 "PayNSA": Contributions of trend components

The data "PayNSA" contains monthly numbers of all employees, total nonfarm payrolls, thousands of persons. Since the trend is complex, we will use Sequential SSA.

To extract a trend we take a small window length $L = 36$ and a frequency range $[0, 0.06]$ noting that the frequency $1/12 \approx 0.0833$ is related to seasonality. Figure 2.43 shows ordered contributions of these frequency ranges. A sharp drop after the fifth ordered component is clearly seen. Therefore, we choose the threshold equal to 0.7. The extracted trend is depicted in Fig. 2.44 together with the original series.

After performing trend extraction, let us investigate the residual. To extract the seasonal component, we choose the frequency range consisting of small intervals containing the frequencies $1/12$, $2/12$, $3/12$, $4/12$, $5/12$, and $6/12$. Figure 2.45 shows the contributions of the components in the initial order (right) and the contributions ordered by the magnitude of their values for each frequency range (left). In the right figure, one can see that the components with large contributions come in pairs with each pair corresponding to one frequency, except for a single component for the frequency $1/2$. The extracted seasonal component is presented in Fig. 2.46.

Note that if we subtract the seasonal component from the original time series, then we obtain the so-called seasonally adjusted component. To confirm that the seasonal component was really extracted, we depict the log-spectra of the original series and seasonally-adjusted series together (Fig. 2.47).

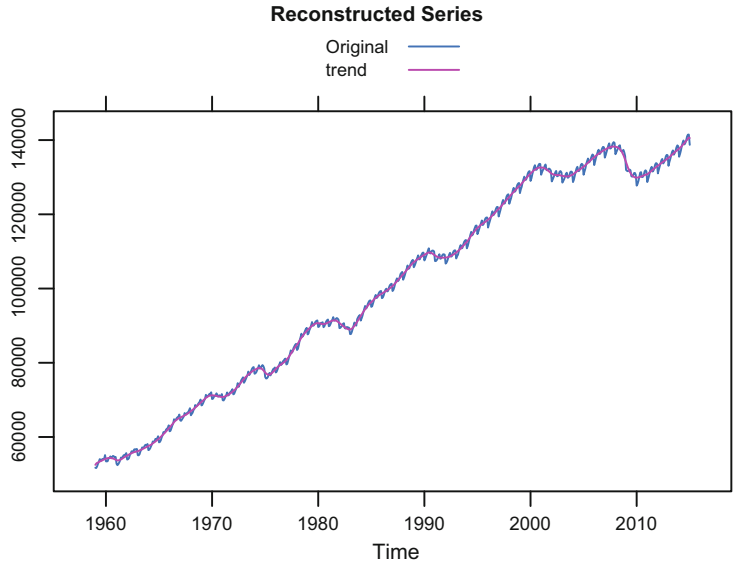

Fig. 2.44 "PayNSA": Automatically identified trend

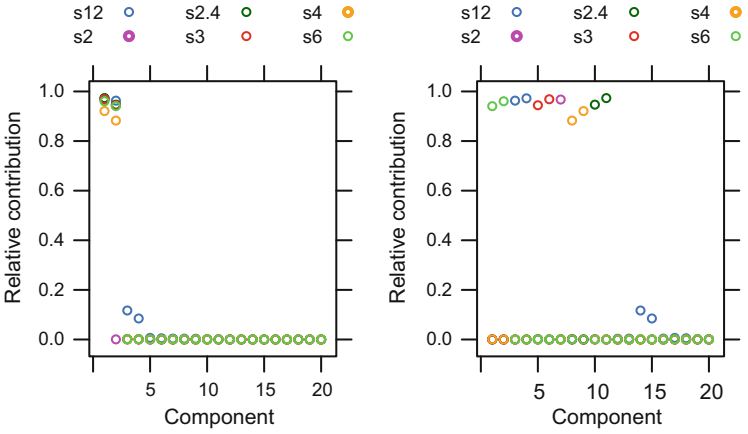

Fig. 2.45 "PayNSA": Contributions of seasonal components, ordered by their values (left) and ordered by component numbers (right) for different frequency ranges

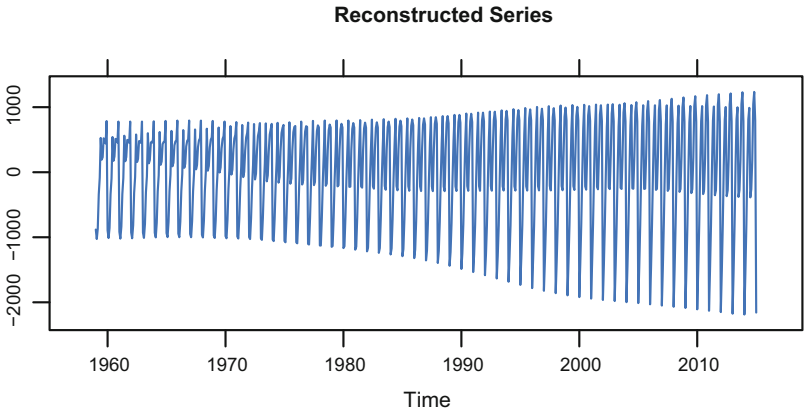

Fig. 2.46 "PayNSA": Automatically identified seasonal component

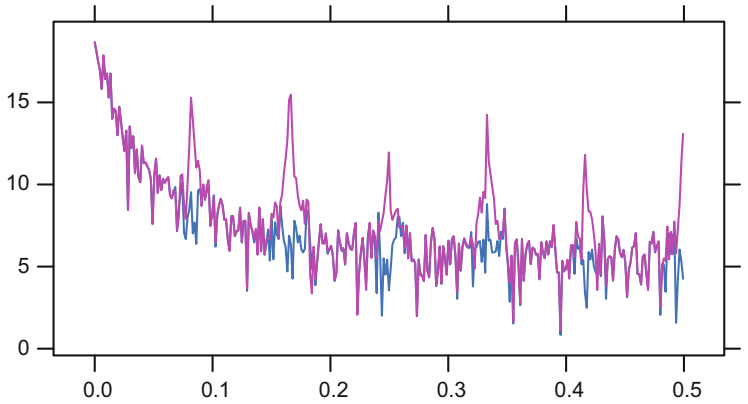

Fig. 2.47 "PayNSA": Log-periodogram of original and seasonally-adjusted series

Fragment 2.8.9 ("PayNSA": Automatically Identified Trend)

```
> data("paynsa", package = "ssabook")
> n <- 241
> pay <- window(paynsa, start = time(paynsa)[n])
> s <- ssa(pay, L = 36)
> g1 <- grouping.auto(s, base = "series",
+                     freq.bins = list(trend = 0.06),
+                     threshold = 0.7)
> print(g1$trend)
[1]  1  2  3  8 12
> plot(g1, order = TRUE, type = "b")
> r1 <- reconstruct(s, g1)
> plot(r1, plot.method = "xyplot", superpose = TRUE,
+      add.residuals = FALSE)
> s1 <- ssa(pay - r1$trend, L = 120)
```

```
> coef <- c(1 - 0.02, 1 + 0.02)
> freq.bins.seas = list(s12 = 1/12 * coef, s6 = 1/6 * coef,
+                       s4 = 1/4 * coef, s3 = 1/3 * coef,
+                       s2.4 = 1/2.4 * coef, s2 = 1/2 * coef)
> g3 <- grouping.auto(s1, base = "series", groups = 1:20,
+                     freq.bins = freq.bins.seas,
+                     threshold = list(0.6))
> p1 <- plot(g3, order = TRUE, scales = NULL,
+            auto.key = list(columns = 3))
> p2 <- plot(g3, order = FALSE, scales = NULL,
+            auto.key = list(columns = 3))
> plot(p1, split = c(1, 1, 2, 1), more = TRUE)
> plot(p2, split = c(2, 1, 2, 1), more = FALSE)
> r3 <- reconstruct(s1, groups = list(unlist(g3)))
> plot(r3, plot.method = "xyplot", add.residuals = FALSE,
+      add.original = FALSE)
> specNSA <- spectrum(pay - r3$F1, plot = FALSE)
> specSA <- spectrum(pay, plot = FALSE)
> w.pay <- seq(0, length.out = length(specNSA$spec),
+              by = 1/length(pay))
> xyplot(log(specNSA$spec) + log(specSA$spec) ~ w.pay,
+        type = "l", xlab = NULL, ylab = NULL)
```

2.8.8 Log-Transformation

As mentioned in Golyandina and Zhigljavsky (2013; Section 2.3.1.3), any multiplicative model can be considered as an additive model:

$$x_n = t_n(1+s_n)(1+r_n) = t_n + t_n s_n + (t_n + t_n s_n) r_n, \tag{2.21}$$

where t_n is a trend, s_n consists of regular oscillations, and r_n is a homogeneous noise. Since $t_n s_n$ can be considered as modulated regular oscillations and $(t_n + t_n s_n) r_n$ is a heterogeneous noise, SSA is able to perform such a decomposition.

On the other hand, one can consider the log-transformed series $\widetilde{x}_n = \ln x_n = \widetilde{t}_n + \widetilde{s}_n + \widetilde{r}_n$, where $\widetilde{t}_n = \log(t_n)$ is a trend, $\widetilde{s}_n = \log(1+s_n)$, and $\widetilde{r}_n = \log(1+r_n) \approx r_n$ is noise; if s_n is small, then $\widetilde{s}_n = \log(1+s_n) \approx s_n$ can still be treated as oscillations.

Thus, if a series follows the multiplicative model (2.21), then SSA-family methods can be applied both to the initial series and to the log-transformed data to obtain a decomposition into a sum of a trend, oscillations, and noise.

If the log-transformation makes the structure of the series components simpler, then it can be recommended. For example, if the model is multiplicative and the time series trend has a complex form, then the log-transformation may eliminate the modulation. Note that if the trend is exponential, then the log-transformed trend becomes linear and therefore SSA with double centering can be recommended for its extraction.

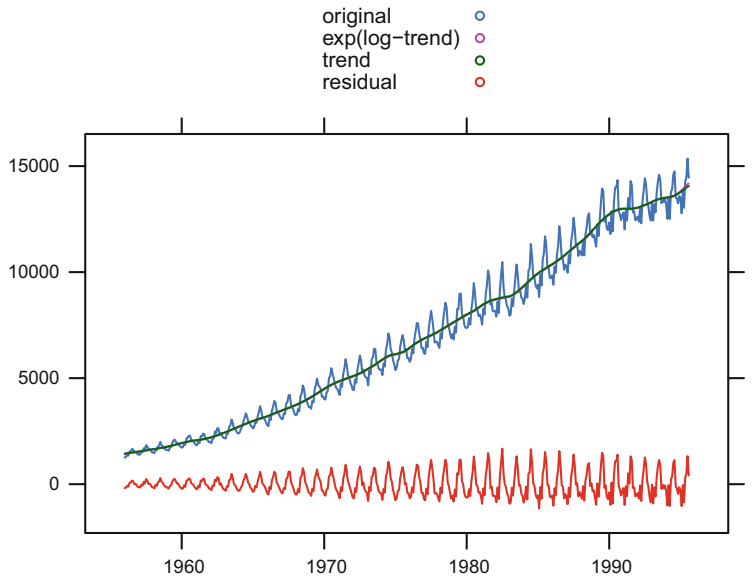

Fig. 2.48 "Elec": Decomposition for initial and log-transformed data

Let us consider the series "Elec" of Australian monthly electricity production (Jan 1956 – Aug 1995).

In Fragment 2.8.10, the trend is estimated by the decomposition of the original data and of the log-transformed data. Figure 2.48 shows that the trend estimations almost coincide. This is typical in cases when the model is not purely multiplicative. In the present example, the multiplicativity breaks down after 1980 when the range of oscillations slightly decreases.

The log-transformation can only be applied when the original data is positive. After making the log-transformation we have to apply the exponential transformation to the series, which we obtain from the SSA analysis, in order to return to the initial scale. This makes the resulted data positive; this is a very attractive feature of the log-transformation in many applications.

Fragment 2.8.10 ("Elec": Log-Transformation)

```
> data("elec", package = "fma")
> elec.log <- log(elec)
> Time <- time(elec)
> s <- ssa(elec, L = 12)
> r <- reconstruct(s, groups = list(trend = c(1)))
> sl <- ssa(elec.log, L = 12)
> rl <- reconstruct(sl, groups = list(trend = c(1)))
> xyplot(elec + exp(as.vector(rl$trend)) + r$trend +
+          (elec - r$trend) ~ Time,
+        superpose = TRUE, type ="l", ylab = NULL, xlab = NULL,
```

```
+         auto.key = list(text = c("original", "exp(log-trend)",
+                                  "trend", "residual")))
```

References

Alexandrov T, Golyandina N (2005) Automatic extraction and forecast of time series cyclic components within the framework of SSA. In: Proceedings of the 5th St.Petersburg workshop on simulation. St. Petersburg State University, pp 45–50

Andrew AL (1973) Eigenvectors of certain matrices. Linear Algebra Appl 7(2):151–162

Anderson E, Bai Z, Bischof C, Blackford L, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999) LAPACK Users' guide, 3rd edn. Society for Industrial and Applied Mathematics

De Moor BLR, Golub GH (1991) The restricted singular value decomposition: Properties and applications. SIAM J Matrix Anal Appl 12(3):401–425

Ghil M, Allen RM, Dettinger MD, Ide K, Kondrashov D, Mann ME, Robertson A, Saunders A, Tian Y, Varadi F, Yiou P (2002) Advanced spectral methods for climatic time series. Rev Geophys 40(1):1–41

Golyandina N (2010) On the choice of parameters in singular spectrum analysis and related subspace-based methods. Stat Interface 3(3):259–279

Golyandina N, Korobeynikov A (2013) Basic singular spectrum analysis and forecasting with R. Comput Stat Data Anal 71:943–954

Golyandina N, Lomtev M (2016) Improvement of separability of time series in singular spectrum analysis using the method of independent component analysis. Vestnik St. Petersburg Univ Math 49(1):9–17

Golyandina N, Shlemov A (2015) Variations of singular spectrum analysis for separability improvement: Non-orthogonal decompositions of time series. Stat Interface 8(3):277–294

Golyandina N, Shlemov A (2017) Semi-nonparametric singular spectrum analysis with projection. Stat Interface 10(1):47–57

Golyandina N, Zhigljavsky A (2013) Singular spectrum analysis for time series. Springer briefs in statistics. Springer

Golyandina N, Nekrutkin V, Zhigljavsky A (2001) Analysis of time series structure: SSA and related techniques. Chapman&Hall/CRC

Golyandina N, Pepelyshev A, Steland A (2012a) New approaches to nonparametric density estimation and selection of smoothing parameters. Comput Stat Data Anal 56(7):2206–2218

Harris T, Yan H (2010) Filtering and frequency interpretations of singular spectrum analysis. Physica D 239:1958–1967

Korobeynikov A, Larsen RM, Wu KJ, Yamazaki I (2016) SVD: Interfaces to various state-of-art SVD and eigensolvers. URL http://CRAN.R-project.org/package=svd, R package version 0.4

Larsen RM (1998) Efficient algorithms for helioseismic inversion. PhD thesis, University of Aarhus, Denmark

Roy R, Kailath T (1989) ESPRIT: estimation of signal parameters via rotational invariance techniques. IEEE Trans Acoust 37:984–995

Vautard R, Yiou P, Ghil M (1992) Singular-spectrum analysis: A toolkit for short, noisy chaotic signals. Physica D 58:95–126

Yamazaki I, Bai Z, Simon H, Wang LW, Wu K (2008) Adaptive projection subspace dimension for the thick-restart Lanczos method. Tech. rep., Lawrence Berkeley National Laboratory, University of California, One Cyclotron road, Berkeley, California 94720

Chapter 3
Parameter Estimation, Forecasting, Gap Filling

Similarly to Chap. 2, this chapter is devoted to applications of SSA for one-dimensional series; that is, to 1D-SSA. The SSA analysis of time series, which is considered in Chap. 2, can be classified as model-free. In this chapter, on the contrary, we consider the methodologies within the 1D-SSA approach, which require a model. These methodologies include the common problems of forecasting, interpolation, low-rank approximation, and parameter estimation. The model used is based on properties of the approximating subspace constructed in the process of 1D-SSA analysis of Chap. 2 and so the methodologies of this chapter belong to the class of subspace-based methods of time series analysis and signal processing.

The main parametric model of 1D-SSA is a linear recurrence relation (LRR) which a time series should approximately satisfy. In Sect. 3.1, we describe how to estimate the LRR coefficients and parameters of a series component satisfying such LRR.

Section 3.2 is devoted to forecasting, the most practically important application of time series analysis. In 1D-SSA, the problem of forecasting coincides with the problem of continuation of the signal $\mathbb{S}$ extracted from the observed series $\mathbb{S} + \mathbb{R}$, where $\mathbb{R}$ is the residual (or noise). To do that, we estimate the trajectory space of $\mathbb{S}$ and make the continuation based on the estimated subspace. A straightforward manner to make a forecast is to directly use the parametric form of the signal estimated using the methods of Sect. 3.1. However, the class of series suitable for forecasting is much wider than the class of series where the parametric model is adequate and some of the forecasting methods of Sect. 3.2 only use certain features of the estimated subspaces and not the estimators of the signal parameters; hence, a medium-term forecast could be quite accurate even if the given series cannot be approximated by a signal which globally satisfies an LRR. Section 3.2 also thoroughly discusses the problem of assessing stability of forecasts, which is the key issue in understanding of how much the forecasts can be trusted.

N. Golyandina et al., *Singular Spectrum Analysis with R*, Use R!,
https://doi.org/10.1007/978-3-662-57380-8_3

Section 3.3 is devoted to the problem of imputation of missing values, or gap filling. The methods of Sect. 3.3 extend the methods of Sect. 3.2 as the problem of forecasting can be considered as a particular case of the problem of missing value imputation when the missing values are located at the end of the series.

Section 3.4 is devoted to the problem of structured low-rank approximation of Hankel matrices which arises when the parametric model of the signal is of the main interest. The most common method of solving this problem is a repeated application of the SSA algorithm. In Sect. 3.4 we stress that choosing appropriate weights in defining the matrix norm can make a significant improvement in the accuracy of the approximation, especially at the points close to both ends of the series.

In Sect. 3.5 we carefully analyze several real-world time series to illustrate the main points of previous sections. We also discuss the problem of choosing parameters of the algorithms.

As in Chap. 2, for the sake of brevity, in this chapter we will refer to 1D-SSA simply as SSA. In contrast to the SSA analysis, the input for the algorithms of this chapter is not necessarily a collection $\mathbb{X}_N = (x_1, \ldots, x_N)$ of N real numbers; it can be an estimated subspace. This subspace is, as a rule, obtained after Grouping step of any of the SSA algorithms and has the form $\mathrm{span}(U_i, i \in I)$.

3.1 Parameter Estimation

In Sect. 2.1.2, in the process of explaining the concepts related to the SSA analysis such as SSA decomposition and separability, we have described a class of series $\mathbb{S} = (s_n)$ governed by linear recurrence relations (LRRs) $s_n = \sum a_i s_{n-i}$. In this section, we describe how to estimate the LRR coefficients and parameters of a series components governed by an LRR. We will assume that the SSA method is able to approximately extract the investigated series component; that is, the component of interest is approximately separated and the window length together with the SSA modification are chosen appropriately.

A series governed by an LRR can be expressed in the parametric form (1.9). A particular case is a series $\mathbb{S} = (s_n)$ with $s_n = \sum_{i=1}^{r} C_i \mu_i^n$, $\mu_i \in \mathsf{C}$, or, in the real-valued form, $s_n = \sum_{i=1}^{p} A_i \exp(\alpha_i n) \cos(2\pi n \omega_i + \phi_i)$, where A_i, α_i, ω_i and ϕ_i $(i = 1, \ldots, p)$ are unknown parameters whose values may be (and often are) of interest to the investigator. Hence the problem of parameter estimation arises.

3.1.1 Method

We describe the so-called subspace-based methods of parameter estimation, where only the estimated subspace of the series components is of concern but Reconstruction step of the SSA algorithm is of no importance.

Let a set of indices I correspond to the component of interest in the constructed decomposition of the trajectory matrix $\mathbf{X} = \sum_{i=1}^{d} \sigma_i P_i Q_i^{\mathrm{T}}$. For simplicity of notation we assume $I = \{1, 2, \ldots, r\}$. Then the estimated subspace is $\widetilde{\mathcal{S}} = \mathrm{span}(P_1, \ldots, P_r)$. We always consider the generating set $\{P_i\}$ of $\widetilde{\mathcal{S}}$ to be orthonormal as otherwise we can orthonormalize it. Since the original vectors P_i may be linearly dependent (for example, in the method of SSA with projection), the procedure of orthogonalization may reduce the number of vectors. We will consider r to be equal to the number of vectors after orthogonalization.

We consider two kinds of parametrization; first, in the form of a governing LRR and, second, in the form (1.9). Correspondingly, we describe how to estimate the coefficients of a governing LRR and the parameters of (1.9).

3.1.1.1 Estimation of the Governing LRR

The trajectory space $\mathcal{S}$ of a signal $\mathbf{S}$ governed by a particular LRR corresponds to many LRRs. More precisely, any vector from $\mathcal{S}^{\perp}$ with the last coordinate -1 produces such an LRR; in other words, any such vector from $\mathcal{S}^{\perp}$ provides a set of coefficients for a linear combination of the first $L-1$ coordinates of a vector from $\mathcal{S}$ to obtain the last coordinate; see Golyandina et al. (2001; Section 5) and Golyandina and Zhigljavsky (2013; Chapter 3) for detailed explanations.

Among all these LRRs (generating the same trajectory space $\mathcal{S}$) there is the best LRR with minimal sum of squared coefficients (the so-called min-norm LRR). The min-norm LRR suppresses possible perturbations in the initial data as much as possible, which is important if we use this LRR for series generation or continuation, on the base of SSA approximation of the initial data.

For a chosen window length L, the signal subspace $\mathcal{S} \in \mathsf{R}^L$ and therefore the min-norm LRR has order $L-1$. For each column vector P_i of $\mathbf{P}_r$, denote π_i the last coordinate of P_i, $\underline{P}_i \in \mathsf{R}^{L-1}$ the vector P_i with the last coordinate removed, and $\nu^2 = \sum_{i=1}^{r} \pi_i^2$. Then the elements of the vector

$$\mathcal{R} = (a_{L-1}, \ldots, a_1) = \frac{1}{1-\nu^2} \sum_{i=1}^{r} \pi_i \underline{P}_i \tag{3.1}$$

provide the coefficients of the *min-norm* governing LRR: $s_n = \sum_{i=1}^{L-1} a_i s_{n-i}$.

For an estimated subspace $\widetilde{\mathcal{S}}$, the estimated LRR is calculated in the same way, on the base of an orthonormal basis of $\widetilde{\mathcal{S}}$.

3.1.1.2 Estimation of Frequencies

Let $\mathbb{X}_N = \mathbb{S}_N + \mathbb{R}_N$, where $s_n = \sum_{j=1}^{r} c_j \mu_j^n$ and the series $\mathbb{S}_N$ and $\mathbb{R}_N$ are approximately separable for a given window length L. Generally, the signal roots of the characteristic polynomial of a governing LRR allow estimation of the signal

parameters μ_j, $j = 1, \ldots, r$ (see Sect. 2.1.2.2). However, the min-norm LRR is not minimal and therefore we should somehow distinguish between the signal roots and the extraneous roots. Usually, the signal roots of the min-norm LRR have maximal moduli (e.g., see Usevich (2010)). Therefore, one can find roots of the min-norm LRR, arrange them in the order of decrease, and take the first r roots.

However, the ordering is never guaranteed. Therefore, the methods that are able to separate the signal and extraneous roots could be very useful.

Let us describe one of these methods called ESPRIT (Roy and Kailath 1989). This method is implemented in two versions, LS-ESPRIT and TLS-ESPRIT, where LS means least squares, TLS means total least squares (see, e.g., the method description in Golyandina and Zhigljavsky (2013; Section 3.8.2)). Other names are HSVD (Barkhuijsen et al. 1987) and HTLS (Van Huffel et al. 1994). Here we describe the LS version (HSVD).

Denote $\{P_1, \ldots, P_r\}$ an orthonormal basis of the estimated subspace of the component under interest. Set $\mathbf{P}_r = [P_1 : \ldots : P_r]$ and let $\underline{\mathbf{P}_r}$ be the matrix with the last row removed and $\overline{\mathbf{P}_r}$ be the matrix with the first row removed. Then μ_i can be estimated by the eigenvalues of the matrix $\underline{\mathbf{P}_r}^{\dagger}\overline{\mathbf{P}_r}$, where $\dagger$ denotes pseudo-inversion. Correspondingly, the estimated frequencies are the arguments of μ_i.

Note that the matrix $\mathbf{P}_r$ conventionally consists of the chosen eigenvectors U_i in the Basic SSA algorithm. However, any basis of the subspace, which estimates the signal subspace, is suitable.

Let us mention a simple and fast method of frequency estimation which is used for identification of the eigentriples at Grouping step. Two vectors $U^{(1)}$ and $U^{(2)}$ forming an orthogonal basis of the trajectory space of an exponentially-modulated sine wave have similar forms and their phases differ by approximately $\pi/2$. Let A and B be defined by $a_n = \rho^n \sin(2\pi\omega n + \phi)$ and $b_n = \rho^n \cos(2\pi\omega n + \phi)$. Denote the angle between vectors by $\angle$. Then $\omega = \angle\left(\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}, \begin{pmatrix} a_2 \\ b_2 \end{pmatrix}\right) \Big/ (2\pi)$. Therefore, we can estimate the frequency using the basis vectors $U^{(1)}$ and $U^{(2)}$. Since these vectors do not have exactly the same form as A and B, the sequence of angles $\angle\left(\begin{pmatrix} u_i^{(1)} \\ u_i^{(2)} \end{pmatrix}, \begin{pmatrix} u_{i+1}^{(1)} \\ u_{i+1}^{(2)} \end{pmatrix}\right) \Big/ (2\pi)$, $i = 1, \ldots, L-1$, can be considered and then the mean or median can be taken as an estimate of the frequency; see Golyandina et al. (2001; Section 1.6) for details. In Rssa, the median is considered and the median of absolute deviations from the median is used as a measure of accuracy.

3.1.2 Algorithms

Although the LRR approximating the time series is usually used for forecasting, it can also be helpful for construction of the signal model. Hence we introduce an algorithm for calculation of the min-norm LRR coefficients.

Algorithm 3.1 Estimation of the signal LRR

Input: Matrix $\mathbf{P}_r \in \mathsf{R}^{L\times r}$ consisting of orthonormal column vectors, which form a basis of the estimated signal subspace.

Output: Coefficients $\mathcal{R} = (a_{L-1}, \ldots, a_1)$ of the corresponding LRR.

1: For each column vector P_i of $\mathbf{P}_r$, calculate π_i and $\underline{P}_i$, $\nu^2 = \sum_{i=1}^{r} \pi_i^2$. If ν^2 is equal to 1, then STOP with the error message "Verticality coefficient equals 1."

2: Compute $\mathcal{R} = \frac{1}{1-\nu^2} \sum_{i=1}^{r} \pi_i \underline{P}_i$.

The next algorithm shows how the parameters μ_i in $s_n = \sum_{i=1}^{r} C_i \mu_i^n$ can be estimated from the roots of the characteristic polynomial of an LRR governing this time series (see Sect. 2.1.2.2 for a description of the relation between LRRs and their characteristic polynomials). The given LRR is an estimate of an LRR governing a series of rank r; therefore, only r roots correspond to the signal, while the other roots are extraneous. Since frequently (but not always!) the signal roots for the min-norm LRR have larger moduli than the extraneous roots, we can select signal roots with large absolute values among the whole set of roots.

Algorithm 3.2 Estimation of the signal roots through characteristic polynomial of LRR

Input: Coefficients $A = (a_1, \ldots, a_m)$ of the LRR $s_n = \sum_{i=1}^{m} a_i s_{n-i}$, rank r.

Output: Signal roots $\mu_i, i = 1, \ldots, r$.

1: Construct the characteristic polynomial $P(\mu) = \mu^d - \sum_{i=1}^{m} a_i \mu^{n-i}$.

2: Find the roots $\mu_1, \ldots, \mu_m$ of $P(\mu)$.

3: Order the roots so that $|\mu_1| \geq \ldots \geq |\mu_m|$.

4: The leading roots $\mu_i, i = 1, \ldots, r$, are the candidates for the signal roots.

Algorithm 3.3 is one of the most known high-resolution subspace-based algorithms of estimation of frequencies and damping factors.

Algorithm 3.3 ESPRIT

Input: Matrix $\mathbf{P}_r \in \mathsf{R}^{L\times r}$ consisting of orthonormal column vectors, which form a basis of the estimated signal space.

Output: r roots in the form (ρ_i, ω_i).

1: Using either LS or TLS method, find a matrix $\mathbf{M} \in \mathsf{R}^{r\times r}$ satisfying $\overline{\mathbf{P}_r} \approx \underline{\mathbf{P}_r}\mathbf{M}$. For the LS-method, $\mathbf{M} = \underline{\mathbf{P}_r}^{\dagger}\overline{\mathbf{P}_r}$.

2: Find eigenvalues $\overline{\mu_i}, i = 1, \ldots, r$, of $\mathbf{M}$.

3: Set $\rho_i = \mathrm{Mod}(\mu_i)$, $\omega_i = \mathrm{Arg}(\mu_i)$.

The next algorithm is a complementary to Decomposition step used for helping to gather sine-waves with similar frequencies.

Algorithm 3.4 Fast ("pairs") estimation of frequencies

Input: Two orthonormal vectors $U^{(1)}$ and $U^{(2)}$ forming an estimated trajectory space of a sine wave.

Output: Frequency ω, period T.

1: Compute $\phi_i = \angle\left(\begin{pmatrix} u_i^{(1)} \\ u_i^{(2)} \end{pmatrix}, \begin{pmatrix} u_{i+1}^{(1)} \\ u_{i+1}^{(2)} \end{pmatrix}\right)$, $i = 1, \ldots, L-1$.

2: Calculate $\bar{\phi}$ as the mean or median of $\{\phi_i\}$.

3: $\omega = \bar{\phi}/(2\pi)$, $T = 1/\omega$.

3.1.3 Estimation in RSSA

3.1.3.1 Description of Functions

After the `ssa` object `s` has been constructed by the call of the `ssa` (alternatively, `iossa` or `fossa`) function, the min-norm LRR can be constructed by the call of the form

```
lrr.coef <- lrr(s, groups = list(2:3))
```

Arguments:

`s` is an `ssa` object holding the full one-dimensional SSA decomposition.
`groups` is a list defining the group of selected eigentriples.

The function `lrr` returns a list of objects of the `lrr` class, which contain coefficients of the LRRs for each given group.

Complex roots of the characteristic polynomial of an LRR, which are ordered by their moduli, are calculated by the call

```
lrr.roots <- roots(lrr.coef, method = "companion")
```

Arguments:

`lrr.coef` is an `lrr` object.
`method` is a method used for calculation of the polynomial roots: via eigenvalues of the companion matrix or via R's standard `polyroot` routine.

Estimation of parameters can be performed by means of this typical call:

```
est <- parestimate(s, groups = list(c(2, 3, 5, 6)),
                   method = "esprit")
```

Arguments:

`s` is an `ssa` object holding the full one-dimensional SSA decomposition.
`groups` is a list of eigentriples groups; for `method = "pairs"` each group should consist of exactly two components.
`method` is a method of estimation of frequencies and damped factors; it can have the following values: `"esprit"`, `"pairs"`.

`subspace` indicates which space, column or row, will be used for parameter estimation by the ESPRIT method. The default value `"column"` is standard for ESPRIT.

`solve.method` is the method of shift matrix estimation; it can be set as `"ls"` for the least squares solution and `"tls"` for the total least squares approach.

For an `lrr` object, the function `print` prints the LRR coefficients and `plot` draws the produced complex roots, both signal and extraneous.

For the result of `parestimate`, the function `print` prints the estimated parameters, while `plot` draws the estimated signal roots on the complex plane.

3.1.3.2 Typical Code

We start with a simple example to show a relation between LRRs and roots (Fragment 3.1.1). For the exponential series $s_n = 1.01^n = e^{n \ln 1.01}$ of rank $r = 1$, the minimal LRR is $x_n = 1.01x_{n-1}$; that is, the vector of its coefficients is $A = (1.01)$. The characteristic polynomial has the form $P(\mu) = \mu - 1.01$, its root is 1.01. The minimal LRR can be obtained for $L = r + 1$. Note that this choice is an inappropriate choice for noisy series, since it would most likely provide a poor separability between the signal and noise.

For $L = 6$ we have 4 extraneous roots. All five roots are depicted in Fig. 3.1. Moduli of all four extraneous roots are smaller than 1.

The second simple example produces LRR coefficients and signal roots for a linear function. In this example, rank $r = 2$, the minimal LRR does not depend on the coefficients of the linear function and is $x_n = 2x_{n-1} - x_{n-2}$; that is, $A = (2, -1)^{\mathrm{T}}$. The characteristic polynomial is $P(\mu) = \mu^2 - 2\mu + 1$; it has root 1 of multiplicity 2. Since all methods are numerical, it is impossible to obtain exactly equal roots. Therefore, a linear function numerically generates two different roots, each close to 1. In this example, the linear function is approximated by a sum of two exponentials (the case of two different real roots). In the case of two conjugate complex roots instead of one root 1 of multiplicity 2, it can be approximated by one sine wave with low frequency.

Fragment 3.1.1 (LRRs and Roots of Characteristic Polynomials)

```
> # Minimal LRR
> x <- 1.01^(1:10)
> s <- ssa(x, L = 2)
> l <- lrr(s, groups = list(1))
> print(l)
[1] 1.01
attr(,"class")
[1] "lrr"
> print(roots(l))
[1] 1.01
> # Extraneous roots
> x <- 1.01^(1:10)
```

```
> s <- ssa(x, L = 6)
> l <- lrr(s, groups = list(1))
> r <- roots(l)
> plot(l)
> # Multiple roots
> x <- 2.188 * (1:10) + 7.77
> s <- ssa(x, L = 3)
> l <- lrr(s, groups = list(1:2))
> print(l)
[1] -1  2
attr(,"class")
[1] "lrr"
> print(roots(l))
[1] 1.0000003 0.9999997
```

Fragment 3.1.2 demonstrates the methods of parameter estimation for the real-life series "CO2." For this series, ET1,4 correspond to a trend, while ET2,3 are related to a sine-wave with period 12 (annual periodicity) and ET5,6 correspond to half-year periodicity. One can see that both estimation methods provide almost equal

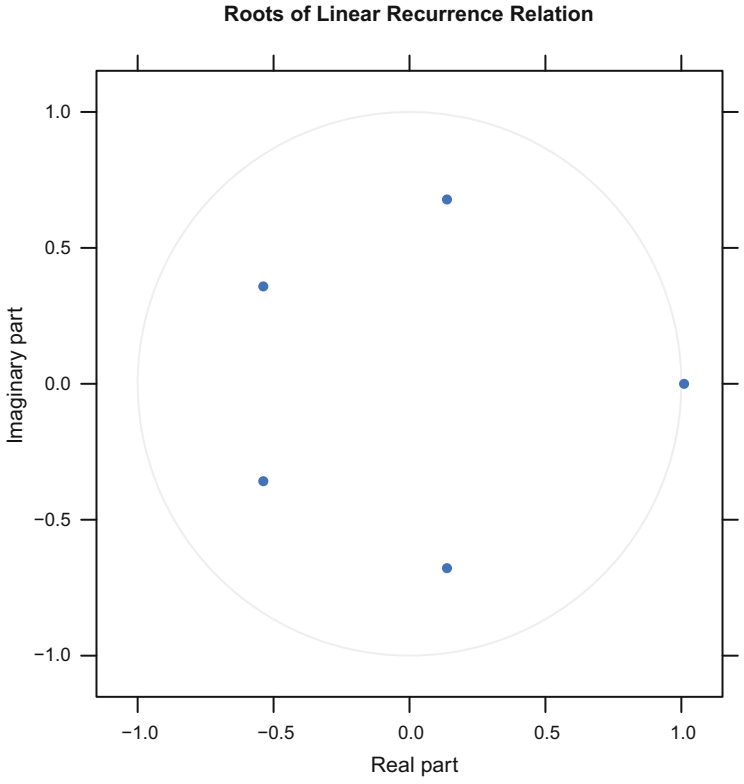

Fig. 3.1 Exponential signal: One signal and four extraneous roots

estimated periods of the annual component close to 12. Note that for the method `"pairs"`, the values `rate` and `Mod` mean nothing.

The ESPRIT method results in complex signal roots μ_j, which are presented in convenient exponential form $\mu_j = \rho_j e^{\mathrm{i}2\pi\omega_j}$: `period` is $1/\omega_j$, `Mod` is ρ_j, `rate` is $\ln \rho_j$, `Arg` means $2\pi\omega_j$. Since for real-valued series complex roots form conjugate pairs, two rows with parameter estimates have equal absolute values but have opposite signs. One can see that the trend part of the chosen components is approximated by a sum of two real exponentials (their periods equal `Inf`), the first one is increasing (the rate is positive), while the second one is decreasing.

Fragment 3.1.2 (Parameter Estimation for "CO2")

```
> # Decompose "co2" series with default window length L
> s <- ssa(co2)
> # Estimate the periods from 2nd and 3rd eigenvectors
> # using default "pairs" method
> print(parestimate(s, groups = list(c(2, 3)), method = "pairs"))
   period      rate   |     Mod     Arg   |      Re         Im
   11.995   0.000000  |  1.00000    0.52  |  0.86592    0.50019
> # Estimate the periods and rates using ESPRIT
> pe <- parestimate(s, groups = list(1:6),
+                   method = "esprit")
> print(pe)
   period      rate   |     Mod     Arg   |      Re         Im
   11.995   0.000542  |  1.00054    0.52  |  0.86638    0.50047
  -11.995   0.000542  |  1.00054   -0.52  |  0.86638   -0.50047
    5.999   0.000512  |  1.00051    1.05  |  0.50015    0.86653
   -5.999   0.000512  |  1.00051   -1.05  |  0.50015   -0.86653
      Inf   0.000375  |  1.00037    0.00  |  1.00037    0.00000
      Inf  -0.008308  |  0.99173    0.00  |  0.99173    0.00000
> plot(pe)
```

Figure 3.2 depicts six estimated signal roots on the complex plane, where two trend real-valued roots can hardly be distinguished.

3.2 Forecasting

The problem of forecasting is the problem of continuation of the signal $\mathbb{S}$ extracted from the observed series $\mathbb{X} = \mathbb{S} + \mathbb{R}$. To do that it is sufficient to estimate the trajectory space of $\mathbb{S}$ and then to construct the forecasted series based on the estimated subspace.

An obvious way to perform forecasting would be to estimate series parameters and use them for forecasting. However, the class of series suitable for forecasting is considerably wider than the class of series for parameter estimation and hence the forecasting methods considered below do not estimate series parameters but only use some features of the estimated subspace.

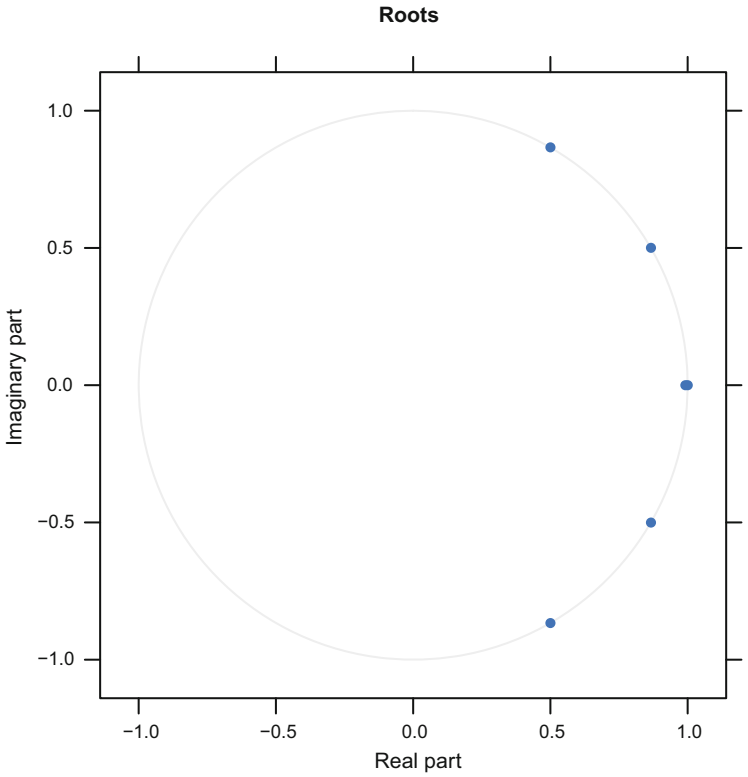

Fig. 3.2 "CO2": Six signal roots

Generally, the forecasted series should have a structure to forecast. In the framework of SSA, we say that $\mathbb{S}$ has a structure if $\mathbb{S}$ is governed by an LRR. However, it is very important to note that SSA forecasts are meaningful for a much wider class of series when an LRR gives an adequate description of the structure of the series only locally rather than globally, which is a requirement for parameter estimation. For example, as a rule, a trend does not satisfy an LRR on the whole time range but it can be locally approximated by a smooth series governed by an LRR. In particular, relatively reliable forecasts can be made for the series which can be approximated by a series of the form $s_n = \sum_{i=1}^{r} C_i(n)\mu_i^n$, $\mu_i \in \mathsf{C}$, where $C_i(n)$ are slowly varying functions of n.

In any version of the forecasting algorithm, we should assume that the series $\mathbb{S}$ is approximately separated from $\mathbb{R}$ by a chosen modification of the SSA method.

3.2.1 Method

The methods of SSA forecasting are closely related to construction of LRRs described in Sect. 3.1. Below we describe two forecasting algorithms. One of them directly uses the constructed LRR for forecasting, whereas the other algorithm does it implicitly. Both algorithms provide the same forecasts of series governed by LRRs, if parameters of SSA are chosen properly.

3.2.1.1 Approach

We will use the same notation as in Sect. 3.1.

Let $\{P_i\}$ be a basis of a subspace $\widetilde{\mathcal{S}}$ of R^L. Then we can state the problem of forecasting in this subspace.

Different modifications of SSA described in Chap. 2 (except for SSA with projection) provide an estimate of a basis of the signal subspace.

If an SSA modification is applied and a set of eigentriples $\{(\sigma_i, P_i, Q_i),\ i \in I\}$ is chosen for reconstruction, then $\widetilde{\mathcal{S}} = \mathrm{span}\{P_i, i \in I\}$. The set of vectors $\{P_i, i \in I\}$ is not necessarily an orthonormal basis. The first mandatory step is therefore the ortho-normalization of the set of vectors $\{P_i, i \in I\}$. After making this step we can construct forecasting algorithms considering an orthonormal basis as an input.

The subspace $\widetilde{\mathcal{S}}$ produces coefficients of a linear combination for reconstruction of the last coordinates of vectors from $\widetilde{\mathcal{S}}$ through their first $L-1$ coordinates. The linear combination used for forecasting has minimal Euclidean norm of coefficients among all linear combinations that correspond to the subspace $\widetilde{\mathcal{S}}$. If $\widetilde{\mathcal{S}}$ is exactly the trajectory subspace of a series governed by an LRR, then the linear combination with minimal norm corresponds to the min-norm LRR. Moreover, the continuation of the series in this subspace is unique.

However, if the series subspace is estimated approximately, several versions of forecasting can be suggested. If the estimation of the subspace was accurate enough, then different forecasting versions will be close. Otherwise, they can differ considerably.

Now we formally describe the forecasting algorithms. For detailed explanation, see Golyandina et al. (2001; Chapter 2).

3.2.1.2 Recurrent Forecasting

The recurrent SSA forecasting is performed by means of the min-norm LRR defined in (3.1).

The *recurrent forecasting method* can be formulated as follows:

1. The time series $\mathbb{Y}_{N+M} = (y_1, \ldots, y_{N+M})$ is defined by

$$y_i = \begin{cases} \widetilde{x}_i & \text{for } i = 1, \ldots, N, \\ \sum\limits_{j=1}^{L-1} a_j y_{i-j} & \text{for } i = N+1, \ldots, N+M. \end{cases} \tag{3.2}$$

2. The numbers $y_{N+1}, \ldots, y_{N+M}$ form the M terms of the recurrent forecast.

Thus, the recurrent forecasting is performed by the direct use of the forecasting LRR with coefficients taken from $\mathcal{R} = (a_{L-1}, \ldots, a_1)$.

Remark 3.1 Let us define the linear operator $\mathcal{P}_{\mathrm{Rec}} : \mathsf{R}^L \mapsto \mathsf{R}^L$ by the formula

$$\mathcal{P}_{\mathrm{Rec}} Z = \begin{pmatrix} \overline{Z} \\ \mathcal{R}^{\mathrm{T}} \overline{Z} \end{pmatrix}, \tag{3.3}$$

where $\overline{Z}$ consists of the last $L-1$ coordinates of Z. Set

$$Y_i = \begin{cases} \widetilde{X}_i & \text{for } i = 1, \ldots, K, \\ \mathcal{P}_{\mathrm{Rec}} Y_{i-1} & \text{for } i = K+1, \ldots, K+M. \end{cases} \tag{3.4}$$

It is easily seen that the matrix $\mathbf{Y} = [Y_1 : \ldots : Y_{K+M}]$ is the trajectory matrix of the series $\mathbb{Y}_{N+M}$. Therefore, (3.4) can be regarded as a vector version of (3.2).

Remark 3.2 In recurrent forecasting, the original series can be taken instead of the reconstructed series as the initial data for the forecasting LRR. This may be sensible only if the leading components are chosen for forecasting. This option can reduce the bias caused by the reconstruction inaccuracy but the volatility of forecasts may increase.

If the LRR is not minimal, then only r of the roots correspond to the signal. Other roots are extraneous and can influence the forecast. Extraneous roots that have moduli larger than 1 can lead to instability.

3.2.1.3 Vector Forecasting

Let $\mathcal{L}_r = \mathrm{span}(P_i, i \in I)$ and $\widehat{X}_i$ be the projection of the lagged vector X_i on $\mathcal{L}_r$. Consider the matrix

$$\Pi = \underline{\mathbf{P}}\,\underline{\mathbf{P}}^{\mathrm{T}} + (1 - \nu^2)\mathcal{R}\mathcal{R}^{\mathrm{T}}, \tag{3.5}$$

where $\underline{\mathbf{P}} = [\underline{P_1} : \ldots : \underline{P_r}]$ and $\mathcal{R}$ is defined in (3.1). The matrix Π defines the linear operator that performs the orthogonal projection $\mathsf{R}^{L-1} \mapsto \underline{\mathcal{L}_r}$, where $\underline{\mathcal{L}_r} =$

span($\underline{P_i}, i \in I$). Finally, we define the linear operator $\mathcal{P}_{\mathrm{Vec}} : \mathsf{R}^L \mapsto \mathcal{L}_r$ by the formula

$$\mathcal{P}_{\mathrm{Vec}} Z = \begin{pmatrix} \Pi \overline{Z} \\ \mathcal{R}^{\mathrm{T}} \overline{Z} \end{pmatrix}. \tag{3.6}$$

The *vector forecasting method* can be formulated as follows:

1. In the notation above, define the vectors

$$Y_i = \begin{cases} \widehat{X}_i & \text{for } i = 1, \ldots, K, \\ \mathcal{P}_{\mathrm{Vec}} Y_{i-1} & \text{for } i = K+1, \ldots, K+M+L-1. \end{cases} \tag{3.7}$$

2. By constructing the matrix $\mathbf{Y} = [Y_1 : \ldots : Y_{K+M+L-1}]$ and making its diagonal averaging we obtain the series $y_1, \ldots, y_{N+M+L-1}$.
3. The numbers $y_{N+1}, \ldots, y_{N+M}$ form the M terms of the vector forecast.

In recurrent forecasting, we perform diagonal averaging to obtain the reconstructed series and then apply the LRR. In the vector forecasting algorithm, these steps are applied in the reverse order. The vector forecast is typically slightly more stable. The current fast implementation of the vector forecasting makes the vector forecasting comparable with recurrent forecasting it terms of the computational cost, see Golyandina et al. (2015).

If the time series component is fully separated from the residual and is governed by an LRR, both recurrent and vector forecasting coincide and provide the exact continuation. In the case of an approximate separability, the recurrent and vector forecasting algorithms give different forecasts.

3.2.1.4 Specificity of SSA Modifications

Basic SSA, Toeplitz SSA, and Filter-adjusted SSA provide orthogonal bases. The basis obtained by Iterated O-SSA needs ortho-normalization. After the subspace is chosen, the forecasting algorithms do not depend on the modification of SSA used.

The algorithms of forecasting for SSA with projection are constructed and implemented in the Rssa package by Alex Shlemov but are not yet properly studied and even properly described. In view of this, we do not discuss these algorithms in this book. For row centering, the algorithm is a generalization of the forecasting with centering described in Golyandina et al. (2001; Section 1.7.1).

3.2.1.5 Bootstrap Confidence and Prediction Intervals

Assume again $\mathbb{X}_N = \mathbb{S}_N + \mathbb{R}_N$. Let us describe the construction of bootstrap confidence intervals for the signal $\mathbb{S}_N$ and its forecast assuming that the signal has

rank r and the residuals are white noise. The algorithm consists of the following steps.

- Fix L, $I = \{1, \dots, r\}$, apply SSA, reconstruct the signal and obtain the decomposition $\mathbb{X}_N = \widetilde{\mathbb{S}}_N + \widetilde{\mathbb{R}}_N$.
- Fix $\widetilde{\mathbb{S}}_N$, calculate the empirical distribution of the residual $\widetilde{\mathbb{R}}_N$.
- Simulate Q independent copies $\widetilde{\mathbb{R}}_{N,i}$, $i = 1, \dots, Q$, using the empirical distribution, construct $\widetilde{\mathbb{X}}_{N,i} = \widetilde{\mathbb{S}}_N + \widetilde{\mathbb{R}}_{N,i}$.
- Apply SSA with the same L and I to $\widetilde{\mathbb{X}}_{N,i}$, reconstruct the signal, then perform M-step ahead forecasting and obtain $\widetilde{\mathbb{S}}_{N+M,i}$, $i = 1, \dots, Q$.
- For each time point j consider the sample $\widetilde{s}_{j,i}$, $i = 1, \dots, Q$, and construct the *bootstrap γ-confidence interval* as the interval defined by $(1-\gamma)/2$- lower and upper sample quantiles. The sample mean is called *average bootstrap forecast*.

Remark 3.3 In the same manner as for linear regression, the prediction intervals can be considered in addition to the confidence intervals. The prediction intervals are constructed as the confidence intervals enlarged by the values of quantiles of the noise distribution. While the confidence intervals show the bounds for the signal and its forecast, the prediction intervals determine the bounds for the whole series and its prediction. Note that in the case of linear regression, this is the theoretical approach; for SSA, this is an empirical approach. To estimate quantiles of the noise distribution, we use $(1-\gamma)/2$- lower and upper sample quantiles of the residuals $\widetilde{R}_N = (r_1, \dots, r_N)$. To construct the *bootstrap γ-prediction intervals*, these quantiles are added to the lower and upper bounds of the γ-confidence interval, correspondingly.

Note that for cross-validation of SSA forecasts future values should not be involved for construction of forecasts and choice of parameters; in this respect, see a discussion in a recent paper (Du et al. 2017).

3.2.2 Algorithms

Let a version of SSA be applied to the time series $\mathbb{X}$ and let an eigentriple group $\{(\sigma_i, P_i, Q_i),\ i \in I\}$ be chosen for reconstruction. The suggested forecasting algorithms are formulated for forecasting in the subspace $\mathcal{L}_r = \mathrm{span}\{P_i, i \in I\} \subset \mathsf{R}^L$. For simplicity, we assume that $I = \{1, \dots, r\}$ and the vectors P_i, $i \in I$, are orthonormal. Note that the forecasting values do not depend on the choice of basis in $\mathcal{L}_r$.

Algorithm 3.5 is written in the form, when the reconstructed series is taken as a base for forecasting. If the original series is used as the base of forecasting, x_n are taken instead of $\widetilde{x}_n$ for $n = N - L + 2, \dots, N$ at Step 3.

Algorithm 3.5 constructs a forward recurrent forecasting. Backward recurrent forecasting is obtained by applying the forward forecasting to the reversed series.

Algorithm 3.5 Recurrent SSA forecasting

Input: Time series $\mathbb{X}$ of length N, window length L, orthonormal system of vectors $\{P_i\}_{i=1}^r$, forecast horizon M.

Output: Forecast values $(\widetilde{x}_{N+1}, \ldots, \widetilde{x}_{N+M})$.

1: Construct the vector $\mathcal{R} = (a_{L-1}, \ldots, a_1)^{\mathrm{T}}$ of coefficients of the min-norm LRR by Algorithm 3.1 applied to $\{P_i, i \in I\}$.

2: Construct the reconstructed matrix $\widehat{\mathbf{X}} = \mathbf{P}\mathbf{P}^{\mathrm{T}}\mathbf{X}$, where $\mathbf{P} = [P_1 : \ldots : P_r]$, and the reconstructed series $\widetilde{\mathbb{X}} = (\widetilde{x}_1, \ldots, \widetilde{x}_N)$ by $\widetilde{\mathbb{X}} = \mathcal{T}_{\mathrm{SSA}}^{-1} \circ \Pi_{\mathcal{H}}(\widehat{\mathbf{X}})$.

3: Calculate the forecast values by applying the min-norm LRR:

$$\widetilde{x}_n = \sum_{i=1}^{L-1} a_i \widetilde{x}_{n-i}, \ n = N+1, \ldots, N+M$$

The next algorithm implements the algorithm of vector forecasting, where the application of the min-norm LRR and the hankelization operation are taken in the reverse order.

Algorithm 3.6 Vector SSA forecasting

Input: Time series $\mathbb{X}$ of length N, window length L, orthonormal system of vectors $\{P_i\}_{i=1}^r$, forecast horizon M.

Output: Forecast values $(\widetilde{x}_{N+1}, \ldots, \widetilde{x}_{N+M})$.

1: Obtain the vector $\mathcal{R} = (a_{L-1}, \ldots, a_1)^{\mathrm{T}}$ of coefficients of the min-norm LRR by Algorithm 3.1 applied to $\{P_i, i \in I\}$.

2: Calculate the matrix Π of projection given in (3.5).

3: Construct the reconstructed matrix $\widehat{\mathbf{X}} = \mathbf{P}\mathbf{P}^{\mathrm{T}}\mathbf{X}$, where $\mathbf{P} = [P_1 \ldots : P_r]$.

4: Extend the reconstructed matrix $\widehat{\mathbf{X}} = [\widehat{X}_1 : \ldots : \widehat{X}_K]$ by column vectors:

$$\widehat{X}_n = \mathcal{P}_{\mathrm{Vec}} \widehat{X}_{n-1} \text{ for } n = K+1, \ldots, K+M+L-1,$$

where $\mathcal{P}_{\mathrm{Vec}}$ is given in (3.6) and uses Π and $\mathcal{R}$. Denote the extended matrix $\widehat{\mathbf{X}}_{\mathrm{ext}} \in \mathsf{R}^{L \times (K+M+L-1)}$.

5: Obtain the extended reconstructed series $\widetilde{\mathbb{X}}_{\mathrm{ext}} = (\widetilde{x}_1, \ldots, \widetilde{x}_{N+M+L-1})$ as $\widetilde{\mathbb{X}}_{\mathrm{ext}} = \mathcal{T}_{\mathrm{SSA}}^{-1} \circ \Pi_{\mathcal{H}}(\widehat{\mathbf{X}}_{\mathrm{ext}})$.

6: Return the forecast values $(\widetilde{x}_{N+1}, \ldots, \widetilde{x}_{N+M})$.

Additional $L-1$ vectors $\widehat{X}_n$ at Step 4 are calculated to make the forecast values independent on the forecast horizon.

In Algorithm 3.6, the reconstructed series is taken as the base for forecasting. For vector forecasting, it makes little sense to use the original series as the base for forecasting.

Note that in this straightforward form, Algorithm 3.6 has a much larger computational cost than Algorithm 3.5. However, a fast implementation described in Golyandina et al. (2015; Section 6.3) and realized in Rssa makes the vector forecasting as fast as the recurrent one.

3.2.3 Forecasting in RSSA

3.2.3.1 Description of Functions

Forecasting becomes available after an SSA decomposition is performed. RSSA implements two methods of SSA forecasting, the recurrent and vector ones, with construction of bootstrap confidence intervals if the signal is forecasted. The package provides different interfaces for forecasting.

Let the decomposition `s` be constructed by one of SSA modifications. For example, we construct the decomposition by Basic SSA as `s <- ssa(x)`. Then typical calls of forecasting functions are

```
# Recurrent forecasting
fr <- rforecast(s, groups = list(1, c(2:3)), len = 1,
                only.new = TRUE)

# Vector forecasting
fv <- vforecast(s, groups = list(trend = c(1,4)), len = 12,
                only.new = FALSE, drop = FALSE)
```

Arguments

`s` is an `ssa` object holding the decomposition;
`groups` is a list of groups of eigentriples to be used in the forecast;
`len` is a number of terms to forecast;
`base` is a series used as a "seed" of forecast: `"original"` or `"reconstructed"` (default) according to the value of groups argument;
`reverse` : `TRUE` means that the recurrent forecast is backward;
`only.new` : if `TRUE`, only the forecast values are returned; otherwise, the forecasted values from the parameter `base` are added;
`drop` acts only if one group is chosen; `TRUE` (default) value means that the result is transformed from the list of the forecasted series to the forecasted series itself.

The following function can perform the chosen forecasting algorithm along with construction of bootstrap confidence intervals. For example, one can call

```
# Bootstrap confidence intervals
bf <- bforecast(s, groups = list(1, c(2:3)), len = 1,
                R = 100, level = 0.95,
                type = "recurrent",
                interval = "confidence", only.intervals = FALSE)
```

In addition to parameters of `rforecast` and `vforecast`, the parameter `R` defines the number of simulations and `level` denotes the confidence level. The parameter `type`, which might take either `"recurrent"` or `"vector"` values, indicates what kind of forecasting should be used. The parameter `only.intervals` influences the construction of the forecast. If `only.intervals = TRUE`, then the forecast coincides with the result of the function `rforecast` (or `vforecast`, in the case of vector forecast). For the default value `only.intervals = FALSE`, the forecasting values are obtained by averaging the forecasts, which were performed during the

construction of bootstrap confidence intervals. Since these forecasts are constructed for simulated series, both the bootstrap intervals and the forecasting values can differ for different calls of `bforecasts`.

The argument `interval` takes the value from `c("confidence","prediction")`. The value `"confidence"` means that the bootstrap confidence bounds for the reconstructed signal/forecast are calculated; the value `"prediction"` means that the bootstrap prediction intervals for the whole series are computed.

One of the parameters of `rforecast` and `vforecast`, which can be added to the arguments of the functions `bforecast`, is the parameter `only.new`. If `only.new = FALSE`, then the bootstrap intervals are constructed for both the signal/series and the forecast.

The following all-in-one function is designed to form an input for visualization of the forecast results by means of the FORECAST package (Hyndman 2017).

```
# All-in-one forecasting
f <- forecast(s,
              groups = list(trend = 1:4), len = 12,
              method = "recurrent",
              interval = "confidence",
              level = c(0.8, 0.99))
```

The function `predict` is exactly the same as `forecast` except for the form of the returned value.

The parameter `method` can have values `"recurrent"` (by default) and `"vector"`. The value of `interval` is from `c("none", "confidence", "prediction")`. If `interval = "none"`, then bootstrap intervals are not constructed. Opposite to the function `bforecast`, the default value of the parameter `only.intervals` in the function `forecast` is `TRUE`. Parameters of `rforecast`, `vforecast`, or `bforecast` can be added to the parameters of the functions `forecast` and `predict` depending on the values of `method` and `interval`. One of the parameters, which can be formally added, is `only.new`. However, for the function `forecast` the value of this parameter is forced to `TRUE`.

Note that the help information for `forecast` and `predict` can be obtained in R as `?forecast.ssa` and `?predict.ssa`.

Like the function `reconstruct`, all the forecasting routines try to use the attributes of the initial series for the resulting series (in particular, they try to add to the result the time index of the series). Unfortunately, this cannot be done in class-neutral way as it is done in the `reconstruct` case and needs to be handled separately for each possible type of time series. The forecasting routines know how to impute the time indices for some standard time series classes like `ts` and `zooreg`.

3.2.3.2 Typical Code

Let us demonstrate the result of application of the family of forecasting functions to the series "CO2," see Fragment 3.2.1.

Fragment 3.2.1 (Forecasting of "CO2")

```
> # Decomposition stage
> s <- ssa(co2, L = 120)
> # Recurrent forecast, the result is the forecast values only
> # The result is the set of forecasts for each group
> for1 <- rforecast(s, groups = list(1, c(1,4), 1:4, 1:6),
+                   len = 12)
> matplot(data.frame(for1), type = "b",
+         pch = c("1", "2", "3", "4"), ylab = "")
> # Vector forecast, the forecasted points are
> # added to the base series
> for1a <- vforecast(s,
+                    groups = list(1, trend = c(1,4), 1:4, 1:6),
+                    len = 36, only.new = FALSE)
> # Plot of the forecast based on the second group c(1,4)
> plot(cbind(co2, for1a$trend), plot.type = "single",
+      col = c("black", "red"), ylab = NULL)
> # Reverse recurrent forecast
> len <- 60
> for2 <- rforecast(s, groups = list(1:6), len = len,
+                   only.new = TRUE, reverse =  TRUE)
> initial <- c(rep(NA, len), co2)
> forecasted <- c(for2, rep(NA, length(co2)))
> matplot(data.frame(initial, forecasted), ylab = NULL,
+         type = "l", col = c("black", "red"), lty = c(1, 1))
> set.seed(3)
> for3 <- forecast(s, groups = list(1:6),
+                  method = "recurrent", interval = "confidence",
+                  only.intervals = FALSE,
+                  len = 24, R = 100, level = 0.99)
> plot(for3, include = 36, shadecols = "green", type = "l",
+      main = "Confidence intervals")
> set.seed(3)
> for4 <- forecast(s, groups = list(1:6),
+                  method = "recurrent", interval = "prediction",
+                  only.intervals = FALSE,
+                  len = 24, R = 100, level = 0.99)
> plot(for4, include = 36, shadecols = "green", type = "l",
+      main = "Prediction intervals")
```

Analysis of the Basic SSA decomposition (see Sect. 2.1 for recommendations) shows that ET1,4 can be referred to a trend, while ET2-3,5–6 make the seasonality group. Figure 3.3 shows a set of the forecast values for different eigentriple groups. The forecast for trend (ET1 and ET4) is shown in Fig. 3.4 together with the reconstructed series. Figure 3.5 demonstrates backward recurrent forecast. Figure 3.6 shows the bootstrap confidence and prediction intervals for the forecasts; it uses the graphical tools from the FORECAST package. Recall that the confidence intervals are constructed for the signal forecast, while the prediction intervals are constructed for the forecast of the whole signal. Therefore, prediction intervals are wider.

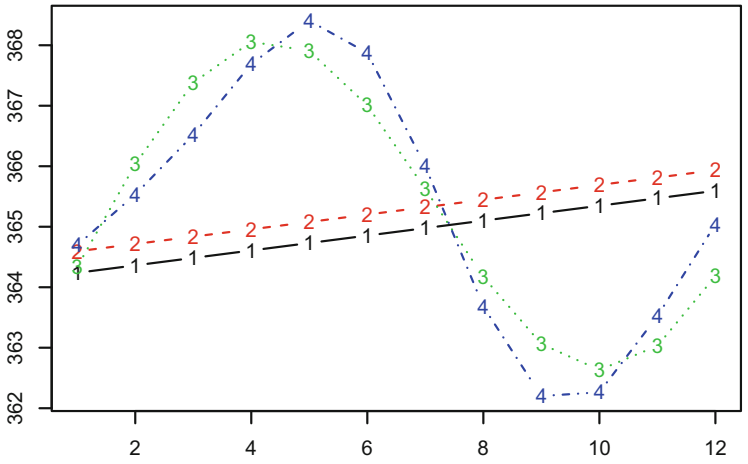

Fig. 3.3 "CO2": A set of recurrent forecasts

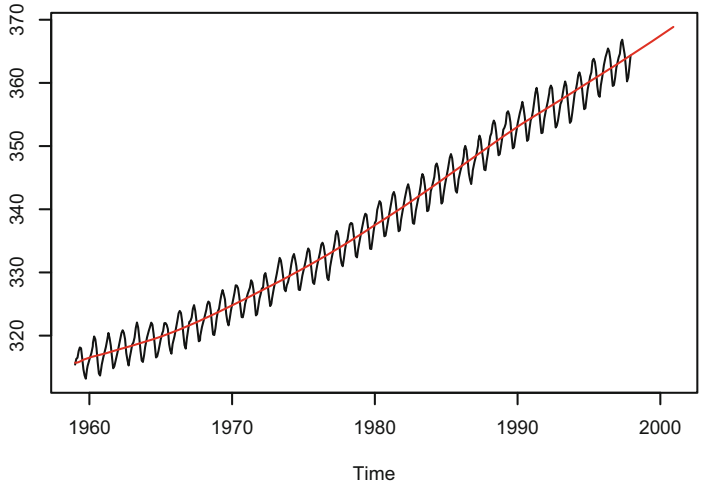

Fig. 3.4 "CO2": Forecast of trend

3.3 Gap Filling

This section is devoted to the extension of the SSA forecasting algorithms for the analysis of time series with missing data.

There are three approaches for solving this problem. The first approach was suggested in Schoellhamer (2001). This approach is suitable for stationary time series only and uses the following simple idea: in the process of the calculation of the inner products of vectors with missing components we use only pairs of valid

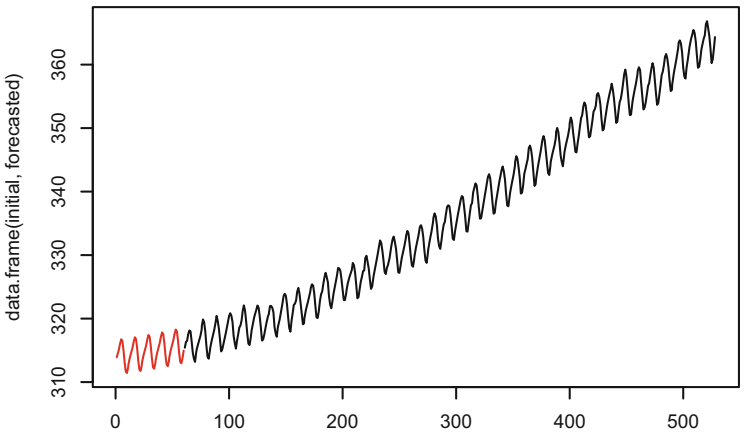

Fig. 3.5 "CO2": Backward forecast of the signal

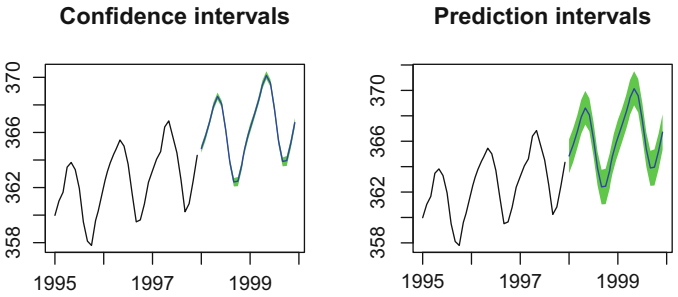

Fig. 3.6 "CO2": Plots of confidence and prediction intervals for the forecast

vector components and omit the others. The Rssa package does not implement this approach in view of its limitations. We hence concentrate on the other two approaches, the subspace-based (Golyandina and Osipov 2007) approach and the iterative (Kondrashov and Ghil 2006) one.

Usually, the problem of missing data imputation is stated as the problem of filling-in the signal data. However, the problem of imputation is more general. For example, one can be interested in imputation of missing data in the trend or seasonality only. To do it, the structure, which we are interested in, should be detected by the method. From the viewpoint of SSA, it means that the interesting series component should be separated from the residual and also the rule for the component extraction should be fixed (e.g., the indices of the eigentriples for reconstruction should be set in Basic SSA). Therefore, it makes sense to combine the considered methods with the SSA modifications described in Chap. 2 that improve separability.

For detection of structure prior to performing gap filling, Shaped SSA can be applied to the series if the location of the gaps allows the decomposition (see

Sect. 2.6). If Shaped SSA gives unsatisfactory results (for example, if the number of complete lagged vectors is too small and therefore detection of the structure is impossible), then the subspace-based approach is not applicable. However, the following general technique can be applied in the framework of the iterative approach: artificial gaps can be added and parameters of the method of gap filling can be chosen to minimize the error of imputation.

3.3.1 Method

3.3.1.1 Subspace-Based Approach

The subspace-based method of gap filling suggested in Golyandina and Osipov (2007) (see also Golyandina and Zhigljavsky (2013; Section 3.7)) is an extension of SSA forecasting algorithms. For forecasting, the last vector coordinate in a chosen subspace can be uniquely imputed as a linear combination of the first $L-1$ coordinates. The approach can be extended for imputing a set of unknown (missing) vector coordinates as linear combinations of known coordinates. Here we use a found signal structure (in the form of a subspace) to fill the gaps. In a particular case, when missing values are located at the end of the series, the problem of filling-in of these values coincides with the problem of forecasting.

The assumptions for the gap filling are the same as for forecasting; that is, SSA should be able to approximately separate the series component of interest.

Note that imputation of gaps in separate signal components can be performed as the following two-step procedure: first, we fill gaps in the whole signal and then decompose the reconstructed signal into desired components.

Clusters of Missing Data

In the subspace-based approach, the gap filling method can be applied to different groups of missing data independently. To introduce such independent clusters, let us give several definitions following (Golyandina and Osipov 2007).

Definition 3.1 For a fixed L, a sequence of missing data of a time series is called a *cluster of missing data* if every two adjacent missing values from this sequence are separated by less than L non-missing values and there is no missing data among L neighbors (if they exist) of the left/right element of the cluster.

Thus, a group of not less than L successive non-missing values of the series separates clusters of missing data.

A cluster is called *left/right* if its left/right element is located at a distance of less than L from the left/right end of the series. A cluster is called *continuous* if it does not contain non-missing data.

The Layout of the Algorithm

Let us describe the algorithm layout. Note that the description below is different from the descriptions provided in Golyandina and Osipov (2007) and Golyandina and Zhigljavsky (2013; Section 3.7); we present it here in the form that is implemented in the RSSA package.

Assume that we have the initial time series $\mathbb{X}_N = (x_1, \ldots, x_N)$ consisting of N elements, some part of which is unknown. Let us describe the scheme of the algorithm assuming that we are reconstructing the first component $\mathbb{X}_N^{(1)}$ of the observed series $\mathbb{X}_N = \mathbb{X}_N^{(1)} + \mathbb{X}_N^{(2)}$.

The scheme of the method is as follows. Parameters are the window length L and a group I of components in the SSA decomposition. We assume that the location of missing data allows application of Shaped SSA for the chosen L. Two versions, "sequential" and "simultaneous" are suggested. These versions correspond to sequential recurrent forecasting and simultaneous vector forecasting, respectively.

Scheme of Subspace-Based Gap Filling

1. **Shaped SSA.** For the series, the shaped version of SSA is applied for the given window length L and group I. Any modification described in Chap. 2 and consistent with Shaped SSA can be used. As a result, we obtain a reconstructed series and a set of orthonormal vectors providing a basis for the approximated signal subspace.
2. **Detection of clusters of missing data.** All missing entries are split into clusters. For sequential version, each cluster is transformed into a continuous one; that is, non-missing values within the cluster are changed to NA.
3. **Forecasting.** For forecasting or filling-in several values, two approaches can be used, sequential and simultaneous. Both of these approaches can use either the recurrent or vector forecasting methods. In the current version of the RSSA package, the sequential approach uses the recurrent forecasting method, while the simultaneous approach uses a method similar to the vector forecasting one.
   Sequential gap filling-in makes forecasting from the left and from the right for each cluster with subsequent weighted averaging of the forecasting results; the subspace used for forecasting is estimated by Shaped SSA. If a cluster is left or right, then only one forecast is used for gap-filling.
   For simultaneous gap filling the so-called simultaneous forecasting is used. In Golyandina and Zhigljavsky (2013; Section 3.1), simultaneous forecasting is described in its recurrent version. Since here we use the vector version, the simultaneous gap filling coincides with the description in Golyandina and Osipov (2007), where the method consists of two operations called "Π-projection" and "simultaneous filling-in."

Discussion

Note that for successful imputation, an approximate separability of the imputed component is necessary. For exactly separated component, the missing values can be reconstructed with no error. The location of missing data is very important for the possibility of imputation by the subspace method, since the number of non-missing values should be large enough for achieving separability by Shaped SSA. At least, the number of the complete lagged vectors should be larger than the rank of the imputed time series component.

3.3.1.2 Iterative Approach

A natural and simple idea for filling-in missing values is the iterative approach, when the missing entries are initially filled-in using some reasonable values and then these values are iteratively improved by updating the SSA approximations for underlying structure of the object. This idea was suggested in Beckers and Rixen (2003) for the imputation of missing values in noisy rank-deficient matrices and was later extended to time series in Kondrashov and Ghil (2006).

For a rank-deficient matrix, the structure is defined by its rank and therefore the improvement is performed by the SVD, where the first r SVD components describe this structure. Time series of finite rank r can be considered in the form of its trajectory matrix, which has rank r and also is a Hankel matrix. Therefore, the improvement can be obtained with the help of the SVD of the trajectory matrix with subsequent hankelization. Note that this is exactly the Basic SSA algorithm with reconstruction. Also, Toeplitz SSA or SSA with projection can be used at iterations, if the series is stationary or we partly know the series model.

At each iteration, we insert the improved values at the places of missing entries and restore the initially used data at the places of non-missing entries. The initially used data may be of two types. First, the original values are used. Second, if application of Shaped SSA is possible, then the reconstructed values can be used instead of the original ones.

The approach described above can be formally applied for almost any location of missing values. Numerical experiments shows that the iterative approach can fail if missing data are located at the ends of the time series.

The iterative approach has no rigorous proof of convergence. Another drawback of the iterative approach is its impossibility to fill-in the gaps exactly even for noiseless signals. Moreover, the iterative method has large computational cost.

3.3.2 Algorithms

Let us start with describing a simpler iterative gap-filling algorithm. For a collection $\mathbb{Y}$ and a set of indices P we denote by $\mathbb{Y}\big|_P$ the part of the collection with the indices from P. Set $\mathcal{N} = \{1, \dots, N\}$.

Algorithm 3.7 Iterative gap filling

Input: Time series $\mathbb{X}$ of length N containing gaps, set of indices of missing values P, window length L, version of SSA, series $\mathbb{G}$ of length N as the source of initial values for gaps, rank for reconstruction r, stop criterion STOP.

Output: Reconstructed series component $\widetilde{\mathbb{X}}$ with no gaps.

1: $k \leftarrow 0$, $\widetilde{\mathbb{G}}^{(k)}\big|_P = \mathbb{G}\big|_P$, $I = \{1, \dots, r\}$.
2: Set $\widetilde{\mathbb{X}}^{(k+1)}$ such that $\widetilde{\mathbb{X}}^{(k+1)}\big|_{\mathcal{N}\setminus P} = \mathbb{X}\big|_{\mathcal{N}\setminus P}$ and $\widetilde{\mathbb{X}}^{(k+1)}\big|_P = \mathbb{G}^{(k)}\big|_P$.
3: Apply the selected version of SSA with the chosen L and I to $\widetilde{\mathbb{X}}^{(k+1)}$ and obtain the reconstructed series $\mathbb{G}^{(k+1)}$.
4: $k \leftarrow k + 1$
5: If not STOP, go to Step 2; else $\widetilde{\mathbb{X}} = \mathbb{G}^{(k)}$.

Input of the algorithm can contain several groups of indices I_k, $k = 1, \dots m$. Then the iterations are performed for $r = \max\{i : i \in I_k, k = 1, \dots, m\}$. In this case, the reconstruction at the last step before STOP is performed for each group I_k separately.

There is a modification of Algorithm 3.7, where $\widetilde{\mathbb{X}}^{(k+1)}\big|_{\mathcal{N}\setminus P}$ at step 2 is taken from the reconstructed series, which is calculated by Shaped SSA applied to the initial time series $\mathbb{X}$.

Below we only provide a short description of the algorithms of subspace-based filling-in. More comprehensive description and mathematical details of the algorithms can be found in Golyandina and Osipov (2007) and Golyandina and Zhigljavsky (2013; Section 3.7). The algorithms can deal with several gaps. We only describe their versions for one internal gap.

The following algorithm corresponds to a combination of methods "sequential filling-in from the left" and "sequential filling-in from the right."

Algorithm 3.8 Sequential recurrent subspace-based gap filling

Input: Time series $\mathbb{X}$ of length N containing a gap, which starts from ith and finished in jth points, set of gap indices $P = \{i, \dots, j\}$, $p = |P|$, window length L, version of SSA, group of eigentriples I.

Output: Reconstructed series component $\widetilde{\mathbb{X}}$ with a filled gap.

1: Apply the shaped form of the chosen SSA version to $\widetilde{\mathbb{X}}$ (Algorithms 2.13 and 2.14) and obtain the subspace $\mathcal{L} = \operatorname{span}\{P_i, i \in I\}$ and the reconstructed series $\widetilde{\mathbb{X}}$ with gaps.
2: Apply the forward recurrent forecasting algorithm in the subspace $\mathcal{L}$ starting from $(\widetilde{x}_{i-L+1}, \dots, \widetilde{x}_{i-1})$ and construct the p-step recurrent forecast $\mathbb{G}^{\text{left}}$.
3: Apply the backward recurrent forecasting algorithm in the subspace $\mathcal{L}$ starting from $(\widetilde{x}_{j+L-1}, \dots, \widetilde{x}_{j+1})$ and construct the p-step recurrent forecast $\mathbb{G}^{\text{right}}$.
4: Combine $\mathbb{G}^{\text{left}}$ and $\mathbb{G}^{\text{right}}$ to obtain $\mathbb{G}$. For example, $g_i = (1-\alpha_i)g_i^{\text{left}} + \alpha_i g_i^{\text{right}}$, $i = 1, \dots, p$, where $\alpha_i = i/(p+1)$.
5: Set $\widetilde{\mathbb{X}}\big|_P = \mathbb{G}$.

Note that if the gap is right (or left), then only forward (or backward) recurrent forecasting is applied.

There is a modification of the algorithm, when the initial data for the forecasting formula at Steps 2 and 3 is taken from the initial series but not from the reconstructed series as in Algorithm 3.8.

The following algorithm corresponds to the combination of the method "Π-projector" and "simultaneous filling-in" introduced in Golyandina and Osipov (2007).

Algorithm 3.9 Simultaneous vector subspace-based gap filling

Input: Time series $\mathbb{X}$ of length N containing a gap, which starts from ith and finishes in jth points, set of gap indices $P = \{i, \dots, j\}$, $p = |P|$, window length L, version of SSA, group of eigentriples I.

Output: Reconstructed series component $\widetilde{\mathbb{X}}$ with a filled gap.

1: Apply the shaped form of the chosen SSA version to $\widetilde{\mathbb{X}}$ (Algorithms 2.13 and 2.14) and obtain the subspace $\mathcal{L} = \mathrm{span}\{P_i, i \in I\}$ and the reconstructed matrix $\widehat{\mathbf{X}}$ consisting of vectors with non-missing values at all positions.
2: Continue the values of the complete reconstructed vectors according to Hankel structure of the matrix to obtain partly filled reconstructed vectors.
3: Project the valid parts of vectors by means of the Π-projector.
4: Simultaneously fill-in the missing parts of the vectors. If it is impossible, then put `NA` ("not available").
5: Hankelize the matrix $\widehat{\mathbf{X}}$ to obtain the series $\widetilde{\mathbb{X}}$. Hankelization is performed by averaging by non-missing values. If there are no non-missing values, then the result is `NA`.

3.3.3 Gap-Filling in RSSA

3.3.3.1 Description of Functions

Since subspace-based gap filling can be considered as interpolation (that is, as forecasting of the time series to gaps), the call of `gapfill` is similar to a call of the forecasting functions.

Similarly to forecasting, one should estimate the trajectory space of an interpolated series component by an SSA-modification; for example, by a call `s <- ssa(ts, L=120)`. Since series contains gaps, the shaped version of SSA is applied. Shaped SSA results in reconstruction of a set of series points, which can be covered by the chosen window length. Therefore, the set of uncovered points can be wider than the set of missing values. The sequential method considers uncovered points as gaps, while the simultaneous method imputes the initial gaps.

Subspace-based gap filling-in is applied to each cluster of missing values, which are detected automatically. A typical call is as follows:

```
# Subspace-based gap filling
g <- gapfill(s, groups = list(c(1,4),c(2:3,5:6)),
        base = "reconstructed",
        method = "sequential",
        alpha = function(len) seq.int(0, 1, length.out = len))
```

Arguments:

`s` is a shaped `ssa` object holding the decomposition; this kind of the object is obtained for a series with `NA` values.

`groups` is a list of groups of eigentriples used for estimation of the component subspaces.

`base` is a series used as a "seed" for gap filling: original or reconstructed according to the value of groups argument; e.g., for sequential filling-in, this is simply the series a forecasting LRR is applied to.

`method` is a method used for gap filling, `"sequential"` or `"simultaneous"`.

`alpha` is used for `method = "sequential"` and sets weights used for combining forecasts from the left and from the right. It can have the following values: 0 (1) means that only the forecast from the left (respectively, from the right) is used; 0.5 means that the forecasts are averaged. It can be a function of `len`, where `len` is the number of missing data in one gap. By default, the function provides linearly decreasing weights from 1 to 0 from both sides.

Note that the computational cost of subspace-based gap filling is very low.

Iterative gap filling is used when the signal subspace is hard (or even impossible) to estimate. Suppose that we know the rank of the signal r. Then the short call has the form

```
# Iterative gap filling
ig <- igapfill(s, groups=list(1:r),  base = "original")
```

Arguments:

`s` is a shaped `ssa` object holding the decomposition; at least, the window length L is taken from `s`. Decomposition in `s` can be empty if `base = "original"` is used.

`groups` is a list of groups of eigentriples to be used for reconstruction at iterations. Each group I is used at the first iteration. Next iterations use groups $\{1, \ldots, |I|\}$.

`fill` is a time series of the length coinciding with the length of the original series; initial values for the missing entries at the first iteration are taken from this series; if `fill = NULL`, the average of the base series is used.

`tol` is a tolerance for the difference between reconstructions at subsequent iterations.

`maxiter` is an upper bound for the number of iterations; if `maxiter = 0`, the upper bound is not used.

`norm` is a distance function used in the convergence criterion;

`base` is a series used for forced values at non-missing positions at each iteration. `"original"` is used if a shaped decomposition is impossible; `"reconstructed"` is used if Shaped SSA yields an appropriate estimation of the component subspace.

`trace` is logical; specifies whether the convergence process should be traced. For example, the number of iterations for convergence can be found in the trace.

3.3.3.2 Typical Code

Let us demonstrate the methods of gap filling by inserting artificial gaps into the time series "CO2." The methods fill gaps in a series component such as signal or trend; noise component can not be recovered.

Fragment 3.3.1 demonstrates how the subspace-based filling method is able to reconstruct a gap. We take one gap of length 100 in the middle of the time series and use the window length $L = 72$. In this case we have enough data to estimate the signal subspace.

Fragment 3.3.1 (Subspace-Based Gap Filling)

```
> F <- co2
> F[201:300] <- NA
> s <- ssa(F, L = 72)
> g0 <- gapfill(s, groups = list(c(1, 4)), method = "sequential",
+                alpha = 0, base = "reconstructed")
> g1 <- gapfill(s, groups = list(c(1, 4)), method = "sequential",
+                alpha = 1, base = "reconstructed")
> g <- gapfill(s, groups = list(c(1, 4)), method = "sequential",
+              base = "reconstructed")
> plot(co2, col = "black")
> lines(g0, col = "blue", lwd = 2)
> lines(g1, col = "green", lwd = 2)
> lines(g, col = "red", lwd = 2)
```

Figure 3.7 shows the result of sequential filling-in. Sequential filling-in is constructed as a linear combination of forecasts from the left and from the right in the subspace estimated by Shaped SSA applied to the whole series. To choose the subspace, identification of eigentriples should be performed as it was demonstrated in Sect. 2.6. Here ET1,4 correspond to a trend.

It can be seen that the accuracy of forecasts from the left and from the right gets worse as the distance from the corresponding edge increases. Therefore, one should

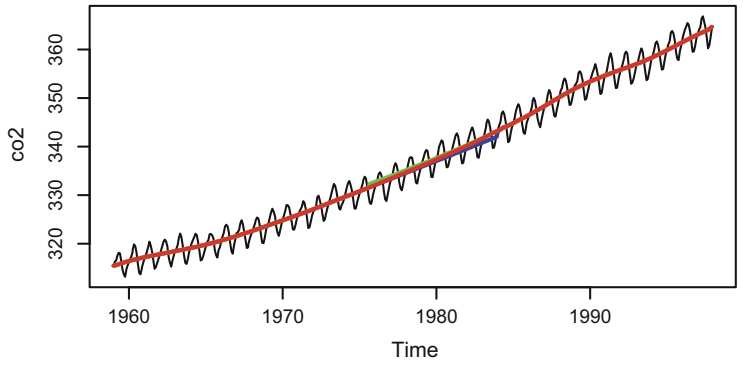

Fig. 3.7 "CO2": Subspace-based gap filling, from the left, from the right, and their combination

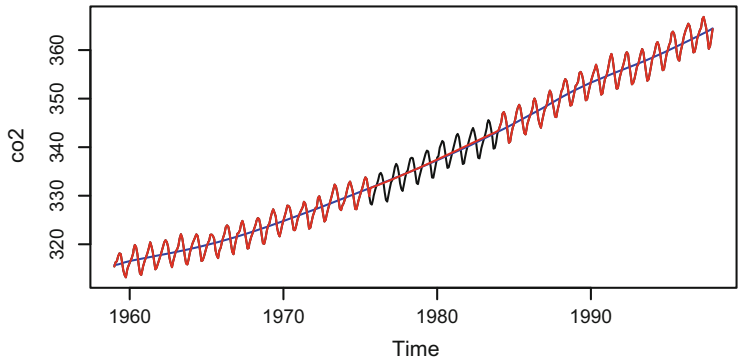

Fig. 3.8 "CO2": Iterative gap filling of trend

combine these forecasts with assigning weights which decrease while moving far from the edges (this combination is discussed in Rodrigues and de Carvalho (2013)). We take the sets of weights linearly decreasing from 1 to 0 with the sum of weights equal to 1. The linear combination of forecasts becomes more accurate.

Since for the recurrent forecast the forecasting LRR is applied to reconstructed series by default, here we also choose `base = "reconstructed"`. The mode `base = "original"` can be used only if the gaps in the signal are imputed, while `base = "reconstructed"` provides a reasonable result for reconstruction of any separable series component, e.g., seasonality.

Simultaneous gap filling cannot fill-in the considered missing data, since the window length is smaller than the size of the gap.

Let us demonstrate how the iterative method works (Fragment 3.3.2). The first example of gaps location is the same as in Fragment 3.3.1 and Fig. 3.7. At each iteration, non-missing values are returned to either reconstructed or original values. The mode with reconstructed values is available only if Shaped SSA for estimation of the component subspace is applicable. The mode with original series values is used when the component subspace cannot be estimated by Shaped SSA because of the gap location. Note that the latter cannot be used for filling-in gaps in components, which are not the leading ones. Figure 3.8 shows that both options yield approximately the same result as the subspace method.

Fragment 3.3.2 (Iterative Gap Filling, One Gap)

```
> F <- co2
> F[201:300] <- NA
> is <- ssa(F, L = 72)
> ig <- igapfill(is, groups = list(c(1,4)),
+                base = "reconstructed")
> igo <- igapfill(is, groups = list(c(1,4)),
+                 base = "original")
> # Compare the result
> plot(co2, col="black")
```

```
> lines(ig, col = "blue", lwd = 1)
> lines(igo, col = "red", lwd = 1)
> ig1 <- igapfill(is, groups = list(c(1, 4)),
+                 base = "original", maxiter = 1)
> ig5 <- igapfill(is, groups = list(c(1, 4)), fill = ig1,
+                 base = "original", maxiter = 4)
> ig10 <- igapfill(is, groups = list(c(1, 4)), fill = ig5,
+                  base = "original", maxiter = 5)
> init.lin <- F
> init.lin[200:301] <- F[200] + (0:101) / 101 * (F[301] - F[200])
> ig.lin <- igapfill(s,
+                    fill = init.lin,
+                    groups = list(c(1, 4)),
+                    base = "original", maxiter = 10)
> # Compare the result
> plot(co2, col = "black")
> lines(ig1, col = "green", lwd = 1)
> lines(ig5, col = "blue", lwd = 1)
> lines(ig10, col = "red", lwd = 1)
> lines(ig.lin, col = "darkred", lwd = 1)
```

By default, iterations are run until the difference between filled-in values becomes smaller than the given accuracy. The reconstructed trend in Fig. 3.8 was obtained with accuracy `tol = 1e-6` and used about 200 iterations; that is, 200 calls of SSA (to obtain this information, one can add `trace = TRUE` to the function parameters).

Results of filling-in after performing 1, 5 and 10 iterations are depicted in Fig. 3.9. It seems that performing 10 iterations is probably enough. The difference between filled-in values in consecutive iterations is 0.17. Note that we continue the iterations ($1 + 4 = 5$, $5 + 5 = 10$) using the results of the previous iterations as the initial values for the next iterations. These iterations start from the initial values by default; these initial values are the mean for the whole time series. Evidently, for

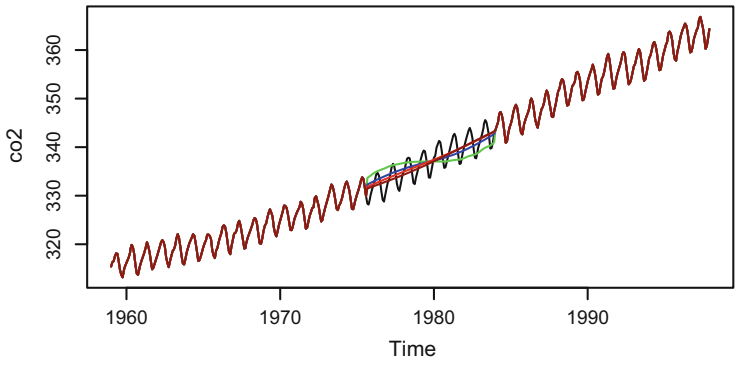

Fig. 3.9 "CO2": Iterative gap filling of trend: convergence

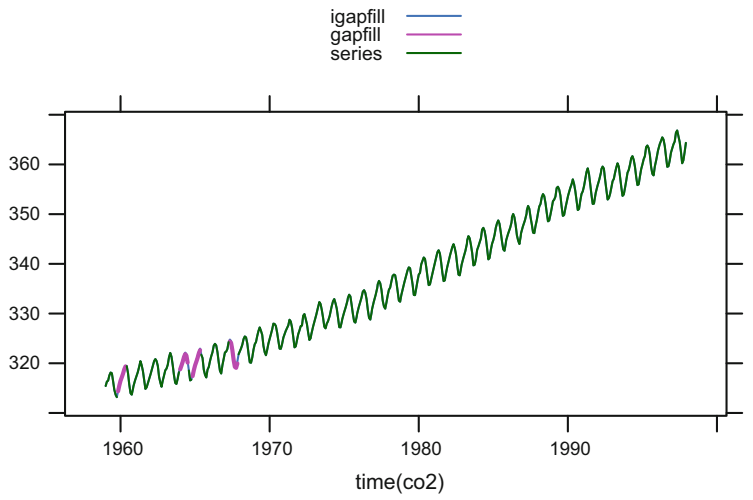

Fig. 3.10 "CO2": Iterative and simultaneous subspace-based gap filling of trend: randomly located gaps

non-stationary time series it is not a good choice. If we take a linear combination of edge values, then 10 iterations give the difference 0.1.

In the second example (Fragment 3.3.3), gaps are located arbitrarily and their location may make it difficult to estimate the signal subspace. Here we use an additional information that the components ET1–6 correspond to a signal. The result of imputation is shown in Fig. 3.10. Recall that in the case of unknown rank of the signal, one can add artificial gaps and choose the number of components to minimize errors for artificial gaps (Kondrashov and Ghil 2006).

Fragment 3.3.3 (Iterative Gap Filling, Several Gaps)

```
> F <- co2
> loc <- c(11:17, 61:67, 71:77, 101:107)
> F[loc] <- NA;
> sr <- ssa(F, L = 200)
> igr <- igapfill(sr, groups = list(c(1:6)), fill = 320,
+                 base = "original", maxiter = 10)
> gr <- gapfill(sr, groups = list(c(1:6)),
+               method = "simultaneous", base = "original")
> G <- rep(NA, length(F)); G[loc] = gr[loc]
> print(mean((gr[loc] - co2[loc])^2)) #MSE of gapfill
[1] 0.1132225
> print(mean((igr[loc] - co2[loc])^2)) #MSE of igapfill
[1] 0.1425962
> xyplot(igr + G + F ~ time(co2), type = "l",
+        lwd = c(1, 2, 1), ylab = NULL,
+        auto.key = list(lines = TRUE, points = FALSE,
+                        text = c("igapfill", "gapfill", "series")))
```

Note that it is typical for the iterative approach to use the parameter value `force.decompose = FALSE` in the preliminary call of `ssa` to avoid the decomposition, which may be impossible due to the gap location. Here we used the default value `TRUE` of `force.decompose`, since it was necessary for demonstration of `gapfill` and `igapfill` with `base = "reconstructed"`.

In addition to iterative gap filling-in, we depict the result of simultaneous subspace-based gap filling, with the original series as the base series. One can see that the results almost coincide. Moreover, the mean-squared error is slightly smaller for the simultaneous filling-in.

3.4 Structured Low-Rank Approximation

3.4.1 Cadzow Iterations

Let us consider the problem of extraction of a finite-rank signal $\mathbb{S}_N$ of rank r from an observed noisy signal $\mathbb{X}_N = \mathbb{S}_N + \mathbb{R}_N$.

The problem of finite-rank approximation can be reduced to the problem of approximation of the L-trajectory matrix $\mathbf{X}$ of the observed time series $\mathbb{X}$ by a Hankel matrix of rank r. This problem belongs to the class of problems of structured low-rank approximation (SLRA), see, e.g., Markovsky et al. (2006), Markovsky (2012).

The Hankel SLRA problem can be stated in two forms: (a) vector form and (b) matrix form. The vector (time series) form of this problem is

$$f_w(\mathbb{Y}) \to \min_{\mathbb{Y}:\operatorname{rank}\mathbb{Y}\le r}, \quad f_w(\mathbb{Y}) = \|\mathbb{X}-\mathbb{Y}\|_w^2 = \sum_{i=1}^{N} w_i (x_i - y_i)^2, \tag{3.8}$$

where $\mathbb{Y} = (y_1, \dots, y_N)$ and $w_1, \dots, w_N$ are some non-negative weights.

The Hankel SLRA problem in the matrix form is the following optimization problem:

$$f_{\mathbf{M}}(\mathbf{Y}) \to \min_{\mathbf{Y}\in\mathcal{M}_r\cap\mathcal{H}}, \quad f_{\mathbf{M}}(\mathbf{Y}) = \|\mathbf{X}-\mathbf{Y}\|_{\mathbf{M}}^2 = \sum_{l=1}^{L}\sum_{k=1}^{K} m_{lk}(x_{lk}-y_{lk})^2, \tag{3.9}$$

where $\mathbf{M} = [m_{lk}]$ is a matrix of size $L \times K$, $\mathcal{H} = \mathcal{M}^{(\mathrm{H})}_{L,K} \subset \mathsf{R}^{L\times K}$ is the space of Hankel matrices of size $L \times K$, $\mathcal{M}_r \subset \mathsf{R}^{L\times K}$ is the set of matrices of rank not larger than r. Matrices $\mathbf{X}$ and $\mathbf{Y}$ are related to vectors (time series) $\mathbb{X}$ and $\mathbb{Y}$ by, respectively, $\mathbf{X} = \mathcal{T}(\mathbb{X})$ and $\mathbf{Y} = \mathcal{T}(\mathbb{Y})$, where $\mathcal{T} = \mathcal{T}_{\mathrm{SSA}}$ is the SSA embedding operator defined in (2.1).

Each $\mathbf{M}$ in (3.9) generates a set of weights w_i in (3.8) by the equality $f_{\mathbf{M}}(\mathbf{Y}) = f_w(\mathbb{Y})$. If $m_{lk} = 1$ for all l and k, then the weights w_i are as in (2.2).

The iterative step of the method of alternating projections for solving (3.9) has the following form:

$$\mathbf{Y}_{k+1} = \breve{\Pi}_{\mathcal{H}} \circ \breve{\Pi}_{\mathcal{M}_r}(\mathbf{Y}_k), \tag{3.10}$$

where $\mathbf{Y}_0 = \mathbf{X}$ and $\breve{\Pi}_{\mathcal{H}}$ and $\breve{\Pi}_{\mathcal{M}_r}$ are projectors to the corresponding sets with respect to the norm $\|\cdot\|_{\mathbf{M}}$.

For the matrix $\mathbf{M}$ with $m_{lk} = 1$ for all l and k, we obtain the well-known method of Cadzow iterations (Cadzow 1988). In this case, $\|\cdot\|_{\mathbf{M}}$ is the conventional Frobenius norm and $\breve{\Pi}_{\mathcal{H}} = \Pi_{\mathcal{H}}$. We will keep the same name "Cadzow iterations" for general $\mathbf{M}$.

The projector $\breve{\Pi}_{\mathcal{H}}$ is calculated in a straightforward way: for $\widehat{\mathbf{Y}} = \breve{\Pi}_{\mathcal{H}}\mathbf{Y}$ we have

$$\hat{y}_{ij} = \frac{\sum_{l,k:\,l+k=i+j} m_{lk}\, y_{lk}}{\sum_{l,k:\,l+k=i+j} m_{lk}}. \tag{3.11}$$

The projector $\breve{\Pi}_{\mathcal{M}_r}$ is easily calculated if there exist $\mathbf{P}$ and $\mathbf{Q}$ such that $\|\mathbf{X} - \mathbf{Y}\|^2_{\mathbf{M}}$ corresponds to the Frobenius norm with respect to the oblique inner products $(X, Y)_{\mathbf{P}} = X^{\mathrm{T}}\mathbf{P}Y$ in R^L and $(X, Y)_{\mathbf{Q}} = X^{\mathrm{T}}\mathbf{Q}Y$ in R^K. Then $\breve{\Pi}_{\mathcal{M}_r}$ is calculated as the r leading terms of the $(\mathbf{P}, \mathbf{Q})$-SVD defined by (2.14) in Definition 2.5.

One of examples of such $\mathbf{M}$ is the case, when $\mathbf{P} = \mathrm{diag}(p_1, \ldots, p_L)$, $\mathbf{Q} = \mathrm{diag}(q_1, \ldots, q_K)$ and therefore $m_{ij} = p_i q_j$. In this case, as explained in Zhigljavsky et al. (2016a), $\|\cdot\|_{\mathbf{M}} = \|\cdot\|_w$ if and only if $W = P \star Q$, where $W = (w_1, \ldots, w_N)$, $P = (p_1, \ldots, p_L)$, $Q = (q_1, \ldots, q_K)$, and $\star$ denotes convolution of vectors.

A natural choice of equal weights w_i in (3.8) corresponds to the ordinary least-squares method. As there are no vectors P and Q with positive elements (in this case $(\cdot,\cdot)_{\mathbf{P}}$ and $(\cdot,\cdot)_{\mathbf{Q}}$ would be norms) such that $(1, 1, \ldots, 1) = P \star Q$ only approximately equal weights w_i, $i = 1, \ldots, N$, can be achieved; see, e.g., Zhigljavsky et al. (2016b) or Gillard and Zhigljavsky (2016).

As a rule, Cadzow iterations do not converge to the optimal solution, see, e.g., Gillard and Zhigljavsky (2011, 2013). There are some partial results on convergence of Cadzow iterations to the set $\mathbf{Y} \in \mathcal{M}_r \cap \mathcal{H}$, but even this question is hard, see, e.g., Zvonarev and Golyandina (2017). One may try to solve (3.8) by applying standard techniques of global optimization in a parametric space (assuming the model of the sum of damped sinusoids, for example) but the arising optimization problem is far too difficult, see Gillard and Kvasov (2016). Another general approach to the numerical solution of the weighted SLRA problems can be found in Markovsky and Usevich (2014).

Assume we have stopped after k iterations of (3.10) and have mapped the matrices to R^N. Then the resulting series can be written as

$$\widetilde{\mathbb{S}}_N = \mathcal{T}^{-1} \circ \left(\breve{\Pi}_{\mathcal{H}} \circ \breve{\Pi}_{\mathcal{M}_r}\right)^k \circ \mathcal{T}\left(\mathbb{X}_N\right). \tag{3.12}$$

The vector (series) $\widetilde{\mathbb{S}}_N$ obtained by (3.12) can be considered as an estimator of the signal. If $k = 1$ and $m_{lk} = 1$ for all l and k, then (3.12) is simply the Basic SSA reconstruction with $I = \{1, \ldots, r\}$.

Let the series length N be divisible by the window length L. Choose some $\alpha > 0$ and set $p_i = 1$ for each i and

$$q_k = q_k(\alpha) = \begin{cases} 1, & \text{if } k = jL + 1 \text{ for some } j, \\ \alpha, & \text{otherwise.} \end{cases} \tag{3.13}$$

According to Zvonarev and Golyandina (2017), we will call the method (3.12) with these $P = (p_1, \ldots, p_L)$ and $Q = (q_1, \ldots, q_K)$ "Cadzow(α) iterations." In this notation, Cadzow(1) corresponds to the conventional Cadzow iterations, while Cadzow(0) corresponds to the Cadzow iterations that would attempt to solve the problem (3.8) with $w_i = 1$ ($i = 1, \ldots, N$). Cadzow(0) does not solve the problem (3.8) in view of the degeneracy of some weights in Q. As an approximation to Cadzow(0) iterations, Cadzow(α) iterations with small $\alpha > 0$ are considered.

It is shown in Zvonarev and Golyandina (2017) that Cadzow(α) with smaller α has slower convergence rate and better accuracy in the limit than Cadzow(α) with larger α.

3.4.2 Algorithms

We call Algorithm 3.10 Cadzow iterations. It implements the iterations (3.12), where projections are performed with respect to the norm, induced by the left and right weight vectors P and Q.

Algorithm 3.10 Cadzow iterations

Input: Time series $\mathbb{X}$ of length N, window length L, version of SSA, weight vectors $P \in \mathsf{R}^L$ and $Q \in \mathsf{R}^K$, rank for reconstruction r, stop criterion STOP.
Output: Approximation $\widetilde{\mathbb{X}}$ of rank r for the series $\mathbb{X}$.

1: $k \leftarrow 0$, $\mathbf{X}^{(0)} = \mathcal{T}(\mathbb{X})$.
2: $\mathbf{X}_r^{(k+1)} = \breve{\Pi}_{\mathcal{M}_r} \mathbf{X}^{(k)}$ taking the leading r components of the $(\mathbf{P}, \mathbf{Q})$-SVD, see Algorithm 2.6.
3: $\mathbf{X}^{(k+1)} = \breve{\Pi}_{\mathcal{H}} \mathbf{X}_r^{(k)}$ by (3.11) with $m_{ij} = p_i q_j$.
4: $k \leftarrow k + 1$;
5: If not STOP, go to Step 2; else $\widetilde{\mathbb{X}} = \mathcal{T}^{-1}(\mathbf{X}^{(k)})$.

Note that in the RSSA package the input data are partly taken from the `ssa` object (e.g., the window length L).

3.4.3 *Structured Low-Rank Approximation in* RSSA

3.4.3.1 Description of Functions

A typical call is as follows:

```
c <- cadzow(s, rank = 2, correct = FALSE, tol = 1e-6,
       maxiter = 100, norm = function(x) max(abs(x)))
```

Arguments:

`s` is an `ssa` object holding the decomposition parameters. Decomposition in `s` can be empty.

`tol` is a tolerance for the difference between reconstructions at sequential iterations.

`maxiter` is an upper bound for the number of iterations; if `maxiter = 0`, this upper bound is not used.

`norm` is a distance function used for the convergence criterion;

`trace` is logical; it indicates whether the convergence process should be traced. For example, the number of iterations for convergence can be found in the trace, which is printed at the output window.

There are no parameters related to weights in Cadzow iterations. This is because the weights are fully specified in the preliminary call of the `ssa` function. For example, in the call

```
s <- ssa(x, column.oblique = 'identity', row.oblique = weights)
```

the parameter `row.oblique = weights` provides the vector of right weights of length K.

3.4.3.2 Typical Code

Let us demonstrate how to perform forecasting by means of the result of weighted Cadzow iterations. We take the first 360 points of the "CO2" series and $L = 60$. Small values of α provide better accuracy of the signal reconstruction but slow convergence, see Zvonarev and Golyandina (2017). Therefore, we take 10 iterations with small $\alpha = 0.01$ and then continue iterations with $\alpha = 1$ to reach convergence. Fragment 3.4.1 contains the code for such iterations. The weights are constructed according to (3.13).

Fragment 3.4.1 (Weighted Cadzow Approximation)

```
> cut <- 49 + 60
> x <- window(co2, end = time(co2)[length(co2) - cut + 1])
> L <- 60
> K <- length(x) - L + 1
> alpha <- 0.01
> weights <- vector(len = K)
> weights[1:K] <- alpha
> weights[seq(K, 1, -L)] <- 1
> xyplot(weights ~ 1:K, type = "l")
> s1 <- ssa(x, L = L) #to detect the series rank
> ncomp <- 6
> s01 <- ssa(x, L = L, column.oblique = "identity",
+            row.oblique = weights)
> c01 <- cadzow(s01, rank = ncomp, maxiter = 10)
> s01.1 <- ssa(c01, L = L, column.oblique = NULL,
+              row.oblique = NULL)
> c01.1 <- cadzow(s01.1, rank = ncomp, tol = 1.e-8 * mean(co2))
> print(t(ssa(c01.1, L = ncomp + 1)$sigma), digits = 5)
      [,1]   [,2]   [,3]  [,4]  [,5]     [,6]       [,7]
[1,] 16472 73.537 41.096 12.29 4.674 0.044457 2.2465e-06
> ss01.1<- ssa(c01.1, L = ncomp + 1)
> fr <- rforecast(ss01.1, groups = list(1:ncomp), len = cut)
> xyplot(cbind(Original = co2, Cadzow1and01 = c01.1,
+              ForecastCadzow = fr), superpose = TRUE)
> print(parestimate(ss01.1, groups = list(1:ncomp),
+                   method = "esprit"))
   period     rate   |    Mod     Arg  |     Re        Im
   11.998   0.000525 |  1.00053   0.52 |  0.86643   0.50035
  -11.998   0.000525 |  1.00053  -0.52 |  0.86643  -0.50035
      Inf   0.000450 |  1.00045  -0.00 |  1.00045  -0.00000
    5.998   0.000287 |  1.00029   1.05 |  0.49986   0.86644
   -5.998   0.000287 |  1.00029  -1.05 |  0.49986  -0.86644
      Inf  -0.004656 |  0.99535  -0.00 |  0.99535  -0.00000
```

The resultant signal estimate `c01.1` has rank $r = 6$: the 7-th singular value is practically zero. The trajectory subspace of this signal for any $L > 6$ uniquely defines the LRR, which governs the estimated signal and provides the forecasting formula. To obtain this formula by means of Rssa, we apply Basic SSA with $L = r + 1$ and then construct forecast by any forecasting method, see Fig. 3.11. In addition to forecasting, parameter estimation can be performed by applying the corresponding function to the `ssa` object.

3.4.3.3 Simulated Example

One example presented in Sect. 3.4.3.2 cannot show if Cadzow(α) iterations are better than the conventional Cadzow(1) iterations. Therefore, we perform simulations for a noisy sine wave. The parameters are chosen to repeat one of the numerical results of Zvonarev and Golyandina (2017). Simulations show that

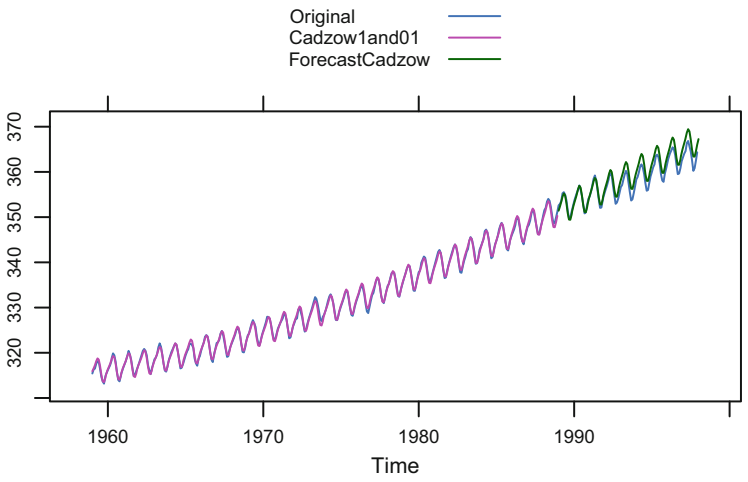

Fig. 3.11 "CO2": Approximation of rank 6 by the weighted Cadzow method and its forecast

$\alpha = 0.01$ provides approximately 10% improvement in the MSE of the signal estimator.

Fragment 3.4.2 (Accuracy of Weighted Cadzow Approximation)

```
> SIMUL <- FALSE
> set.seed(3)
> L <- 20
> N <- 2 * L
> K <- N - L + 1
> alpha <- 0.01
> sigma <- 1
> signal <- 5 * sin(2 * pi * (1:N) / 6)
> weights <- vector(len = K)
> weights[1:K] <- alpha
> weights[seq(1, K, L)] <- 1
> M <- 1000
> norm.meansq <- function(x) mean(x^2)
> if (SIMUL) {
+   RMSE <- sqrt(rowMeans(replicate(M, {
+     x <- signal + sigma * rnorm(N)
+     s.alpha <- ssa(x, L = L, column.oblique = NULL,
+                    row.oblique = weights)
+     c.alpha <- cadzow(s.alpha, rank = 2, tol = 1.e-8,
+                       norm = norm.meansq,
+                       correct = FALSE)
+     s <- ssa(x, L = L)
+     cc <- cadzow(s, rank = 2, norm = norm.meansq, tol = 1.e-8,
+                  correct = FALSE)
+     c("err" = mean((cc - signal)^2),
+       "err.alpha" = mean((c.alpha - signal)^2))
+   })))
```

```
+
+   cadzow.sim <- as.data.frame(t(RMSE))
+ } else {
+   data("cadzow.sim", package = "ssabook")
+ }
> print(cadzow.sim)
        err err.alpha
1 0.3753331 0.3222088
```

3.5 Case Studies

3.5.1 Forecasting of Complex Trend and Seasonality

Let us consider the series "Elec," which was analyzed in Sect. 2.8.8. We will construct forecasts obtained using the subseries with the last two years removed and then compare the two-year forecasts with the known data. The series "Elec" has complex trend and therefore the trend can be extracted by means of Basic SSA with a small window length. Unfortunately, the window length $L = 12$ is too small to obtain a stable forecast. However, larger window lengths would make the signal and residual to be badly mixed. Iterated O-SSA can help to better separate the trend from the residual. Fragment 3.5.1 contains the code, which performs two forecasts, on the base of the whole time series and on the base of the second half of the series. Figure 3.12 shows that Iterative O-SSA allows to obtain accurate forecasts. Moreover, both forecasts are almost the same.

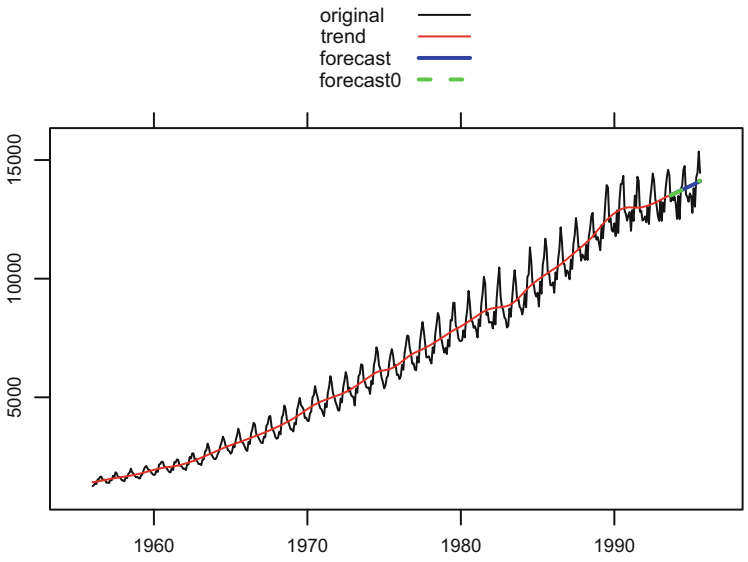

Fig. 3.12 "Elec": Trend forecasting

Fragment 3.5.1 ("Elec": Trend Forecasting and `iossa`)

```
> data("elec", package = "fma")
> N <- length(elec)
> len <- 24
> L <- 24
> s <- ssa(window(elec, end = c(1993, 8)), L = L)
> si <- iossa(s, nested.groups = list(c(1, 4), c(2, 3, 5:10)))
> fi <- rforecast(si, groups = list(trend = c(1:2)),
+                 len = len, only.new = FALSE)
> s0 <- ssa(window(elec, start = c(1972, 8), end = c(1993, 8)),
+           L = L)
> f0 <- vforecast(s0, groups = list(trend = c(1)),
+                 len = len, only.new = TRUE)
> si0 <- iossa(s0, nested.groups = list(c(1,4), c(2,3,5:10)))
> fi0 <- vforecast(si0, groups = list(trend = c(1:2)),
+                  len = len, only.new = TRUE)
> theme <- simpleTheme(col = c("black", "red", "blue", "green"),
+                      lwd = c(1, 1, 2, 2),
+                      lty = c("solid", "solid",
+                              "solid", "dashed"))
> xyplot(cbind(elec,
+              window(fi, end = c(1993, 8)),
+              window(fi, start = c(1993, 9)),
+              fi0),
+        superpose = TRUE, type ="l", ylab = NULL, xlab = NULL,
+        auto.key = list(text = c("original", "trend",
+                                 "forecast", "forecast0"),
+                        type = c("l", "l", "l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme)
```

Unlike the trend, seasonality has a stable structure. The window length L equal to two periods is still too small to obtain an accurate forecast. Therefore, a combined forecast similar to Sequential SSA briefly described in Sect. 2.1.3.2 can be recommended. In Fragment 3.5.2, the trend estimated in Fragment 3.5.1 is subtracted from the series and the residual is forecasted with large window $L = 240$. Figure 3.13 contains the forecast obtained by the sum of trend and seasonality forecasts. Note that the choice of eigentriples for forecasting was performed by the standard technique, which involves the analysis of eigenvectors as demonstrated in Sect. 2.1.5.3.

Fragment 3.5.2 ("Elec": Combined Forecasting)

```
> L <- 240
> elec_sa <- elec - fi
> s_sa <- ssa(window(elec_sa, end = c(1993, 8)), L = L)
> f_sa <- rforecast(s_sa, groups = list(trend = c(1:13)),
+                   len = len, only.new = FALSE)
> theme <- simpleTheme(col = c("black", "red", "green"),
+                      lwd = c(1, 2, 2),
+                      lty = c("solid", "solid", "solid"))
> xyplot(cbind(window(elec, start = c(1985, 12)),
```

```
+                window(fi, start = c(1985, 12), end = c(1993, 8)),
+                window(fi, start = c(1993, 9)) +
+                  window(f_sa, start = c(1993, 9))),
+          superpose = TRUE, type = "l", ylab = NULL, xlab = NULL,
+          auto.key = list(text = c("original", "trend",
+                                   "forecast"),
+                          type = c("l", "l", "l"),
+                          lines = TRUE, points = FALSE),
+          par.settings = theme)
```

3.5.2 *Different Methods of Forecasting*

In the example considered in Sect. 3.5.1, it was not important which forecasting method to apply, since the series "Elec" has a reasonably stable structure. Generally, the closeness of vector and recurrent forecasts can be considered as an indication of structural stability of the series component we have chosen to forecast. This does not guarantee, however, that the forecasts are accurate, since the structure of the series may change in the future.

Here we demonstrate a difference between recurrent and vector forecasts. From the theoretical point of view, the recurrent forecast is simpler, since it is just a continuation by an LRR. An explicit formula for this continuation can be constructed in a similar manner to what is done in Sect. 3.5.5 below. Vector forecasting, however, can be more accurate for continuation of a stable structure; in particular, it performs extra $L - 1$ steps; the extra steps are done for making the

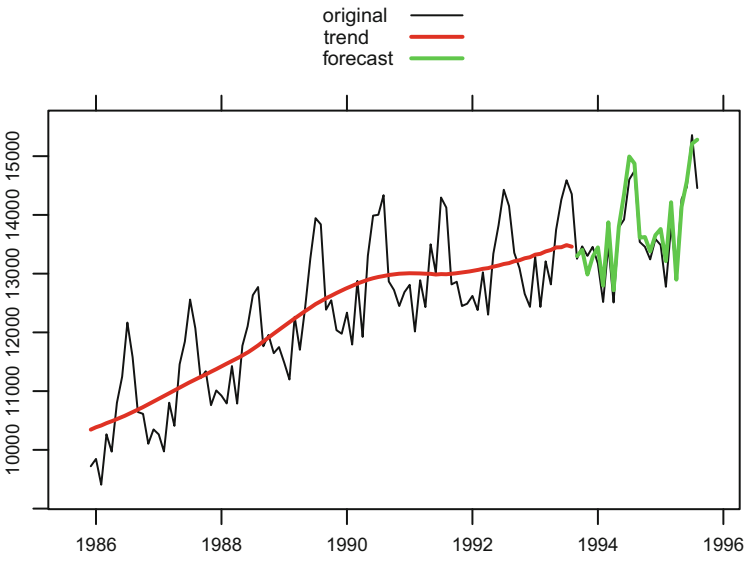

Fig. 3.13 "Elec": Combined forecasting

coincidence between an M-step forecast and the first M points of $(M+1)$-step forecast.

Let us consider a small example "Cowtemp" (daily morning temperature of a cow).

We have removed the last 14 points (2 weeks) and constructed the recurrent and vector forecasts of the 61-term series. Window length was chosen to be equal to 4 weeks. It is large and therefore we can expect the global tendency to be captured well. In addition to the two main forecasts, we consider the forecast based on Toeplitz SSA (see Sect. 2.2). Recall that Toeplitz SSA is designed to analyze stationary series. The considered series probably has a trend. Therefore, we take a small window length for Toeplitz SSA. The code, which implements three methods of forecasting, is contained in Fragment 3.5.3. In all cases, we will construct forecasts based on one leading eigentriple.

Figure 3.14 shows several typical effects. First, if the separability is not good, then the recurrent forecast can have a jump at its first point. We can see that the vector forecast (a) has no jumps and (b) changes the last $L-1$ points of the reconstructed series. Nevertheless, both forecasts are similar. They have the form $C \cdot 0.995^n$.

The estimator of the signal root by Toeplitz SSA is much closer to 1. Note that `parestimate` for Toeplitz SSA force unit moduli of roots by default. For the option `normalize.roots = FALSE`, which we have chosen, the root estimate is almost 1. The root estimator obtained as the maximum-modulus root of the characteristic polynomial of the estimated LRR (which is the forecasted LRR for the recurrent forecast) is equal to 0.999. Thus, we see that the vector and recurrent Basic SSA

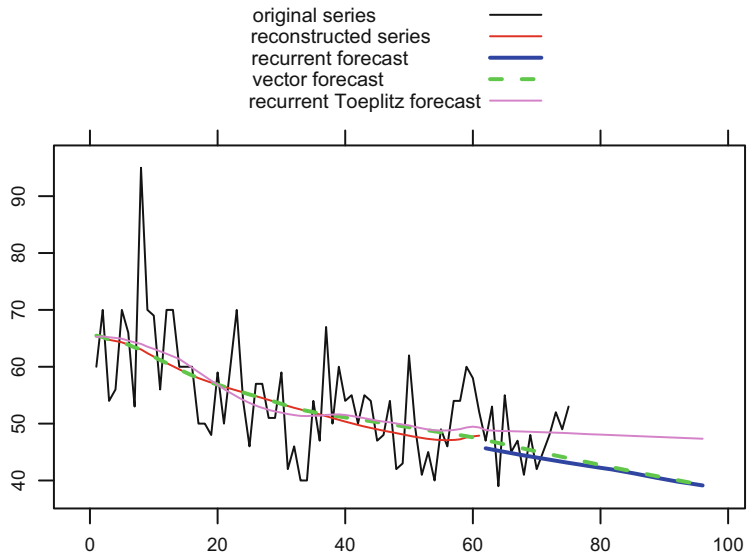

Fig. 3.14 "Cowtemp": Basic SSA and Toeplitz SSA forecasting

forecasts decrease as 0.995^n, while the recurrent Toeplitz SSA forecast decreases slower as 0.999^n.

On the time interval [62, 75], the forecasts are more or less similar. However, the long-term forecast is much more appropriate in the case of Toeplitz SSA, since the temperature is likely to oscillate around a constant and cannot rapidly decrease. Therefore, the method, which adds the limitation of stationarity, wins in this particular example.

Fragment 3.5.3 ("Cowtemp": Different Methods of Forecasting)

```
> data("cowtemp", package = "fma")
> series <- cowtemp
> N <- length(series)
> cut <- 14
> future <- 21
> len <- cut + future
> r <- 1
> L <- 28
> Lt <- 14
> s <- ssa(window(series, end = N - cut), L = L)
> parestimate(s, groups = list(trend = c(1:r)),
+             method = "esprit")$moduli
[1] 0.9946241
> roots(lrr(s, groups = list(trend = c(1:r))))[1]
[1] 0.9953519+0i
> rec <- reconstruct(s, groups = list(1:r))
> st <- ssa(window(series, end = N - cut),
+           kind = "toeplitz-ssa", L = Lt)
> parestimate(st, groups = list(trend = c(1:r)),
+             method = "esprit")$moduli
[1] 1
> parestimate(st, groups = list(trend = c(1:r)),
+             normalize.roots = FALSE,
+             method = "esprit")$moduli
[1] 0.9999984
> roots(lrr(st, groups = list(trend = c(1:r))))[1]
[1] 0.9990458+0i
> fr <- rforecast(s, groups = list(trend = c(1:r)),
+                 len = len, only.new = TRUE)
> fv <- vforecast(s, groups = list(trend = c(1:r)),
+                 len = len, only.new = FALSE)
> ftr <- rforecast(st, groups = list(trend = c(1:r)),
+                  len = len, only.new = FALSE)
> print(sqrt(mean((window(fr, start = 62) -
+               window(series, start = 62))^2)))
[1] 5.711485
> print(sqrt(mean((window(fv, start = 62) -
+               window(series, start = 62))^2)))
[1] 5.253602
> print(sqrt(mean((window(ftr, start = 62) -
+               window(series, start = 62))^2)))
[1] 4.785783
```

```
> theme <- simpleTheme(col = c("black", "red", "blue",
+                              "green", "violet"),
+                      lwd = c(1, 1, 2, 2, 1),
+                      lty = c("solid", "solid", "solid",
+                              "dashed", "solid"))
> future.NA <- rep(NA, future)
> xyplot(cbind(series, rec$F1, fr, fv, ftr),
+        superpose = TRUE, type = "l", ylab = NULL, xlab = NULL,
+        auto.key = list(text = c("original series",
+                                 "reconstructed series",
+                                 "recurrent forecast",
+                                 "vector forecast",
+                                 "recurrent Toeplitz forecast"),
+                        type = c("l", "l", "l", "l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme)
```

3.5.3 Choice of Parameters and Confidence Intervals

Since SSA is a model-free method, there are no theoretical confidence intervals. As shown in Sect. 3.2.1.5 we can naturally construct bootstrap confidence intervals for the signal. The assumption for the adequacy of bootstrap confidence intervals is that the signal is extracted and a model for the residual distribution is built. In practice, these assumptions are not valid (or we cannot check them) and therefore bootstrap confidence intervals can only be considered as indicative measures of accuracy of the forecasts.

By considering bootstrap confidence intervals, the user investigates forecasting stability. As an alternative to bootstrap confidence intervals, one can study forecast response to a perturbation of the initial series. To do this, we add to the initial series a perturbation (e.g., a white noise with some standard deviation σ) and look at the resulting variability of forecasts. Similarly to the case of bootstrap confidence intervals, for a confidence level γ, we will consider the intervals between $(1-\gamma)/2$ lower and upper quantiles and call them *perturbed forecasting intervals* (see Fragment 3.5.4). Note that the influence of perturbation on the signal subspace estimation is theoretically studied in Nekrutkin (2010).

Fragment 3.5.4 (Function for Perturbed Forecasting Intervals)

```
> perturbation <-
+   function(s, noise, R, Qfor, num.comp, L, level, template) {
+   r <- reconstruct(s, groups = list(1:num.comp))
+   stopifnot(length(r) == 1)
+
+   delta <- sd(residuals(r))
+   res <- matrix(nrow = Qfor, ncol = R)
+
+   ser <- s$F; attributes(ser) <- s$Fattr
+   for (j in 1:R) {
```

```
+     si <- ssa(ser + delta * noise[, j], L = L)
+     res[, j] <- rforecast(si, groups = list(1:num.comp),
+                           len = Qfor)
+   }
+
+   cf <- apply(res, 1, quantile,
+               probs = c((1 - level) / 2, (1 + level) / 2))
+   out <- template
+   out$x[] <- ser
+   out$fitted[] <- r[[1]]
+   out$residuals[] <- residuals(r)
+   out$lower[] <- cf[1, ]
+   out$upper[] <- cf[2, ]
+   out$level[] <- 100 * level
+   out$mean[] <- rowMeans(res)
+
+   out
+ }
```

Fragment 3.5.5 shows how the bootstrap and perturbed intervals depend on the number of eigentriples, which are used for reconstruction, for the total wine sales "Total" from the dataset "AustralianWine".

Fragment 3.5.5 ("Total": Stability of Forecasting)

```
> data("AustralianWine", package = "Rssa")
> wine <- window(AustralianWine, end = time(AustralianWine)[174])
> ser0 <- wine[, "Total"]
> Q <- 66
> l <- length(ser0)
> ser <- window(ser0, end = time(ser0)[l-Q])
> include <- min(1000, l - Q)
> L <- 48
> s <- ssa(ser, L = L)
> plot(wcor(s, groups = 1:min(nu(s), 50)),
+      scales = list(at = c(10, 20, 30, 40, 50)))
> set.seed(1)
> R <- 100
> noise <- matrix(rnorm(length(ser) * R), nrow = length(ser))
> range <- 1:30
> err.pert <- numeric(length(range))
> err <- numeric(length(range))
> k <- 1
> for (num.comp in range) {
+   bf0 <- forecast(s, groups = list(1:num.comp),
+                   method = "recurrent",
+                   interval = "confidence",
+                   len = Q, R = R, level = 0.9)
+
+   bf0.pert <- perturbation(s, noise, R, Q, num.comp, L,
+                            level = 0.9, bf0)
+   err.pert[k] <- sqrt(mean((bf0.pert$upper - bf0.pert$lower)^2))
+   err[k] <- sqrt(mean((bf0$upper - bf0$lower)^2))
+   k <- k + 1
```

```
+ }
> bf0.pert1 <- perturbation(s, noise, R, Q, 1, L,
+                           level = 0.9, bf0)
> plot(bf0.pert1, include = include, shadecols = "green",
+      main = paste("Perturbed SSA forecast, 1 component"))
> bf0.pert12 <- perturbation(s, noise, R, Q, 12, L,
+                            level = 0.9, bf0)
> plot(bf0.pert12, include = include, shadecols = "green",
+      main = paste("Perturbed SSA forecast, 12 components"))
> bf0.pert14 <- perturbation(s, noise, R, Q, 14, L,
+                            level = 0.9, bf0)
> plot(bf0.pert14, include = include, shadecols = "green",
+      main = paste("Perturbed SSA forecast, 14 components"))
> start <- 48
> theme <- simpleTheme(col = c("black","red","blue"),
+                      lwd = c(2, 1, 2),
+                      lty = c("solid", "solid", "solid"))
> xyplot(cbind(window(ser0, start = c(1984, 1)),
+              bf0.pert12$mean,
+              bf0.pert14$mean),
+        superpose = TRUE, type = "l", ylab = NULL, xlab = NULL,
+        auto.key = list(text = c("`Total'",
+                                 "forecast, ET1-12",
+                                 "forecast, ET1-14"),
+                        type = c("l", "l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme)
> xyplot(err + err.pert ~ range, type = "l",
+        ylab = NULL, xlab = NULL,
+        auto.key = list(text = c("Bootstrap errors",
+                                 "Perturbation errors"),
+                        type = c("l", "l"),
+                        lines = TRUE, points = FALSE),
+        scales = list(y = list(log = TRUE)))
```

We take the first 108 points and consider 66-term forecasting. Window length is $L = 48$. Figure 3.15 contains the graph of **w**-correlations. One can see that there are several natural candidates for the number of leading components which we can choose. For example, if we choose the number of leading components to be equal to 12, 16, 17, or 19, then the reconstructed signal looks to be almost unmixed with the residual. Let us calculate the mean size of bootstrap and perturbed 90% confidence intervals for the forecasts performed for different numbers of leading components. For construction of the perturbed intervals, we will take σ equal to the standard deviation of the residuals after the reconstruction by the corresponding number of components. Figure 3.16 contains square roots of mean squared confidence ranges as a function of the number of components. One can see that after 12 components the variability of forecasts sharply increases. On the base of this, the choice of 12 components can be recommended. It is interesting that there is only a small difference between bootstrap and perturbed intervals.

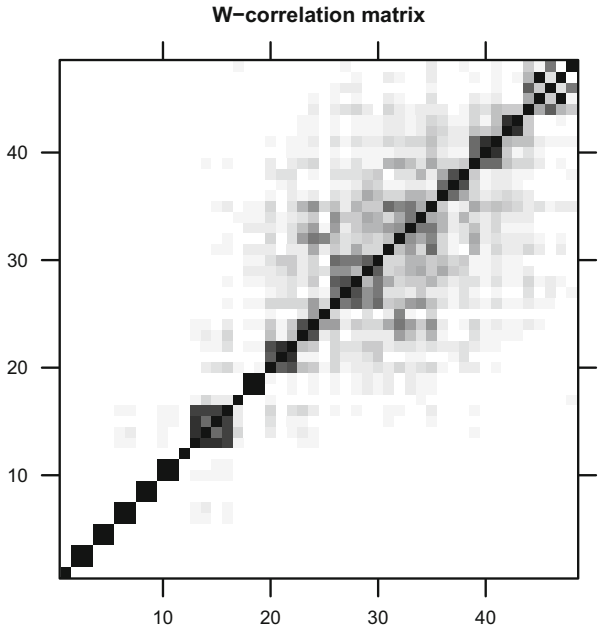

Fig. 3.15 "Total": **w**-Correlations

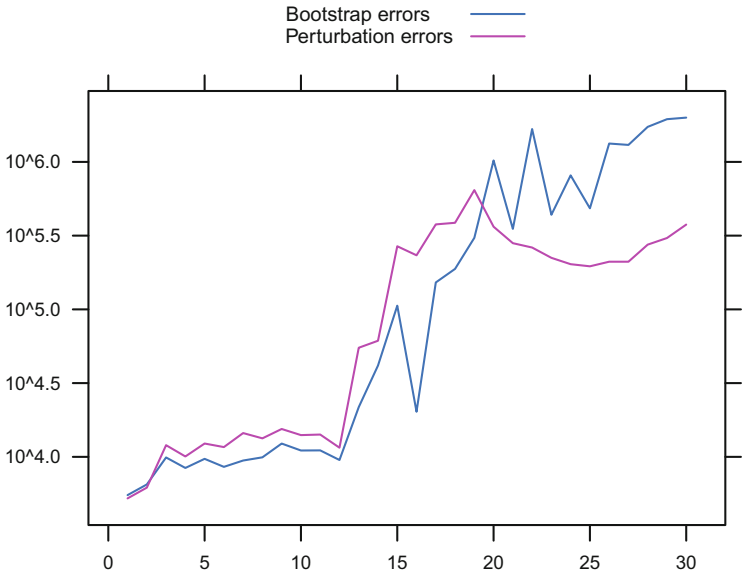

Fig. 3.16 "Total": Sizes of 90% forecasting intervals in dependence on the number of components

Let us demonstrate how the confidence intervals look like for forecasts which use ET1, ET1–12, and ET1–14 (Figs. 3.17, 3.18, and 3.19, respectively). Figure 3.19 shows that the long-term forecasting by 14 components is likely to be wrong.

Let us compare the obtained forecasts with the known series values. The mean forecasts which are calculated by averaging simulated forecast values are depicted. Figure 3.20 shows that, first, the long-term forecast by ET1–12 is more or less adequate, but the forecast by ET1–14 fails. On the other hand, short-term forecasts by ET1–12 and 1–14 are similar. Moreover, the "wrong" forecast (by ET1–14) is slightly more accurate at short horizons.

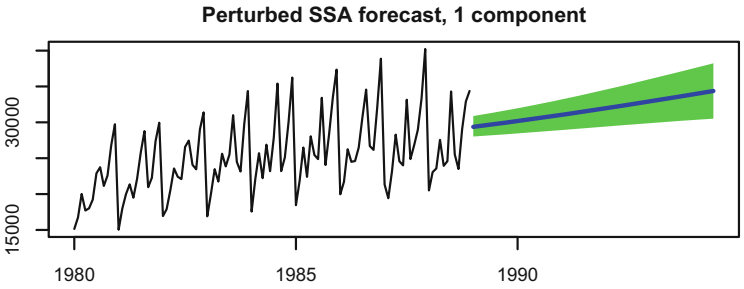

Fig. 3.17 "Total": Perturbed forecasting intervals, ET1

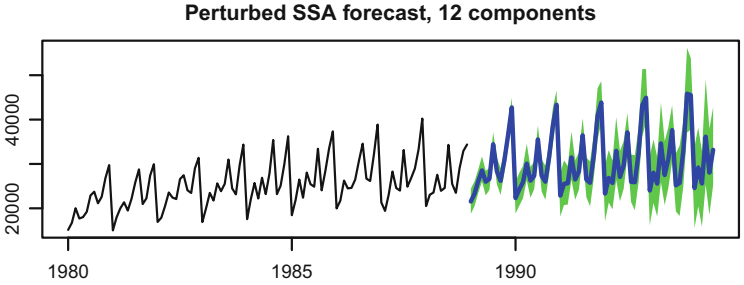

Fig. 3.18 "Total": Perturbed forecasting intervals, ET1–12

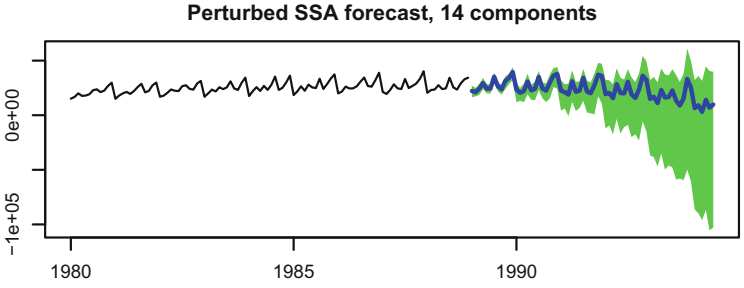

Fig. 3.19 "Total": Perturbed forecasting intervals, ET1–14

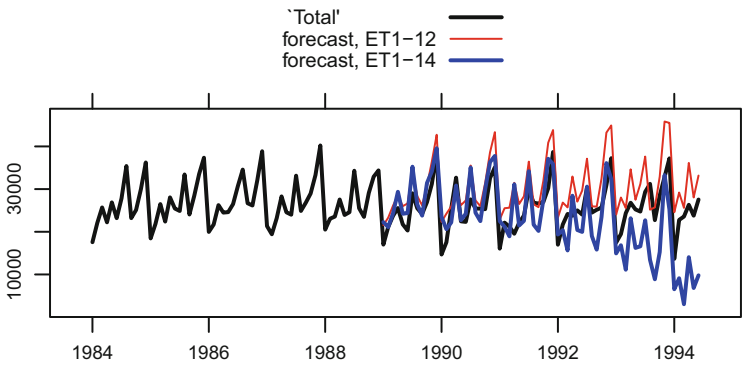

Fig. 3.20 "Total": Comparison of forecasts by ET1–12 and ET1–14

3.5.4 Gap Filling

Let us consider the data "Glonass" provided by the satellite navigation system Glonass (the investigated data `igs<wwww><d>.clk.Z` contains final corrections of time in the format Clock_RINEX obtained in IGS). Data are presented with the step 5 min (300 s) so that any 24-h period consists of 288 points. This data can be used for correcting future time values. However, the data contain gaps. Trends of corrections can be of different form. In Fragment 3.5.6, we demonstrate how Rssa can help for data imputation on a simple example with an almost linear trend. Data consists of 104832 points, taken from 02/01/2014 to 31/12/2014, the GLONASS satellite number 15.

Fragment 3.5.6 ("Glonass": Gap Filling)

```
> data("g15", package = "ssabook")
> xyplot(g15 ~ 1:length(g15), type = "l",
+        ylab = NULL, xlab = NULL)
> range1 <- 14950:15050
> g15_short <- g15[range1]
> g15_un <- na.omit(as.vector(g15))
> g15_un_short <- g15_un[range1]
> p1 <- xyplot(g15_short ~ range1, type = "l",
+              ylab = NULL, xlab = NULL)
> p2 <- xyplot(g15_un_short ~ range1, type = "l",
+              ylab = NULL, xlab = NULL)
> plot(p1, split = c(1, 1, 2, 1), more = TRUE)
> plot(p2, split = c(2, 1, 2, 1), more = FALSE)
> L <- 72
> neig <- min(L, 100)
> s <- ssa(g15, L = 72, neig = neig)
> plot(s, type = "vectors", idx = 1:8, plot.contrib = FALSE)
> g <- gapfill(s, groups = list(1:2))
> xyplot(g[range1] + g15[range1] ~ range1, type = "l",
+        ylab = NULL, xlab = NULL,
```

```
+         par.settings = simpleTheme(col = c("red", "black")))
> spec.pgram(g15_un, detrend = FALSE, log = "no",
+            xlim = c(0.00, 0.02), ylim = c(0, 1e-14),
+            main = "", sub = "")
> axis(1, at = c(1/144, 1/72), labels = c("1/144", "1/72"),
+      las = 2)
> spec.pgram(as.vector(g), detrend = FALSE, log = "no",
+            xlim = c(0.00, 0.02), ylim = c(0, 1e-14),
+            main = "", sub = "")
> axis(1, at = c(1/144, 1/72), labels = c("1/144", "1/72"),
+      las = 2)
```

The whole time series is depicted in Fig. 3.21. It contains several gaps; the total number of missing points is 895.

Let us consider two ways for dealing with missing data: suppressing it (simply by removing time points with missing observations and thus reducing the sample size) or properly treating it as missing and imputing the missing observations. A part of the data in these two cases is depicted in Fig. 3.22.

We start with filling-in the gaps and then show why suppressing the missing data is a wrong strategy, at least for this data set.

Since the time series is very long and the missing data has several compact locations, we will use the subspace-based method of gap filling, which is implemented in the function `gapfill`. To cover by the window all the points of the time series, we

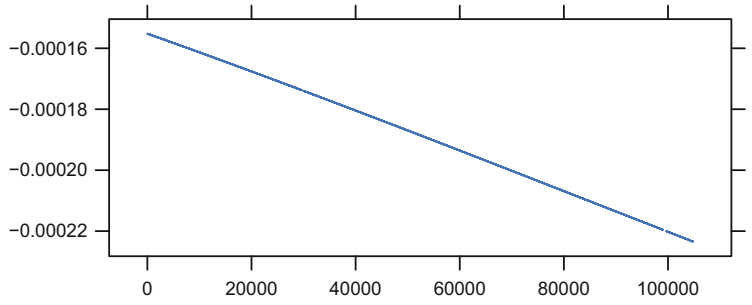

Fig. 3.21 "Glonass": Initial series with gaps

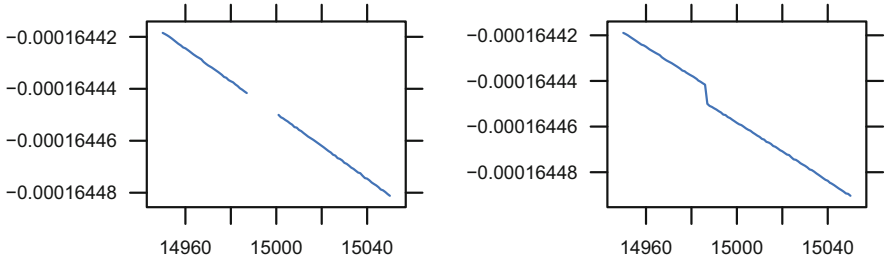

Fig. 3.22 "Glonass": A subseries with a gap (left) and with the suppressed gap (right)

will take a moderate window length L equal to 120. Figure 3.23 shows the leading eight eigenvectors. One can see that the first two eigenvectors correspond to a linear trend. Let us implement the gaps on the base of ET1,2. Figure 3.24 demonstrates imputation for one of the gaps. Certainly, for a trend as simple as this one, many methods can impute the gaps. An advantage of SSA is that the method does not use any trend model and therefore can be applied to trends of other shapes in exactly the same way.

Periodograms of the series with suppressed gaps (Fig. 3.25) and with filling-in gaps (Fig. 3.26) clearly demonstrate that the suppression of gaps corrupts the trend and hides periodicities, while the time series with properly filled gaps is

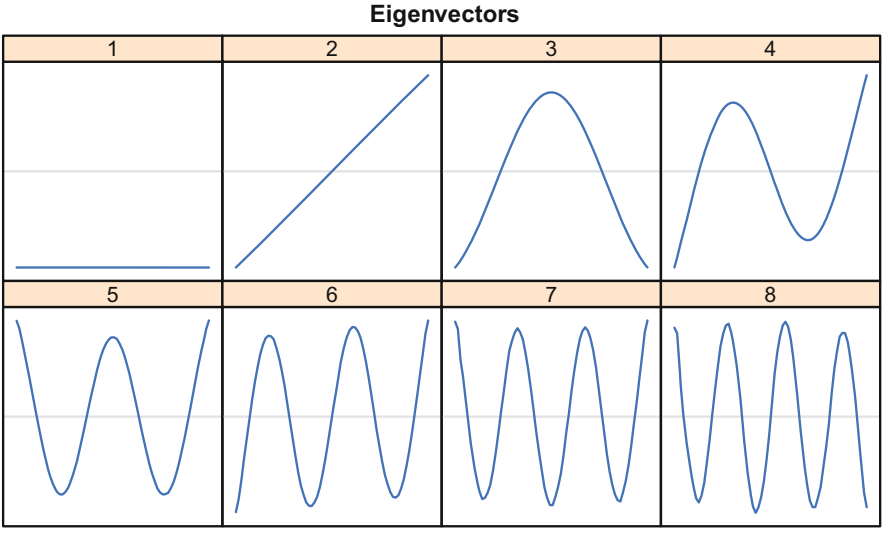

Fig. 3.23 "Glonass": Eigenvectors for the series with gaps, $L = 72$

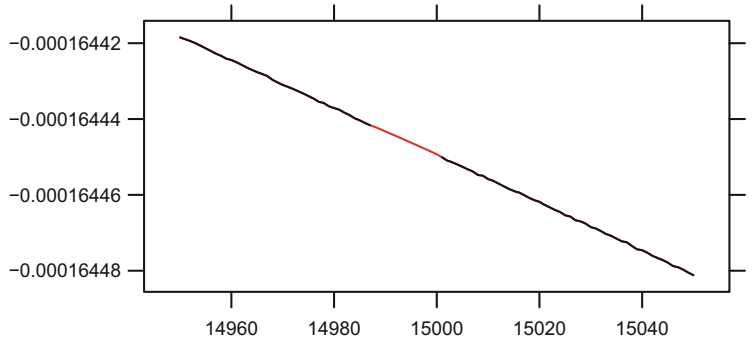

Fig. 3.24 "Glonass": A subseries with the filled gap

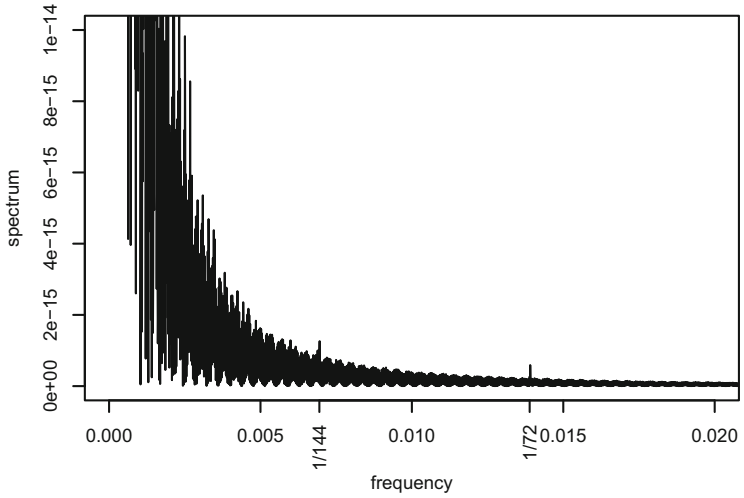

Fig. 3.25 "Glonass": Periodogram of the series with suppressed gaps

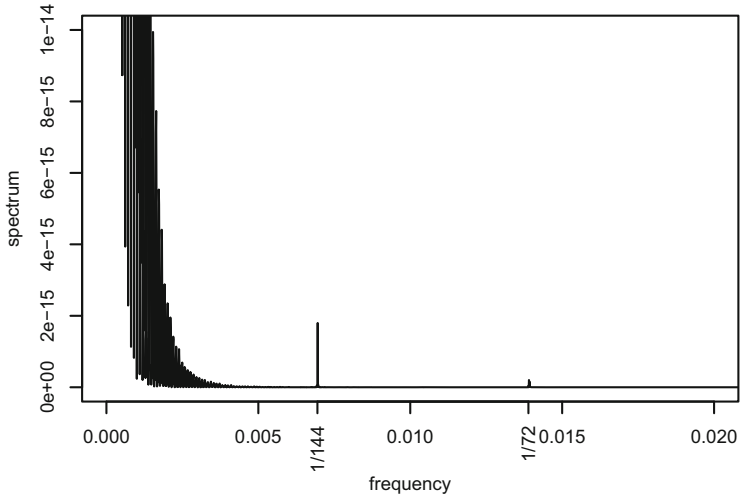

Fig. 3.26 "Glonass": Periodogram of the series with filled gaps

well-structured. Figure 3.26 shows that there are strong peaks at frequencies 1/144 and 1/72, which corresponds to a 12-h periodicity.

Let us extract the periodicity (Fragment 3.5.7) from the obtained time series with no gaps. Since the trend is simple, we take a large window length $L = 52416$ equal to the half of the time series length and divisible by 288. As a guess, we use the period estimates obtained from all pairs of eigenvectors with numbers $(i, i + 1)$.

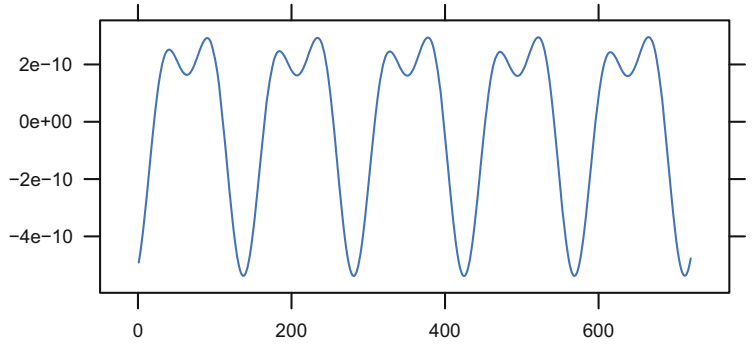

Fig. 3.27 "Glonass": A subseries with the 12-h periodicity; it is extracted from the series with filled gaps and $L = 52416$

The pairs $(32, 33)$ and $(59, 60)$, where the estimated period is smaller than 300, have the expected periods 144 and 72, approximately.

Figure 3.27 demonstrates the extracted 12-h periodicity.

Fragment 3.5.7 ("Glonass": Periodicity Extraction After the Gap Filling)

```
> s_filled = ssa(g, L = 52416, neig = 100)
> pg <- vector(length = 99)
> for (i in 1:99) {
+   pg[i] <- parestimate(s_filled, groups = list(i:(i + 1)),
+                        method = "esprit")$period[1]
+ }
> print(ind <- which(pg < 1/0.003))
[1] 32 58
> print(pg[ind], digits = 0)
[1] 144  72
> r <- reconstruct(s_filled, groups = list(day = c(ind, ind+1)))
> xyplot(r$day[1:720] ~ 1:720, type = "l",
+        ylab = NULL, xlab = NULL)
```

3.5.5 *Parameter Estimation and Low-Rank Approximation*

Low-rank approximation works well if the signal is governed by an LRR exactly. Such kind of signals, e.g., a sum of modulated sinusoids, is a common case in engineering.

To demonstrate the method, we choose a simple series "FORT" from the dataset "AustralianWine," which is very similar to a noisy signal of finite rank. First, let us apply the Cadzow(α) method described in Sect. 3.4 (see Fragment 3.5.8). In Zvonarev and Golyandina (2017), the value $\alpha = 0.2$ is recommended for the

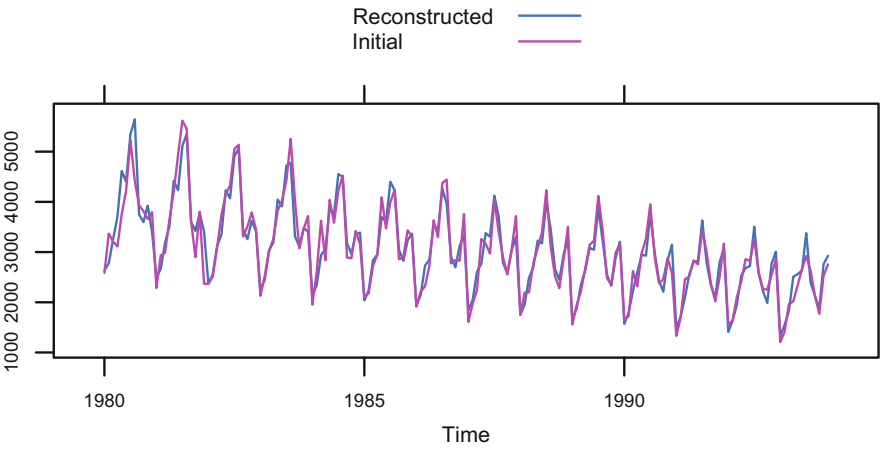

Fig. 3.28 "FORT": Approximation by a series of finite rank

reconstruction of the signal in "FORT." The result of the Cadzow(0.2) iterations is depicted in Fig. 3.28.

Fragment 3.5.8 ("FORT": Cadzow Iterations)

```
> wine <- window(AustralianWine, end = time(AustralianWine)[168])
> ser <- wine[, "Fortified"]
> N <- length(ser)
> L <- 84
> K <- N - L + 1
> rank <- 11
> # Basic SSA
> s0 <- ssa(ser, L = L)
> r0 <- reconstruct(s0, groups = list(signal = 1:rank))$signal
> # Cadzow iterations with series weights close to equal.
> alpha <- 0.1
> weights <- numeric( K)
> weights[1:K] <- alpha
> weights[seq(from = K, to = 1, by = -L)] <- 1
> s <- ssa(ser, L = L, column.oblique = "identity",
+          row.oblique = weights, decompose.force = FALSE)
> c <- cadzow(s, rank = rank)
> sc <- ssa(c, L = rank + 1)
> rc <- reconstruct(sc, groups = list(signal = 1:rank))$signal
> xyplot(cbind(rc, ser), type = "l",
+        superpose = TRUE,
+        auto.key = list(text = c("Reconstructed",
+                                 "Initial"),
+                        type = c("l", "l"),
+                        lines = TRUE, poinwts = FALSE))
```

Let us describe how an explicit form of the estimated signal can be obtained. We consider two cases: (a) estimation of the signal parameters by means of one Cadzow iteration (this iteration coincides with the Basic SSA reconstruction) in Fragment 3.5.9, and (b) estimation of the signal parameters using a finite-rank approximation in Fragment 3.5.10.

To estimate r signal roots of a signal of rank r, it is sufficient to take the window length equal to $r + 1$. Thus, we apply Basic SSA to the limit series of the Cadzow iterations and then call the function `parestimate` for the group consisting of ET1–r. Since we apply Basic SSA to a noisy signal, we should take a large window length L to separate the signal from noise. The results are quite similar for different choices of L, as long as L is large enough. Negative periods are calculated formally as $1/\pm\omega$.

Fragment 3.5.9 ("FORT": Estimation of Parameters by Basic SSA)

```
> # Estimation by means of the first iteration of Cadzow
> # iterations (SSA)
> par <- parestimate(s0, groups = list(1:rank),
+                    method = "esprit")
> print(par)
   period      rate    |     Mod     Arg   |      Re         Im
    5.972   0.004238 |  1.00425    1.05 |  0.49781   0.87218
   -5.972   0.004238 |  1.00425   -1.05 |  0.49781  -0.87218
    2.388   0.001744 |  1.00175    2.63 | -0.87403   0.48945
   -2.388   0.001744 |  1.00175   -2.63 | -0.87403  -0.48945
    4.000   0.000359 |  1.00036    1.57 | -0.00008   1.00036
   -4.000   0.000359 |  1.00036   -1.57 | -0.00008  -1.00036
      Inf  -0.003318 |  0.99669   -0.00 |  0.99669  -0.00000
   12.006  -0.005931 |  0.99409    0.52 |  0.86104   0.49680
  -12.006  -0.005931 |  0.99409   -0.52 |  0.86104  -0.49680
    3.015  -0.011285 |  0.98878    2.08 | -0.48570   0.86127
   -3.015  -0.011285 |  0.98878   -2.08 | -0.48570  -0.86127
> o <- order(abs(par$periods), decreasing = TRUE)
> periods <- (par$periods[o])
> moduli <- par$moduli[o]
> len <- rank
> vars <- matrix(nrow = len, ncol = rank)
> for (i in 1:rank) {
+   if (periods[i] == Inf)
+     vars[, i] <- moduli[i]^(1:len)
+   else if (periods[i] == 2)
+     vars[, i] <- (-moduli[i])^(1:len)
+   else if (periods[i] > 0)
+     vars[, i] <-
+       moduli[i]^(1:len) * sin(2 * pi * (1:len) / periods[i])
+   else
+     vars[, i] <-
+       moduli[i]^(1:len) * cos(2 * pi * (1:len) / periods[i])
+ }
> lm0 <- lm(r0[1:len] ~ 0 + ., data = data.frame(vars))
> coefs0 <- coef(lm0)
> print(round(coefs0[1:6], digits = 2))
```

```
     X1       X2       X3       X4       X5       X6
3969.23 -717.10 -927.57  105.52  137.98 -287.11
> print(round(coefs0[7:11], digits = 2))
     X7       X8       X9      X10      X11
 215.64 -254.51 -205.12   90.44   10.95
```

Fragment 3.5.10 ("FORT": Estimation of Parameters by Cadzow Iterations)

```
> # Estimation by means of the limit of Cadzow iterations
> parc <- parestimate(sc, groups = list(1:rank),
+                     method = "esprit")
> print(parc)
   period     rate   |    Mod     Arg   |     Re         Im
    5.975   0.004986 |  1.00500   1.05 |  0.49863   0.87258
   -5.975   0.004986 |  1.00500  -1.05 |  0.49863  -0.87258
    2.389   0.003402 |  1.00341   2.63 | -0.87506   0.49102
   -2.389   0.003402 |  1.00341  -2.63 | -0.87506  -0.49102
    3.998   0.000311 |  1.00031   1.57 | -0.00076   1.00031
   -3.998   0.000311 |  1.00031  -1.57 | -0.00076  -1.00031
      Inf  -0.003356 |  0.99665   0.00 |  0.99665   0.00000
   12.009  -0.005985 |  0.99403   0.52 |  0.86105   0.49669
  -12.009  -0.005985 |  0.99403  -0.52 |  0.86105  -0.49669
    3.018  -0.010620 |  0.98944   2.08 | -0.48394   0.86301
   -3.018  -0.010620 |  0.98944  -2.08 | -0.48394  -0.86301
> oc <- order(abs(parc$periods), decreasing = TRUE)
> periodsc <- (parc$periods[o])
> modulic <- parc$moduli[o]
> lenc <- rank
> varsc <- matrix(nrow = lenc, ncol = rank)
> for (i in 1:rank) {
+   if (periodsc[i] == Inf)
+     varsc[, i] <- modulic[i]^(1:lenc)
+   else if (periodsc[i] == 2)
+     varsc[, i] <- (-modulic[i])^(1:lenc)
+   else if (periodsc[i] > 0)
+     varsc[, i] <-
+       modulic[i]^(1:lenc) * sin(2 * pi * (1:lenc) / periodsc[i])
+   else
+     varsc[, i] <-
+       modulic[i]^(1:lenc) * cos(2 * pi * (1:lenc) / periodsc[i])
+ }
> lm.c <- lm(rc[1:lenc] ~ 0 + ., data = data.frame(varsc))
> #lm.c
> coefs.c <- coef(lm.c)
> print(round(coefs.c[1:6], digits = 2))
     X1       X2       X3       X4       X5       X6
4005.56 -721.77 -940.64   68.30  184.45 -269.52
> print(round(coefs.c[7:11], digits = 2))
     X7       X8       X9      X10      X11
 325.92 -251.10 -255.41  154.28   61.90
```

By looking at the signal root estimates, we suggest the following model for the signal:

$$\begin{aligned} s_n &= C_0\rho_0^n + \sum_{k=1}^{5} C_k\rho_k^n \sin\left(\frac{2\pi n}{T_k} + \phi_k\right) \\ &= C_0\rho_0^n + \sum_{k=1}^{5} \rho_k^n \left(A_k \sin\left(\frac{2\pi n}{T_k}\right) + B_k \cos\left(\frac{2\pi n}{T_k}\right)\right). \end{aligned} \tag{3.14}$$

Results of `parestimate` are the estimates of ρ_k and T_k in (3.14). To estimate C_k and ϕ_k, one can first estimate A_k and B_k by the least-squares method and then find $C_k = \sqrt{A_k^2 + B_k^2}$, $\phi_k = \arctan(B_k/A_k)$ (Fragment 3.5.11). Note that for Basic SSA we take the whole time series to estimate the coefficients, while in the case of series of finite rank it is sufficient to take any r sequential series points to find r unknown parameters.

Fragment 3.5.11 ("FORT": Estimation of Parametric Real-Valued Form)

```
> idx <- seq(2, 11, 2)
> coefs.c.phase <- numeric(length(idx))
> phases.c <- numeric(length(idx))
> periods.c.phase <- numeric(length(idx))
> moduli.c.phase <- numeric(length(idx))
> for (i in seq_along(idx)) {
+   periods.c.phase[i] <- periodsc[idx[i]]
+   moduli.c.phase[i] <- modulic[idx[i]]
+   coefs.c.phase[i] <- sqrt(coefs.c[idx[i]]^2 +
+                              coefs.c[idx[i] + 1]^2)
+   phases.c[i] <- atan2(coefs.c[idx[i] + 1], coefs.c[idx[i]])
+ }
> print("trend:")
[1] "trend:"
> print("coefficient * modulus^n")
[1] "coefficient * modulus^n"
> print(data.frame(coefficients = coefs.c[1],
+                  moduli = modulic[1]))
   coefficients    moduli
X1     4005.561 0.9966493
> print("periodics:")
[1] "periodics:"
> print("coefficient * modulus^n * cos(2 * pi* n/period + phase)")
[1] "coefficient * modulus^n * cos(2 * pi* n/period + phase)"
> print(data.frame(periods = periods.c.phase, phases = phases.c,
+                  coefficients = coefs.c.phase,
+                  moduli = moduli.c.phase))
    periods    phases coefficients    moduli
1 12.008804 -2.225294    1185.6458 0.9940328
2  5.974637  1.216171     196.6835 1.0049988
3  3.998061  2.261753     422.9253 1.0003111
4  3.018060 -2.347688     358.1681 0.9894363
5  2.388825  0.381569     166.2372 1.0034081
```

3.5.6 Subspace Tracking

Let us consider the problem of finding changes in a time series by the SSA subspace tracking. There are many algorithms for change-point detection in time series, see Basseville et al. (1993), Tartakovsky et al. (2014). We advocate SSA for change-point detection and structure monitoring. The principal technique of using SSA for sequential detection of structural changes was developed in Moskvina and Zhigljavsky (2003); in what follows, we pursue the approach thoroughly described in Golyandina et al. (2001; Chapter 3). We take the annual sunspots data 1700–2015 and try to find changes in the oscillations. Trend needs to be extracted as a preprocessing step. Trend extraction and further analysis are performed in Fragment 3.5.12.

Fragment 3.5.12 ("Sunspots": Subspace Tracking)

```
> data("sunspot2", package = "ssabook")
> s <- ssa(sunspot2, L = 11)
> r <- reconstruct(s, groups = list(Trend = 1))
> plot(r, plot.method = "xyplot", superpose = TRUE)
> sun.oscill <- residuals(r)
> N <- length(sun.oscill)
> rank <- 2
> periods <- function(M, L) {
+   ts(sapply(1:(N - M),
+             function (i) {
+               s <- ssa(sun.oscill[i:(i + M - 1)], L = L)
+               par <- parestimate(s, groups = list(c(1:rank)),
+                                  method = "esprit")
+               abs(par$periods[1])
+             }),
+      start = time(sunspot2)[M + 1], delta = 1)
+ }
> per22 <- periods(22, 11)
> per44 <- periods(44, 22)
> xyplot(cbind(per22, per44), type = "l", xlim = c(1677, 2040),
+        strip = strip.custom(factor.levels =
+                               c("B = 22", "B = 44")))
> M <- 22; L <- M / 2
> hm <- hmatr(sun.oscill, B = M, T = M, L = L, neig = rank)
> plot(hm)
> M <- 44; L <- M / 2
> hm <- hmatr(sun.oscill, B = M, T = M, L = L, neig = rank)
> plot(hm)
```

Figure 3.29 (top) contains the extracted trend and the residual. We choose the window length $L = 11$ as approximately equal to the main period. This choice corresponds to the extraction of a more detailed trend, which can be reconstructed by the leading eigentriple.

Further we consider the residual. Let us check the behavior of the main frequency. First, we consider the sliding subseries $\mathbb{X}_n = (x_{n-B+1}, \ldots, x_n)$, $n = B, \ldots N$, with

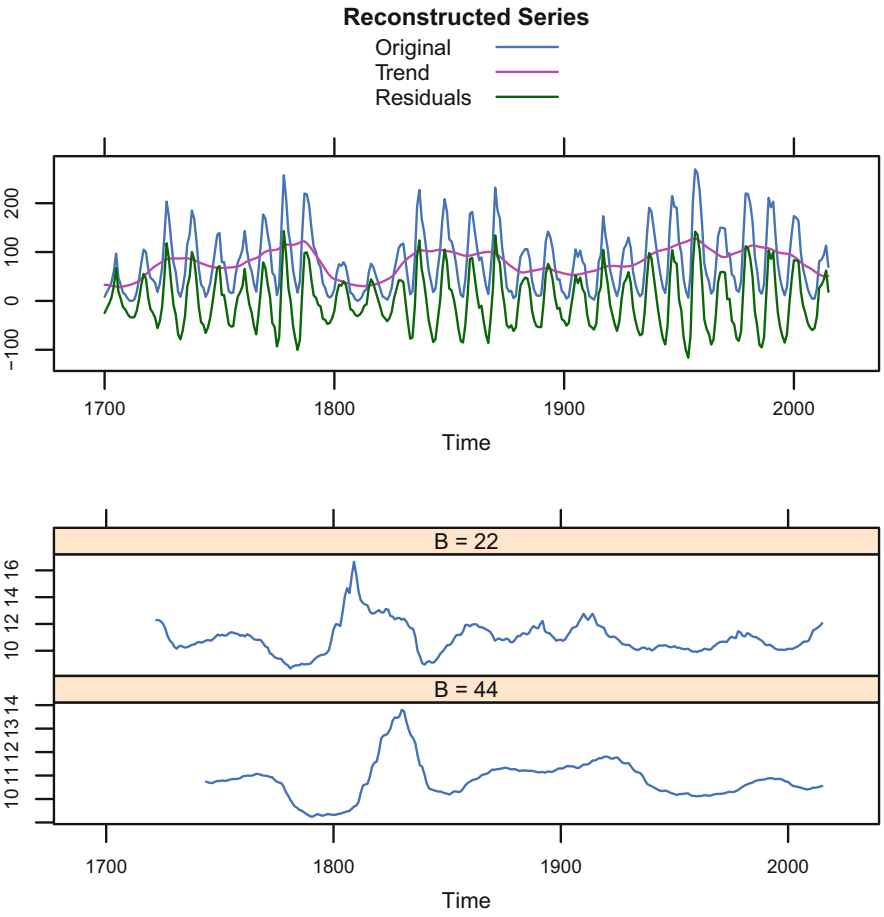

Fig. 3.29 "Sunspots": Trend extraction (top), subspace tracking of residuals with $B = 22$ (middle) and $B = 44$ (bottom)

$B = 22$ and $B = 44$, choosing window length L equal to $B/2$. Then, we apply SSA to each series $\mathbb{X}_n$, find the subspace produced by the first two eigenvectors and estimate the main period P_n by the LS-ESPRIT method. Graphs of periods P_n as functions of n are shown in Fig. 3.29 (middle and bottom). One can see that the period is oscillating around $P = 11$. However, after 1800 we see that the period sharply increases. This corresponds to small values of sun activities (Fig. 3.29, top). If we use the tracking of frequencies for a-priori change detection, then we can clearly see that the delay for $B = 44$ (bottom) is larger than that for $B = 22$ (middle).

Let us apply the techniques suggested in Golyandina et al. (2001; Chapter 3) for a visualization of a-posteriori change-point detection. Visual change-point detection can be performed by means of the so-called heterogeneity matrix. The

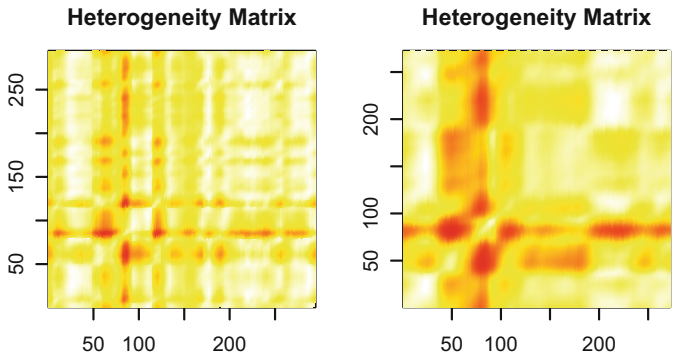

Fig. 3.30 "Sunspots": Heterogeneity matrices $B = 22$ (left) and $B = 44$ (right)

rows correspond to sliding base subseries of length B. We chose the same two values, $B = 22$ and $B = 44$. Each subseries produces a subspace, which is exactly the subspace used for frequency tracking. The columns correspond to sliding test subseries of length T. We take $T = B$. Each test series produces a set of L-lagged vectors. The heterogeneity index is defined as the sum of distances from the L-lagged vectors of the test series to the base subspace, see (3.1) in Golyandina et al. (2001). Large values of the heterogeneity index correspond to large dissimilarity between the test and base subseries. In Fig. 3.30, large values of the heterogeneity index are depicted by red color, while small values of this index are depicted by white and yellow colors. If we consider rows or columns of the heterogeneity matrix, then we see that there is a change-point starting from the subseries $(x_{80}, \ldots, x_{80+B-1})$. This corresponds to the same change at around year 1800, which we have found by the frequency tracking. Note that after the subseries passes the intervals that include the change-point, the values of the heterogeneity index become smaller again. This means that the change in the series was temporal.

Fast algorithms of subspace tracking were developed in many papers (e.g., Real et al. (1999), Badeau et al. (2004), among others), since it can be expected that the subspace estimate for the $(n + 1)$th subseries can be calculated more effectively if one uses the information obtained at the previous nth step. However, our experience shows that despite the fast algorithms implemented in the RSSA and SVD packages cannot be improved on the base of the use of previously constructed subspaces, these implementations of RSSA and SVD can be faster than the improved conventional algorithms.

3.5.7 Automatic Choice of Parameters for Forecasting

Since SSA can be applied without requiring validity of any model for the time series, the choice of parameters should be non-specific. The most conventional

model-free way of parameter choice is the minimization of forecasting errors within the validation (training) period. This approach can be applied if the time series length is large enough.

Fragment 3.5.13 contains the code of functions, which may help in finding the optimal parameters (which are the window length and the number of leading components). The function `forecast.mse` performs forecasting on the base of a given `ssa` object `x` and calculates the mean square of the difference between the forecast and a given time series `F.check`. The function `forecast.sliding.mse` call `forecast.mse` for sliding subseries, given set of window lengths L, and set of numbers of components *ncomp* used for forecasting. The function `forecast.mse` is designed to be applied to one series (subseries), one window length, one number of leading components. In `forecast.sliding.mse`, `forecast.mse` is applied to many (`K.sliding`) subseries of a given series, many window lengths (stored in the vector `L`), many numbers of components (stored in the vector `ncomp`), which is done in an effective manner. In this way, we obtain a 3D array of MSE errors. Note that the number of SSA decompositions is equal to the number of sliding windows `K.sliding` multiplied by the number of window lengths.

Finally, the function `optim.par` on the base of this 3D array calculates the matrix of mean MSE errors, which are obtained by computing the average of the MSE errors corresponding to different sliding subseries, and finds the optimal window length and the number of components, which correspond to the minimal mean MSE. The matrix of mean MSE errors provides a possibility to plot the dependence of accuracy as a function of parameters and hence to check if this accuracy is stable (that is, does not change much) in the chosen range of parameters.

To use the function `optim.par`, the user should choose the length of the validation period. This validation period may correspond to the whole series. In the example considered (see Fragment 3.5.13) the whole series is divided into training and test periods to check the forecast accuracy.

Fragment 3.5.13 (Functions for the Search of Optimal Parameters)

```
> library("plyr")
> forecast.mse <- function(x, F.check,
+                          forecast.len = 1, ...) {
+   stopifnot(length(F.check) == forecast.len)
+   f <- forecast(x, h = forecast.len, ...)
+   mean((f$mean - F.check)^2)
+ }
> forecast.sliding.mse <- function(F,
+                                  L, ncomp,
+                                  forecast.len = 1,
+                                  K.sliding = N %/% 4,
+                                  .progress = "none",
+                                  .parallel = FALSE,
+                                  ...) {
+   N <- length(F)
+   sliding.len <- N - K.sliding - forecast.len + 1
+   L.max = max(L); L.min = min(L); ncomp.max = max(ncomp)
+   stopifnot(sliding.len > L.max)
```

```
+   stopifnot(ncomp.max + 1 < min(L.min, N - L.max + 1))
+   g <- expand.grid(L = L, i = 1:K.sliding)
+   aaply(g, 1,
+         splat(function(L, i) {
+             F.train <- F[seq(from = i, len = sliding.len)]
+             F.check <- F[seq(from = sliding.len + i,
+                              len = forecast.len)]
+             s <- ssa(F.train, L = L)
+             sapply(ncomp,
+                    function(ncomp) {
+                        res <- forecast.mse(s, F.check,
+                                            forecast.len =
+                                              forecast.len,
+                                            groups =
+                                              list(1:ncomp),
+                                            ...)
+                        names(res) <- as.character(ncomp)
+                        res
+                    })
+         }),
+         .progress = .progress, .parallel = .parallel)
+ }
> optim.par <- function(m0) {
+   m <- apply(m0, c(1, 3), mean)
+   mpos <- which(m == min(m), arr.ind = TRUE)
+   L.opt <- Ls[mpos[1]]
+   ncomp.opt <- ncomp[mpos[2]]
+   list(L.opt = L.opt, ncomp.opt = ncomp.opt, m = m)
+ }
```

Fragment 3.5.14 shows how this function is applied to "Bookings" data to obtain optimal parameters. The data contains the numbers of hourly hotel bookings through a particular web-site during 6 months. We can expect the main period of the data to be equal to $168 = 24 \cdot 7$ (`frequency` of the data is equal to 168 observations per week). We forecast the series for 2 weeks. To find the parameters, we minimize the RMSE errors of two forecasts for $336 = 2 \cdot 168$ steps ahead (`K.sliding = 2`).

Fragment 3.5.14 ("Bookings": Search for Optimal Parameters)

```
> data("bookings", package = "ssabook")
> K.sliding <- 2
> forecast.base.len <- 2*frequency(bookings)
> base.len <- length(bookings)
> sliding.len <- base.len - K.sliding - forecast.base.len + 1
> print(sliding.len)
[1] 4007
> ncomp <- 1:100
> L.min <- frequency(bookings)
> Ls <- seq(L.min, 10*L.min, by = frequency(bookings))
> m0 <- forecast.sliding.mse(bookings,
+                            K.sliding = K.sliding,
+                            L = Ls, ncomp = ncomp,
+                            method = "recurrent",
```

```
+                                forecast.len = forecast.base.len,
+                                .progress = "none")
> p <- optim.par(m0)
> print(c(p$L.opt, p$ncomp.opt, sqrt(min(p$m))))
[1] 504.0000  90.0000 116.0134
> matplot(Ls, sqrt(p$m), ylab = "", xlab = "Window lengths",
+         type = "l", col = topo.colors(100))
```

The dependence of RMSE errors on window lengths for different number of components r chosen for decomposition is depicted in Fig. 3.31. Colors are changing from blue to yellow; this corresponds to values of r from 1 to 100. The optimal window length us equal to 504, the optimal number of components is equal to 90.

In Fragment 3.5.15, the forecast is constructed with the parameters found. The forecasts are shown in Fig. 3.32 for the whole series. One can see that the series has some irregularities. The last points are depicted in Fig. 3.33.

Fragment 3.5.15 ("Bookings": Forecast with Optimal Parameters)

```
> forecast.len <- 2*frequency(bookings)
> ssa.obj <- ssa(bookings, L = p$L.opt)
> ssa.for <- rforecast(ssa.obj, groups = list(1:p$ncomp.opt),
+                      len = forecast.len)
> xyplot(cbind(bookings, ssa.for),
+        type = "l", superpose = TRUE)
> xyplot(cbind(bookings, ssa.for), type = "l",
+        superpose = TRUE, xlim = c(21,29))
```

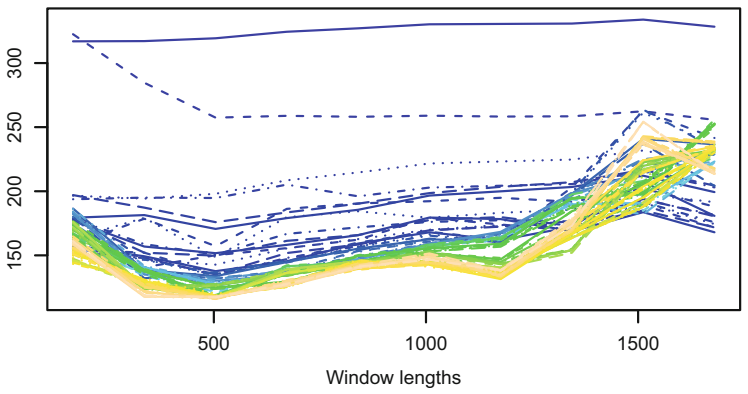

Fig. 3.31 "Bookings": Dependence of RMSE on L for different numbers of components

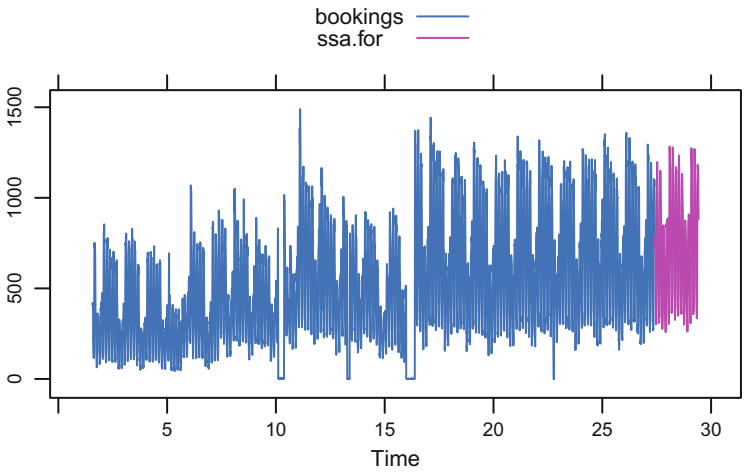

Fig. 3.32 "Bookings": Forecast with optimal parameters

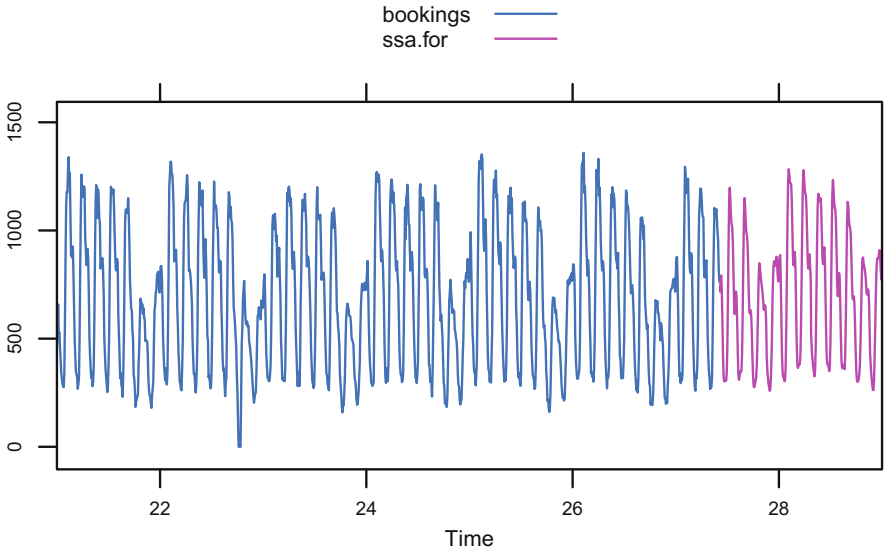

Fig. 3.33 "Bookings": Forecast with optimal parameters for last points

3.5.8 Comparison of SSA, ARIMA, and ETS

As was mentioned in Sect. 1.6.3, real-world time series do not satisfy any model exactly. Therefore, comparing application of ARIMA and SSA with totally different models, we compare approximations by these models.

Seasonal ARIMA and exponential smoothing (ETS) methods correspond to concrete model families. In particular, the frequency of the periodic component (e.g., seasonality) should be specified. Therefore, different information criteria are developed for these models. The idea of these criteria is to use some measure of correspondence between the model and the time series and then adjust it by the number of parameters in the model. In real-world problems, these information criteria can be formally applied for the choice of model from the corresponding family.

For SSA, we will use the approach described in Sect. 3.5.7. Since the minimized forecasting errors are calculated for series points which do not participate in the construction of the forecasts, this procedure partly avoids over-fitting. Minimization of reconstruction errors is senseless, since the minimum can be reached by over-fitting (the larger is the number of the reconstructed components, the smaller is the reconstruction error).

ARIMA and ETS models provide theoretical prediction intervals for the whole series. SSA provides bootstrap confidence intervals for the signal and bootstrap prediction for the whole series in the model "signal + noise," see Sect. 3.2.1.5. For comparability, we will consider prediction intervals for all considered methods. To compare accuracy of the methods, we consider the mean squared difference between the series and mean forecasts for ARIMA and ETS and the mean squared difference between the series and signal forecasts for SSA.

Let us consider the series "Sweetwhite" from the dataset "AustralianWine." This time series contains monthly sales of sweet white wines and has a changing structure. Therefore, it does not suit any model. We divide the series into two parts: training (base) and test ones (Fragment 3.5.16). Models will be constructed by means of the training part, while the methods will be compared by the forecasting of the test part values.

Fragment 3.5.16 ("Sweetwhite": Training and Test Periods)

```
> name <- "Sweetwhite"
> wine <- window(AustralianWine, end = time(AustralianWine)[174])
> series <- wine[, name]
> set.seed(1)
> forecast.len <- 12
> base.len <- length(series) - forecast.len
> F.base <- window(series,
+                  end = time(series)[base.len]) # training
> F.new <- window(series,
+                 start = time(series)[base.len + 1]) # test
```

Fragment 3.5.17 contains the code, which seeks the parameters of SSA, the window length and the number of eigentriples for forecasting. Similar to the example in Sect. 3.5.7, the parameters correspond to minimal MSE forecasting errors on the training part. Since the series is short enough and has a changing

behavior, we minimize the mean MSE of 12 two-step ahead forecasts. The obtained values are $L = 108$, $r = 8$.

Fragment 3.5.17 ("Sweetwhite": Search for SSA Parameters)

```
> K.sliding <- 12
> forecast.base.len <- 2
> ns <- base.len - K.sliding - forecast.base.len + 1
> ncomp <- 1:15
> L.min <- 24
> Ls <- seq(L.min, ns - L.min, by = 12)
> method <- "recurrent"
> m0 <- forecast.sliding.mse(F.base, K.sliding = K.sliding,
+                            L = Ls, ncomp = ncomp,
+                            method = method,
+                            forecast.len = forecast.base.len)
> p <- optim.par(m0)
> print(c(p$L.opt, p$ncomp.opt, sqrt(min(p$m))))
[1] 108.00000   8.00000  24.80634
> # These parameters provides the best forecast
> # L.opt <- 132; ncomp.opt <- 13
```

In Fragment 3.5.18, the forecasts are constructed and the methods are compared by RMSE. Since the parameters of SSA were constructed by forecasting of sliding shortened time series of length 149, we will use the last 149 points of the base series to construct the forecast for comparison. One can see that the forecast accuracy is approximately the same. Since ARIMA and ETS models provides prediction intervals for the whole series, we consider the SSA prediction intervals too. Figures 3.34, 3.35, and 3.36 depict the series, the forecast (in blue color), and the prediction intervals. Note that for ARIMA forecasting the prediction intervals are very large.

Fragment 3.5.18 ("Sweetwhite": Comparison of SSA, ARIMA and ETS)

```
> # SSA forecast
> F.base.short <-
+   window(F.base, start =
+            time(series)[K.sliding + forecast.base.len])
> ssa.obj <- ssa(F.base.short, L = p$L.opt)
> ssa.for <- forecast(ssa.obj,
+                     groups = list(1:p$ncomp.opt),
+                     method = method, h = forecast.len,
+                     interval = "prediction",
+                     level=c(0.8, 0.95))
> err.ssa <- (ssa.for$mean - F.new)^2
> # ARIMA forecast
> sarima.fit <- auto.arima(F.base, trace = FALSE,
+                          lambda = 0, stepwise = FALSE)
> sarima.for <- forecast(sarima.fit, h = forecast.len)
> err.sarima <- (sarima.for$mean - F.new)^2
> # ETS forecast
> ets.fit <- ets(F.base)
> ets.for <- forecast(ets.fit, h = forecast.len)
```

```
> err.ets <- (ets.for$mean - F.new)^2
> # Models
> print(sarima.fit)
Series: F.base
ARIMA(1,1,0)(2,0,0)[12]
Box Cox transformation: lambda= 0
Coefficients:
          ar1    sar1    sar2
      -0.4165  0.4847  0.2123
s.e.   0.0722  0.0765  0.0813
sigma^2 estimated as 0.03897:  log likelihood=30.78
AIC=-53.55   AICc=-53.29   BIC=-41.22
> print(ets.fit)
ETS(M,N,M)
Call:
 ets(y = F.base)
  Smoothing parameters:
    alpha = 0.5571
    gamma = 1e-04
  Initial states:
    l = 123.5798
    s=1.4262 1.2408 1.0236 1.0546 1.1188 1.0852
           0.7795 0.8136 0.8903 0.8834 0.8241 0.8598
  sigma:  0.1732
     AIC     AICc      BIC
2026.600 2029.887 2072.913
> print(c("SSA(L,r)", p$L.opt, p$ncomp.opt))
[1] "SSA(L,r)" "108"      "8"
> # RMSE for test periods
> print(c("ssa", sqrt(mean(err.ssa))))
[1] "ssa"               "55.3572366330604"
> print(c("sarima", sqrt(mean(err.sarima))))
[1] "sarima"            "60.1534785331151"
> print(c("ets", sqrt(mean(err.ets))))
[1] "ets"               "54.2338009066311"
> # Plot of forecasts with confidence intervals
> plot(ets.for); lines(series,col="black");
> plot(sarima.for); lines(series,col="black");
> plot(ssa.for); lines(series,col="black");
```

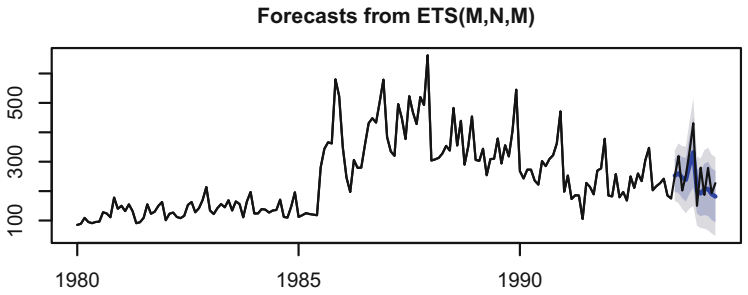

Fig. 3.34 "Sweetwhite": ETS forecast with optimal parameters

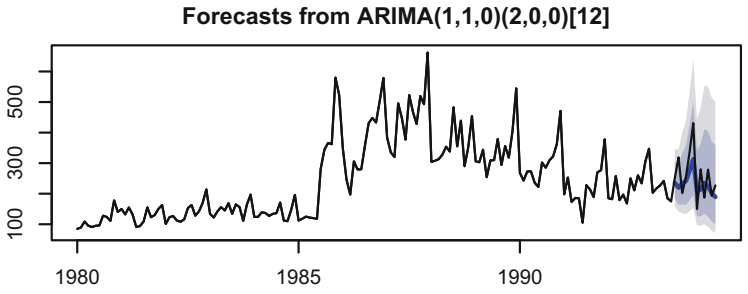

Fig. 3.35 "Sweetwhite": ARIMA forecast with optimal parameters

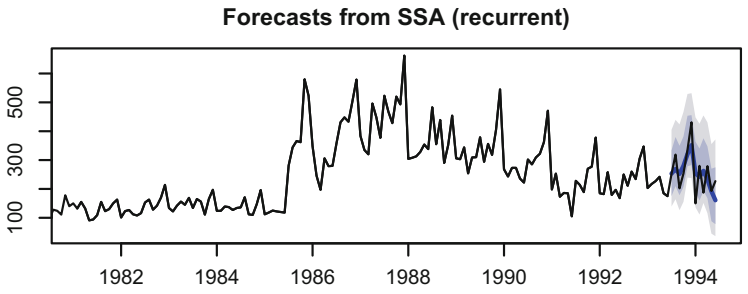

Fig. 3.36 "Sweetwhite": SSA forecast with optimal parameters

References

Badeau R, Richard G, David B (2004) Sliding window adaptive SVD algorithms. IEEE Trans Signal Process 52(1):1–10

Barkhuijsen H, de Beer R, van Ormondt D (1987) Improved algorithm for noniterative time-domain model fitting to exponentially damped magnetic resonance signals. J Magn Reson 73:553–557

Basseville M, Nikiforov IV, et al (1993) Detection of abrupt changes: theory and application, vol 104. Prentice Hall, Englewood Cliffs

Beckers J, Rixen M (2003) EOF calculations and data filling from incomplete oceanographic data sets. Atmos Ocean Technol 20:1839–1856
Cadzow JA (1988) Signal enhancement: a composite property mapping algorithm. IEEE Trans Acoust 36(1):49–62
Du K, Zhao Y, Lei J (2017) The incorrect usage of singular spectral analysis and discrete wavelet transform in hybrid models to predict hydrological time series. J Hydrol 552:44–51
Gillard J, Kvasov D (2016) Lipschitz optimization methods for fitting a sum of damped sinusoids to a series of observations. Stat Interface 10(1):59–70
Gillard J, Zhigljavsky A (2011) Analysis of structured low rank approximation as an optimization problem. Informatica 22(4):489–505
Gillard J, Zhigljavsky A (2013) Optimization challenges in the structured low rank approximation problem. J Global Optim 57(3):733–751
Gillard J, Zhigljavsky AA (2016) Weighted norms in subspace-based methods for time series analysis. Numer Linear Algebra Appl 23(5):947–967
Golyandina N, Osipov E (2007) The "Caterpillar"-SSA method for analysis of time series with missing values. J Stat Plan Inference 137(8):2642–2653
Golyandina N, Zhigljavsky A (2013) Singular spectrum analysis for time series. Springer briefs in statistics. Springer
Golyandina N, Nekrutkin V, Zhigljavsky A (2001) Analysis of time series structure: SSA and related techniques. Chapman&Hall/CRC
Golyandina N, Korobeynikov A, Shlemov A, Usevich K (2015) Multivariate and 2D extensions of singular spectrum analysis with the Rssa package. J Stat Softw 67(2):1–78
Hyndman RJ (2017) FORECAST: Forecasting functions for time series and linear models. URL http://CRAN.R-project.org/package=forecast, R package version 8.1, with contributions from Slava Razbash and Drew Schmidt
Kondrashov D, Ghil M (2006) Spatio-temporal filling of missing points in geophysical data sets. Nonlinear Process Geophys 13(2):151–159
Markovsky I (2012) Low rank approximation. Springer
Markovsky I, Usevich K (2014) Software for weighted structured low-rank approximation. J Comput Appl Math 256:278–292
Markovsky I, Willems JC, Van Huffel S, De Moor B (2006) Exact and approximate modeling of linear systems: A behavioral approach, vol 11. SIAM
Moskvina V, Zhigljavsky A (2003) An algorithm based on singular spectrum analysis for change-point detection. Commun Stat Simul Comput 32(2):319–352
Nekrutkin V (2010) Perturbation expansions of signal subspaces for long signals. Stat Interface 3:297–319
Real E, Tufts D, Cooley JW (1999) Two algorithms for fast approximate subspace tracking. IEEE Trans Signal Process 47(7):1936–1945
Rodrigues PC, de Carvalho M (2013) Spectral modeling of time series with missing data. Appl Math Modell 37(7):4676–4684
Roy R, Kailath T (1989) ESPRIT: estimation of signal parameters via rotational invariance techniques. IEEE Trans Acoust 37:984–995
Schoellhamer D (2001) Singular spectrum analysis for time series with missing data. Geophys Res Lett 28(16):3187–3190
Tartakovsky A, Nikiforov I, Basseville M (2014) Sequential analysis: Hypothesis testing and changepoint detection. CRC Press
Usevich K (2010) On signal and extraneous roots in singular spectrum analysis. Stat Interface 3(3):281–295
Van Huffel S, Chen H, Decanniere C, van Hecke P (1994) Algorithm for time-domain NMR data fitting based on total least squares. J Magn Reson Ser A 110:228–237
Zhigljavsky A, Golyandina N, Gillard J (2016a) Analysis and design in the problem of vector deconvolution. In: Kunert J, Müller HC, Atkinson CA (eds) mODa 11 - advances in model-oriented design and analysis. Springer International Publishing, pp 243–251

Zhigljavsky A, Golyandina N, Gryaznov S (2016b) Deconvolution of a discrete uniform distribution. Stat Probab Lett 118:37–44

Zvonarev N, Golyandina N (2017) Iterative algorithms for weighted and unweighted finite-rank time-series approximations. Stat Interface 10(1):5–18

Chapter 4
SSA for Multivariate Time Series

In this chapter we consider the problem of simultaneous decomposition, reconstruction, and forecasting for a collection of time series from the viewpoint of SSA. The main method of this chapter is usually called either Multichannel SSA or Multivariate SSA, shortly MSSA. The principal idea of the algorithm is the same as for Basic SSA, the difference is in the way of how the trajectory matrix is constructed. The aim of MSSA is to take into consideration the combined structure of a multivariate series to obtain more accurate results.

MSSA is usually considered as an extension of 1D-SSA. However, the algorithm of MSSA was published even earlier than the algorithm of 1D-SSA; see Weare and Nasstrom (1982), where MSSA was named Extended Empirical Orthogonal Function (EEOF) analysis. The MSSA algorithm in the framework of SSA was formally formulated in Broomhead and King (1986b). Here we consider the algorithm of MSSA for the analysis and forecasting of multivariate time series following the approach described in Golyandina et al. (2001; Chapter 2) for one-dimensional series and in Golyandina and Stepanov (2005) for multidimensional ones.

Section 4.1 starts this chapter by describing the complex-valued version of 1D-SSA (called Complex SSA), which is a natural generalization of 1D-SSA for the analysis and forecasting of a system of two time series (Keppenne and Lall 1996).

In Sect. 4.2 we expand the methodology of Chap. 2 for the SSA analysis of a system of several time series. It is important to note that there are two main ways of stacking individual trajectory matrices into a joint trajectory matrix: horizontal stacking and vertical stacking. For the MSSA analysis, these two stacking procedures are equivalent.

Section 4.3 considers forecasting in MSSA. There are four main variants of MSSA forecasting: recurrent column forecasting, recurrent row forecasting, vector column forecasting, and vector row forecasting. We carefully describe the commonalities and differences between these four variants and make their comparison on a simulated data. Finally, in Sect. 4.4 we discuss features of the MSSA analysis,

N. Golyandina et al., *Singular Spectrum Analysis with R*, Use R!,
https://doi.org/10.1007/978-3-662-57380-8_4

forecasting and data filling on several real-world data sets. The examples considered in Sects. 4.3 and 4.4 and the discussion of Sect. 4.3.4 demonstrate that essentially all the techniques that have been developed in Chaps. 2 and 3 for 1D-SSA can be naturally extended to the multivariate case. This concerns, in addition to the SSA analysis and forecasting, the practical problems of smoothing, filtering, imputation of missing values, estimation of parameters of the signal, and also subspace tracking for monitoring stability and change-point detection.

4.1 Complex SSA

Any real-valued 1D-SSA variation can be transferred to the complex-valued case. At present, only Basic 1D-SSA is implemented in the RSSA package in the complex-valued form. Therefore, in this section we briefly discuss the complex-valued version of Basic 1D-SSA and call it Complex SSA. A comparison of Complex SSA with other methods of multivariate SSA will be made in subsequent sections of this chapter.

4.1.1 Method

Assume that a system of two time series of the same length N is given. Then we can consider the one-dimensional complex-valued series $\mathbb{X} = \mathbb{X}^{(1)} + \mathrm{i}\mathbb{X}^{(2)}$ and apply the complex version of 1D-SSA to this one-dimensional series. Since the Basic SSA algorithm in Sect. 2.1 is written in the real-valued form, there is some difference in the form of the SVD performed in the complex-valued space, where the transposition should be Hermitian.

Also, there is a specificity related to the uniqueness of the SVD expansion. If the singular values are different, then the SVD is unique up to multiplication of left and right singular vectors by c, where $|c| = 1$. In the real-valued case, $c = \pm 1$, while in the complex-valued case there are infinitely many complex numbers c with $|c| = 1$.

Complex-valued SSA forecasting and parameter estimation are straightforward extensions of the corresponding techniques for the real-valued time series. In the current version of RSSA, forecasting (recurrent and vector ones), Cadzow iterations and gap-filling are implemented but the parameter estimation and the shaped version of SSA are not.

4.1.2 Separability

Separability in Complex SSA is defined exactly as in the real-valued 1D-SSA variations. However, the conditions for separability are different. Conditions for

Complex SSA separability of time series are more restrictive than the separability conditions for the one-dimensional series (Golyandina et al. 2015; Appendix A.1). In particular, the following sufficient condition for weak separability is valid.

Proposition 4.1 *If time series* $\mathbb{F}^{(1)}$ *and* $\mathbb{F}^{(2)}$, $\mathbb{G}^{(1)}$ *and* $\mathbb{G}^{(2)}$, $\mathbb{F}^{(1)}$ *and* $\mathbb{G}^{(2)}$, *and also* $\mathbb{G}^{(1)}$ *and* $\mathbb{F}^{(2)}$ *are weakly L-separable by 1D-SSA, then the complex-valued time series* $\mathbb{F}^{(1)} + \mathrm{i}\mathbb{F}^{(2)}$ *and* $\mathbb{G}^{(1)} + \mathrm{i}\mathbb{G}^{(2)}$ *are weakly L-separable for Complex SSA.*

The conditions of Proposition 4.1 can be extended to conditions for asymptotic separability ($N \to \infty$) and therefore for approximate separability for fixed N.

The most important difference between 1D-SSA and Complex SSA is related to the separability of imaginary exponential functions with the term $s_n = Ae^{\mathrm{i}(2\pi\omega n+\phi)} = A\cos(2\pi\omega n + \phi) + \mathrm{i}A\sin(2\pi\omega n + \phi)$, $0 < \omega < 0.5$. Such series have the Complex SSA-rank 1, which is smaller than the 1D-SSA-ranks of real and imaginary parts of these functions; these ranks are equal to 2. Recall that the rank of a series is equal to the rank of its trajectory matrix constructed by the chosen method. In particular, this feature implies that Complex SSA provides better separability of imaginary exponential functions, in comparison with other 1D-SSA variations, see Golyandina and Stepanov (2005). Note that Complex SSA for extraction of imaginary exponential functions is used as a step of the "f-xy eigenfiltering" method for noise suppression in stacked 3-D volumes of seismic traces (Trickett 2003).

Another difference is related to the eigenvalues produced by the complex exponentials like $s_n = A\cos(2\pi\omega n + \phi_1) + \mathrm{i}B\sin(2\pi\omega n + \phi_2)$. In contrast to Basic SSA, such time series frequently produce quite different eigenvalues depending on relation between amplitudes and phases (Golyandina et al. 2015; Appendix A.1). This can influence strong separability, either for the better or for the worse.

4.1.3 Algorithm

Complex SSA formally differs from Basic SSA, presented in Sect. 2.1.4, by the use of the Hermitian transpose, which we will denote by "*."

Algorithm 4.1 Complex SSA: decomposition

Input: Complex-valued time series $\mathbb{X}$ of length N and rank d, window length L.
Output: Decomposition of the trajectory matrix on elementary matrices $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where where $d = \operatorname{rank}\mathbf{X}$ and $\mathbf{X}_i = \sqrt{\lambda_i}U_iV_i^*$.
1: Construct the trajectory matrix $\mathbf{X} = \mathcal{T}_{\mathrm{SSA}}(\mathbb{X})$.
2: Compute the SVD $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, $\mathbf{X}_i = \sqrt{\lambda_i}U_iV_i^*$.

Note that despite the difference between Algorithms 2.1 and 4.1 is just in the change of "T" by "*," numerical complex-valued algorithms can be much more complicated and less stable.

The reconstruction algorithm in Complex SSA is standard, see Algorithm 2.2.

Since the algorithms of forecasting for Complex SSA are very similar to that in the real-valued case, we do not formulate them here.

4.1.4 Complex SSA in RSSA

4.1.4.1 Description of Functions

A typical call of the `ssa` function has the form

```
s <- ssa(x, L = (N + 1) %/% 2, kind = "cssa", svd.method = "svd")
```

where `N` is the series length.

Arguments:

`x` is an object to be decomposed. For Complex SSA it is assumed to be a simple vector or vector-like object of complex numbers. Everything else is coerced to vector.

`L` is a window length. By default it is fixed to half of the series length.

`kind` specifies the kind of SSA to apply.

`svd.method` selects the SVD method to use. Unlike for the case of real-valued SSA, only straightforward implementations of the complex SVD at Decomposition step are included into the RSSA package (called `svd` and `eigen`).

Since Complex SSA is an extension of Basic SSA to the complex-valued case, calls of `reconstruct`, `forecast`, and `cadzow` functions are exactly the same as in the real-valued case; see details in Sect. 2.1.5.

4.1.4.2 Typical Code

The following code demonstrates how to extract trends from two series simultaneously. The data "Stocks" includes daily closing prices of major European stock indices, 1991–1998, included into the RSSA package. The first series is related to Germany DAX (Ibis), the second series contains data for Switzerland SMI.

Fragment 4.1.1 ("Stocks": Reconstruction)

```
> library("Rssa")
> s <- ssa(EuStockMarkets[, 1] + 1i*EuStockMarkets[, 2],
+          kind = "cssa", svd.method = "svd")
> r <- reconstruct(s, groups = list(Trend = 1:2))
> plot(r, plot.method = "xyplot", layout = c(2, 3))
> plot(s, type = "vectors", idx = 1:8)
> len = 2
> print(rforecast(s, groups = list(Trend = 1:2), len = len)[1:len])
[1] 6156.061+8492.425i 6169.006+8507.808i
```

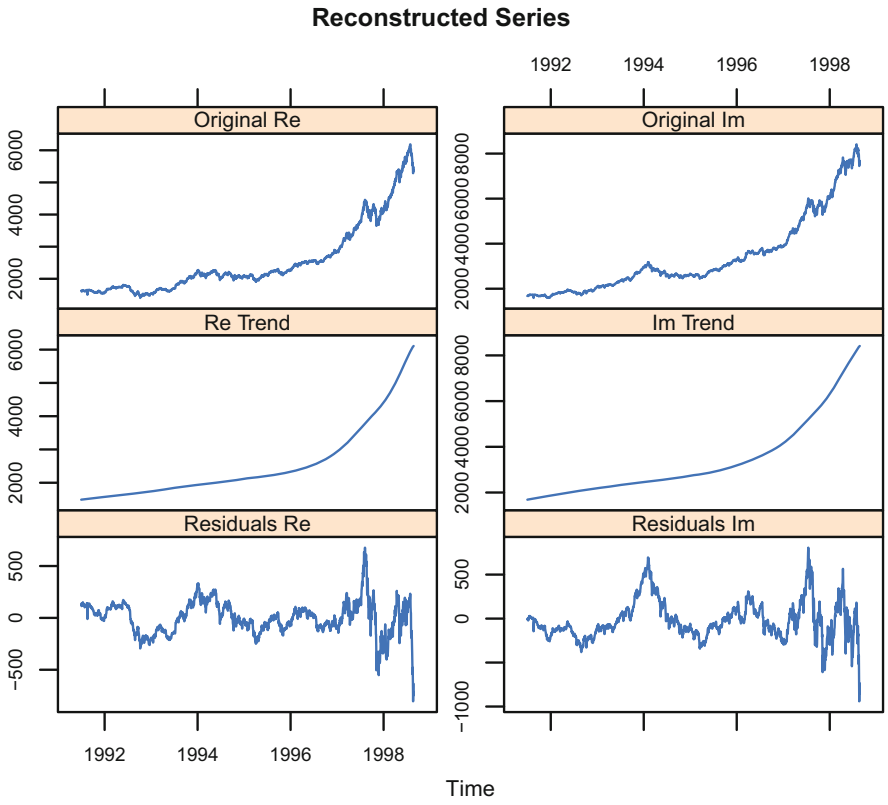

Fig. 4.1 "Stocks": Reconstructed trends

The estimated common trends are depicted in Fig. 4.1.

We choose ET1–2 by analyzing eigenvectors shown in Fig. 4.2. Similarly to the real-valued case, for trend extraction we should specify eigentriples with slowly-varying real and imaginary parts of eigenvectors; they are depicted using different colors. Note that the real part refers to the first series and the imaginary part corresponds to the second series.

However, paired scatterplots can no longer be used for detecting the sine-wave components, since eigenvectors are defined up to multiplication by a unit complex number and therefore the real and imaginary parts of two eigenvectors can differ by an arbitrary phase not equal to $\pi/2$.

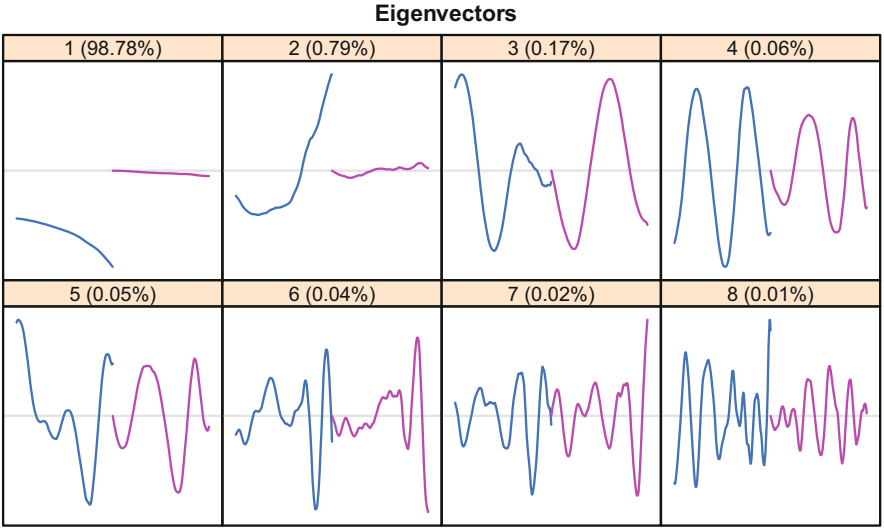

Fig. 4.2 "Stocks": Eigenvectors, real and imaginary parts

4.2 MSSA Analysis

4.2.1 Method

Consider a multivariate time series; that is, a collection $\{\mathbb{X}^{(p)} = \left(x_j^{(p)}\right)_{j=1}^{N_p}, \quad p = 1, \ldots, s\}$ of s time series of length N_p, $p = 1, \ldots, s$.

Denote $\mathbb{X} = (\mathbb{X}^{(1)}, \ldots, \mathbb{X}^{(s)})$ the initial data for the MSSA algorithm. The generic scheme of the algorithm described in Sect. 1.1 holds for MSSA; we only need to define the embedding operator $\mathfrak{T}_{\mathrm{MSSA}}(\mathbb{X}) = \mathbf{X}$.

4.2.1.1 Embedding

Let L be an integer called window length, $1 < L < \min(N_p, p = 1, \ldots, s)$. For each time series $\mathbb{X}^{(p)}$, we form $K_p = N_p - L + 1$ L-lagged vectors $X_j^{(p)} = (x_j^{(p)}, \ldots, x_{j+L-1}^{(p)})^{\mathrm{T}}$, $1 \le j \le K_p$. Denote $K = \sum_{p=1}^{s} K_p$. The *trajectory matrix* of the multidimensional series $\mathbb{X}$ is a matrix of size $L \times K$ and has the form

$$\mathfrak{T}_{\mathrm{MSSA}}(\mathbb{X}) = \mathbf{X} = [X_1^{(1)} : \ldots : X_{K_1}^{(1)} : \ldots : X_1^{(s)} : \ldots : X_{K_s}^{(s)}] = [\mathbf{X}^{(1)} : \ldots : \mathbf{X}^{(s)}], \quad (4.1)$$

where $\mathbf{X}^{(p)} = \mathfrak{T}_{\mathrm{SSA}}(\mathbb{X}^{(p)})$ is the trajectory matrix of the one-dimensional series $\mathbb{X}^{(p)}$ defined by (2.1). Thus, the trajectory matrix of a system of time series has

stacked Hankel structure. Note that

$$\mathfrak{T}^{-1}_{\mathrm{MSSA}}(\mathbf{X}) = [\mathfrak{T}^{-1}_{\mathrm{SSA}}(\mathbf{X}^{(1)}) : \ldots : \mathfrak{T}^{-1}_{\mathrm{SSA}}(\mathbf{X}^{(s)})]. \tag{4.2}$$

Let us make an important comment concerning Embedding step formulated above.

In papers devoted to MSSA, Embedding step can differ. First, in some papers, including papers on climatological applications of MSSA (see, e.g., Broomhead and King (1986); Allen and Robertson (1996); Hannachi et al. (2007)), the trajectory matrix is transposed; that is, the trajectory matrices $\mathbf{X}^{(p)}$ are stacked vertically. We stack them horizontally with the purpose of getting the same structure of the column space in MSSA as in 1D-SSA. Since the time series can have different lengths, one dimension of their trajectory matrices $\mathbf{X}^{(p)}$, $p = 1, \ldots, s$, is the same, while the other dimension can differ. We call the coinciding dimension the window length L and stack the matrices horizontally to obtain the trajectory space (the column subspace of the trajectory matrix, i.e., the space produced by L-lagged vectors of the system of series) of dimension L.

In the horizontally-stacked case, the column trajectory space of a system of identical series coincides with the trajectory space of one time series, while in the vertically-stacked case it is not so. Also, the horizontal stacking is consistent with the continuation of time series, since the increase of series lengths changes the number of lagged vectors and does not change their dimension.

The discussion on similarities and dissimilarities of the horizontally-stacked and vertically-stacked versions of MSSA will be continued at the end of this section (see Remarks 5 and 6) and in Sect. 4.2.4.1.

4.2.1.2 Decomposition

The conventional rank-one matrix decomposition at Decomposition step of MSSA is constructed by applying the SVD to the trajectory matrix; that is, the standard MSSA is an extension of Basic SSA and therefore it can be called Basic MSSA.

Oblique modifications of MSSA are the same as in the 1D case; that is, one can perform nested decompositions by Iterative O-SSA and Filter-adjusted O-SSA. The use of these nested variations is exactly the same as in the 1D case and we refer the reader to Sects. 2.4 and 2.5 for details.

4.2.1.3 Reconstruction

Since $\mathcal{M}^{(\mathrm{H})}_{L,K}$ in MSSA is the set of stacked Hankel matrices, the orthogonal projector $\Pi_{\text{stacked } \mathcal{H}}$ to $\mathcal{M}^{(\mathrm{H})}_{L,K}$ has the form

$$\Pi_{\text{stacked } \mathcal{H}}(\mathbf{Y}) = [\Pi_{\mathcal{H}}(\mathbf{Y}^{(1)}) : \ldots : \Pi_{\mathcal{H}}(\mathbf{Y}^{(s)})], \tag{4.3}$$

where $\Pi_{\mathcal{H}}$ is defined in (2.2). The equality (4.3) follows from the general form of the projection described in Sect. 1.1.2.6. Similar to the 1D case, the reconstructed series are obtained by means of the composition of $\mathfrak{T}^{-1}_{\mathrm{MSSA}}$ and $\Pi_{\text{stacked }\mathcal{H}}$.

4.2.1.4 Comments

Let us make some comments concerning characteristic features of MSSA.

The eigenvectors $\{U_i\}$ in the SVD of the trajectory matrix $\mathbf{X} = \sum_i \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$ form the common basis of the column trajectory spaces of all time series from the system. Factor vectors $\{V_i\}$ (often called extended empirical orthogonal functions (EEOF) in climatology applications, starting from Weare and Nasstrom (1982)) consist of parts related to each time series separately; that is,

$$V_i = \begin{pmatrix} V_i^{(1)} \\ \vdots \\ V_i^{(s)} \end{pmatrix}, \tag{4.4}$$

where the pth factor subvector $V_i^{(p)} \in \mathsf{R}^{K_p}$ belongs to the row trajectory space of the pth series.

The eigenvectors U_i reflect the common features of time series, while the factor subvectors $V_i^{(p)}$ show how these common features appear in each series. It is natural to transform a factor vector to a *factor system* of factor subvectors $V_i^{(p)}$. Then the form of transformed factor vectors will be similar to the initial system of series.

Similarly to the one-dimensional case, the main result of application of MSSA is a decomposition of the multivariate time series into a sum of m multivariate series; the parameters are the window length L and the way of grouping. For the frequently used case of two groups, we denote by $\widetilde{\mathbb{X}}^{(k)} = \left(\tilde{x}_j^{(k)}\right)_{j=1}^{N}$, $k = 1, \ldots, s$, the reconstructed series (usually, the signal) corresponding to the first group of eigentriples I_1.

4.2.1.5 Remarks

1. The indexing of time points $1, \ldots, N_p$ $(p = 1, \ldots, s)$ starting from 1 does not mean that all s series start at the same time; they can also finish at different times. The resultant decomposition obtained by the MSSA algorithm does not depend on the shift between the one-dimensional series and therefore this indexing is only a formality. In particular, MSSA decompositions of two one-dimensional series measured at the same time interval and at disjoint time intervals do not differ.

2. The original time ranges for series $\mathbb{X}^{(p)}$ can be useful for depicting and interpreting them. The reconstructed series have the same time ranges as the original series. Factor subvectors from the factor system can also be synchronized for plotting based on the ranges of the initial series; although factor vectors are shorter than the initial series, their time shifts are the same.
3. For simultaneous analysis of several time series, it is recommended to transfer them into the same scale. Otherwise, the structure of one particular time series will have too much influence on the MSSA results. To balance the time series, they can be either standardized (centered and normalized, in additive models) or only normalized (in multiplicative models). From the other viewpoint, scaling of individual series can be used to influence the importance of a particular series of the system when, for example, this particular series is more important or has a smaller noise component.
4. The MSSA algorithm can be modified in the same ways as the 1D-SSA algorithm. For example, Toeplitz MSSA and MSSA with projection (including centering) can be considered. (However, these options are not implemented in the current version of Rssa.) Nested oblique variations of 1D-SSA (Iterative O-SSA and Filter-adjusted O-SSA) are implemented in Rssa.
5. In climatology, the SVD of the transposed (vertically-stacked) trajectory matrix defined in (4.1) is traditionally considered (Hannachi et al. 2007) as the trajectory matrix. Therefore, the eigenvectors $\{U_i\}$ correspond to normalized extended principal components in Hannachi et al. (2007), while the factor vectors $\{V_i\}$ are called Extended Empirical Orthogonal Functions (EEOFs).
6. The computational cost of the SVD at Decomposition step depends on the size of the matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and hence this computational cost may be significantly different for the horizontally-stacked and vertically-stacked versions of MSSA.

4.2.2 Multi-Dimensional Time Series and LRRs

The model of a system of time series which well suits MSSA is related to times series governed by LRRs. Instead of one LRR of the form (1.8) in the 1D case, see Sects. 1.4 and 2.1.2.2, we have a system of LRRs, which can reflect similarity of time series in the system.

Consider a system of infinite time series $\mathbb{X}^{(1)}, \dots, \mathbb{X}^{(s)}$, choose the window length L, and denote $\mathcal{X}^{(1)}, \dots, \mathcal{X}^{(s)}$ the column trajectory spaces of the series. Let $\mathcal{X} = \mathrm{span}(\mathcal{X}^{(1)}, \dots, \mathcal{X}^{(s)})$ be the column trajectory space of the collection of time series $(\mathbb{X}^{(1)}, \dots, \mathbb{X}^{(s)})$. Similarly to the 1D case, we call the dimension of the trajectory space $\mathcal{X}$ (equal to the rank of the trajectory matrix $\mathbf{X}$ of the series collection) the MSSA-rank of the series collection, see Sect. 1.1.2 for short description of general notions.

Denote the 1D-SSA-ranks of $\mathbb{X}^{(l)}$ by $r_l = \dim \mathcal{X}^{(l)} \le L$, $l = 1, \ldots, s$. For each time series $\mathbb{X}^{(l)}$, we can write the minimal LRR governing the series:

$$x^{(l)}_{j+r_l} = \sum_{k=1}^{r_l} a^{(l)}_k x^{(l)}_{j+r_l-k}, \quad \text{where} \quad a^{(l)}_{r_l} \neq 0, \quad l = 1, \ldots, s. \tag{4.5}$$

The characteristic polynomials of the LRRs (4.5) are

$$P^{(l)}_{r_l}(\mu) = \mu^{r_l} - \sum_{k=1}^{r_l} a^{(l)}_k \mu^{r_l-k}, \quad l = 1, \ldots, s. \tag{4.6}$$

Recall that the roots of the characteristic polynomials of the minimal LRRs governing the series are called signal roots.

Let

$p^{(l)}$ be the number of different roots of the polynomial $P^{(l)}_{r_l}(\mu)$,
$\mu^{(l)}_m$ be the m-th root of the polynomial $P^{(l)}_{r_l}(\mu)$,
$k^{(l)}_m$ be the multiplicity of the root $\mu^{(l)}_m$.

Then

$$k^{(l)}_1 + \ldots + k^{(l)}_{p^{(l)}} = r_l, \quad l = 1, \ldots, s.$$

The characteristic roots determine the series behavior. For example, if $k^{(l)}_m = 1$, then

$$x^{(l)}_n = \sum_{m=1}^{r_l} C^{(l)}_m \left(\mu^{(l)}_m\right)^n.$$

Also let

$\mu_1, \ldots, \mu_p$ be the pooled set of roots of polynomials $P^{(1)}_{r_1}, \ldots, P^{(s)}_{r_s}$,
$k_1, \ldots, k_p$ be the multiplicities of the roots $\mu_1, \ldots, \mu_p$,

where multiplicity of a root in the pooled set is equal to the maximum of multiplicities of the corresponding roots in the initial sets.

Since the roots are determined by the structure of the trajectory space, the following proposition holds, see Golyandina et al. (2015; Appendix A.2).

Proposition 4.2 *Let $r = \sum_{i=1}^{p} k_i < L$. Then the rank of the infinite multi-dimensional time series $(\mathbb{X}^{(1)}, \mathbb{X}^{(2)}, \ldots, \mathbb{X}^{(s)})$ is equal to r.*

Consider a simple example.

Example 4.1 Let $\mathbb{F}^{(1)} = (f_1^{(1)}, \ldots, f_N^{(1)})$ and $\mathbb{F}^{(2)} = (f_1^{(2)}, \ldots, f_N^{(2)})$ with

$$f_k^{(1)} = A\cos(2\pi\omega_1 k + \varphi_1), \quad f_k^{(2)} = B\cos(2\pi\omega_2 k + \varphi_2), \tag{4.7}$$

where $0 < \omega < 1/2$, $0 \le \varphi_1, \varphi_2 < 2\pi$ and $A, B \ne 0$. Let us fix the window length $L > 4$ and find the 1D-SSA-rank of the time series $\mathbb{F}^{(1)}$, the MSSA-rank of the system $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ and the Complex SSA-rank of $\mathbb{F}^{(1)} + \mathrm{i}\mathbb{F}^{(2)}$:

1. For $\omega_1 = \omega_2$, the 1D-SSA ranks of $\mathbb{F}^{(1)}$ and $\mathbb{F}^{(2)}$, as well as the MSSA-rank of $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$, are equal to 2. The Complex SSA-rank of $\mathbb{F}^{(1)} + \mathrm{i}\mathbb{F}^{(2)}$ is equal to 1 if $A = B$ and $|\varphi_1 - \varphi_2| = \pi/2 \bmod \pi$ and is equal to 2 otherwise.
2. For $\omega_1 \ne \omega_2$ the 1D-SSA-rank of $\mathbb{F}^{(1)}$ and $\mathbb{F}^{(2)}$ is equal to 2. The MSSA rank of $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ and the Complex SSA-rank of $\mathbb{F}^{(1)} + \mathrm{i}\mathbb{F}^{(2)}$ are both equal to 4.

4.2.2.1 Matching of Series

Simultaneous analysis of several time series is usually performed to identify their inter-relation and to extract their common structure. Recall that for 1D-SSA, a time series has a structure if and only if the trajectory matrix of this series is rank-deficient. Certainly, for a typical real-world series, the trajectory matrix has full rank. Therefore, in what follows we talk about the rank of signal (a part of times series with structure) or its components.

Consider a system of signals $\mathbb{H} = (\mathbb{H}^{(1)}, \mathbb{H}^{(2)})$ with a rank-deficient trajectory matrix. The structure of a series is reflected in its trajectory space. We can say that two time series have the same structure if their trajectory spaces coincide. For example, for two sine waves with equal periods their trajectory spaces coincide, whatever the values of their amplitudes and phases. This follows from the fact that the trajectory space is the span of subseries of length L of the initial series. On the other hand, sine waves with different frequencies have entirely different structure and the combined trajectory space of their system is a direct sum of the series trajectory spaces.

If two time series are fully matched, then the trajectory space of one time series can be used for reconstructing or forecasting of the second series. If two series are unrelated and have totally different structure, then neither series contains any useful information about the other series for the MSSA analysis.

For MSSA, any shift between two time series and any difference between phases of two matched sine waves have no influence on the result of analysis. Therefore, one cannot say anything about the direction of causality. Moreover, asymmetry of influence of one time series to the another series can be caused by different levels of noise. However, the time series $\mathbb{X}^{(2)}$ can be called *supportive* for the time series $\mathbb{X}^{(1)} = \mathbb{H}^{(1)} + \mathbb{R}^{(1)}$ if the accuracy of either reconstruction or forecasting of $\mathbb{H}^{(1)}$ improves if we analyze the system of two series $\mathbb{X} = (\mathbb{X}^{(1)}, \mathbb{X}^{(2)})$ rather than the series $\mathbb{X}^{(1)}$ alone.

Numerical experiments confirm that for two matching signals the series with any level of noise, which is not larger than the other one, is always supportive (see, e.g., Sect. 4.3.3.3).

4.2.3 Separability

The notion of separability for multidimensional time series is similar to that for one-dimensional series; the latter was briefly considered in Sect. 1.1.2 and thoroughly described in Golyandina et al. (2001; Sections 1.5 and 6.1).

Separability is the key notion in the SSA theory. Indeed, separability of $\mathbb{H}$ from $\mathbb{R}$ means the ability of SSA to extract $\mathbb{H}$ from the sum $\mathbb{H}+\mathbb{R}$. Recall that there is a weak separability, which means orthogonality of the trajectory spaces, and a strong separability which is equivalent to empty intersection of the sets of singular values produced by the series which we are trying to separate.

Conditions of separability of multidimensional time series are more restrictive than that for one-dimensional series. The following sufficient condition of weak separability is valid (exactly the same as for Complex SSA, see Sect. 4.1.2), see (Golyandina et al. 2015; Appendix A.1).

Proposition 4.3 *If time series $\mathbb{F}^{(1)}$ and $\mathbb{F}^{(2)}$, $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$, $\mathbb{F}^{(1)}$ and $\mathbb{G}^{(2)}$, and also $\mathbb{G}^{(1)}$ and $\mathbb{F}^{(2)}$ are weakly L-separable by 1D-SSA, then the two-dimensional time series $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ and $(\mathbb{G}^{(1)}, \mathbb{G}^{(2)})$ are weakly L-separable by MSSA.*

Proposition 4.3 can be extended to cover the case of asymptotic separability (as series lengths $N_i \to \infty$) and therefore approximate separability for fixed large N_i.

Example 4.2 Consider an example of four harmonic real-valued time series $\mathbb{F}^{(1)}$, $\mathbb{F}^{(2)}$, $\mathbb{G}^{(1)}$, and $\mathbb{G}^{(2)}$ of length N:

$$f_k^{(1)} = A_1 \cos(2\pi\omega_1 k + \varphi_1), \qquad f_k^{(2)} = B_1 \cos(2\pi\omega_1 k + \varphi_2),$$

$$g_k^{(1)} = A_2 \cos(2\pi k\omega_2 k + \phi_1), \qquad g_k^{(2)} = B_2 \cos(2\pi k\omega_2 k + \phi_2),$$

$\omega_1 \neq \omega_2$, $k = 0, \dots, N-1$, $A_1, A_2, B_1, B_2 \neq 0$. If $L\omega_i$ and $K\omega_i$ $(i = 1, 2)$ are integers, then $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ and $(\mathbb{G}^{(1)}, \mathbb{G}^{(2)})$ are L-separable by MSSA.

Note that if either $L\omega_i$ or $K\omega_i$ (or both) is not integer, then the two series are not L-separable; however, asymptotic (and approximate for finite lengths) separability takes place.

Weak separability is not enough for extraction of time series components. Therefore, let us look at strong separability related to eigenvalues produced by time series components. It appears that the same pair of time series $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ can produce different eigenvalues in 1D-SSA, MSSA, and Complex SSA. Therefore, by applying a more suitable multivariate extension of 1D-SSA we can improve strong separability.

Example 4.3 Let

$$f_k^{(1)} = A\cos(2\pi\omega k + \varphi_1), \qquad f_k^{(2)} = B\cos(2\pi k\omega k + \varphi_2).$$

If $L\omega$ and $K\omega$ are integers, then $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ produces two equal eigenvalues in MSSA: $\lambda_1 = \lambda_2 = (A^2+B^2)LK/4$. This implies that there is no strong separability in the respective version of MSSA. Note, however, that there is strong separability in this example for Complex SSA, see Golyandina et al. (2015).

4.2.4 Comments on 1D-SSA, MSSA and Complex SSA

4.2.4.1 Covariance Structure

Consider in more detail the case of two time series $\mathbb{X} = (\mathbb{F}, \mathbb{G})$ and let $\mathbf{F}$ and $\mathbf{G}$ be the trajectory matrices of $\mathbb{F}$ and $\mathbb{G}$ correspondingly. Then, since in MSSA we stack the individual trajectory matrices horizontally, the trajectory matrix of $\mathbb{X}$ is $\mathbf{X} = [\mathbf{F} : \mathbf{G}]$. In accordance with (2.3), the SVD of $\mathbf{X} = \mathbf{X}^{(\mathrm{H})}$ is $\mathbf{X} = \sum_i \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$, where λ_i and U_i are eigenvalues and eigenvectors of the matrix $\mathbf{S} = \mathbf{S}^{(\mathrm{H})}_{\mathrm{MSSA}} = \mathbf{X}\mathbf{X}^{\mathrm{T}} = \mathbf{F}\mathbf{F}^{\mathrm{T}} + \mathbf{G}\mathbf{G}^{\mathrm{T}}$ and $V_i = \mathbf{X}^{\mathrm{T}} U_i/\sqrt{\lambda_i}$.

Consider now the vertical stacking of the trajectory matrices of $\mathbb{F}$ and $\mathbb{G}$ in the trajectory matrix: $\mathbf{X}^{(\mathrm{V})} = \begin{pmatrix} \mathbf{F}^{\mathrm{T}} \\ \mathbf{G}^{\mathrm{T}} \end{pmatrix} = \left(\mathbf{X}^{(\mathrm{H})}\right)^{\mathrm{T}}$.

The SVD of $\mathbf{X}^{(\mathrm{V})}$ is then $\mathbf{X}^{(\mathrm{V})} = \sum_i \sqrt{\lambda_i} V_i U_i^{\mathrm{T}}$, the transposed SVD of $\mathbf{X}$. Here λ_i, V_i, and U_i are exactly the same as above but now they have different interpretation: in particular, V_i (they are often called EEOFs, see Remark 5 in Sect. 4.2.1) are the eigenvectors of

$$\mathbf{S}^{(\mathrm{V})}_{\mathrm{MSSA}} = \mathbf{X}^{(\mathrm{V})}\left(\mathbf{X}^{(\mathrm{V})}\right)^{\mathrm{T}} = \begin{pmatrix} \mathbf{F}^{\mathrm{T}}\mathbf{F} & \mathbf{F}^{\mathrm{T}}\mathbf{G} \\ \mathbf{G}^{\mathrm{T}}\mathbf{F} & \mathbf{G}^{\mathrm{T}}\mathbf{G} \end{pmatrix}.$$

The last formula clearly demonstrates the relation between the two versions (horizontal stacking and vertical stacking) of MSSA and shows that MSSA takes into consideration cross-covariances of time series (more precisely, we obtain the cross-covariances if centering of the one-dimensional series is done at the preprocessing stage).

Consider now Complex SSA. Since the eigendecomposition of a complex-valued matrix $\mathbf{A} + \mathrm{i}\mathbf{B}$ can be reduced to the eigendecomposition of the real-valued matrix

$$\mathbf{D} = \begin{pmatrix} \mathbf{A} & -\mathbf{B} \\ \mathbf{B} & \mathbf{A} \end{pmatrix},$$

in the case of Complex SSA we in fact analyze eigenvectors of the matrix

$$\mathbf{S}_{\mathrm{CSSA}} = \begin{pmatrix} \mathbf{F}^{\mathrm{T}}\mathbf{F} & \mathbf{F}^{\mathrm{T}}\mathbf{G} \\ \mathbf{G}^{\mathrm{T}}\mathbf{F} & \mathbf{G}^{\mathrm{T}}\mathbf{G} \end{pmatrix} + \begin{pmatrix} \mathbf{G}^{\mathrm{T}}\mathbf{G} & -\mathbf{G}^{\mathrm{T}}\mathbf{F} \\ -\mathbf{F}^{\mathrm{T}}\mathbf{G} & \mathbf{F}^{\mathrm{T}}\mathbf{F} \end{pmatrix}.$$

We can observe that the structures of the two one-dimensional series in Complex SSA are mixed more than in MSSA.

4.2.4.2 Separability

Conditions of separability for multidimensional time series are more restrictive than that for one-dimensional series. In particular, a sufficient condition for separability of a two-series system is the separability of each series from the first collection with each series from the second one. However, for matched signals, their (weak) 1D-SSA-separability from noise can be considerably improved by their simultaneous MSSA analysis.

Since weak separability is not enough for extraction of time series components, we should pay attention to strong separability related to eigenvalues produced by the time series components. It appears (see Example 4.3) that the two-dimensional time series $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ typically produce different eigenvalues in 1D-SSA, MSSA, and Complex SSA. Therefore, an application of more suitable multidimensional version of SSA can improve strong separability. However, non-matching of one-dimensional time series in a system of series increases the number of eigenvalues related to the signal and hence increases the chance of mixing the signal with the residual. To overcome this effect, the modification DerivSSA (see Sect. 2.5 for the 1D case), which is able to considerably improve strong separability, is implemented in Rssa for the MSSA analysis of a collection of time series.

4.2.4.3 Ranks

Rank of a signal is a very important notion in SSA, since it reflects the complexity of signals and hence the difficulty of the problem of their extraction. For MSSA and Complex SSA, the notions of the time series of finite rank and of time series satisfying LRRs are similar to the related notions in 1D-SSA, although the rank of the same time series may be different and depend on the method used. Let us compare 1D-SSA-, MSSA-, and Complex SSA-ranks, i.e., ranks of the corresponding trajectory matrices. By 1D-SSA-rank for a collection of time series we mean the rank of each one-dimensional series separately.

If two time series have the same structure (and therefore the same 1D-SSA-ranks), then the MSSA-rank is equal to the 1D-SSA-rank of each of the two series. The Complex SSA-rank can be even smaller than the individual 1D-SSA-ranks in the specific case of imaginary exponentials.

Consider a collection $\mathbb{H}^{(k)} = \left(h_j^{(k)}\right)_{j=1}^{N}$, $k = 1, \ldots, s$, of s signals of length N. Let r_k denote the 1D-SSA rank of $\mathbb{H}^{(k)}$ (i.e., the dimension of the trajectory space generated by one-dimensional SSA applied to this time series) and r denote the MSSA rank of $(\mathbb{H}^{(1)}, \ldots, \mathbb{H}^{(s)})$. The relation between r and r_k, $k = 1, \ldots, s$, is considered in Sect. 4.2.2. In particular, it is shown that $r_{\min} \le r \le r_{\max}$, where $r_{\min} = \max\{r_k, k = 1, \ldots, s\}$ and $r_{\max} = \sum_{k=1}^{s} r_k$. The case $r = r_{\max}$ is the least favorable for MSSA and means that different time series do not have matched components. The case $r < r_{\max}$ indicates the presence of matched components and hence simultaneous processing of the time series system can be more effective than their individual analysis.

In terms of matching, if all s series have the same characteristic roots (see Sect. 4.2.2 for the definition), then the time series $\mathbb{H}^{(m)}$, $m = 1, \ldots, s$, consist of additive components of the same 1D-SSA-structure. Such time series are fully matched. For fully matched time series, the MSSA-rank is much smaller than the sum of the 1D-SSA-ranks of the separate time series from the system. On the other hand, if the sets of characteristic roots do not intersect, then the time series have no common structure. In this case, the MSSA-rank is equal to the sum of the 1D-SSA-ranks of the separate time series from the system. For a typical system of real-world time series, we are in-between these two extreme cases.

4.2.4.4 Choice of the Window Length

The choice of the window length for one-dimensional SSA was reviewed in Sect. 2.1.3.2; see Golyandina et al. (2001; Section 1.6) and Golyandina (2010) for more thorough discussions. The problem of the choice of the window length in MSSA is more complicated than that in 1D-SSA. Until now there is no in-depth study of the problem of the choice of the optimal window length for analysis and, to an even greater extent, for forecasting of multidimensional time series. Moreover, the choice of the best window length for MSSA forecasting differs for different types of forecasting methods, see numerical comparison in Sect. 4.4. Some numerical investigation of this problem has been performed in Golyandina and Stepanov (2005), Golyandina et al. (2015); it is extended in Sect. 4.4.

By analogy with the one-dimensional case, we can formulate some key principles for the choice of L. The main principle is the same as for 1D-SSA and states that the choice of L should provide (approximate) separability of series. However, the MSSA case has additional features. Different approaches to the choice of the window length can be partly explained as follows. In 1D-SSA, it makes sense to constrain the window length to the interval $2 \le L \le [(N+1)/2]$, since the SVD expansions for window lengths L and $N - L + 1$ coincide. For the MSSA-analysis of more than one time series, the expansions for all possible window lengths $2 \le L \le \min_i N_i - 1$ are generally different. In particular, while in the 1D-SSA analysis it makes no sense to take $L > (N+1)/2$, in the MSSA analysis it makes

perfect sense choosing large L (and hence small $K_i = N_i - L + 1$) for trend extraction and smoothing.

Since L in 1D-SSA does not exceed half of the time series length, the divisibility of $L = \min(L, K)$ on possible periods of oscillations is recommended in 1D-SSA. In MSSA, $\min(L, K_i)$ is not necessarily equal to L and therefore one also has to pay attention to the values of K_i.

In 1D-SSA, the most detailed decomposition can be obtained if the trajectory matrix $\mathbf{X}$ has maximal rank. In the general case of SSA-family methods, this corresponds to the case of a square trajectory matrix. Thus, for a system of s time series of length N the window length in MSSA providing the square trajectory matrix $\mathbf{X}$ is approximately $sN/(s+1)$. For the case of two time series this corresponds to $2L/3$ for MSSA, while for Complex SSA applied to one complex-valued series this gives $N/2$.

Numerical investigations show that the formula $L \simeq sN/(s+1)$ is appropriate for the decomposition of a small number of time series (see simulation results in Sect. 4.4), but does not look suitable for the system of many short series (the values of K_i become too small for achieving separability). Generally, the choice $L \simeq N/2$ is still appropriate for MSSA.

Various special techniques can be transferred from 1D-SSA to MSSA, such as Sequential SSA, see Sect. 2.8 for examples of Sequential 1D-SSA analysis. Sequential 1D-SSA is based on successive application of 1D-SSA with different window lengths, see Golyandina and Zhigljavsky (2013; Section 2.5.5) for more details. Sequential MSSA can be applied in a similar manner. In addition to the reasons which are similar to the 1D case, we may find extra arguments in favor of Sequential MSSA. In particular, if trends of different one-dimensional series are of different structure, a smaller window length can be applied to achieve similarity of eigenvectors and to improve separability. After that, the residuals with a common structure (e.g., containing the seasonality) can be simultaneously decomposed with a larger window length.

4.2.5 Algorithm

The algorithm of MSSA decomposition differs from Algorithm 2.1 of Basic SSA decomposition only by the form of the embedding operator.

Algorithm 4.2 MSSA: decomposition

Input: Collection $\{\mathbb{X}^{(p)} = \left(x_j^{(p)}\right)_{j=1}^{N_p},\ p = 1, \ldots, s\}$ of s time series of length N_p, $p = 1, \ldots, s$, window length L.

Output: Decomposition of the trajectory matrix on elementary matrices $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.

1: Construct the trajectory matrix $\mathbf{X} = \mathfrak{T}_{\mathrm{MSSA}}(\mathbb{X})$, where $\mathfrak{T}_{\mathrm{MSSA}}$ is defined by (4.1).
2: Compute the SVD $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.

Reconstruction stage is also very similar to Algorithm 2.2 for Basic SSA reconstruction.

Algorithm 4.3 MSSA reconstruction

Input: Decomposition $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where $\mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}}$ and $\|U_i\| = \|V_i\| = 1$, grouping $\{1, \ldots, d\} = \bigsqcup_{j=1}^{m} I_j$.
Output: Decomposition of the time series system on identifiable components $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$.
1: Construct the grouped matrix decomposition $\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_m}$, where $\mathbf{X}_I = \sum_{i \in I} \mathbf{X}_i$.
2: $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$, where $\mathbb{X}_i = \mathfrak{T}^{-1}_{\mathrm{MSSA}} \circ \Pi_{\text{stacked } \mathcal{H}}(\mathbf{X}_{I_i})$.

Recall that $\mathfrak{T}_{\mathrm{MSSA}}$ and $\Pi_{\text{stacked } \mathcal{H}}$ can be expressed through $\mathfrak{T}_{\mathrm{SSA}}$ and $\Pi_{\mathcal{H}}$ introduced in Chap. 2 for 1D-SSA (see Sect. 4.2.1).

4.2.6 MSSA Analysis in Rssa

4.2.6.1 Description of Functions

Typical call of `ssa` for the MSSA analysis is

```
s <- ssa(x, L = (min(N) + 1)%/%2, kind = "mssa")
```

where `N` is the vector of the series lengths.

Arguments:

`x` is an object containing a collection of time series to be decomposed.
`L` is a window length. By default it is fixed to half of the minimal series length.
`neig` is the number of desired eigentriples. If `neig = NULL`, a sane default value which depends on L and N will be used.
`kind` specifies the kind of SSA to apply.
`svd.method` selects the SVD method to use. Full description is given in Sect. 2.1.5.2.

Additional details and the description of the `reconstruct` function can be found in Sect. 2.1.5, since there is no difference between one-dimensional and multivariate cases here.

4.2.6.2 Typical Code

Here we demonstrate how the MSSA decomposition of a system of time series can be performed by means of the Rssa package. Since the analysis and forecasting of one-dimensional time series by Rssa are thoroughly described in previous chapters and in Golyandina and Korobeynikov (2013), we pay more attention to the differences between 1D-SSA and MSSA.

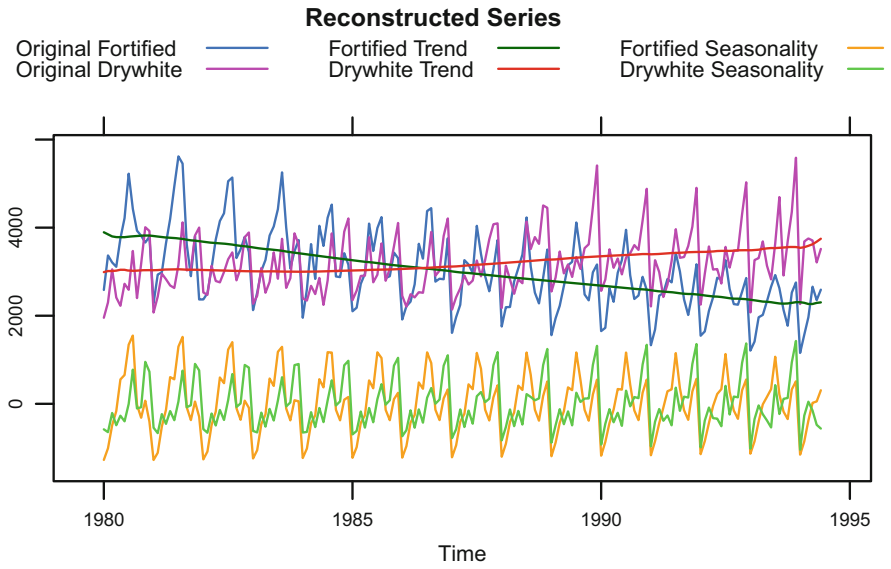

Fig. 4.3 "FORT" and "DRY": Reconstructed trend and seasonality

In Sect. 2.1.5.3 we decomposed the one-dimensional series "FORT" (sales of fortified wines) from the dataset "AustralianWine." Here we add one more series, the sales of dry wines (shortly "DRY"), for simultaneous analysis.

For loading the data we use the code from Fragment 2.1.1.

Fragment 4.2.1 ("FORT" and "DRY": Reconstruction)

```
> wineFortDry <- wine[, c("Fortified", "Drywhite")]
> L <- 84
> s.wineFortDry <- ssa(wineFortDry, L = L, kind = "mssa")
> r.wineFortDry <- reconstruct(s.wineFortDry,
+                              groups = list(Trend = c(1, 6),
+                                 Seasonality = c(2:5, 7:12)))
> plot(r.wineFortDry, add.residuals = FALSE,
+      plot.method = "xyplot",
+      superpose = TRUE, auto.key = list(columns = 3))
```

Fragment 4.2.1 contains a typical code for simultaneous extraction of the trend and seasonality (compare with Fragment 2.1.2) and produces Fig. 4.3. A clear difference between the two fragments is in the indicated value of the parameter `kind` in the `ssa` function. A more significant difference is related to plotting the results. For a multivariate series, there is, in a sense, a matrix of series, where one index is the series number in the system and the second index indicates the component number in the decomposition. The `plot` function for the reconstruction object allows to indicate which subset (slice) of this matrix one wants to depict by

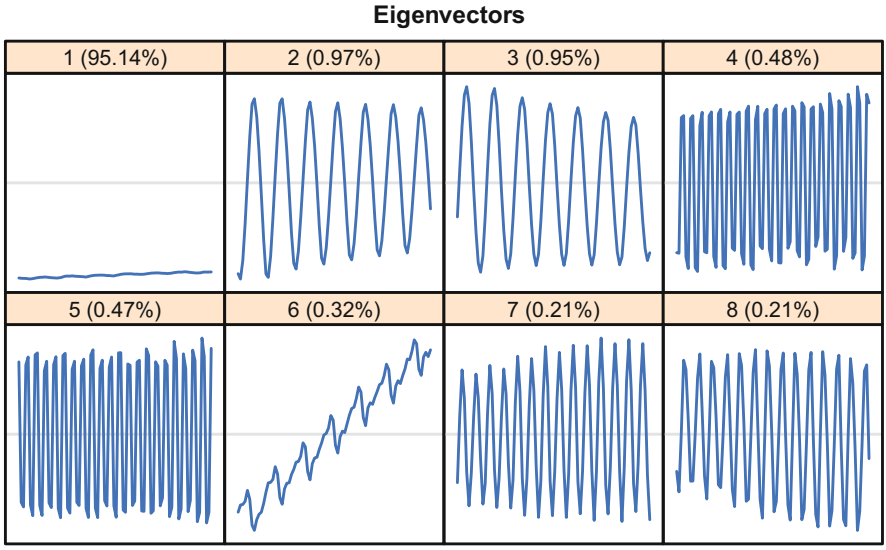

Fig. 4.4 "FORT" and "DRY": 1D graphs of eigenvectors

means of the parameter `slice`. The parameter `slice` consists of the list of indices of series and indices of decomposition components. The use of `slice` is demonstrated on several examples in Sects. 4.4.1 and 4.4.3.

The code for the component identification in MSSA is very similar to that in SSA, compare Fragments 4.2.2 and 2.1.3. The difference is in the structure of the factor vectors; however, they are not necessary for the identification. Figure 4.4 (compare it with Fig. 2.2) shows that the trend is described by ET1 and ET6, which is slightly mixed with seasonality. Figure 4.5 (compare with Fig. 2.3) demonstrates those pairs of ETs that are related to seasonality.

The results of MSSA analysis are similar to the results of 1D-SSA analysis. However, the separability is sometimes slightly worse for MSSA. Iterative O-SSA (Sect. 2.4) and DerivSSA (Sect. 2.5) can be applied to MSSA objects to improve separability. One can see in Fig. 4.6 (compare ET2 here with ET6 in Fig. 4.4) that after application of the Iterative O-SSA the trend is no longer mixed with seasonality. The eigentriples are reordered and the trend is described by the first two eigentriples. Note that if the signal components are mixed up between themselves, then the signal forecasting is not affected by this. However, if one wants to forecast the trend only, then the mixture would typically worsen the forecast accuracy.

Since the implemented methods of parameter estimation are based on eigenvectors only, they can be applied to eigenvectors in multidimensional case in exactly the same way as in the one-dimensional case.

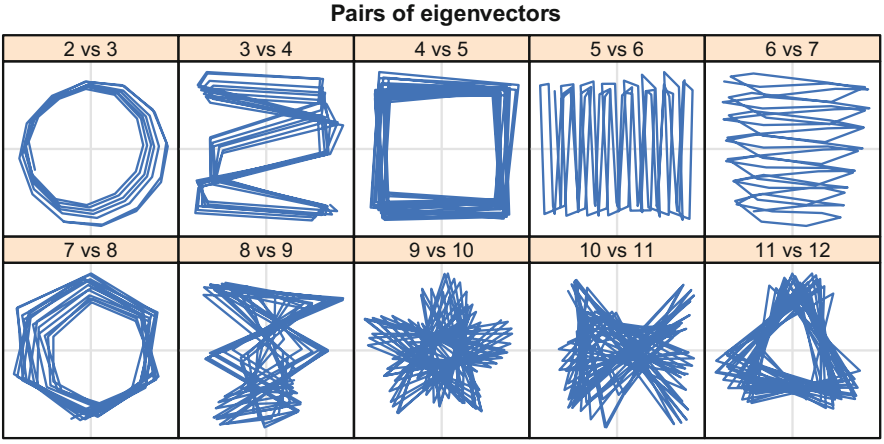

Fig. 4.5 "FORT" and "DRY": 2D scatterplots of eigenvectors

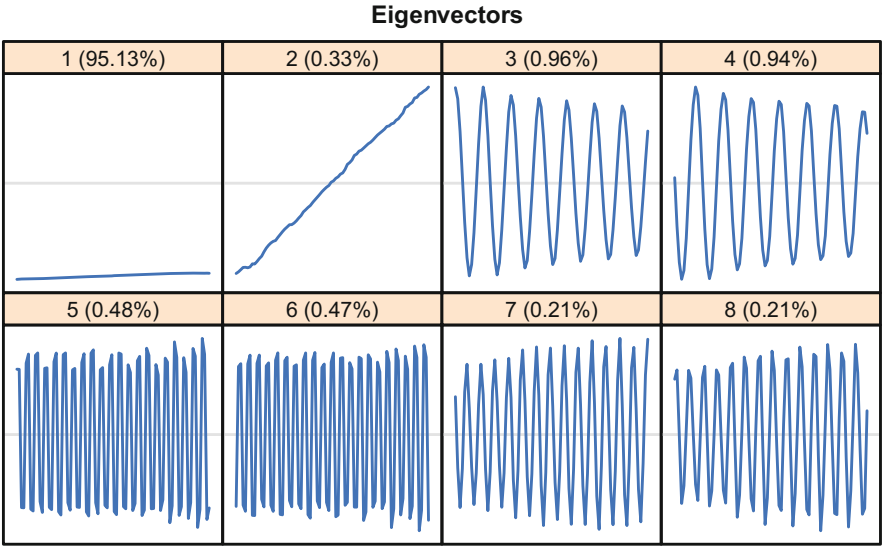

Fig. 4.6 "FORT" and "DRY": 1D graphs of eigenvectors after Iterative O-SSA

Fragment 4.2.2 ("FORT" and "DRY": Identification)

```
> plot(s.wineFortDry, type = "vectors", idx = 1:8)
> plot(s.wineFortDry, type = "paired", idx = 2:11,
+      plot.contrib = FALSE)
> print(parestimate(s.wineFortDry, groups = list(2:3, 4:5),
+                   method = "esprit"))
$F1
   period      rate    |     Mod      Arg   |      Re         Im
```

```
   12.128  -0.004789 |  0.99522   0.52 |  0.86463   0.49283
  -12.128  -0.004789 |  0.99522  -0.52 |  0.86463  -0.49283
$F2
   period     rate    |    Mod     Arg  |     Re        Im
    4.007  -0.001226 |  0.99877   1.57 |  0.00279   0.99877
   -4.007  -0.001226 |  0.99877  -1.57 |  0.00279  -0.99877
> plot(wcor(s.wineFortDry, groups = 1:30),
+      scales = list(at = c(10, 20, 30)))
> si.wineFortDry <- iossa(s.wineFortDry,
+                         nested.groups = list(c(1,6), c(2:5, 7:12)))
> plot(si.wineFortDry, type = "vectors", idx = 1:8)
```

4.2.6.3 Comments

Formats of Input and Output Data

While the representation of a one-dimensional time series in R is pretty obvious, there are many possible ways of defining a multivariate time series. Let us outline some common choices.

- A matrix with separate series in the columns. Optionally, an additional time structure like in `mts` objects can be embedded.
- A matrix-like (e.g., a `data.frame`) object with series in the columns. In particular, `data.frame` would be a result of reading the series from a file via the `read.table` function.
- A list of separate time series objects (e.g., a `list` of `ts` or `zoo` objects).

Also, the time scales of the individual time series can be normalized via head or tail padding with `NA` (for example, as a result of the `ts.union` call) or specified via time series attributes.

The package is designed to allow any of the input cases outlined above and produces the reconstructed series in the same format. All the attributes, names of the series, `NA` padding, etc. are carefully preserved. For forecasted series, the time scale attributes for several known time series objects (e.g., `ts`) are inferred automatically where possible.

The examples in Fragments 4.2.1 and 4.3.2 provide an overview of the possible input series formats.

Plotting Specifics

Plotting of the reconstructed series is performed by the function `plot` applied to the reconstructed collection of time series. The parameter `plot.method` can be `"native"` or `"xyplot"`. Keep in mind that the default (`"native"`) plotting method for reconstruction objects may or may not be suitable for multivariate time series plotting. For example, it provides many useful plotting possibilities for `ts` and `mts` objects, but may be totally unusable in the case of `data.frame` objects, because it will only call the `pairs` function on the resulting data frame at the end.

Efficient Implementation

All ideas from the one-dimensional case can be extended to the multivariate case. In the one-dimensional case, the complexity is determined by the series length N and the window length L and the worst case corresponds to $L \sim K \sim N/2$ with overall complexity of $O(L^3 + L^2K) = O(N^3)$.

In the multidimensional case (for simplicity assume that all the series have equal lengths N), the worst case corresponds to $L \sim K \sim sN/(s+1)$; the order of complexity is the same $O(N^3)$ but the constant can be considerably larger. Therefore, the speed-up (due to efficient implementation) giving the order $O(kN\log(N) + k^2N)$, where k is the number of calculated eigentriples, in the multivariate case can be much higher than in the one-dimensional case.

Note that MSSA can be viewed as a special case of Shaped 2D-SSA (see Sect. 5.2.1.3) and the current implementation in the package implicitly uses this.

4.3 MSSA Forecasting

Recall from Sect. 3.2 that forecasting in 1D-SSA is performed for a signal component which can be separated by 1D-SSA and is governed, perhaps approximately, by an LRR. For brevity, we will talk about forecasting of the whole signal. 1D-SSA provides an estimate of the signal subspace and thereby an estimate of one of LRRs governing the signal. The recurrent 1D-SSA forecasting continues the estimated signal by the estimated LRR. The vector 1D-SSA forecasting continues the reconstructed vectors in the given subspace.

Methods of one-dimensional SSA forecasting in a given subspace are described in Sect. 3.2. For CSSA, the forecasting algorithms are straightforward extensions of 1D-SSA forecasting algorithms to the complex-valued case, therefore we do not discuss them here. On the other hand, the methods of MSSA forecasting require special attention.

As in 1D-SSA, methods of MSSA forecasting can be subdivided into recurrent and vector forecasting. In contrast with 1D-SSA, rows and columns of the trajectory matrix in MSSA have different structure. Therefore, there exist two kinds of MSSA forecasting: row forecasting and column forecasting; this depends on which of the two spaces the forecasting is made (row or column space respectively). In total, there are four main variants of MSSA forecasting: recurrent column forecasting, recurrent row forecasting, vector column forecasting, and vector row forecasting.

There are different names for the same forecasting methods. In Golyandina and Stepanov (2005), column and row forecasting are called L- and K-forecasting. In Hassani and Mahmoudvand (2013), these methods are called horizontal and vertical forecasts and the trajectory matrix is transposed. In Sects. 4.2.1 and 4.2.4.1 we have explained the choice of orientation of the MSSA trajectory matrix and the connection between the horizontally-stacked and vertically-stacked trajectory matrices of separate time series. We use the name "column" and "row" with respect to the horizontally-stacked trajectory matrices as defined in Sect. 4.2.1.

In the column forecasting methods, each time series in the system is forecasted separately but in a given common subspace (i.e., using the common LRR). In the row forecasting methods, each series is forecasted with the help of its own LRR applied to the whole set of series from the system. Let us describe all four variants of MSSA forecasting.

4.3.1 Method

4.3.1.1 Common Notation

First, we introduce some common notation used for description of all the variants of MSSA forecasting.

Denote by $\overline{A} \in \mathsf{R}^{Q-1}$ the vectors consisting of the last $Q-1$ coordinates of $A \in \mathsf{R}^Q$; that is, the vectors with the first coordinate removed are indicated by the line on the top of the vector. Denote by $\underline{A} \in \mathsf{R}^{Q-1}$ the vectors consisting of the first $Q-1$ coordinates of A; by $\pi(A)$ we denote the last coordinate of the vector. For a matrix $\mathbf{A} = [A_1 : \ldots : A_r]$, we denote $\overline{\mathbf{A}} = [\overline{A}_1 : \ldots : \overline{A}_r]$ and $\underline{\mathbf{A}} = [\underline{A}_1 : \ldots : \underline{A}_r]$ and let $\boldsymbol{\pi}(\mathbf{A}) = (\pi(A_1), \ldots, \pi(A_r))^{\mathrm{T}}$ be the last row of the matrix $\mathbf{A}$.

Consider the following form of $B \in \mathsf{R}^K$, where $K = \sum_{i=1}^s K_i$, induced by the structure of the row trajectory space:

$$B = \begin{pmatrix} B^{(1)} \\ B^{(2)} \\ \vdots \\ B^{(s)} \end{pmatrix}, \qquad \underline{B} = \begin{pmatrix} \underline{B}^{(1)} \\ \underline{B}^{(2)} \\ \vdots \\ \underline{B}^{(s)} \end{pmatrix}, \qquad \overline{\underline{B}} = \begin{pmatrix} \overline{\underline{B}}^{(1)} \\ \overline{\underline{B}}^{(2)} \\ \vdots \\ \overline{\underline{B}}^{(s)} \end{pmatrix}, \tag{4.8}$$

where $B^{(j)} \in \mathsf{R}^{K_j}$, and let $\boldsymbol{\mu}(B) = (\pi(B^{(1)}), \ldots, \pi(B^{(s)}))^{\mathrm{T}}$. Also, for $\mathbf{B} = [B_1 : \ldots : B_r]$ let $\underline{\mathbf{B}} = [\underline{B}_1 : \ldots : \underline{B}_r]$ and $\mathbf{B}^{(j)} = [B_1^{(j)} : \ldots : B_r^{(j)}]$.

Assume that the group I corresponding to the forecasted component is given by the set of the leading components at Decomposition step of Algorithm 4.2; this assumption is made just for simplifying the formulas. Thus, let r leading eigentriples $(\sqrt{\lambda_j}, U_j, V_j)$ be identified and chosen as related to the signal of rank r so that $I = I_1 = \{1, \ldots, r\}$, $\mathbf{U} = [U_1 : \ldots : U_r]$, $\mathbf{V} = [V_1 : \ldots : V_r]$. The reconstructed series $\widetilde{\mathbb{X}}$, its trajectory matrix $\widetilde{\mathbf{X}}$, and the reconstructed matrix $\widehat{\mathbf{X}}$ are defined in Sect. 1.1.1. Define $\mathcal{L}^{\mathrm{col}} = \mathrm{span}(U_i, i \in I)$, $\mathcal{L}^{\mathrm{row}} = \mathrm{span}(V_i, i \in I)$. The reconstructed matrix $\widehat{\mathbf{X}} = [\widehat{\mathbf{X}}_1 : \ldots : \widehat{\mathbf{X}}_K]$ consists of the column vectors which are the projections of the column vectors of the trajectory matrix on the chosen subspace $\mathcal{L}^{\mathrm{col}}$.

To avoid repeating the transpose sign, denote $\widetilde{\mathbf{Y}} = [\widetilde{\mathbf{Y}}_1 : \ldots : \widetilde{\mathbf{Y}}_L] = \widetilde{\mathbf{X}}^{\mathrm{T}}$, $\widehat{\mathbf{Y}} = [\widehat{\mathbf{Y}}_1 : \ldots : \widehat{\mathbf{Y}}_L] = \widehat{\mathbf{X}}^{\mathrm{T}}$, $\widehat{\mathbf{Y}}_k = \widehat{\mathbf{X}}_k^{\mathrm{T}}$.

4.3.1.2 Recurrent MSSA Forecast

We denote the vector of forecasted signal values for each time series by $R_N = \left(\tilde{x}^{(1)}_{N_1+1}, \tilde{x}^{(2)}_{N_2+1}, \ldots, \tilde{x}^{(s)}_{N_s+1}\right)^{\mathrm{T}}$. Recurrent forecasting is closely related to missing data imputation for components of vectors from the given subspace and in fact uses the formula (1) from Golyandina and Osipov (2007). Following Golyandina and Stepanov (2005), we will write the forecasting formulas for two versions of the recurrent MSSA forecast: row (generated by $\{U_j\}_{j=1}^r$) and column (generated by $\{V_j\}_{j=1}^r$). These one-term ahead forecasting formulas can be applied for M-term ahead forecasting by using the recurrence.

The column recurrent forecasting performs forecast by an LRR of order $L-1$ applied to the last $L-1$ points of the reconstructed signal; that is, the same LRR and different initial data. The row recurrent forecasting constructs s different linear relations, each is applied to the set of $K_i - 1$ last points of series; that is, the LRRs are different but the initial data for them is the same.

Column Forecast

Denote by $\mathbf{Z}$ the matrix consisting of the last $L-1$ values of the reconstructed signals:

$$\mathbf{Z} = \begin{pmatrix} \tilde{x}^{(1)}_{N_1-L+2} & \cdots & \tilde{x}^{(1)}_{N_1} \\ \tilde{x}^{(2)}_{N_2-L+2} & \cdots & \tilde{x}^{(2)}_{N_2} \\ \vdots & \vdots & \vdots \\ \tilde{x}^{(s)}_{N_s-L+2} & \cdots & \tilde{x}^{(s)}_{N_s} \end{pmatrix},$$

$\nu^2 = \sum_{j=1}^{r} \pi(U_j)^2$. If $\nu^2 < 1$, then the column MSSA forecast is uniquely defined and can be calculated by the formula

$$R_N = \mathbf{Z}\mathcal{R}_L, \qquad \text{where} \qquad \mathcal{R}_L = \frac{1}{1-\nu^2}\sum_{j=1}^{r} \pi(U_j)\underline{U_j} \ \in \mathsf{R}^{L-1}. \tag{4.9}$$

Note that (4.9) implies that the forecasting of all individual signals is made using the same linear recurrent formula which is generated by the whole system.

Row Forecast

Introduce the vectors of the last $K_m - 1$ values of the reconstructed signals

$$Z^{(m)} = \left(\tilde{x}^{(m)}_{N-K_m+2}, \ldots, \tilde{x}^{(m)}_{N_m}\right)^{\mathrm{T}}, \quad m = 1, \ldots, s,$$

and denote

$$Z = \begin{pmatrix} Z^{(1)} \\ Z^{(2)} \\ \vdots \\ Z^{(s)} \end{pmatrix}, \qquad \mathbf{S} = [\boldsymbol{\mu}(V_1) : \ldots : \boldsymbol{\mu}(V_r)].$$

In this notation, $Z = \overline{\overline{\widetilde{Y}}}_L$.

If the inverse matrix $(\mathbf{I}_s - \mathbf{S}\mathbf{S}^{\mathrm{T}})^{-1}$ exists and $r \le K - s$, then the row MSSA recurrent forecast exists and can be calculated by the formula

$$R_N = \mathcal{R}_K Z, \quad \text{where } \mathcal{R}_K = (\mathbf{I}_s - \mathbf{S}\mathbf{S}^{\mathrm{T}})^{-1} \mathbf{S}\underline{\underline{V}}^{\mathrm{T}}. \tag{4.10}$$

Note that (4.10) implies that the forecasting of the individual signals is made using the LRRs which are different for different series. The forecasting value generally depends on the last values of all time series from the system of time series.

4.3.1.3 Vector MSSA Forecasting

Denote $\underline{\mathcal{L}}^{\mathrm{col}} = \mathrm{span}(\underline{U}_1, \ldots, \underline{U}_r)$ and $\underline{\underline{\mathcal{L}}}^{\mathrm{row}} = \mathrm{span}(\underline{\underline{V}}_1, \ldots, \underline{\underline{V}}_r)$. Let Π^{col} be the orthogonal projector of R^{L-1} on $\underline{\mathcal{L}}^{\mathrm{col}}$ and Π^{row} be the orthogonal projector of R^{K-s} on $\underline{\underline{\mathcal{L}}}^{\mathrm{row}}$.

An explicit form of the matrices of the column and row projectors can be found in Golyandina and Osipov (2007; formula (4)). However, the calculation by that formula is time-consuming. A fast algorithm of calculation is presented in Golyandina et al. (2015; Section 6.3).

Column Forecast

We have mentioned above that for a given subspace ($\mathcal{L}^{\mathrm{col}}$ in our case) the column forecast is performed independently for each time series. Define the linear operator $\mathcal{P}^{\mathrm{col}}_{\mathrm{Vec}} : \mathsf{R}^L \mapsto \mathcal{L}^{\mathrm{col}}$ by the formula

$$\mathcal{P}^{\mathrm{col}}_{\mathrm{Vec}} Z = \begin{pmatrix} \Pi^{\mathrm{col}} \overline{Z} \\ \mathcal{R}_L^{\mathrm{T}} \overline{Z} \end{pmatrix}, \tag{4.11}$$

where $\mathcal{R}_L$ is defined in (4.9).

The *vector forecasting algorithm* for jth series is as follows:

1. In the notation above, define vectors Z_i as follows:

$$Z_i = \begin{cases} \widehat{X}_i^{(j)} & \text{for } i = 1, \ldots, K_j, \\ \mathcal{P}^{\mathrm{col}}_{\mathrm{Vec}} Z_{i-1} & \text{for } i = K_j + 1, \ldots, K_j + M + L - 1. \end{cases} \tag{4.12}$$

2. By constructing the matrix $\mathbf{Z} = [Z_1 : \ldots : Z_{K_j+M+L-1}]$ and making its diagonal averaging we obtain the series $z_1, \ldots, z_{N_j+M+L-1}$.
3. The numbers $z_{N_j+1}, \ldots, z_{N_j+M}$ form the M terms of the vector forecast.

Row Forecast

Define the linear operator $\mathcal{P}^{\mathrm{row}}_{\mathrm{Vec}} : \mathsf{R}^K \mapsto \mathcal{L}^{\mathrm{row}}$ by the formula

$$\mathcal{P}^{\mathrm{row}}_{\mathrm{Vec}} Z = A, \tag{4.13}$$

such that $\underline{\underline{A}} = \Pi^{\mathrm{row}} \overline{\overline{Z}}$ and $\boldsymbol{\mu}(A) = \mathfrak{R}_K \overline{\overline{Z}}$, where $\mathfrak{R}_K$ is defined in (4.10).

The *vector forecasting algorithm* is as follows:

1. In the notation above, define vectors Z_i as follows:

$$Z_i = \begin{cases} \widehat{Y}_i & \text{for } i = 1, \ldots, L, \\ \mathcal{P}^{\mathrm{row}}_{\mathrm{Vec}} Z_{i-1} & \text{for } i = L+1, \ldots, L+M+K^*-1, \end{cases} \tag{4.14}$$

   where $K^* = \max(K_i, i = 1, \ldots, s)$.
2. By constructing the matrix $\mathbf{Z} = [Z_1 : \ldots : Z_{L+M+K^*-1}]$ and making Reconstruction step we obtain the series $z_1^{(j)}, \ldots, z_{N_j+M+K^*-1}^{(j)}$, $j = 1, \ldots, s$.
3. The numbers $z_{N_j+1}^{(j)}, \ldots, z_{N_j+M}^{(j)}$, $j = 1, \ldots, s$, form the M terms of the vector forecast.

Remark 4.1 For the M-step ahead vector forecast, $M + K^* - 1$ new lagged vectors for the row forecasting and $M + L - 1$ ones for the column forecasting are constructed. The reason for this is to make the M-step forecast inheriting the $(M-1)$-step forecast as its part. This specific feature of the vector forecasting provides its stability and accuracy if the accurately extracted component of finite rank is forecasted; that is, if a long-term forecast is appropriate. Otherwise (if the MSSA approximation is inadequate), the long-term vector forecasting can be misleading and even a short-term vector forecasting can be inaccurate for large K^* or L correspondingly.

4.3.2 Algorithms

Algorithms of MSSA column forecasting are very similar to the algorithms of 1D-SSA forecasting (see Algorithms 3.5 and 3.6). Let a version of MSSA be applied to the system of s time series $\mathbb{X}$ and let the eigentriples $\{(\sigma_i, P_i, Q_i),\ i \in I\}$ be chosen for reconstruction. The suggested forecasting algorithms are formulated for the forecasting in the subspace $\mathcal{L}_r = \mathrm{span}\{P_i, i \in I\} \subset \mathsf{R}^L$. For simplicity, we assume that $I = \{1, \ldots, r\}$ and the vectors $P_i, i \in I$, are orthonormal. Note that the forecasting values do not depend on the choice of a basis of $\mathcal{L}_r$.

Algorithm 4.4 Recurrent MSSA column forecasting

Input: Collection of time series $\mathbb{X}^{(p)}$ of length N_p, where $p = 1, \dots, s$, window length L, orthonormal system of vectors $\{P_i\}_{i=1}^r$, forecast horizon M.

Output: Forecast values $(\widetilde{x}_{N_p+1}^{(p)}, \dots, \widetilde{x}_{N_p+M}^{(p)})$, $p = 1, \dots, s$.

1: Construct the vector $\mathcal{R} = (a_{L-1}, \dots, a_1)^{\mathrm{T}}$ of coefficients of the min-norm LRR by Algorithm 3.1 applied to $\{P_i, i \in I\}$.
2: Construct the reconstructed matrices $\widehat{\mathbf{X}}^{(p)} = \mathbf{P}\mathbf{P}^{\mathrm{T}}\mathbf{X}^{(p)}$, where $\mathbf{P} = [P_1 : \dots : P_r]$, and the reconstructed series $\widetilde{\mathbb{X}}^{(p)} = (\widetilde{x}_1^{(p)}, \dots, \widetilde{x}_{N_p}^{(p)})$ as $\widetilde{\mathbb{X}}^{(p)} = \mathcal{T}_{\mathrm{SSA}}^{-1} \circ \Pi_{\mathcal{H}}(\widehat{\mathbf{X}}^{(p)})$; $p = 1, \dots, s$.
3: Calculate the forecast values recurrently by

$$\widetilde{x}_n^{(p)} = \sum_{i=1}^{L-1} a_i \widetilde{x}_{n-i}^{(p)}, \ n = N_p + 1, \dots, N_p + M; \ p = 1, \dots, s.$$

Algorithm 4.4 is written for the version, where the reconstructed series is taken as the base for forecasting. In the other version, where the original series itself is the base of forecasting, $x_n^{(p)}$ are taken instead of $\widetilde{x}_n^{(p)}$ at Step 3 for $n = N_p - L + 1, \dots, N_p$.

The next algorithm implements vector forecasting.

Algorithm 4.5 Vector MSSA column forecasting

Input: Collection of time series $\mathbb{X}^{(p)}$ of length N_p, where $p = 1, \dots, s$, window length L, orthonormal system of vectors $\{P_i\}_{i=1}^r$, forecast horizon M.

Output: Forecast values $(\widetilde{x}_{N_p+1}^{(p)}, \dots, \widetilde{x}_{N_p+M}^{(p)})$, $p = 1, \dots, s$.

1: Obtain the vector $\mathcal{R} = (a_{L-1}, \dots, a_1)^{\mathrm{T}}$ of coefficients of the min-norm LRR by Algorithm 3.1 applied to $\{P_i, i \in I\}$.
2: Calculate the matrix Π defining the projection in (3.5).
3: Compute the reconstructed matrices $\widehat{\mathbf{X}}^{(p)} = \mathbf{P}\mathbf{P}^{\mathrm{T}}\mathbf{X}^{(p)}$, where $\mathbf{P} = [P_1 \dots : P_r]$; $p = 1, \dots, s$.
4: Extend the reconstructed matrices $\widehat{\mathbf{X}}^{(p)} = [\widehat{X}_1^{(p)} : \dots : \widehat{X}_{K_p}^{(p)}]$ by column vectors:

$$\widehat{X}_n^{(p)} = \mathcal{P}_{\mathrm{Vec}} \widehat{X}_{n-1}^{(p)} \text{ for } n = K_p + 1, \dots, K_p + M + L - 1; \ p = 1, \dots, s,$$

where $\mathcal{P}_{\mathrm{Vec}}$ is given in (3.6) and uses the vector $\mathcal{R}$ of coefficients of the min-norm LRR. Denote the extended matrices by $\widehat{\mathbf{X}}_{\mathrm{ext}}^{(p)}$; $\widehat{\mathbf{X}}_{\mathrm{ext}}^{(p)} \in \mathsf{R}^{L \times (K_p + M + L - 1)}$; $p = 1, \dots, s$.
5: Obtain the extended reconstructed series $\widetilde{\mathbb{X}}_{\mathrm{ext}}^{(p)} = (\widetilde{x}_1^{(p)}, \dots, \widetilde{x}_{N_p+M+L-1}^{(p)})$ by $\widetilde{\mathbb{X}}_{\mathrm{ext}}^{(p)} = \mathcal{T}_{\mathrm{SSA}}^{-1} \circ \Pi_{\mathcal{H}}(\widehat{\mathbf{X}}_{\mathrm{ext}}^{(p)})$; $p = 1, \dots, s$.
6: Return the forecast values $(\widetilde{x}_{N_p+1}^{(p)}, \dots, \widetilde{x}_{N_p+M}^{(p)})$; $p = 1, \dots, s$.

Algorithms of row forecasting have more complicated form and follow the steps outlined in Sect. 4.3.1.

4.3.3 *MSSA Forecasting in* Rssa

4.3.3.1 Description of Functions

MSSA forecasting functions differ from the functions described in Sect. 3.2.3.1 for one-dimensional case by the additional parameter `direction`, which can be equal to either `"row"` or `"column"`. The other parameters are exactly the same as described in Sect. 3.2.3.1. For example, a call for the recurrent row forecast based on the `ssa` object `s` can be made as follows:

```
f <- rforecast(s, groups = list(1:2, 3:4),
               direction = "row", len = 12, only.new = FALSE)
```

The wrappers `predict` is able to construct forecasts in a unified manner. For example, the presented call of `rforecast` is similar to the call

```
f <- predict(s, groups = list(1:2, 3:4),
             method = "recurrent", direction = "row", len = 12)
```

The function `forecast`, the bootstrap intervals, and backward forecasting are not implemented for MSSA forecasting.

4.3.3.2 Typical Code

The code for forecasting is very similar to that in 1D-SSA, compare Fragments 4.3.1 and 3.2.1. For demonstration, we use the monthly sales of fortified ("FORT") and dry ("DRY") wines taken from the dataset "AustralianWine" (Fig. 4.7).

Fragment 4.3.1 ("FORT" and "DRY": Forecast)

```
> f.wineFortDry <- rforecast(s.wineFortDry,
+                            groups = list(1, 1:12),
+                            len = 60, only.new = TRUE)
> plot(cbind(wineFortDry[, "Fortified"],
+            f.wineFortDry$F2[, "Fortified"]),
+      plot.type = "single",
+      col = c("black", "red"), ylab = "Fort")
> plot(cbind(wineFortDry[, "Drywhite"],
+            f.wineFortDry$F2[, "Drywhite"]),
+      plot.type = "single",
+      col = c("black", "red"), ylab = "Dry")
> par(mfrow = c(1, 1))
```

4.3.3.3 Simulated Example: Numerical Comparison

In this section, we demonstrate how the accuracy of MSSA is related to the structure of the multivariate time series. The aim is to compare accuracy for separate analysis and forecasting of time series with simultaneous processing of the series system. We

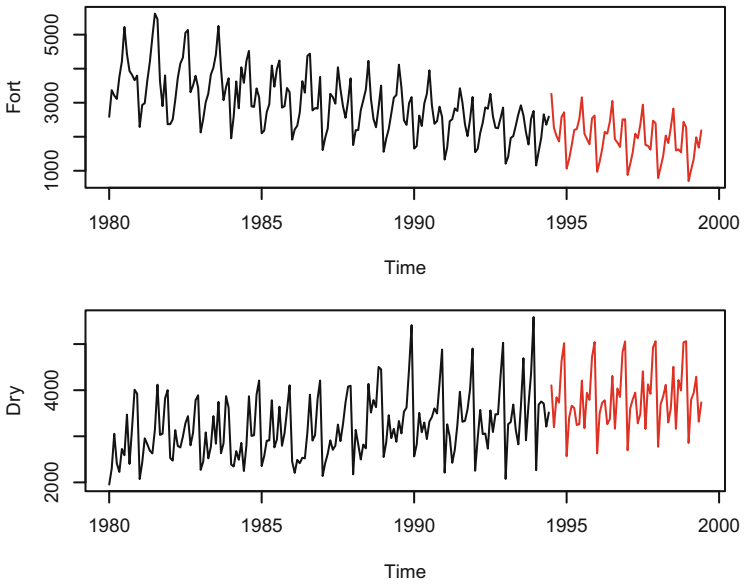

Fig. 4.7 "FORT" and "DRY": Forecast of the signal

summarize the results from Golyandina and Stepanov (2005) and Golyandina et al. (2015) and supplement them with new comparisons. In particular, the comparison results explain the choice of the default forecasting method.

In the study below, we consider the case $s = 2$ and examine the following SSA methods: (a) 1D-SSA applied twice, (b) MSSA, and (c) CSSA. The investigated model examples include the least favorable and the most favorable cases for MSSA as well as some cases well suited for the application of CSSA.

Let us assume that we observe $(\mathbb{X}^{(1)}, \mathbb{X}^{(2)}) = (\mathbb{H}^{(1)}, \mathbb{H}^{(2)}) + (\mathbb{N}^{(1)}, \mathbb{N}^{(2)})$, where $(\mathbb{H}^{(1)}, \mathbb{H}^{(2)})$ is a two-dimensional signal consisting of two harmonic time series, $\mathbb{N}^{(1)}$ and $\mathbb{N}^{(2)}$ are realizations of independent white Gaussian noises. Then we can use the standard simulation techniques to obtain estimates of the mean square errors (MSE) for the reconstruction and forecasting of $(\mathbb{H}^{(1)}, \mathbb{H}^{(2)})$ by the indicated SSA methods. The resultant MSE is calculated as the mean of $\mathrm{MSE}^{(1)}$ and $\mathrm{MSE}^{(2)}$ for $\mathbb{H}^{(1)}$ and $\mathbb{H}^{(2)}$ correspondingly.

We take the following parameters for the simulation of the time series: $N = 71$, the variance of all noise components is $\sigma^2 = 25$, the number of replications is 10000. We consider the following three versions of the signal $(\mathbb{H}^{(1)}, \mathbb{H}^{(2)})$.

Example A (the same periods; the difference between the phases is different from $\pi/2$):

$$h_k^{(1)} = 30\cos(2\pi k/12), \quad h_k^{(2)} = 20\cos(2\pi k/12 + \pi/4), \quad k = 1, \ldots, N.$$

Example B (the same periods and amplitudes; the difference between the phases is equal to $\pi/2$):

$$h_k^{(1)} = 30\cos(2\pi k/12), \quad h_k^{(2)} = 30\cos(2\pi k/12 + \pi/2), \quad k = 1, \ldots, N.$$

Example C (different periods):

$$h_k^{(1)} = 30\cos(2\pi k/12), \quad h_k^{(2)} = 20\cos(2\pi k/8 + \pi/4), \quad k = 1, \ldots, N.$$

The choice of these examples is determined by the observation that the dimensions of the signal trajectory spaces (i.e., ranks) are different for different extensions of the 1D-SSA method, see Table 4.1. For each example the rank in blue corresponds to the method with the best accuracy for this example. Cells in the row corresponding to 1D-SSA contain one number, since the ranks of the times series from the considered collections coincide.

The results of investigation for different window lengths L are summarized in Tables 4.2 and 4.3. The 24 term-ahead forecast was performed. For each example, the cells corresponding to the method with the reconstruction/forecast accuracy, which is closed to the best one, are shown in bold and the overall minimum is in blue color.

Comparison of Tables 4.2 and 4.3 with Table 4.1 clearly demonstrates the relation between the accuracy of the signal reconstruction (forecast) and the dimension of the signal trajectory space. Since the structure of the series from Example B and Example A is the same from the viewpoint of MSSA and 1D-SSA, we omit the corresponding results for Example B in Table 4.3.

Table 4.1 Dimension of the signal trajectory space

	Example A	Example B	Example C
MSSA	**2**	2	4
1D-SSA	2	2	**2**
CSSA	2	**1**	4

Table 4.2 MSE of signal reconstruction

Example A	$L = 12$	$L = 24$	$L = 36$	$L = 48$	$L = 60$
MSSA	3.18	1.83	1.59	**1.47**	2.00
1D-SSA	3.25	**2.01**	**2.00**	**2.01**	3.25
CSSA	3.25	**2.02**	**2.01**	**2.02**	3.25
Example B	$L = 12$	$L = 24$	$L = 36$	$L = 48$	$L = 60$
MSSA	3.18	1.82	1.58	**1.47**	1.97
1D-SSA	3.25	**2.01**	**2.00**	**2.01**	3.25
CSSA	1.57	**1.00**	**0.99**	**1.00**	1.57
Example C	$L = 12$	$L = 24$	$L = 36$	$L = 48$	$L = 60$
MSSA	6.91	3.77	3.07	**2.88**	3.84
1D-SSA	3.23	**2.01**	**2.00**	**2.01**	3.23
CSSA	6.98	4.06	**3.82**	4.06	6.98

Table 4.3 MSE of signal forecast

Example A	$L = 12$	$L = 24$	$L = 36$	$L = 48$	$L = 60$
Recurrent					
MSSA-column	5.36	**3.67**	3.73	3.70	4.43
MSSA-row	6.02	4.25	3.83	**3.32**	3.98
1D-SSA	7.24	**5.59**	6.30	6.42	7.93
CSSA	7.30	**5.60**	6.32	6.41	7.86
Vector					
MSSA-column	5.93	3.77	3.62	**3.11**	3.65
MSSA-row	4.00	**3.03**	3.39	3.17	4.24
1D-SSA	7.74	5.43	5.85	**5.14**	6.76
CSSA	7.79	5.44	5.86	**5.12**	6.87
Example C	$L = 12$	$L = 24$	$L = 36$	$L = 48$	$L = 60$
Recurrent					
MSSA-column	25.76	**7.39**	7.55	7.43	9.00
MSSA-row	19.82	8.47	8.00	**6.66**	8.30
1D-SSA	7.36	**5.61**	6.28	6.44	8.00
CSSA	38.79	**11.21**	13.37	13.09	24.89
Vector					
MSSA-column	25.34	7.56	7.57	**6.20**	7.67
MSSA-row	57.59	**6.04**	7.03	6.30	8.69
1D-SSA	7.84	5.47	5.84	**5.18**	6.88
CSSA	35.77	10.89	13.44	**10.22**	69.04
Example B	$L = 12$	$L = 24$	$L = 36$	$L = 48$	$L = 60$
CSSA recurrent	3.48	**2.76**	3.10	3.19	3.99
CSSA vector	3.82	2.70	2.89	**2.56**	3.18

Note that the reconstructions by 1D-SSA and CSSA are the same for window lengths L and $N - L + 1$ (12 and 60, 24 and 48 for the considered examples). Reconstructions by MSSA are different for different L. Also note that the trajectory matrix for 1D-SSA has rank $\min(L, N - L + 1)$ and the rank is maximal for $L \simeq N/2$. The MSSA-trajectory matrix has rank equal to $\min(L, (N - L + 1)s)$, where s is the number of time series in the system. This rank is maximal for $L \simeq sN/(s+1)$. Although the maximality of the rank does not guarantee the minimality of errors, this consideration means that to achieve better separability the choice of the window length L larger than $N/2$ can often be recommended. Simulations confirm this: the minimum of the reconstruction error for MSSA is achieved at $L = 48 = 72 \times 2/3$.

The forecasting errors have much more complicated structure, see Golyandina (2010). In particular, these errors for forecasting depend on the reconstruction errors for the last time series points; therefore, the error may have a dependence on L, which is different from that for the average reconstruction errors. The considered examples show that the vector forecast is more accurate than the recurrent one and that the row MSSA forecast is slightly more accurate than the column MSSA forecast.

The considered examples confirm the following assertions:

- The accuracy of the SSA-based methods is closely related to the structure of the signal trajectory spaces generated by these methods. MSSA has an advantage if time series from the system have matched components. (Note that we considered equal levels of noise.)
- Optimal window lengths for analysis and forecasting can differ. Despite the accuracy of forecast is related to the accuracy of reconstruction, this relation is not straightforward.
- The vector forecast with the best window length is more accurate than the recurrent forecast. However, it is not always the case if we compare forecast accuracies for the same window length. This is probably valid for forecasting of well-separated signal of finite rank only, see Remark 4.1.
- In MSSA, the recommendations for the choice of the window length (e.g., "take L larger (or smaller) than the half of the time series length") for recurrent forecasting are in a sense opposite to that for the vector forecasting.
- For the row and column forecasting (1D-SSA and CSSA forecasting methods are particular cases of the column forecasting), the recommendations are also opposite. This is not surprising since L and K have swapped places in MSSA relative to 1D-SSA and CSSA.

Fragment 4.3.2 demonstrates how the Rssa package allows estimation of the reconstruction and forecast accuracy on the example of MSSA and CSSA analysis and vector forecasting applied to Example A with $R = 10$ replications. Note that the numbers in Tables 4.2 and 4.3 were obtained by another complicated code, where $R = 10000$ (see the replicated code to Golyandina et al. (2015)).

Fragment 4.3.2 (Simulation for Accuracy Estimation)

```
> N <- 71
> sigma <- 5
> Ls <- c(12, 24, 36, 48, 60)
> len <- 24
> signal1 <- 30 * cos(2*pi * (1:(N + len)) / 12)
> signal2 <- 30 * cos(2*pi * (1:(N + len)) / 12 + pi / 4)
> signal <- cbind(signal1, signal2)
> R <- 10
> mssa.errors <- function(Ls) {
+   f1 <- signal1[1:N] + rnorm(N, sd = sigma)
+   f2 <- signal2[1:N] + rnorm(N, sd = sigma)
+   f <- cbind(f1, f2)
+   err.rec <- numeric(length(Ls)); names(err.rec) <- Ls
+   err.for <- numeric(length(Ls)); names(err.for) <- Ls
+   for (l in seq_along(Ls)) {
+     L <- Ls[l]
+     s <- ssa(f, L = L, kind = "mssa")
+     rec <- reconstruct(s, groups = list(1:2))[[1]]
+     err.rec[l] <- mean((rec - signal[1:N, ])^2)
+     pred <- vforecast(s, groups = list(1:2), direction = "row",
+                       len = len, drop = TRUE)
```

```
+     err.for[l] <- mean((pred - signal[-(1:N), ])^2)
+   }
+   list(Reconstruction = err.rec, Forecast = err.for)
+ }
> mres <- replicate(R, mssa.errors(Ls))
> err.rec <- rowMeans(simplify2array(mres["Reconstruction", ]))
> err.for <- rowMeans(simplify2array(mres["Forecast", ]))
> print(err.rec)
      12       24       36       48       60
2.869683 1.587789 1.248881 1.153730 1.855115
> print(err.for)
      12       24       36       48       60
2.671251 2.578059 1.501565 2.595378 4.564218
> signal <- signal1 + 1i*signal2
> cssa.errors <- function(Ls) {
+   f1 <- signal1[1:N] + rnorm(N, sd = sigma)
+   f2 <- signal2[1:N] + rnorm(N, sd = sigma)
+   f <- f1 + 1i*f2
+   err.rec <- numeric(length(Ls)); names(err.rec) <- Ls
+   err.for <- numeric(length(Ls)); names(err.for) <- Ls
+
+   for (l in seq_along(Ls)) {
+     L <- Ls[l]
+     s <- ssa(f, L = L, kind = "cssa", svd.method = "svd")
+     rec <- reconstruct(s, groups = list(1:2))[[1]]
+     err.rec[l] <- mean(abs(rec - signal[1:N])^2)
+     pred <- vforecast(s, groups = list(1:2), len = len,
+                       drop = TRUE)
+     err.for[l] <- mean(abs(pred - signal[-(1:N)])^2)
+   }
+   list(Reconstruction = err.rec, Forecast = err.for)
+ }
> cres <- replicate(R, cssa.errors(Ls))
> err.rec <- rowMeans(simplify2array(cres["Reconstruction", ]))
> err.for <- rowMeans(simplify2array(cres["Forecast", ]))
> print(err.rec)
      12       24       36       48       60
7.349316 4.298144 4.101666 4.298144 7.349316
> print(err.for)
      12       24       36       48       60
24.67425 13.60116 14.54819 11.72135 15.86380
```

4.3.4 Other Subspace-Based MSSA Extensions

In view of the common structure of all SSA-family algorithms (see Sect. 1.1), many SSA-related techniques can be naturally extended from 1D objects (i.e., series) to other objects, and particularly to the systems of series.

In Sect. 4.3.1, we have considered methods of MSSA forecasting. Let us now describe some extensions. Since the algorithms and the codes are either exactly or almost identical to the 1D case, we are not writing them out. In particular, all methods and algorithms that are based on the use of the column subspaces are exactly the same in MSSA and 1D-SSA. For example, parameter estimation based on the column subspace can be performed using the same function `parestimate`.

The shaped version of MSSA is almost the same as Shaped 1D-SSA (Sect. 2.6) except for the following difference. If `NA` are placed at the ends of the time series, then the corresponding series is considered as a series of smaller length. If there is no other missing data, then the resultant series of smaller length does not have any missing values and is decomposed by a non-shaped version of MSSA. It is important to mention that if there are `NA` at the right end of a series, then forecasts start from the last not-NA values.

Formally, forecasting can be applied to shaped `ssa` object. However, it is generally recommended to fill gaps first and then forecast the series.

Iterative gap-filling (Sect. 3.3.3, the function `igapfill`) and low-rank approximation by Cadzow iterations (Sect. 3.4, the function `cadzow`) are implemented in a general manner and therefore can be applied to systems of time series with gaps in exactly the same manner as in the 1D case.

Subspace-based gap-filling (Sect. 3.3.3, the function `gapfill`) is implemented on the base of the recurrent column forecasting only and therefore does not have the parameters `direction` and `method`.

4.4 Case Studies

4.4.1 Analysis of Series in Different Scales (Normalization)

Assume that different time series in the system of series are measured in different scales. In statistics, this problem is typically resolved by making a standardization of the data. In SSA, centering may not be an appropriate preprocessing. Therefore, two types of preprocessing can be applied, conventional standardization and normalization, which is the division by the square root of the mean sum of squares. The normalization can be more appropriate for positive series, since it changes only the scale of data.

Let us consider fortified (“FORT”) and rosé (“ROSE”) wine sales from the dataset “AustralianWine.” Sales of fortified wines are measured in thousands but sales of rosé wines are measured in tens and hundreds. Fragment 4.4.1 shows how the scale influences the reconstruction result.

Fragment 4.4.1 ("FORT" and "ROSE": Influence of Series Scales)

```
> wineFortRose <- wine[, c("Fortified", "Rose")]
> summary(wineFortRose)
   Fortified          Rose
 Min.   :1154   Min.   : 30.00
 1st Qu.:2372   1st Qu.: 66.00
 Median :2898   Median : 87.00
 Mean   :3010   Mean   : 93.01
 3rd Qu.:3565   3rd Qu.:114.25
 Max.   :5618   Max.   :267.00
> norm.wineFortRosen <- sqrt(colMeans(wineFortRose^2))
> wineFortRosen <-
+   sweep(wineFortRose, 2, norm.wineFortRosen, "/")
> L <- 84
> s.wineFortRosen <- ssa(wineFortRosen, L = L, kind = "mssa")
> r.wineFortRosen <- reconstruct(s.wineFortRosen,
+                       groups = list(Trend = c(1, 12, 14),
+                                     Seasonality = c(2:11, 13)))
> s.wineFortRose <- ssa(wineFortRose, L = L, kind = "mssa")
> r.wineFortRose <- reconstruct(s.wineFortRose,
+                       groups = list(Trend = 1,
+                                     Seasonality = 2:11))
> wrap.plot <- function(rec, component = 1, series,
+                       xlab = "", ylab, ...)
+   plot(rec, add.residuals = FALSE, add.original = TRUE,
+        plot.method = "xyplot", superpose = TRUE,
+        scales = list(y = list(tick.number = 3)),
+        slice = list(component = component, series = series),
+        xlab = xlab, ylab = ylab, auto.key = "", ...)
> trel1 <- wrap.plot(r.wineFortRosen, series = 2,
+                    ylab = "Rose, norm", main = NULL)
> trel2 <- wrap.plot(r.wineFortRosen, series = 1,
+                    ylab = "Fort, norm", main = NULL)
> trel3 <- wrap.plot(r.wineFortRose, series = 2,
+                    ylab = "Rose", main = NULL)
> trel4 <- wrap.plot(r.wineFortRose, series = 1,
+                    ylab = "Fort", main = NULL)
> plot(trel1, split = c(1, 1, 2, 2), more = TRUE)
> plot(trel2, split = c(1, 2, 2, 2), more = TRUE)
> plot(trel3, split = c(2, 1, 2, 2), more = TRUE)
> plot(trel4, split = c(2, 2, 2, 2))
```

Figure 4.8 demonstrates the result of a trend reconstruction, where the trend was detected in the same way as before; that is, by using the forms of eigenvectors and weighted correlations. The trend of the "ROSE" series is more complicated. However, "FORT" overweighs the decomposition and the eigentriples that refine the "ROSE" trend have very small weight and mix with the common noise. Therefore, the MSSA processing with no normalization is worse for the analysis of the series ROSE, which is measured on a smaller scale.

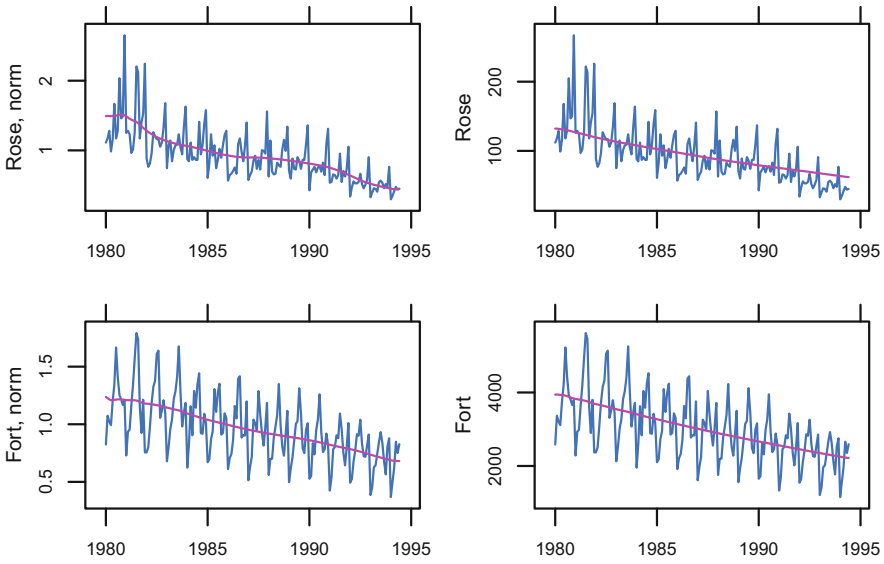

Fig. 4.8 "FORT" and "ROSE": Trends with normalization (ET1,12,14) and without (ET1)

4.4.2 Forecasting of Series with Different Lengths and Filling-In

Fragment 4.3.1 in Sect. 4.3.3.2 shows a typical code when MSSA is used for the series of equal lengths. However, the same code can be applied to series which have different lengths. The full set of "AustralianWine" data has missing values: there is no data for two months (points 175 and 176) for sales of "ROSE" and there is no data for the last 11 months of "Total" sales.

Let us perform the following actions: (A) fill-in missing data in ROSE, (B) calculate the sum of sales of the wines presented in the data, and (C) forecast this sum together with the total series in order to fill-in the missing data.

To use the analysis performed above, let us process the "ROSE" series together with the "FORT" series for (A). Fragment 4.4.2 implements (A). We fill-in the missing data by two methods. The result is presented in Fig. 4.9.

Fragment 4.4.2 ("FORT" and "ROSE": Filling-in the Missing Data in "ROSE")

```
> wineFortRose <- AustralianWine[, c("Fortified", "Rose")]
> L <- 84
> wineFortRose <- AustralianWine[, c("Fortified", "Rose")]
> norm.wineFortRosen <-
+   sqrt(colMeans(wineFortRose^2, na.rm = TRUE))
> wineFortRosen <-
+   sweep(wineFortRose, 2, norm.wineFortRosen, "/")
```

```
> s.wineFortRosen <- ssa(wineFortRosen, L = L, kind = "mssa")
> g.wineFortRosen <-
+   gapfill(s.wineFortRosen, groups = list(1:14))
> ig.wineFortRosen <-
+   igapfill(s.wineFortRosen, groups = list(1:14))
> ig.wineFortRose <-
+   norm.wineFortRosen["Rose"] * ig.wineFortRosen
> g.wineFortRose <-
+   norm.wineFortRosen["Rose"] * g.wineFortRosen
> xyplot(AustralianWine[100:187, "Rose"] +
+          ig.wineFortRose[100:187, "Rose"] +
+          g.wineFortRose[100:187, "Rose"] ~
+          time(AustralianWine)[100:187],
+        type = "l", xlab = "Time", ylab = "Rose",
+        lty = c(1, 2, 1), lwd = c(2, 1, 1),
+        auto.key = list(text = c("`Rose'",
+                                 "Iterative gap filling",
+                                 "Subspace-based gap-filling")))
```

Fragment 4.4.3 implements (B) and (C). Since the series "Total" and "Mainsales" are of different lengths (recall that NA at the end of series correspond to the reduction of the series length), the forecasts correspond to different times (Fig. 4.10). If one is only interested in filling-in the missing data in "Total," then the forecasted values of "Mainsales" can be ignored.

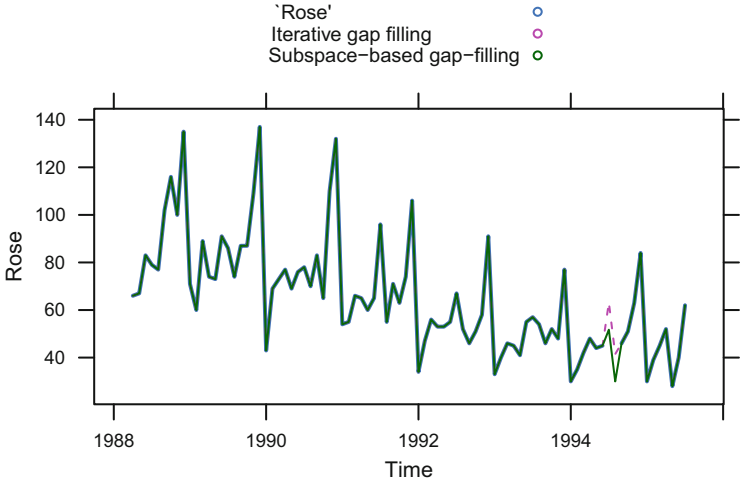

Fig. 4.9 "FORT" and "ROSE": Filling-in of "ROSE" by two methods

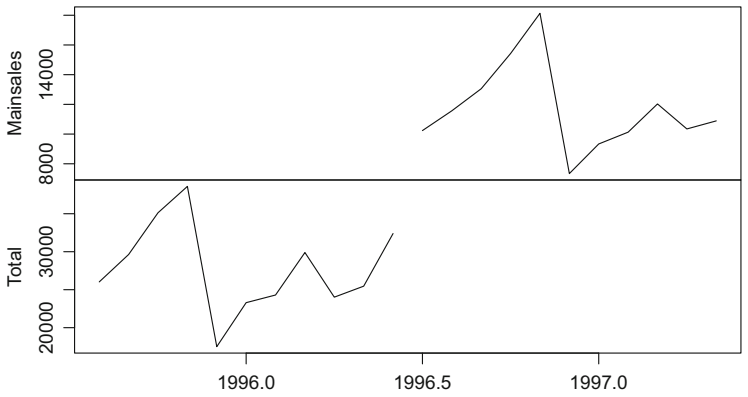

Fig. 4.10 "Total" and "Mainsales": Forecast to fill-in "Total"

Fragment 4.4.3 ("Total" and "Mainsales": Forecast to Fill-in "Total")

```
> FilledRose <- AustralianWine
> FilledRose[175:176, "Rose"] <- g.wineFortRose[175:176]
> mainsales <- ts(rowSums(FilledRose[, -1]))
> tsp(mainsales) <- tsp(AustralianWine)
> wine.add.mainsales <- cbind(FilledRose, mainsales)
> colnames(wine.add.mainsales) <-
+   c(colnames(FilledRose), "Mainsales")
> L <- 84
> s.totalmain <- ssa(wine.add.mainsales[, c("Mainsales",
+                                           "Total")],
+                    L = L, kind = "mssa")
> f.totalmain <- rforecast(s.totalmain, groups = list(1:14),
+                          len = 11, only.new = TRUE)
> plot(f.totalmain, main = "", xlab = NULL, oma = c(3, 1, 1, 1))
```

4.4.3 Simultaneous Decomposition of Many Series

In this example, we consider the system of many time series and show that the decomposition by MSSA helps to look at similar patterns in the series.

Let us consider a collection of $s = 6$ series from the "AustralianWine" dataset, which includes the series of wine sales considered in Fragment 4.2.1, see Sect. 4.2.6.2. A considerable part of this multivariate series can be described as seasonality. Therefore, MSSA can have an advantage over conventional 1D-SSA applied separately to each series from the system.

Since the time series have different scales, it may be advantageous to transform the time series to the same scale by normalizing them. We choose the window length $L = 163$, then $K_i = 12$ $(i = 1, \ldots, 6)$ and $K = 6 \cdot 12 = 72$ and therefore the

number of elementary components is equal to $72 = \min(163, 72)$. This choice of window length does not correspond to the maximal possible number of elementary components.

We can obtain a more detailed decomposition with $144 = \min(151, 144)$ elementary components if we choose $L = 151$ and $K_i = 24$ but in this case elementary components appear with almost equal weights (implying a lack of strong separability). Thus, the choice $L = 163$ and $K_i = 12$ helps to avoid mixing up of the components.

The identification of the trend (ET1,2,5) and seasonality (ET3,4, 6–12) is performed on the base of eigenvectors and uses the principles used in the typical code from Sect. 4.2.6.2. Fragment 4.4.4 contains the code which gives the reconstruction shown in Figs. 4.11 and 4.12.

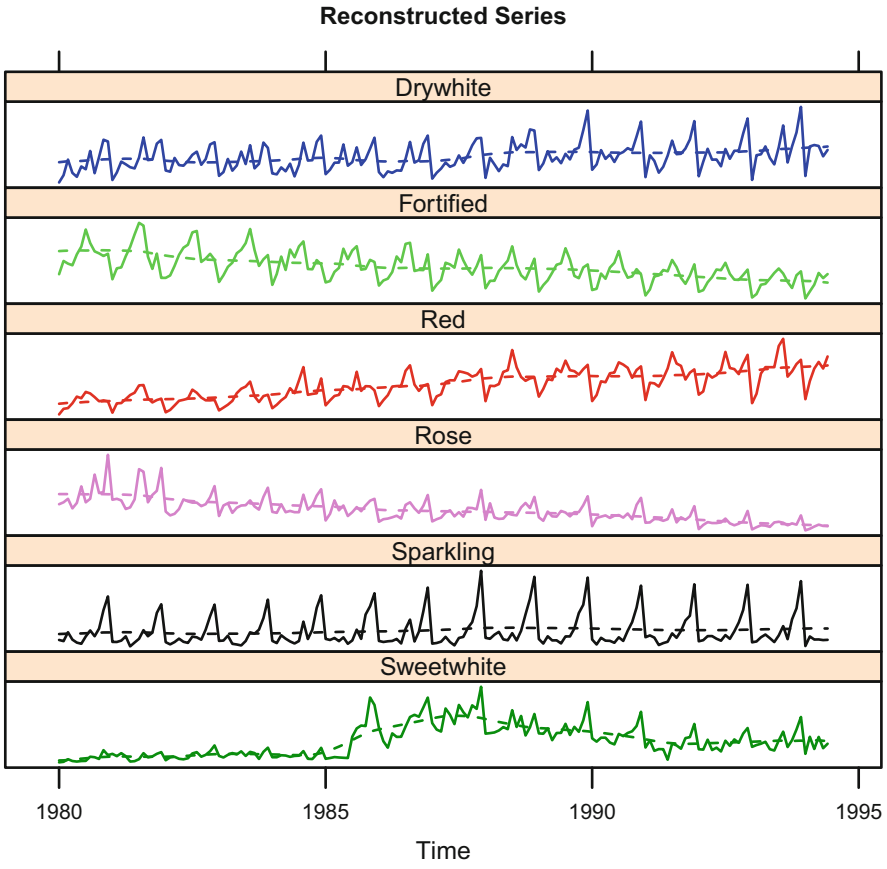

Fig. 4.11 "Australian wines": Extraction of trends

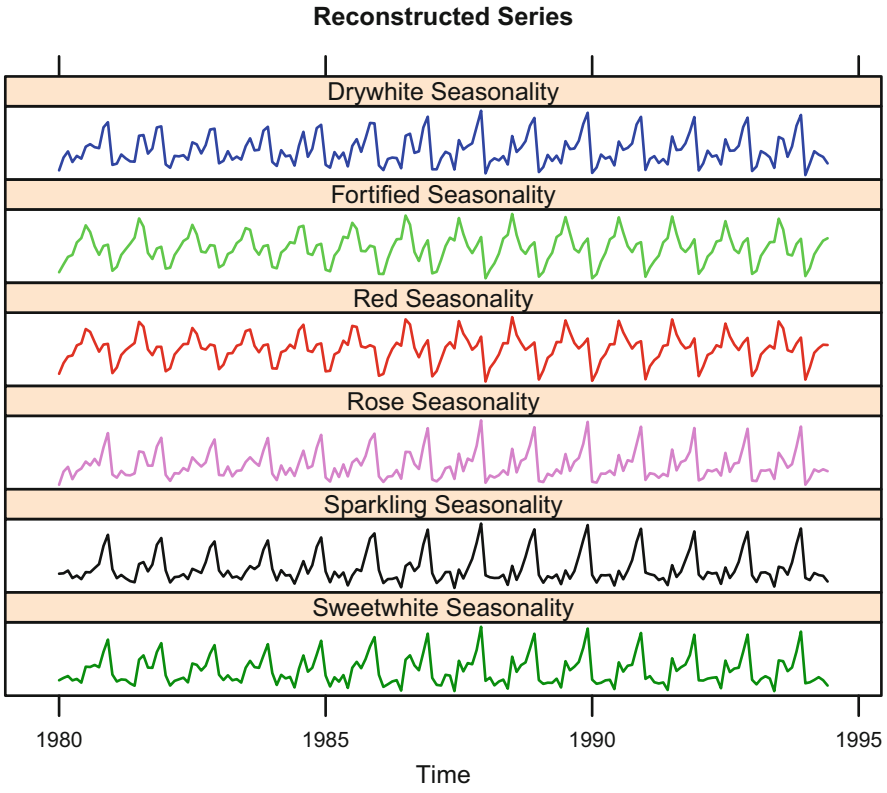

Fig. 4.12 "Australian wines": Extraction of seasonality

Fragment 4.4.4 ("Australian wines": Simultaneous Decomposition by MSSA)

```
> L <- 163
> norm.wine <- sqrt(colMeans(wine[, -1]^2))
> winen <- sweep(wine[, -1], 2, norm.wine, "/")
> s.winen <- ssa(winen, L = L, kind = "mssa")
> r.winen <- reconstruct(s.winen,
+                        groups = list(Trend = c(1, 2, 5),
+                                      Seasonality = c(3:4, 6:12)))
> plot(r.winen, add.residuals = FALSE,
+      plot.method = "xyplot",
+      slice = list(component = 1),
+      screens = list(colnames(winen)),
+      col =
+        c("blue", "green", "red", "violet", "black", "green4"),
+      lty = rep(c(1, 2), each = 6),
+      scales = list(y = list(draw = FALSE)),
+      layout = c(1, 6))
> plot(r.winen, plot.method = "xyplot", add.original = FALSE,
+      add.residuals = FALSE, slice = list(component = 2),
```

```
+       col =
+         c("blue", "green", "red", "violet", "black", "green4"),
+       scales = list(y = list(draw = FALSE)),
+       layout = c(1, 6))
```

The reconstructed trends and seasonal components look adequate. In addition, the simultaneous processing of several time series is very convenient as we obtain similar time series components all at once. In particular, it is clearly seen from Fig. 4.12 that the sales of fortified wines are maximal in June-July (that are winter months in Australia), while the sales of sparkling wines are largest in December.

References

Allen RM, Robertson WA (1996) Distinguishing modulated oscillations from coloured noise in multivariate datasets. Clim Dynam 12(11):775–784

Broomhead D, King G (1986) Extracting qualitative dynamics from experimental data. Physica D 20:217–236

Broomhead D, King G (1986b) On the qualitative analysis of experimental dynamical systems. In: Sarkar S (ed) Nonlinear phenomena and chaos. Adam Hilger, Bristol, pp 113–144

Golyandina N (2010) On the choice of parameters in singular spectrum analysis and related subspace-based methods. Stat Interface 3(3):259–279

Golyandina N, Korobeynikov A (2013) Basic singular spectrum analysis and forecasting with R. Comput Stat Data Anal 71:943–954

Golyandina N, Osipov E (2007) The "Caterpillar"-SSA method for analysis of time series with missing values. J Stat Plan Inference 137(8):2642–2653

Golyandina N, Stepanov D (2005) SSA-based approaches to analysis and forecast of multidimensional time series. In: Proceedings of the 5th St.Petersburg workshop on simulation, June 26–July 2, 2005. St. Petersburg State University, St. Petersburg, pp 293–298

Golyandina N, Zhigljavsky A (2013) Singular spectrum analysis for time series. Springer briefs in statistics. Springer

Golyandina N, Nekrutkin V, Zhigljavsky A (2001) Analysis of time series structure: SSA and related techniques. Chapman&Hall/CRC

Golyandina N, Korobeynikov A, Shlemov A, Usevich K (2015) Multivariate and 2D extensions of singular spectrum analysis with the Rssa package. J Stat Softw 67(2):1–78

Hannachi A, Jolliffe IT, Stephenson DB (2007) Empirical orthogonal functions and related techniques in atmospheric science: A review. Int J Climatol 27(9):1119–1152

Hassani H, Mahmoudvand R (2013) Multivariate singular spectrum analysis: a general view and vector forecasting approach. Int J Energy Stat 01(01):55–83

Keppenne C, Lall U (1996) Complex singular spectrum analysis and multivariate adaptive regression splines applied to forecasting the southern oscillation. In: Exp. long-lead forcst. Bull

Trickett SR (2003) F-xy eigenimage noise suppression. Geophysics 68(2):751–759

Weare BC, Nasstrom JS (1982) Examples of extended empirical orthogonal function analyses. Mon Weather Rev 110(6):481–485

Chapter 5
Image Processing

This chapter is devoted to extensions of 1D-SSA (Chaps. 2 and 3) and MSSA (Chap. 4) for the analysis of objects of dimension 2 and larger. The 2D case corresponds to the digital image processing. The objects with larger dimensions are also widely used. For example, a color image can be considered as a system of 2D images and its analysis can be performed by multivariate 2D-SSA, which is an extension of MSSA designed for analyzing a system of series. The third temporal dimension naturally arises if images are changing in time, which is a typical data in climatology (Hannachi et al. 2007). The data can be 3D, if the measurements are performed at points located in a 3D area, see, e.g., Shlemov et al. (2015a) for a quantitative analysis of a gene expression data. A system of 3D data changing in time can also be considered. The list of extensions can be continued; see, e.g., Oropeza (2010) for high-dimensional SSA algorithms applied to specific problems in seismology.

The scheme of SSA-family methods of Sect. 1.1 can be naturally applied for multidimensional objects of any dimension since the difference is in the embedding operator only. Moreover, the SSA analysis of an object of any dimension and shape is a particular case of the shaped version of SSA, where a window of arbitrary shape goes through an object of arbitrary shape and forms a trajectory matrix consisting of vectorized lagged windows.

The Rssa package implements the so-called nD-SSA for analysis of objects of arbitrary dimensions, in rectangular and shaped versions. In particular, 2D or 3D images, which are not necessarily rectangular and are changing in time (that is, have an additional temporary dimension) can be analyzed. We start this chapter with a description of the 2D-SSA analysis (Sect. 5.1) thus avoiding details related to shapes (these details can be quite complicated). Section 5.2 extends the material of Sect. 5.1 to arbitrary shapes. The algorithms presented in Sect. 5.2 for 2D-shapes could be easily reformulated for objects of any dimension. The same is true for 2D extensions of the ESPRIT method for parameter estimation described in Sect. 5.3.1.

N. Golyandina et al., *Singular Spectrum Analysis with R*, Use R!,
https://doi.org/10.1007/978-3-662-57380-8_5

We do not formally write the algorithms for dimensions larger than two, since they can be obtained by a formal extension of the algorithms from Sects. 5.2 and 5.3. A demonstration of the RSSA code for performing nD-SSA can be found in Sect. 5.4.4.

5.1 2D-SSA

In this section, we consider an extension of the 1D-SSA algorithm for decomposition of two-dimensional data. This extension has the name *2D singular spectrum analysis* (or *2D-SSA* for short). For 2D-SSA, the data object $\mathbb{X}$ is a *two-dimensional data array* of size $N_x \times N_y$ (or simply an $N_x \times N_y$ real-valued matrix), represented as $\mathbb{X} = \mathbb{X}_{N_x,N_y} = (x_{ij})_{i,j=1}^{N_x,N_y}$. A typical example of a 2D-array is a digital 2D monochrome image.

2D-SSA was proposed as an extension of 1D-SSA in Danilov and Zhigljavsky (1997), and was further developed in Golyandina and Usevich (2010), Rodríguez-Aragón and Zhigljavsky (2010). However, related decompositions were developed independently in texture analysis (Ade 1983; Monadjemi 2004), seismic data processing (Trickett 2008; Oropeza 2010), and parameter estimation methods for sums of two-dimensional complex exponentials, see, e.g., Rouquette and Najim (2001).

Until recently, the major drawback of the methods based on 2D-SSA decomposition was its computational complexity. The RSSA package contains an efficient implementation of the 2D-SSA decomposition and reconstruction, which to a great extent overcomes this deficiency (Golyandina et al. 2015).

5.1.1 Method

We mostly use the notation from Golyandina and Usevich (2010) and Golyandina et al. (2015). For a matrix $\mathbf{A} \in \mathsf{R}^{M\times N}$ (or $\mathsf{C}^{M\times N}$), we denote by $\mathrm{vec}(\mathbf{A}) \in \mathsf{R}^{MN}$ (or C^{MN}) its column-major vectorization. For a vector $A \in \mathsf{R}^{MN}$ (or C^{MN}), we define its M *devectorization* as the matrix $\mathrm{vec}_M^{-1}(A) = \mathbf{B} \in \mathsf{R}^{M\times N}$ (or $\mathsf{C}^{M\times N}$) that satisfies $\mathrm{vec}(\mathbf{B}) = A$.

5.1.1.1 The Embedding Operator

The generic scheme of the SSA algorithm is described in Sect. 1.1. Hence, to formally present 2D-SSA, we only need to define the embedding operator $\mathcal{T}_{\text{2D-SSA}}(\mathbb{X}) = \mathbf{X}$.

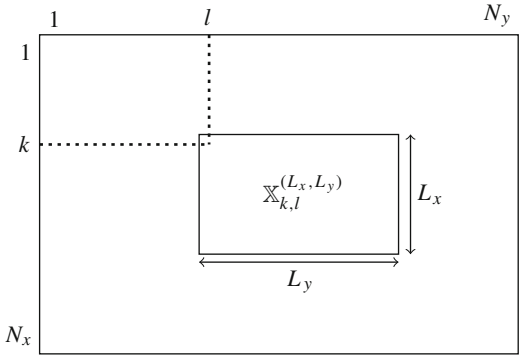

Fig. 5.1 Moving 2D windows

The parameters of the method are the two-dimensional *window sizes* (L_x, L_y), with restrictions $1 \le L_x \le N_x$, $1 \le L_y \le N_y$ and $1 < L_x L_y < N_x N_y$. For convenience, we also denote $K_x = N_x - L_x + 1$, $K_y = N_y - L_y + 1$. As in the general scheme of the algorithms, we define $L = L_x L_y$ (the number of rows of $\mathbf{X}$) and $K = K_x K_y$ (the number of columns of $\mathbf{X}$).

Consider all possible $L_x \times L_y$ submatrices of $\mathbb{X}$ (2D sliding windows). For $k = 1, \ldots, K_x$ and $l = 1, \ldots, K_y$, we define by $\mathbb{X}_{k,l}^{(L_x,L_y)} = (x_{ij})_{i=k,\,j=l}^{L_x+k-1,L_y+l-1}$ the submatrix of size $L_x \times L_y$ shown in Fig. 5.1. Note that the x axis is oriented to the bottom, and the y axis is oriented to the right; the origin is the upper left corner. We use this orientation as it is consistent with the standard mathematical indexing of matrices (Golyandina and Usevich 2010).

Then the trajectory matrix is defined as

$$\mathcal{T}_{\text{2D-SSA}}(\mathbb{X}) = \mathbf{X} = [X_1 : \ldots : X_{K_x K_y}], \tag{5.1}$$

where the columns X_j are vectorizations of the $L_x \times L_y$ submatrices:

$$X_{k+(l-1)K_x} = \operatorname{vec}\left(\mathbb{X}_{k,l}^{(L_x,L_y)}\right).$$

5.1.1.2 Hankel-Block-Hankel Structure

The trajectory matrix (5.1) has the following structure (Golyandina and Usevich 2010):

$$\mathbf{X} = \mathcal{T}_{\text{2D-SSA}}(\mathbb{X}) = \begin{pmatrix} \mathbf{H}_1 & \mathbf{H}_2 & \mathbf{H}_3 & \ldots & \mathbf{H}_{K_y} \\ \mathbf{H}_2 & \mathbf{H}_3 & \mathbf{H}_4 & \ldots & \mathbf{H}_{K_y+1} \\ \mathbf{H}_3 & \mathbf{H}_4 & \iddots & \iddots & \vdots \\ \vdots & \vdots & \iddots & \iddots & \vdots \\ \mathbf{H}_{L_y} & \mathbf{H}_{L_y+1} & \ldots & \ldots & \mathbf{H}_{N_y} \end{pmatrix}, \tag{5.2}$$

where each $\mathbf{H}_j$ is an $L_x \times K_x$ Hankel matrix constructed from $\mathbb{X}_{:,j}$ (the jth column of the 2D array $\mathbb{X}$). More precisely, $\mathbf{H}_j = \mathcal{T}_{\mathrm{SSA}}(\mathbb{X}_{:,j})$, where $\mathcal{T}_{\mathrm{SSA}}$ is defined in (1.1). The matrix (5.1) is called *Hankel-block-Hankel* (shortened to *HbH*), since it is block-Hankel with Hankel blocks. Thus, $\mathcal{M}^{(\mathrm{H})}_{L,K}$ in 2D-SSA is the set of HbH matrices.

Projection Π_{HbH} on the space of HbH matrices can be performed using the general form described in Sect. 1.1.2.6. In this particular case, the projection consists of hankelization of each matrix block at the places of each $\mathbf{H}_j$ and then averaging of the blocks at the places of $\mathbf{H}_j$ with the same index j.

5.1.1.3 Trajectory Space

From (5.1), we observe that the trajectory space is the linear space spanned by the $L_x \times L_y$ submatrices of $\mathbb{X}$. Therefore, the eigenvectors U_i can also be viewed as vectorized $L_x \times L_y$ arrays. Their devectorizations are denoted by $\Psi_i = \mathrm{vec}^{-1}_{L_x}(U_i)$. Similarly, the rows of $\mathbf{X}$ are vectorizations of the (K_x, K_y) submatrices

$$\mathbf{X} = [X^1 : \ldots : X^{L_x L_y}]^{\mathrm{T}}, \quad X^{k+(l-1)L_x} = \mathrm{vec}(\mathbb{X}^{(K_x,K_y)}_{k,l}), \tag{5.3}$$

where X^j is the jth row of the matrix $\mathbb{X}$. The factor vectors V_i can also be viewed as $K_x \times K_y$ arrays. Their devectorizations are denoted by $\Phi_i = \mathrm{vec}^{-1}_{K_x}(V_i)$.

5.1.1.4 Comments

1. The algorithm of 2D-SSA coincides with the algorithm of MSSA for time series of the same length when $L_x = 1$ or $L_y = 1$ (Golyandina and Usevich 2010). This observation will be used in Sect. 5.2.
2. The arrays $\mathbb{X}$ of finite rank in 2D-SSA (i.e., the arrays such that $\mathcal{T}_{\text{2D-SSA}}(\mathbb{X})$ is rank-deficient and has a fixed rank) are sums of products of polynomials, exponentials and cosines, similarly to the one-dimensional case. More details can be found in Sect. 5.1.2.
3. The generic scheme of SSA described in Sect. 1.1 includes a decomposition of the trajectory matrix into a sum of rank-one matrices. If this decomposition is performed by means of the SVD, then we call the method Basic 2D-SSA or simply 2D-SSA. Other versions, such as Shaped 2D-SSA, nested Iterative Oblique 2D-SSA or 2D-SSA with projection, can be also considered.

5.1.2 Elements of 2D-SSA Theory

In the next two sections, we give a short summary of the theory of arrays of finite rank; for detail we refer to Golyandina and Usevich (2009).

5.1.2.1 Arrays of Finite Rank

Consider the class of infinite arrays $\mathbb{X} = (x_{mn})_{m,n=1}^{\infty,\infty}$ given in the parametric form:

$$x_{mn} = \sum_{k=1}^{r} c_k \mu_k^m \nu_k^n, \tag{5.4}$$

where $(\mu_k, \nu_k) \in \mathsf{C}^2$ are distinct pairs of complex numbers and c_k are non-zero. It can be shown that for large enough N_x, N_y, L_x, L_y, the rank of the trajectory matrix $\mathbf{X}$ is equal to r (or equivalently, the arrays of the form (5.4) are *arrays of finite rank*). Note that the exponentials can be also represented in the form

$$\mu_k = \rho_{x,k} \exp(2\pi \mathrm{i} \omega_{x,k}), \quad \nu_k = \rho_{y,k} \exp(2\pi \mathrm{i} \omega_{y,k}), \quad -0.5 < \omega_{y,k} \le 0.5.$$

The class of arrays of finite rank also contains bivariate polynomials and their products with the functions from (5.4), see (1.10). The complete algebraic characterization of infinite arrays of finite rank can be found in Golyandina and Usevich (2009). Similar to the 1D case, we can call (μ_k, ν_k) the characteristic roots.

5.1.2.2 Real Arrays

Now let us summarize what happens in the case of real arrays $\mathbb{X}$. If all elements x_{mn} are real, then for any pair $(\mu, \nu) = (\mu_k, \nu_k) \in \mathsf{C}^2$ in (5.4) its complex conjugate $(\mathrm{conj}(\mu), \mathrm{conj}(\nu))$ is also presented in the right-hand side of (5.4). This is due to the fact that $x_{m,n} = \frac{x_{m,n} + \overline{x}_{m,n}}{2}$ and that the infinite arrays $\mu^m \nu^n$ are linearly independent. The corresponding coefficients associated with (μ, ν) and $(\mathrm{conj}(\mu), \mathrm{conj}(\nu))$ are also complex conjugate. We order the roots so that (μ_k, ν_k) are real for $1 \le k \le d$, and the other $2s$ roots consist of complex conjugate pairs and are arranged so that $(\mu_k, \nu_k) = (\mathrm{conj}(\mu_{k-s}), \mathrm{conj}(\nu_{k-s}))$ for $d + s + 1 \le k \le d + 2s$. Then these roots have the representation

$$(\mu_k, \nu_k) = \begin{cases} (\rho_{x,k}, \rho_{y,k}) = \left(\rho_{y,k} \exp(2\pi \mathrm{i} \omega_{x,k}), \rho_{y,k} \exp(2\pi \mathrm{i} \omega_{y,k})\right), & 1 \le k \le d, \\ \left(\rho_{x,k} \exp(2\pi \mathrm{i} \omega_{x,k}), \rho_{y,k} \exp(2\pi \mathrm{i} \omega_{y,k})\right), & d+1 \le k \le d+s, \\ \left(\rho_{x,k-s} \exp(-2\pi \mathrm{i} \omega_{x,k-s}), \rho_{y,k} \exp(-2\pi \mathrm{i} \omega_{y,k-s})\right), & d+s+1 \le k \le r, \end{cases}$$

where $(\rho_{x,k}, \rho_{y,k}, \omega_{x,k}, \omega_{y,k}) \in \mathsf{R}^2 \times [0; 1/2)^2$ are distinct 4-tuples of real numbers such that $\omega_{x,k} = \omega_{y,k} = 0$ for $1 \le k \le d$.

Then the representation (5.4) becomes a sum of $d + s$ planar modulated sinewaves:

$$x_{mn} = \sum_{k=1}^{d+s} b_k \rho_{x,k}^m \rho_{y,k}^n \cos\left(2\pi(\omega_{x,k}m + \omega_{y,k}n) + \phi_k\right),$$

where b_k and $\phi_k \in [0; 2\pi)$ are unique real coefficients obtained from c_k.

Example 5.1 The product of sines can be uniquely represented as a sum of two planar sines:

$$2\cos(2\pi\omega_x m + \phi_1)\cos(2\pi\omega_y n + \phi_2) =$$
$$\cos(2\pi(\omega_x m + \omega_y n) + \phi_1 + \phi_2) + \cos(2\pi(\omega_x m - \omega_y n) + \phi_1 - \phi_2).$$

5.1.3 Algorithm

The algorithm of 2D-SSA decomposition, as well as 1D-SSA and MSSA, is a particular case of the generic scheme described in Sect. 1.1.1. Let us write down the algorithm of the basic version of 2D-SSA, which is characterized by the use of the SVD for performing the rank-one matrix decomposition.

Algorithm 5.1 Basic 2D-SSA: decomposition

Input: Image $\mathbb{X}$ of size $N_x \times N_y$; window of size $L_x \times L_y$.
Output: Decomposition of the trajectory matrix of sizes $L_x L_y \times K_x K_y$, where $K_x = N_x - L_x + 1$ and $K_y = N_y - L_y + 1$, on elementary matrices $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.
1: Construct the trajectory matrix $\mathbf{X} = \mathcal{T}_{\text{2D-SSA}}(\mathbb{X})$, where $\mathcal{T}_{\text{2D-SSA}}$ is defined by (5.1) or (5.2).
2: Compute the SVD $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.

Note that the presentation of U_i or V_i in plots is performed in the form of matrices $\Psi_i = \mathrm{vec}_{L_x}^{-1}(U_i)$ and $\Phi_i = \mathrm{vec}_{L_y}^{-1}(V_i)$, which are called eigenarrays and factor arrays, respectively.

Reconstruction stage of 2D-SSA is standard.

Algorithm 5.2 2D-SSA reconstruction

Input: Decomposition $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where $\mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}}$ and $\|U_i\| = \|V_i\| = 1$, grouping $\{1, \ldots, d\} = \bigsqcup_{j=1}^m I_j$.
Output: Decomposition of the image on identifiable components: $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$.
1: Construct the grouped matrix decomposition $\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_m}$, where $\mathbf{X}_I = \sum_{i \in I} \mathbf{X}_i$.
2: $\mathbb{X} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$, where $\mathbb{X}_i = \mathcal{T}_{\text{2D-SSA}}^{-1} \circ \Pi_{\text{HbH}}(\mathbf{X}_{I_i})$.

5.1.4 2D-SSA in RSSA

5.1.4.1 Description of Functions

Let `x` be a digital image in a gray scale, which is given by a real matrix. Then a typical call of the `ssa` function is

```
s <- ssa(x, L = c(30,30), kind = "2d-ssa")
```

Arguments:

`x` is an object to be decomposed. If there are any `NA`'s, then the shaped variant of 2D-SSA (see Sect. 5.2) will be used and all non-`NA` elements will be considered as a mask.

`L` is a 2D-window size; it should be a vector of length 2 for 2D-SSA. Each vector element is set to the half of the corresponding image size by default. Default value can be time-consuming. It is recommended to start with window `c(30,30)` or smaller.

`neig` is the number of desired eigentriples. If `neig = NULL`, a default value which depends on L and N will be used.

`kind` specifies the version of SSA to be used; it can be omitted in non-ambiguous cases (e.g., when `x` is a matrix for the 2D case).

`svd.method` selects the SVD method. Full description is given in Sect. 2.1.5.2.

The function returns the `ssa` object, see details in Sect. 2.1.5.

5.1.4.2 Typical Code

Here we demonstrate a typical decomposition of 2D images with the package RSSA. We follow the same route as in the simple example provided in Sect. 2.1.5.3 and stress on the differences that appear in the 2D case.

As an example, we use the image of Mars by Pierre Thierry, from the tutorial of the free IRIS software. The image is of size 258×275, 8-bit grayscale, values from 0 to 255. The input code for this image can be found in Fragment 5.1.1 (the image is included in the package RSSA).

Fragment 5.1.1 ("Mars": Input)

```
> data("Mars", package = "Rssa")
```

We would like to decompose this image with the window of size 25×25. Easy calculations show that even this rather small window produces a 625×58734 trajectory matrix. In Fragment 5.1.2 with `svd.method = "svd"`, we intentionally comment the call to the SSA function, because we do not recommend to use it, unless the trajectory matrix is very small.

Fragment 5.1.2 ("Mars": Decomposition with `svd.method = "svd"`)

```
> # ssa(Mars, kind = "2d-ssa", L = c(25, 25), svd.method = "svd")
```

A remedy for this could be a calculation of just the matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$, and computing its eigendecomposition. In the package Rssa, this is implemented in `svd.method = "eigen"`, see Fragment 5.1.3.

Fragment 5.1.3 ("Mars": Decomposition with `svd.method = "eigen"`)

```
> print(system.time(ssa(Mars, kind = "2d-ssa", L = c(25, 25),
+                       svd.method = "eigen")))
   user  system elapsed
   5.42    0.53    5.95
```

For larger window sizes this approach quickly becomes impractical because the complexity of the full eigendecomposition grows at least as $O(L^3)$. Therefore, in the Rssa package the method `nutrlan` is used by default. This gives a considerable speed-up even for moderate window sizes (25×25), as demonstrated in Fragment 5.1.4. Note that for the 2D-SSA decomposition, `kind = "2d-ssa"` should be used.

Fragment 5.1.4 ("Mars": Decomposition)

```
> print(system.time(s.Mars.25 <- ssa(Mars, kind = "2d-ssa",
+                                    L = c(25, 25))))
   user  system elapsed
   0.67    0.02    0.68
```

Fragment 5.1.5 shows a typical reconstruction code for 2D-SSA.

Fragment 5.1.5 ("Mars": Reconstruction)

```
> r.Mars.25 <-
+     reconstruct(s.Mars.25,
+                 groups = list(Noise = c(12, 13, 15, 16)))
> plot(r.Mars.25, cuts = 255, layout = c(3, 1))
```

The reconstruction results are shown in Fig. 5.2.

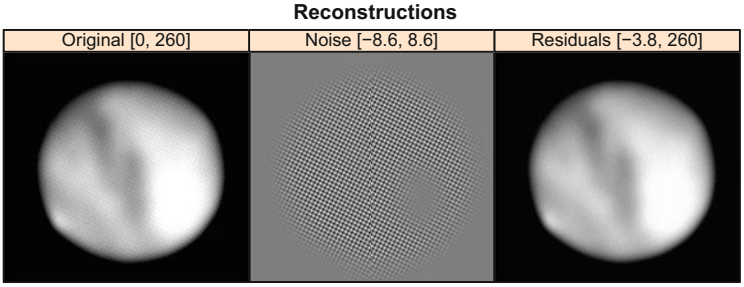

Fig. 5.2 "Mars": Separated periodic noise, $(L_x, L_y) = (25, 25)$

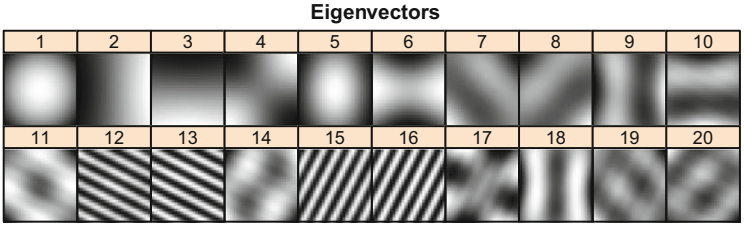

Fig. 5.3 "Mars": Eigenarrays, $(L_x, L_y) = (25, 25)$

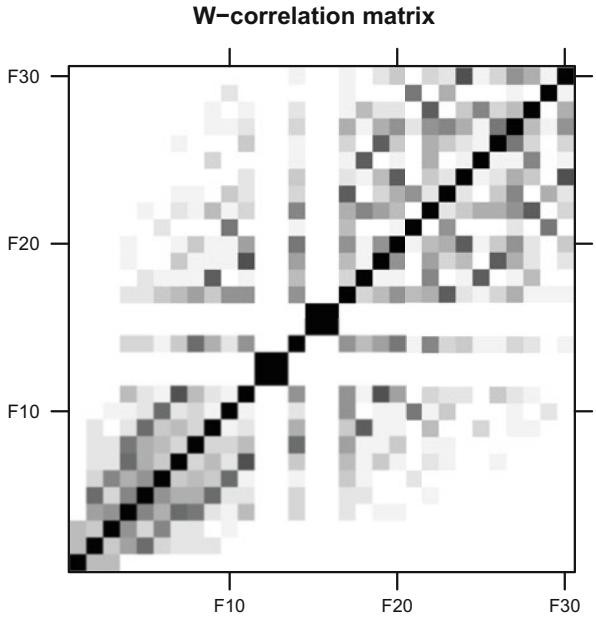

Fig. 5.4 "Mars": **w** Correlations, $(L_x, L_y) = (25, 25)$

The grouping for this decomposition was made on the base of the following information:

- eigenarrays Ψ_i (see Fig. 5.3), and
- the matrix of **w**-correlations, see Fig. 5.4.

Fragment 5.1.6 shows the corresponding code.

Fragment 5.1.6 ("Mars": Identification)

```
> plot(s.Mars.25, type = "vectors", idx = 1:20,
+       cuts = 255, layout = c(10, 2),
+       plot.contrib = FALSE)
> plot(wcor(s.Mars.25, groups = 1:30),
+       scales = list(at = c(10, 20, 30)))
```

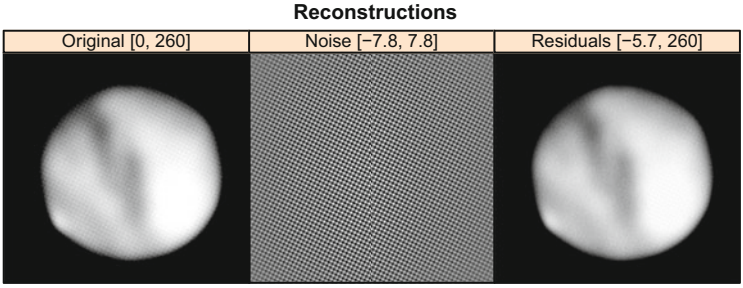

Fig. 5.5 "Mars": Reconstruction, $(L_x, L_y) = (160, 80)$

Let us now try a much more challenging window size $(L_x, L_y) = (160, 80)$. In this case, the trajectory matrix has size 12800×19404 and `svd.method = "eigen"` would take a very long time. However, with the default method `svd.method = "nutrlan"`, the computation of the first 50 eigenvectors can be done in about a second, see Fragment 5.1.7. The results of the reconstruction are shown in Fig. 5.5.

Fragment 5.1.7 ("Mars": Reconstruction)

```
> print(system.time(s.Mars.160.80 <-
+   ssa(Mars, kind = "2d-ssa", L = c(160, 80))))
   user  system elapsed
   0.78    0.03    0.81
> r.Mars.160.80.groups <- list(Noise = c(36, 37, 42, 43))
> r.Mars.160.80 <- reconstruct(s.Mars.160.80,
+                              groups = r.Mars.160.80.groups)
> plot(r.Mars.160.80, cuts = 255, layout = c(3, 1))
```

From Fig. 5.5 we see that in the case of large window sizes, the extracted periodic noise is not modulated (compare to Fig. 5.2). This can be interpreted as follows. We chose the window sizes as (160, 80) (approximately $(0.6N_x, 0.3N_y)$) for which the separability of signal and noise in the parametric model should be better than for small window sizes. On the other hand, if we choose smaller windows (for example, 25×25), then the 2D-SSA decomposition would act more like smoothing.

5.1.4.3 Comments

Formats of Input and Output Data

The input for 2D-SSA is assumed to be a matrix (or an object which can be coerced to a matrix).

Plotting Specifics

By default, all plotting routines use the raster representation (via the argument `useRaster = TRUE` provided to the `LATTICE` plotting functions). In most cases it does not make sense to turn the raster mode off, since the input is a raster image in any case. However, not all graphical devices support this mode.

Efficient Implementation

Most of the ideas from the one-dimensional case can be either transferred directly or generalized to the 2D case. The overall computational complexity of the direct implementation of 2D-SSA is $O(L^3 + KL^2)$ and thus 2D-SSA can be extremely time consuming even for moderate images and window sizes. (Recall that $L = L_x L_y$ and $K = K_x K_y$.) The ideas presented in Golyandina et al. (2015) coupled with Lanczos-based truncated SVD implementations (Larsen 1998; Trickett 2003; Yamazaki et al. 2008; Korobeynikov 2010) allow to dramatically reduce the computational complexity down to $O(kN \log N + k^2 N)$, where $N = N_x N_y$ and k denotes the number of desired eigentriples. Therefore, the achieved speed-up can be much higher than that for the 1D-SSA and MSSA cases.

Note that the Lanczos-based methods have significant overhead for small trajectory matrices, so in this case other SVD methods should be used. For the choice `svd.method="eigen"`, the matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ is computed in $O(LN \log N)$ flops using the fast matrix-vector multiplication (Golyandina et al. 2015). Therefore, the total complexity of the decomposition method is $O(LN \log N + L^3)$, which makes the method applicable for small L and moderate N.

5.2 Shaped 2D-SSA

In its general form, Shaped SSA can be used for analyzing an object given on a grid of dimension $k \geq 1$. The case $k = 1$, which is described in Sect. 2.6, is, to a certain extent, degenerate. For simplicity of notation, we consider the case $k = 2$ as a representative for dimensions $k > 1$.

Shaped 2D-SSA (we will use the same short name *ShSSA* as for the 1D case) is a generalization of 2D-SSA, which allows arbitrary shapes of the input array and of the window (Golyandina et al. 2015). This considerably extends the range of real-life applications of SSA, since it allows to analyze parts of the image with different structures separately, exclude areas with corrupted data, analyze images with gaps, decompose non-rectangular images, etc. In ShSSA, not all values of the rectangular image have to be specified, and the sliding window is not necessarily rectangular. Moreover, there is a circular version of Shaped SSA, when the object can be taken on a circle, a cylinder, or a torus (Shlemov and Golyandina 2014).

5.2.1 Method

5.2.1.1 Shapes and Actions with Them

Formally, we call *shape* $\mathfrak{B}$ a finite non-empty subset of $\{1,\dots,T_x\}\times\{1,\dots,T_y\}$, where $T_x, T_y \in \mathsf{N}\cup\{\infty\}$, N is the set of natural numbers. The values T_x and T_y characterize the topology of the set containing this shape. If $T_x < \infty$ (or $T_y < \infty$), then the topology is circular and the shapes are cyclic with respect to the x-coordinate (or the y-coordinate) with period T_x (respectively, T_y).

A $\mathfrak{B}$-*shaped array* is a partially indexed array $\mathbb{X} = \mathbb{X}_{\mathfrak{B}} = (x_{(i,j)})_{(i,j)\in\mathfrak{B}}$. Let us denote the space of $\mathfrak{B}$-shaped arrays as $\mathsf{R}^{\mathfrak{B}}$. There is an isomorphism $\mathsf{R}^{\mathfrak{B}} \sim \mathsf{R}^{B}$, where $B = |\mathfrak{B}|$ is the cardinality of the set $\mathfrak{B}$. This isomorphism is not unique, since the elements in the shape can be ordered in different ways. However, the result of the algorithm does not depend on the chosen order.

For convenience, we consider the lexicographical order, which fixes the isomorphism

$$\mathcal{J}_{\mathfrak{B}} : \mathsf{R}^{\mathfrak{B}} \mapsto \mathsf{R}^{B}. \tag{5.5}$$

We call $\mathcal{J}_{\mathfrak{B}}$ *vectorization* and its inverse $\mathcal{J}_{\mathfrak{B}}^{-1}$ *shaping*.

Next, we introduce the operation of addition in the considered topology for two pairs of indices $\ell = (\ell_x, \ell_y), \kappa = (\kappa_x, \kappa_y)$. For convenience, we omit the parameters (T_x, T_y) and write $\oplus$ instead of $\oplus^{(T_x,T_y)}$:

$$\ell \oplus \kappa = \big((\ell_x + \kappa_x - 2) \bmod T_x + 1, (\ell_y + \kappa_y - 2) \bmod T_y + 1\big),$$

where mod denotes the remainder in the integer division. Formally, $a \bmod \infty = a$ for any a. Note that -2 and $+1$ in the definition of $\oplus$ is the consequence of the indexing, which starts at 1 (so that $\{1\}\oplus\{1\} = \{1\}$).

For two shapes $\mathfrak{A}$ and $\mathfrak{B}$, we modify the definition of the Minkowski sum in the following way:

$$\mathfrak{A}\oplus\mathfrak{B} = \{\alpha\oplus\beta \mid \alpha\in\mathfrak{A}, \beta\in\mathfrak{B}\}. \tag{5.6}$$

5.2.1.2 Embedding Step

The embedding operator for the Shaped 2D-SSA algorithm is defines as follows:

Input Data and Parameters of the Embedding

The input data are the topology characteristics (T_x, T_y), the shape $\mathfrak{N}$, where $\mathfrak{N}\subset\{1,\dots,T_x\}\times\{1,\dots,T_y\}$, and the $\mathfrak{N}$-shaped array $\mathbb{X}\in\mathsf{R}^{\mathfrak{N}}$. The parameter of the algorithm is a *window shape* $\mathfrak{L}\subset\mathfrak{N}$. It is convenient to consider the window

shape in the form $\mathfrak{L} = \{\ell_1, \ldots, \ell_L\}$, where $L = |\mathfrak{L}|$ and $\ell_i \in \mathsf{N}^2$ are ordered lexicographically.

For each $\kappa \in \mathfrak{N}$, we define a shifted $\mathfrak{L}$-shaped subarray as $\mathbb{X}_{\mathfrak{L}\oplus\{\kappa\}} = (x_\alpha)_{\alpha\in\mathfrak{L}\oplus\{\kappa\}}$. The index κ is a position of the origin for the window. Consider the set of all possible origin positions for the $\mathfrak{L}$-shaped windows:

$$\mathfrak{K} = \{\kappa \in \mathfrak{N} \mid \mathfrak{L} \oplus \{\kappa\} \subset \mathfrak{N}\}.$$

We assume that $\mathfrak{K}$ is written as $\mathfrak{K} = \{\kappa_1, \ldots, \kappa_K\}$, where $K = |\mathfrak{K}| \neq 0$ and κ_j are ordered lexicographically.

If the shapes $\mathfrak{N}$ and $\mathfrak{L}$ are rectangles, then $\mathfrak{K}$ is also rectangular and we call the version of Shaped 2D-SSA *rectangular*. Note that the rectangular version of Shaped 2D-SSA is exactly the ordinary 2D-SSA described in Sect. 5.1.

Embedding Operator

The trajectory matrix $\mathbf{X}$ is constructed by the embedding operator $\mathcal{T}_{\mathrm{ShSSA}} : \mathsf{R}^{\mathfrak{N}} \to \mathsf{R}^{L\times K}$

$$\mathcal{T}(\mathbb{X}) = \mathcal{T}_{\mathrm{ShSSA}}(\mathbb{X}) := \mathbf{X} = [X_1, \ldots, X_K], \tag{5.7}$$

where the columns

$$X_j = (x_{\ell_i\oplus\kappa_j})_{i=1}^{L}$$

are vectorizations of the shaped subarrays $\mathbb{X}_{\mathfrak{L}\oplus\{\kappa_j\}}$.

The embedding operator $\mathcal{T}_{\mathrm{ShSSA}}$ is linear. Denote its range $\mathcal{M}^{(\mathrm{H})}_{L,K}$ as $\mathsf{H}^{\mathfrak{L}\times\mathfrak{K}} \subset \mathsf{R}^{L\times K}$. The subspace $\mathsf{H}^{\mathfrak{L}\times\mathfrak{K}}$ consists of structured matrices, which we will call *quasi-Hankel*. (In fact, they are generalizations of quasi-Hankel matrices from Mourrain and Pan (2000).) If the operator $\mathcal{T}_{\mathrm{ShSSA}}$ is injective, then it sets the isomorphism between the spaces $\mathsf{R}^{\mathfrak{N}}$ and $\mathsf{H}^{\mathfrak{L}\times\mathfrak{K}}$.

Remark 5.1 The embedding operator $\mathcal{T}_{\mathrm{ShSSA}}$ is injective if and only if $\mathfrak{L} \oplus \mathfrak{K} = \mathfrak{N}$, i.e., if each point of the initial shaped array $\mathbb{X}$ can be covered by a window of shape $\mathfrak{L}$.

If there are uncovered points, we can remove them and consider only the decomposition of the restricted array $\mathbb{X}' = (\mathbb{X})_{\mathfrak{N}'}$, where $\mathfrak{N}' = \mathfrak{L}\oplus\mathfrak{K}$. Hereafter we will suppose that $\mathfrak{N} = \mathfrak{L} \oplus \mathfrak{K}$, i.e., all points are covered and the operator $\mathcal{T}_{\mathrm{ShSSA}}$ is injective.

Projection to $\mathsf{H}^{\mathfrak{L}\times\mathfrak{K}}$

Projection $\Pi_{\mathfrak{N}\text{-HbH}}$ to the space of quasi-Hankel matrices is generated by the embedding operator $\mathcal{T}_{\text{ShSSA}}$ and can be performed using the general form described in Sect. 1.1.2.6.

5.2.1.3 Particular Cases

As shown in Golyandina et al. (2015), many versions of SSA can be considered as special cases of Shaped 2D-SSA. Basic SSA corresponds to the shapes

$$\mathfrak{N} = \{1, \dots, N\} \times \{1\}, \qquad \mathfrak{L} = \{1, \dots, L\} \times \{1\}.$$

The shaped arrays $\mathbb{X} \in \mathsf{R}^{\mathfrak{N}}$ in this case are time series, and the set $\mathsf{H}^{\mathfrak{L}\times\mathfrak{K}}$ is the set of ordinary Hankel matrices. 1D-SSA for time series with missing values is obtained by removing the indices of missing elements from the shape $\mathfrak{N}$.

MSSA for multivariate time series of possibly different lengths can be considered as Shaped 2D-SSA for a shaped 2D array consisting of stacked series and a rectangular window shape $\mathfrak{L} = \{1, \dots, L\} \times \{1\}$. Then the space $\mathsf{H}^{\mathfrak{L}\times\mathfrak{K}}$ is the space of horizontally-stacked Hankel matrices.

The 2D-SSA algorithm corresponds to Shaped 2D-SSA with

$$\mathfrak{N} = \{1, \dots, N_x\} \times \{1, \dots, N_y\},$$
$$\mathfrak{L} = \{1, \dots, L_x\} \times \{1, \dots, L_y\}.$$

The shaped arrays $\mathbb{X} \in \mathsf{R}^{\mathfrak{N}}$ are rectangular images, the windows $\mathfrak{L}$ are rectangular, and the set $\mathsf{H}^{\mathfrak{L}\times\mathfrak{K}}$ is the set of *Hankel-block-Hankel* matrices. For more details on different SSA versions as special cases of Shaped 2D-SSA, we refer the reader to Golyandina et al. (2015; Section 5.3).

All these SSA versions can be extended by allowing circular topology. In order to distinguish between different variants, we use the following terminology. If both T_x and T_y are infinite, then we call the SSA version *planar* (see Fig. 5.6, left);

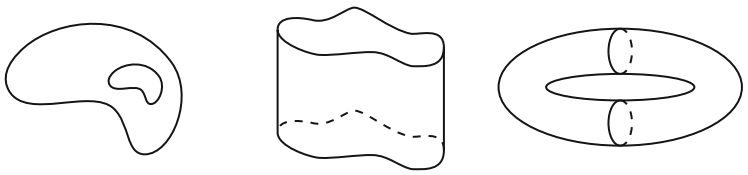

Fig. 5.6 Left: shaped image and window; center: cylindrical topology; right: toroidal topology

otherwise, we use the term *circular*. The case, when only one of T_x and T_y equals infinity, is *cylindrical* (see Fig. 5.6, center), while the case, when both of them are finite, is *toroidal* (see Fig. 5.6, right).

5.2.2 Rank of Shaped Arrays

5.2.2.1 $\mathfrak{L}$-Rank

Following the definition of finite-rank time series (Golyandina et al. 2001; Chapter 5) and finite-rank 2D arrays (Golyandina and Usevich 2010), let us generalize the notion of rank to the shaped and circular shaped cases. Note that generally the theory of infinite arrays of finite rank is closely related to multidimensional arrays satisfying LRRs.

Definition 5.1 The $\mathfrak{L}$-*trajectory space* $\mathsf{S}^{\mathfrak{L}}$ of a shaped array $\mathbb{S} \in \mathsf{R}^{\mathfrak{N}}$ is defined as

$$\mathsf{S}^{\mathfrak{L}}(\mathbb{S}) = \operatorname{span}\{(\mathbb{S})_{\mathfrak{L}\oplus\{\kappa\}}\}_{\kappa|\mathfrak{L}\oplus\{\kappa\}\subset\mathfrak{N}}.$$

The introduced trajectory space corresponds to the column space of the trajectory matrix $\mathbf{S} = \mathcal{T}_{\mathrm{ShSSA}}(\mathbb{S})$. In view of the isomorphism $\mathcal{J}_{\mathfrak{L}}$, $\{\mathbb{P}_1, \dots, \mathbb{P}_r\}$ is a basis of $\mathsf{S}^{\mathfrak{L}}(\mathbb{S})$ if and only if $\{P_1, \dots, P_r\}$, where $P_k = \mathcal{J}_{\mathfrak{L}}(\mathbb{P}_k)$, is a basis of the column space of $\mathbf{S}$. Let $\{Q_1, \dots, Q_r\}$ be a basis of the row space of $\mathbf{S}$, that is, of the row trajectory space. Then $\{\mathbb{Q}_1, \dots, \mathbb{Q}_r\}$, where $\mathbb{Q}_k = \mathcal{J}_{\mathfrak{K}}^{-1}(Q_k)$, forms the basis of a space, which can be denoted as $\mathsf{S}^{\mathfrak{K}}(\mathbb{S})$. We will use the terms *column*/*row trajectory spaces* and *column*/*row shaped trajectory spaces*, depending on the context.

Definition 5.2 The $\mathfrak{L}$-*rank* of a shaped array $\mathbb{S}$ is defined as the dimension of its $\mathfrak{L}$-trajectory subspace:

$$\operatorname{rank}_{\mathfrak{L}} \mathbb{S} = \dim \mathsf{S}^{\mathfrak{L}}(\mathbb{S}) = \operatorname{rank} \mathbf{S}.$$

5.2.2.2 Infinite Arrays of Finite Rank

In order to describe the general form of arrays of finite rank, we consider infinite arrays (and their trajectory spaces) in both sides for both dimensions (i.e., $\mathfrak{N} = \mathsf{Z} \times \mathsf{Z}$). We will consider the planar case with $T_x = T_y = \infty$.

Definition 5.3 Infinite (in both directions) array $\mathbb{S}_\infty$ is called the *array of finite shaped rank* if $r = \max\limits_{\mathfrak{L}} \operatorname{rank}_{\mathfrak{L}} \mathbb{S}_\infty$ (here the maximum is taken over $\mathfrak{L} \subset \{1, \dots, \infty\} \times \{1, \dots, \infty\}$) is finite. In this case, we will write $\operatorname{rank} \mathbb{S}_\infty = r$.

Remark 5.2 The shaped rank for an infinite array is equal to its rectangular rank (when the maximum is taken over rectangular shapes), since a sequence of shapes that contain increasing rectangular shapes and are contained in the rectangular

shapes can be easily constructed. Therefore, we will talk about arrays of finite rank omitting "shaped."

Remark 5.3 If $\mathbb{S}_\infty$ is an infinite array of finite rank r, $\mathbb{S} = (\mathbb{S}_\infty)_{\mathfrak{N}}$ is a sufficiently large finite subarray of $\mathbb{S}_\infty$ and $\mathfrak{L}$ is a sufficiently large shape, then

$$\operatorname{rank}_{\mathfrak{L}} \mathbb{S} = \operatorname{rank}_{\mathfrak{L}} \mathbb{S}_\infty = \operatorname{rank} \mathbb{S}_\infty = r. \tag{5.8}$$

Let $\mathfrak{K} = \{\kappa \mid \mathfrak{L} \oplus \{\kappa\} \subset \mathfrak{N}\}$. A sufficient condition for (5.8) (see, e.g., Golyandina and Usevich (2010) for the proof) is that both $\mathfrak{L}$ and $\mathfrak{K}$ shapes contain at least one $r \times r$ square. More formally, it is sufficient that there exists a 2D index $\alpha = (l, n)$ such that $\{1, \ldots, r\} \times \{1, \ldots, r\} \oplus \{\alpha\} \subset \mathfrak{L}$; the same should be valid for $\mathfrak{K}$.

Proposition 5.1 *Let $T_x = T_y = \infty$. Then an infinite array $\mathbb{S}_\infty$ of finite rank has the form*

$$(\mathbb{S}_\infty)_{m,n} = \sum_{k=1}^{s} P_k(m, n) \mu_k^m \nu_k^n, \tag{5.9}$$

where $\mu_k, \nu_k \in \mathsf{C}$, the pairs (μ_k, ν_k) are different, and $P_k(m, n)$ are complex polynomials of m and n. This representation is unique up to the order of summation.
As was mentioned in Sect. 1.4, this important fact is well known starting from Kurakin et al. (1995; §2.20). Note that the rank of the array $\mathbb{S}_\infty$ given in (5.9) is not determined by the degrees of the polynomials P_k only and is hard to compute (see Golyandina and Usevich (2010)).

An important special case is $(\mathbb{S}_\infty)_{ln} = \sum_{k=1}^{r} A_k \mu_k^l \nu_k^n$; it is widely used in signal processing. In this case, $\operatorname{rank} \mathbb{S}_\infty$ is equal to the number of different (μ_k, ν_k)-pairs.

5.2.3 Algorithm

Algorithm 5.3 Shaped 2D-SSA (planar): decomposition

Input: Image $\mathbb{X}$ of size $N_x \times N_y$ and mask of the same size consisting of TRUE and FALSE; window of size $L_x \times L_y$ and mask of the same size consisting of TRUE and FALSE; if the objects contain "NA," this means that the FALSE values are added to the masks at the places of "NA."

Output: Shaped image $\mathfrak{N}$, which has been decomposed, consisting of N points covered by the shaped window $\mathfrak{L}$ consisting of L points; shape $\mathfrak{K}$ consisting of the K possible positions of the window. Decomposition of the trajectory matrix of size $L \times K$: $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.

1: Construct the shapes $\mathfrak{L}$, $\mathfrak{K}$, $\mathfrak{N}$ and the trajectory matrix $\mathbf{X} = \mathcal{T}_{\mathrm{ShSSA}}(\mathbb{X}|_{\mathfrak{N}})$, where $\mathcal{T}_{\mathrm{ShSSA}}$ is defined by (5.7).
2: Compute the SVD $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.

The circular version of 2D-SSA can be considered in two versions, the cylindric and toroidal ones. Let us comment on the case, when the image is given on a cylinder with the main axis parallel to the axis "y." This case corresponds to $T_x < \infty$ and $T_y = \infty$. It is convenient to take $T_x = N_x$, which means that the image is located on the circular segment of a cylinder. The only difference of the algorithm of the circular 2D-SSA from the rectangular one, see Algorithm 5.3, is in the number K of the windows, which cover the image, and therefore in the form of the trajectory matrix $\mathbf{X}$. For example, in the cylindric topology, the window, which starts on one edge of the image, can continue to the opposite edge.

Note that the presentation of U_i or V_i in plots is performed in the form of shapes $\Psi_i = \mathcal{J}_{\mathfrak{L}}^{-1}(U_i)$ and $\Phi_i = \mathcal{J}_{\mathfrak{K}}^{-1}(V_i)$, which are called eigenshapes and factor shapes, respectively. Here $\mathcal{J}$ is the shaping operator, see (5.5). In the cylindric version, if the window is bounded, then eigenshapes are also bounded, while factors shapes are given on a cylinder.

Reconstruction stage is standard.

Algorithm 5.4 Shaped 2D-SSA reconstruction

Input: Decomposition $\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d$, where $\mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}}$ and $\|U_i\| = \|V_i\| = 1$, grouping $\{1, \ldots, d\} = \bigsqcup_{j=1}^{m} I_j$, $\mathfrak{N}$.

Output: Decomposition of the object on identifiable components: $\mathbb{X}\big|_{\mathfrak{N}} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$.

1: Construct the grouped matrix decomposition $\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_m}$, where $\mathbf{X}_I = \sum_{i \in I} \mathbf{X}_i$.

2: $\mathbb{X}\big|_{\mathfrak{N}} = \mathbb{X}_1 + \ldots + \mathbb{X}_m$, where $\mathbb{X}_i = \mathcal{T}^{-1}_{\text{ShSSA}} \circ \Pi_{\mathfrak{N}\text{-HbH}}(\mathbf{X}_{I_i})$.

5.2.4 *Shaped 2D-SSA in* Rssa

5.2.4.1 Description of Functions

Shaped 2D-SSA is performed if the initial image contains "NA" values. A typical call for Shaped 2D-SSA is

```
s <- ssa(x, mask = x != 0, wmask = circle(50), kind = "2d-ssa",
         circular = c(FALSE, FALSE))
```

Arguments:

`x` is a 2D object to be decomposed; it is assumed to be a matrix. The shape can be set by `NA`. All non-NA elements will be used as a mask.

`L` should be a vector of length 2 for 2D-SSA. If the parameter `wmask` is specified, then its value is considered as a shaped window, while the parameter `L` is ignored.

`mask` is used for the shaped 2D-SSA case only. This parameter indicates which entries in `x` will be considered for decomposition. It is a logical matrix with the same dimension as `x`. If `mask = NULL`, then all non-NA elements of `x` will be used. In the example considered, the mask is given separately and consists of non-zero elements of the input image `x`.

`wmask` should be set for Shaped SSA case only. It is a logical matrix which specifies the window shape. If `wmask = NULL`, then a rectangular window (specified by `L`) will be used. In addition, one may use the following functions as the `wmask` values:

`circle(R)` is the circle mask of radius `R`,

`triangle(side)` is the mask in the form of a isosceles right triangle with the leg of size `side`, where the right angle lays on the top-left corner of the circumscribed square.

`circular` is a logical vector of two elements describing the topology of a 2D object. The value `TRUE` means series circularity by the corresponding coordinate. One `TRUE` value provides the topology of a cylinder, while both `TRUE` values correspond to the topology of torus. Note that in the 1D case `circular` is a logical vector of one element, which describes the series topology for 1D-SSA and Toeplitz SSA. In the 1D case, the value `TRUE` means the topology of circle.

`neig` is the number of desired eigentriples. If `neig = NULL`, a default value which depends on L and N will be used.

`kind` specifies the version of SSA to be used; it can be omitted in non-ambiguous cases (e.g., when `x` is a matrix for the 2D case).

`svd.method` selects the SVD method. Full description is given in Sect. 2.1.5.2.

The function returns the `ssa` object, see details in Sect. 2.1.5. In addition, `s$fmask` provides the mask, which determines the shape $\mathfrak{K}$, and `s$weights` determines the weights of points of the initial object (how many times the window passes through each point); as a result, `s$weights > 0` defines the shape $\mathfrak{N}$.

Note that the ordinary 2D-SSA is implemented as a special case of Shaped 2D-SSA with rectangular window. Therefore, their computational costs are the same.

5.2.4.2 Typical Code

We repeat the experiment from Sect. 5.1.4 (noise removal from the image of Mars); this time using Shaped 2D-SSA. The code for loading the image is the same as in Fragment 5.1.1.

Recall that the array shape can be specified in two different ways:

- by passing the NA values in the input array (these elements are excluded), or
- by specifying the parameter mask which is a logical $N_x \times N_y$ array (the indicator of $\mathfrak{N}$).

If both shape specifications are present, their intersection is considered. The shape of the window is typically passed as an $L_x \times L_y$ logical array (wmask). The shapes can be also specified by a command circle, as shown in Fragment 5.2.2. Fragment 5.2.2 uses the function plot2d, which is presented in Fragment 5.2.1.

Fragment 5.2.1 (Auxiliary Plot of 2D Image)

```
> plot2d <- function(x) {
+   regions <- list(col = colorRampPalette(grey(c(0, 1))));
+   levelplot(t(x[seq(nrow(x), 1, -1), ]), aspect = "iso",
+             par.settings = list(regions = regions),
+             colorkey = FALSE,
+             scales = list(draw = FALSE, relation = "same"),
+             xlab = "", ylab = "")
+ }
```

Fragment 5.2.2 ("Mars": Mask Specification and Decomposition)

```
> mask.Mars.0 <- (Mars != 0)
> mask.Mars.1 <- (Mars != 255)
> Mars[!mask.Mars.0] <- NA
> print(system.time(s.Mars.shaped <-
+   ssa(Mars, kind = "2d-ssa",
+       mask = mask.Mars.1, wmask = circle(15))))
   user  system elapsed
   0.92    0.01    1.00
> mask.Mars.res <- (s.Mars.shaped$weights > 0)
> plot2d(mask.Mars.0)
> plot2d(mask.Mars.1)
> plot2d(mask.Mars.res)
```

In Fig. 5.7 one can see both types of masks and the resulting mask. Note that the resulting mask is constructed as a part of the combined mask, which is covered by the given shaped window.

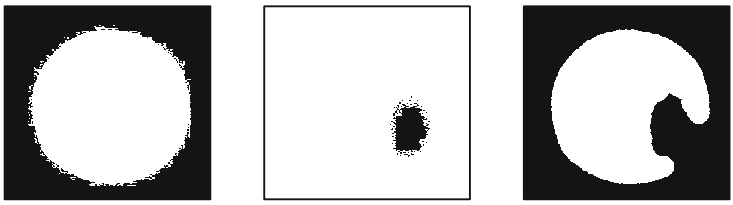

Fig. 5.7 "Mars" masks specification. Left: specified by NA, center: the parameter mask, right: resulting mask. White and black colors correspond to TRUE and FALSE, respectively

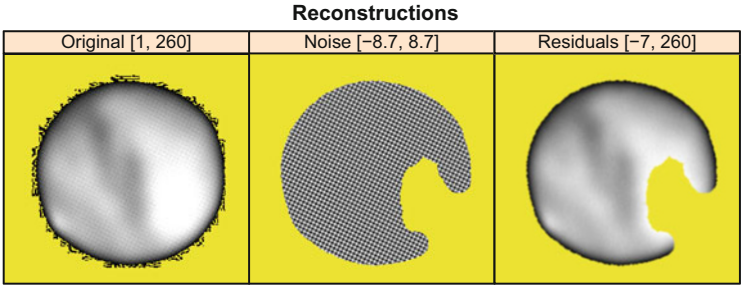

Fig. 5.8 "Mars": Reconstruction, ShSSA

Fragment 5.2.3 shows a typical reconstruction code for ShSSA.

Fragment 5.2.3 ("Mars": Reconstruction)

```
> r.Mars.shaped.groups <- list(Noise = c(7, 8, 9, 10))
> r.Mars.shaped <- reconstruct(s.Mars.shaped,
+                      groups = r.Mars.shaped.groups)
> plot(r.Mars.shaped, cuts = 255, layout = c(3, 1),
+      fill.color = "yellow")
```

The reconstruction results are shown in Fig. 5.8, where we can see that the elements are reconstructed only inside the resulting mask but the original array is drawn for all available elements (except the `NA` values).

The grouping for this decomposition was made based on the following information:

- the eigenarrays (see Fig. 5.9), and
- the matrix of **w**-correlations (see Fig. 5.11 (left)).

The same information can be used for visual estimation of the decomposition quality. Fragment 5.2.4 shows the code that reproduces Figs. 5.9 and 5.11 (left).

Fragment 5.2.4 ("Mars": Identification)

```
> plot(s.Mars.shaped, type = "vectors", idx = 1:30,
+      fill.color = "yellow", cuts = 255, layout = c(10, 3),
+      plot.contrib = FALSE)
> plot(wcor(s.Mars.shaped, groups = 1:30),
+      scales = list(at = c(10, 20, 30)))
> plot(s.Mars.shaped)
```

One can see that in Fig. 5.9 the texture components are slightly mixed with the smooth components, see, e.g., ET12,13,16,17,22. For shaped 2D (and nD) objects, as for the 1D case, we can use the techniques described in Sects. 2.5 and 2.4 for improving the quality of decompositions. Fragment 5.2.5 shows that Filter-adjusted O-SSA can be applied exactly in the same way as for the 1D case. We use the default value of the parameter `filter` in the function `fossa`, which corresponds to discrete derivatives in each direction given in the stacked manner,

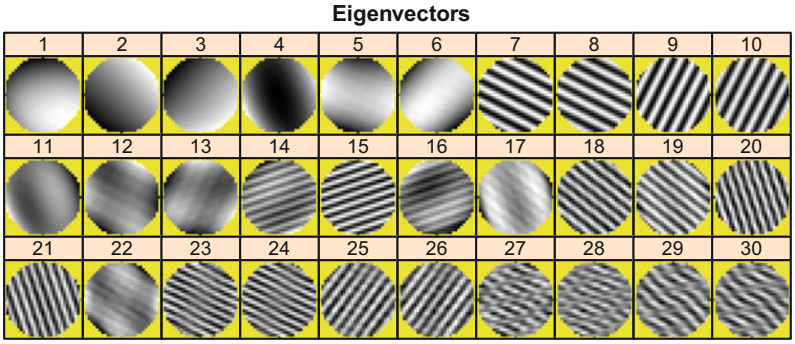

Fig. 5.9 "Mars": Eigenarrays, ShSSA

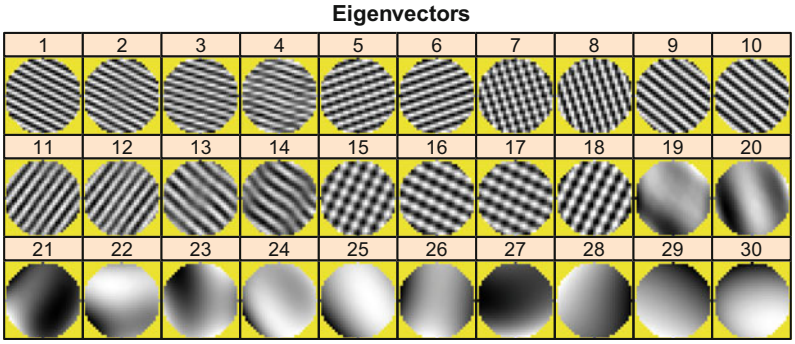

Fig. 5.10 "Mars": Eigenarrays, ShSSA; improvement by DerivSSA

see Remark 2.6. Figure 5.10 shows 30 leading eigenarrays and Fig. 5.11 (right) indicates the improvement in separability. We use the version with normalization (Algorithm 2.11) and therefore the components are ordered by the decrease of their frequencies. Thereby, the texture components are located before the smooth image components. Recall that this ordering does not reflect the component contribution, in contrast to the decomposition obtained with the help of the SVD (Fig. 5.9).

Fragment 5.2.5 ("Mars": Improvement by DerivSSA)

```
> s.Mars.shaped.deriv <-
+   fossa(s.Mars.shaped, nested.groups = list(1:30))
> plot(s.Mars.shaped.deriv, type = "vectors", idx = 1:30,
+      fill.color = "yellow", cuts = 255, layout = c(10, 3),
+      plot.contrib = FALSE)
> plot(wcor(s.Mars.shaped.deriv, groups = 1:30),
+      scales = list(at = c(10, 20, 30)))
> plot(s.Mars.shaped.deriv)
```

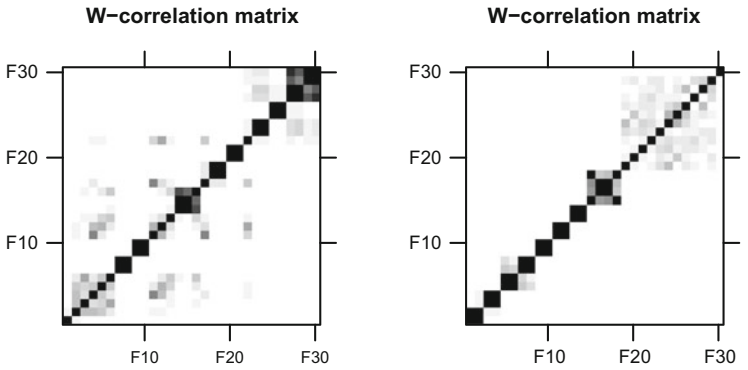

Fig. 5.11 "Mars": **w**-Correlations, ShSSA: initial (left) and after DerivSSA

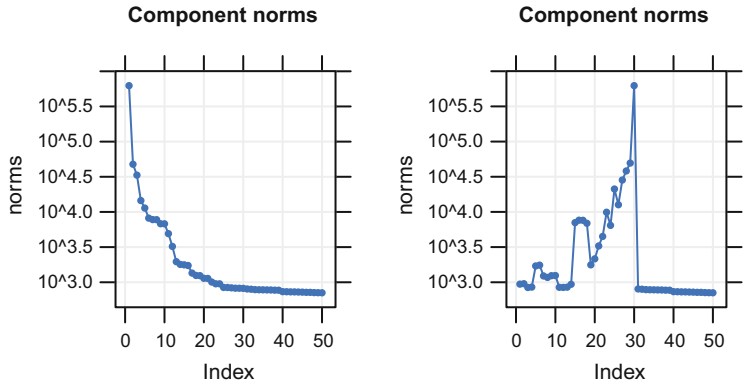

Fig. 5.12 "Mars": Contribution of elementary components, ShSSA: initial (left) and after DerivSSA

Figure 5.12 contains the component norms after application of ShSSA, where the component norms coincide with the ordered squared singular values of the trajectory matrix, and after application of DerivSSA. Note that after DerivSSA the components can be totally reordered, since their order is related to the values of their frequencies.

5.2.5 *Comments on nD Extensions*

We mentioned in the beginning of this chapter that the nD case is very similar to the 2D case. Moreover, many approaches presented in Chaps. 2 and 3 for the 1D case can be applied in the nD case. Let us describe the commonalities and the specificity of the nD case with $n > 2$.

- The `ssa` function has an option `kind`, which can have the value `"nd-ssa"`. This version of SSA is chosen by default if the input object is, e.g., `array` of dimension larger than 2.
- The nD objects can be of non-rectangular shapes; also, non-rectangular windows can be used. That is, the shaped nD-SSA is implemented. Moreover, all other SSA versions can be considered as particular cases of Shaped nD-SSA.
- The method nD-ESPRIT of frequency and damped factor estimation, which is a direct extension of the 2D-ESPRIT and Shaped ESPRIT, is implemented for objects of any dimension and shape.
- For the nD case, Iterative O-SSA and Filter-adjusted O-SSA, which are described in Chap. 2, and also Iterative gap-filling and Cadzow iterations, which are described in Chap. 3, are implemented. The call of the corresponding functions is very similar to the 1D case.
- In the nD case, there is a specificity for setting the parameter `filter` in the function `fossa`. In this case, the filter can be given by an array with the same number of dimensions as for the input object. Also, a list of 1D filters (formally, of vectors) can be provided for application to different dimensions in a stacked manner (see Remark 2.6). If only one 1D filter is given, then it is replicated n times. The value by default is `filter = c(-1,1)`, the same as in DerivSSA.
- Plotting of nD objects is an obviously difficult problem. Plotting functions in RSSA use the parameter `slice` to indicate which 2D slices the user wants to depict.

5.3 Shaped 2D ESPRIT

The well-known ESPRIT (Roy and Kailath 1989) and 2D-ESPRIT (Rouquette and Najim 2001) methods are used for estimation of parameters in the model $\mathbb{X} = \mathbb{S} + \mathbb{R}$, where $\mathbb{R}$ is noise and $\mathbb{S}$ is a signal of finite rank, which is of the form $(\mathbb{S})_l = \sum_{k=1}^{r} A_k \mu_k^l$ (for ESPRIT, see Sect. 3.1.1.2) and $(\mathbb{S})_{ln} = \sum_{k=1}^{r} A_k \mu_k^l \nu_k^n$ (for 2D-ESPRIT). In this section, following Shlemov and Golyandina (2014), we briefly describe the general scheme of ESPRIT-like methods on the example of shaped 2D-arrays.

5.3.1 Method

ESPRIT-like methods use an estimation of the signal subspace, which is obtained at Decomposition step of SSA. The described scheme is applicable to the shaped version of 2D-SSA.

5.3.1.1 General Scheme

1. Estimation of the Signal Subspace

We assume that the signal is at least approximately separated from the residual by a version of Shaped 2D-SSA. Let us consider Basic Shaped 2D-SSA. In this case, r left singular vectors $\{U_{i_1}, \ldots, U_{i_r}\}$ of the trajectory matrix (and corresponding shaped arrays $\{\mathbb{U}_{i_1}, \ldots, \mathbb{U}_{i_r}\}$, where $\mathbb{U}_{i_k} = \mathcal{J}_{\mathfrak{Q}}^{-1}(U_{i_k})$) approximate the column signal subspace. In signal processing, typically, the first r components are chosen, i.e. $\{i_1, \ldots, i_r\} = \{1, \ldots, r\}$.

2. Construction of Shifted Matrices

The algorithm uses the notion of shifts for shaped arrays from the space spanned by $\{\mathbb{U}_{i_1}, \ldots, \mathbb{U}_{i_r}\}$. Directions of shifts correspond to the directions in the initial space. Thus, the four matrices $\mathbf{P}_x$, $\mathbf{Q}_x$, $\mathbf{P}_y$, $\mathbf{Q}_y$ are constructed, whose columns span the shifted subspaces. For 2D-ESPRIT this step is considered in Rouquette and Najim (2001). For Shaped ESPRIT, which is more general, we define this step in Sect. 5.3.1.2.

3. Construction of Shift Matrices

The relation between the shifted subspaces allows estimation of the so-called shift matrices $\mathbf{M}_x$ and $\mathbf{M}_y$ of order $r \times r$ as approximate solutions of $\mathbf{P}_x\mathbf{M}_x \approx \mathbf{Q}_x$ and $\mathbf{P}_y\mathbf{M}_y \approx \mathbf{Q}_y$. The method of estimation is based on least squares (LS-ESPRIT) or total least squares (TLS-ESPRIT).

4. Estimation of Parameters

The eigenvalues of the shift matrices provide estimates of the parameters: the eigenvalues of the x-direction shift matrix $\mathbf{M}_x$ provide estimates of μ_k, while the eigenvalues of the y-direction shift matrix $\mathbf{M}_y$ produce estimates of ν_k. However, the method should provide estimates of the pairs (μ_k, ν_k). There are different approaches for pair estimation: simultaneous diagonalization of shift matrices (see Rouquette and Najim (2001; Section IV, method C); we will name it ESPRIT-DIAG) and a pairing of independently calculated eigenvalues of two shift matrices (the latter method is called 2D-MEMP (Rouquette and Najim 2001; Section IV, method A) and is improved in Wang et al. (2005); we will name the improved version ESPRIT-MEMP).

5.3.1.2 Method for Construction of Shifted Matrices

Let us introduce the shaped version of the ESPRIT method and describe its specific features. A difference of the shaped version from the rectangular planar one is in a special shift construction for shapes at Step 2 of the ESPRIT scheme; other steps are exactly the same as in the planar versions. Also, the circular topology influences the algorithm.

We will use the terminology introduced in Sect. 5.2: the topology of the initial 2D array $\mathfrak{N}$ is parameterized by (T_x, T_y); the 2D shape window $\mathfrak{L}$ is equipped with the topology of the initial object; $\mathbb{U}_1, \ldots, \mathbb{U}_r \in \mathsf{R}^{\mathfrak{L}}$ is a basis of the estimated signal subspace. For example, $\{\mathbb{U}_k\}_{k=1}^r$ are the $\mathfrak{L}$-shaped eigenvectors from the chosen eigentriples of the trajectory matrix identified as related to the image component that one wants to analyze.

Let us show how to construct the x-shifted matrix; that is, the matrix obtained by the shift along the x axis; the case of the shift along the y axis is analogous.

First, the subset of points from $\mathfrak{L}$ that have adjacent points of $\mathfrak{L}$ from the right in the x-direction has to be found:

$$\mathfrak{M}_x = \{\ell \in \mathfrak{L} \mid \ell \oplus (2,1) \in \mathfrak{L}\}, \tag{5.10}$$

where $\oplus$ is defined in (5.6). Recall that the sum $\oplus$ with 1 corresponds to no shift. Then the shaped eigenarrays restricted on $\mathfrak{M}_x$ and on $\mathfrak{M}_x \oplus \{(2,1)\}$ are computed:

$$\mathbb{P}_k = (\mathbb{U}_k)_{\mathfrak{M}_x}, \quad \mathbb{Q}_k = (\mathbb{U}_k)_{\mathfrak{M}_x \oplus \{(2,1)\}}, \ k = 1, \ldots, r; \tag{5.11}$$

after that, the arrays $\mathbb{P}_k$ and $\mathbb{Q}_k$ are vectorized into P_k and Q_k correspondingly. Finally, the obtained vectors are stacked into the matrices

$$\mathbf{P}_x = [P_1 : \cdots : P_r], \quad \mathbf{Q}_x = [Q_1 : \cdots : Q_r].$$

This way, the shifted matrices $\mathbf{P}_x$, $\mathbf{Q}_x$, $\mathbf{P}_y$, $\mathbf{Q}_y$ are constructed and the shift matrices $\mathbf{M}_x$ and $\mathbf{M}_y$ can be estimated from the relations

$$\mathbf{P}_x\mathbf{M}_x \approx \mathbf{Q}_x, \quad \mathbf{P}_y\mathbf{M}_y \approx \mathbf{Q}_y. \tag{5.12}$$

Remark 5.4 We have considered the version of ESPRIT based on the column space. The version based on the row space is similar.

5.3.2 *Theory: Conditions of the Algorithm Correctness*

Let an $\mathfrak{N}$-shaped array $\mathbb{S}$ of rank r have the common term in the form $(\mathbb{S})_{ln} = \sum_{k=1}^r A_k \mu_k^l \nu_k^n$ for a set of different pairs $\{(\mu_k, \nu_k)\}_{k=1}^r$, $\mu_k, \nu_k \in \mathsf{C}$, and let the noise $\mathbb{R}$ be zero. Proposition 5.2 (Shlemov and Golyandina 2014) forms the basis

of the ESPRIT-type methods (see, e.g., Rouquette and Najim (2001)) and provides the conditions when Algorithm 5.5 and the ESPRIT scheme as a whole are correct; that is, the equations (5.12) have exact solutions and the constructed shift matrices produce exactly the pairs (μ_k, ν_k) by the exact joint diagonalization at the last step of the scheme.

Proposition 5.2 *Assume that the window $\mathfrak{L}$ can be chosen so that the forms $\mathfrak{M}_x$ and $\mathfrak{M}_y$ given in (5.10) are nonempty and* $\operatorname{rank}_{\mathfrak{M}_x} \mathbb{S} = \operatorname{rank}_{\mathfrak{M}_y} \mathbb{S} = \operatorname{rank}_{\mathfrak{L}} \mathbb{S} = r$. *Denote $\{\mathbb{U}_k\}_{k=1}^r$, $\mathbb{U}_k \in \mathsf{R}^{\mathfrak{L}}$, a basis of $\mathsf{S}^{\mathfrak{L}}(\mathbb{S})$. Then*

(1) the equalities $\mathbf{P}_x\mathbf{M}_x = \mathbf{Q}_x$ and $\mathbf{P}_y\mathbf{M}_y = \mathbf{Q}_y$, see (5.12), have unique exact solutions $\mathbf{M}_x$ and $\mathbf{M}_y$;

(2) there exists a matrix $\mathbf{T} \in \mathsf{C}^{r\times r}$ so that $\mathbf{M}_x = \mathbf{T}\operatorname{diag}(\mu_1, \dots, \mu_r)\mathbf{T}^{-1}$ and $\mathbf{M}_y = \mathbf{T}\operatorname{diag}(\nu_1, \dots, \nu_r)\mathbf{T}^{-1}$.

The following proposition (Shlemov and Golyandina 2014) provides sufficient conditions for the correctness of the Shaped 2D ESPRIT algorithm for both planar and circular versions.

Proposition 5.3 *Let the shape $\mathfrak{L}$ contain at least one $(r+1)\times(r+1)$ square and the shape $\mathfrak{K} = \{\kappa \mid \mathfrak{L} \oplus \{\kappa\} \subset \mathfrak{N}\}$ contain at least one $r \times r$ square. Then the conditions of Proposition 5.2 are valid.*

5.3.3 Algorithm

Let us show how to construct the x-shifted matrix; that is, the matrix obtained by the shift along the x axis; the case of the shift along the y axis is analogous.

Algorithm 5.5 Construction of x-shifted matrices

Input: Topology of the initial 2D array $\mathfrak{N}$ parameterized by (T_x, T_y); 2D shape $\mathfrak{L}$ equipped with the topology of the initial object; basis $\mathbb{U}_1, \dots, \mathbb{U}_r \in \mathsf{R}^{\mathfrak{L}}$ of the estimated signal subspace.
Output: Matrices $\mathbf{P}_x, \mathbf{Q}_x$.

1: Find the subset of points from $\mathfrak{L}$ that have adjacent points of $\mathfrak{L}$ from the right in the x-direction:

$$\mathfrak{M}_x = \{\ell \in \mathfrak{L} \mid \ell \oplus (2,1) \in \mathfrak{L}\} \tag{5.13}$$

2: Consider the shaped eigenarrays restricted on $\mathfrak{M}_x$ and on $\mathfrak{M}_x \oplus \{(2,1)\}$:

$$\mathbb{P}_k = (\mathbb{U}_k)_{\mathfrak{M}_x}, \quad \mathbb{Q}_k = (\mathbb{U}_k)_{\mathfrak{M}_x \oplus \{(2,1)\}}, \ k = 1, \dots, r. \tag{5.14}$$

3: Vectorize the arrays $\mathbb{P}_k$ and $\mathbb{Q}_k$ into P_k and Q_k correspondingly.
4: Stack the obtained vectors to the matrices

$$\mathbf{P}_x = [P_1 : \cdots : P_r], \quad \mathbf{Q}_x = [Q_1 : \cdots : Q_r].$$

Now let us write down the scheme of the Shaped 2D ESPRIT algorithm according to the scheme described in Sect. 5.3.1.1.

Algorithm 5.6 Shaped 2D-ESPRIT

Input: Topology of the initial 2D array $\mathfrak{N}$ parameterized by (T_x, T_y); 2D shape $\mathfrak{L}$ equipped with the topology of the initial object; rank r.

Output: Set of pairs (μ_k, ν_k), $k = 1, \ldots, r$.

1: Find a basis $\mathbb{U}_1, \ldots, \mathbb{U}_r \in \mathsf{R}^{\mathfrak{L}}$ of the estimated signal subspace by Algorithm 5.3.
2: Construct the shifted matrices $\mathbf{P}_x$, $\mathbf{Q}_x$ by Algorithm 5.5 and the shifted matrices $\mathbf{P}_y$, $\mathbf{Q}_x$ in a similar way.
3: Calculate the shift matrices $\mathbf{M}_x$ and $\mathbf{M}_y$ using the least-squares or total least squares method for $\mathbf{P}_x\mathbf{M}_x \approx \mathbf{Q}_x$ and $\mathbf{P}_y\mathbf{M}_y \approx \mathbf{Q}_y$.
4: Obtain the estimates (μ_k, ν_k) by applying one of the pairing methods to the shift matrices $\mathbf{M}_x$ and $\mathbf{M}_y$.

5.3.4 2D-ESPRIT in RSSA

5.3.4.1 Description of Functions

Estimation of parameters by 2D-ESPRIT is performed by the function `parestimate`, which is also used for 1D-ESPRIT (see Sect. 3.1.3). The difference is in the type of the `ssa` object and additional parameters. A typical call has the form

```
est <- parestimate(s, groups = list(c(1:4)))
```

Arguments:

`s` is an `ssa` object holding the full 2D-SSA decomposition (possibly in the shaped version).

`groups` is a list of groups of eigentriples.

`method` is a method of estimation of frequencies and damped factors; it can only have the value `"esprit"` for nD objects.

`subspace` indicates which space, column or row, will be used for parameter estimation by the ESPRIT method. The default value `"column"` is standard for ESPRIT.

`solve.method` is the method of shift matrices estimation (see Step 3 in Sect. 5.3.1.1); it can be `"ls"` for the least squares solution or `"tls"` for the total least squares solution.

`pairing.method` is the method for roots pairing; it can be `"diag"` for 2D-ESPRIT diagonalization or `"memp"` for MEMP with the improved pairing step (see Step 4 in Sect. 5.3.1.1).

`beta` is a coefficient in the convex linear combination of the shifted matrices (Rouquette and Najim 2001; Eq. (48)).

5.3.4.2 Typical Code

Let us consider a simple rectangular case.

We continue the example "Mars" from Fragment 5.1.7. This example demonstrates advantages of RSSA because of the possibility to choose large window sizes, which was always considered to be problematic for the ESPRIT-type methods. Fragment 5.3.1 shows the corresponding code that outputs the estimated exponentials. In Fig. 5.13, the estimated pairs of complex exponentials (μ_k, ν_k) are shown on separate complex plane plots and the points for μ_k and ν_k have the same color (for the same k).

Fragment 5.3.1 ("Mars": Parameter Estimation with 2D-ESPRIT)

```
> pe.Mars.160.80 <- parestimate(s.Mars.160.80,
+                               groups = r.Mars.160.80.groups)
> print(pe.Mars.160.80)
       x: period       rate |        y: period       rate
           -5.000  -0.000169 |           10.003  -0.000111
            5.000  -0.000169 |          -10.003  -0.000111
            9.995   0.000175 |            4.999  -0.000093
           -9.995   0.000175 |           -4.999  -0.000093
> print(pe.Mars.160.80[[1]])
   period     rate   |    Mod     Arg   |     Re        Im
   -5.000  -0.000169 |  0.99983  -1.26  |  0.30906  -0.95087
    5.000  -0.000169 |  0.99983   1.26  |  0.30906   0.95087
    9.995   0.000175 |  1.00017   0.63  |  0.80897   0.58814
   -9.995   0.000175 |  1.00017  -0.63  |  0.80897  -0.58814
> print(pe.Mars.160.80[[2]])
   period     rate   |    Mod     Arg   |     Re        Im
   10.003  -0.000111 |  0.99989   0.63  |  0.80905   0.58755
  -10.003  -0.000111 |  0.99989  -0.63  |  0.80905  -0.58755
    4.999  -0.000093 |  0.99991   1.26  |  0.30879   0.95103
   -4.999  -0.000093 |  0.99991  -1.26  |  0.30879  -0.95103
> plot(pe.Mars.160.80, col = c(11, 12, 13, 14))
> plot(s.Mars.160.80, type = "vectors",
+      idx = r.Mars.160.80.groups$Noise,
+      cuts = 255, layout = c(4, 1), plot.contrib = FALSE)
```

In Fragment 5.3.1 and Fig. 5.13, it can be seen that each pair (μ_k, ν_k) has its conjugate counterpart $(\mu_{k'}, \nu_{k'}) = \mathrm{conj}((\mu_k, \nu_k))$. In particular, it is the case for $(k, k') = (1, 2)$, and for $(k, k') = (3, 4)$. Therefore, the periodic noise is a sum of two planar sines, as explained in Sect. 5.1.2. This is also confirmed by the plots of the eigenarrays in Fig. 5.14.

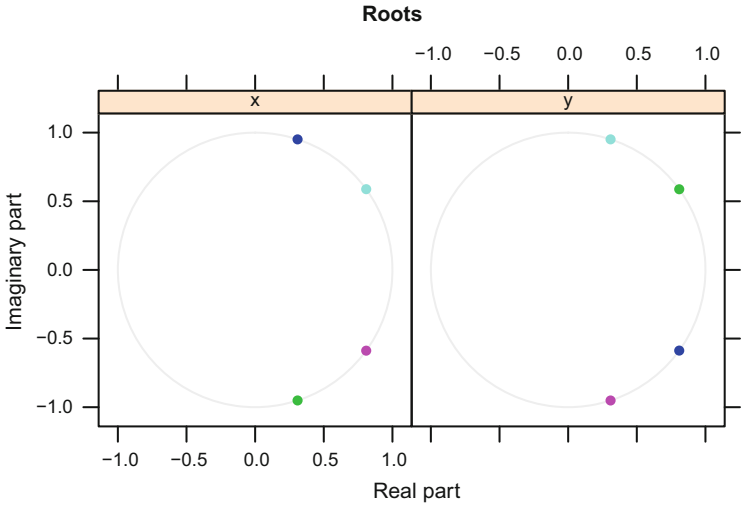

Fig. 5.13 "Mars": Parameter estimation with 2D-ESPRIT, $(L_x, L_y) = (160, 80)$

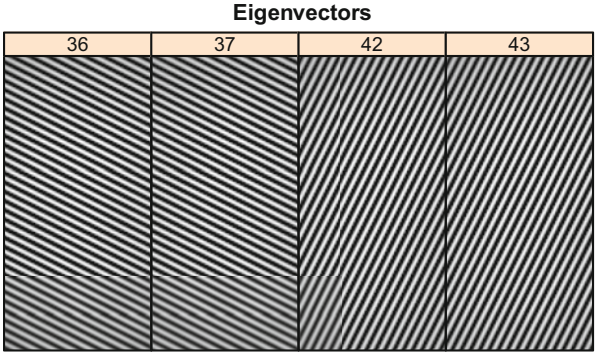

Fig. 5.14 "Mars": Eigenarrays corresponding to the periodic noise, $(L_x, L_y) = (160, 80)$

Fragment 5.3.2, which continues Fragment 5.1.7, demonstrates that similar estimates of parameters can be obtained by Shaped 2D-SSA.

Fragment 5.3.2 ("Mars": Parameter Estimation with Shaped 2D-ESPRIT)

```
> pe.Mars.shaped <- parestimate(s.Mars.shaped,
+                               r.Mars.shaped.groups)
> print(pe.Mars.shaped)
       x: period       rate |        y: period       rate
          -5.007  -0.001403 |           10.015  -0.000408
           5.007  -0.001403 |          -10.015  -0.000408
          10.008   0.000350 |            5.004  -0.001084
         -10.008   0.000350 |           -5.004  -0.001084
```

5.4 Case Studies

5.4.1 Extraction of Texture from Non-Rectangle Images

Let us compare the results of shaped and rectangular versions of 2D-SSA applied to "Mars" data, which were obtained in Sects. 5.1.4.2 and 5.2.4.2.

The quality of the texture extraction and therefore of the image recovery by Shaped SSA (Fig. 5.8) is considerably better than that for 2D-SSA (Fig. 5.2). The improvement in the reconstruction accuracy can be explained by the edge effect that is caused by a sharp drop of intensity near the boundary of Mars. In Fig. 5.15, we compare magnified reconstructed images for 2D-SSA and ShSSA. In the left subfigure, a yellow shadow is shown for the background area in order to indicate the Mars boundary. In the right subfigure, the pure yellow color corresponds to NA. The code that reproduces Fig. 5.15 is shown in Fragment 5.4.1.

Fragment 5.4.1 ("Mars": Magnified Reconstructions by 2D-SSA and ShSSA)

```
> Mars.sh <- r.Mars.shaped$Noise
> Mars.rect.sh <- Mars.rect <- r.Mars.25$Noise
> Mars.rect.sh[is.na(Mars.sh)] <- NA
> library("latticeExtra")
> p.part.rect <- plot2d(Mars.rect[60:110, 200:250]) +
+                   layer(panel.fill(col = "yellow",
+                                    alpha = 0.2),
+                         under = FALSE) +
+                   plot2d(Mars.rect.sh[60:110, 200:250])
> p.part.shaped <- plot2d(r.Mars.shaped[[1]][60:110, 200:250]) +
+                     layer(panel.fill(col = "yellow"),
+                           under = TRUE)
> plot(c(Rectangular = p.part.rect, Shaped = p.part.shaped))
```

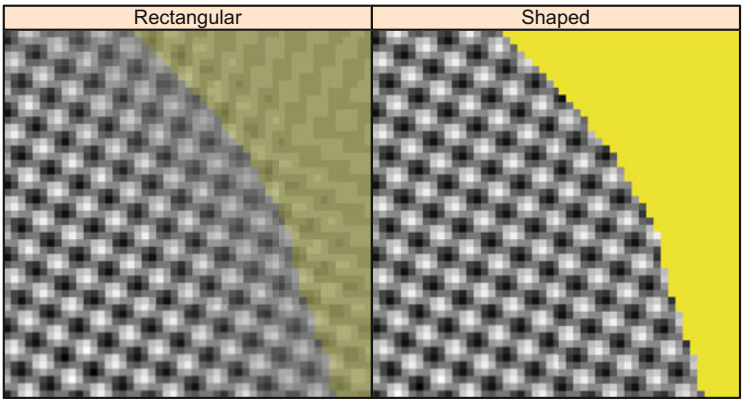

Fig. 5.15 "Mars": comparison of texture reconstructions by 2D-SSA and ShSSA

5.4.2 *Adaptive Smoothing*

2D-SSA can be used for adaptive smoothing of two-dimensional data. It was used for this purpose in Golyandina et al. (2007) for smoothing digital terrain models (DTM) as well as in Holloway et al. (2011) and Golyandina et al. (2012b) for smoothing spatial gene expression data.

Similarly to the example in Golyandina et al. (2007), we consider an image extracted from the SRTM database. The test DTM of a region in South Wales, UK, is extracted by the function `getData` of the package RASTER (Hijmans 2016). The DTM is 80×100, and it includes the Brecon Beacons national park. The point $(N_x, 1)$ lies in a neighborhood of Port Talbot, (N_x, N_y) is in a neighborhood of Newport, and $(1, N_y)$ is near Whitney-on-Wye. In Fragment 5.4.2, we decompose the image with a small window and plot the eigenvectors in Fig. 5.16.

Fragment 5.4.2 ("Brecon Beacons": Decomposition)

```
> data("brecon", package = "ssabook")
> s.brecon <- ssa(brecon, kind = "2d-ssa", L = c(8, 8),
+                 svd.method = "eigen")
> plot(s.brecon, type = "vectors", idx = 1:32,
+      cuts = 255, layout = c(8, 4), plot.contrib = FALSE)
```

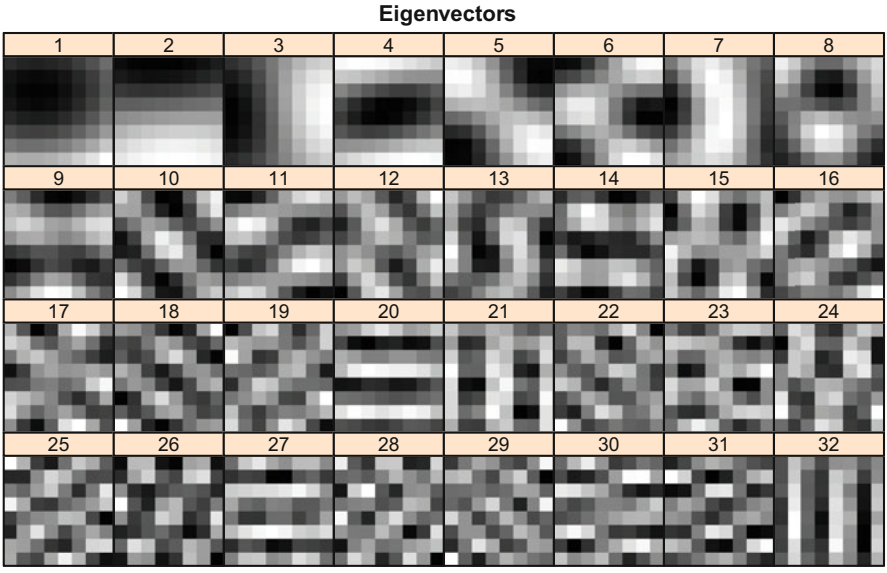

Fig. 5.16 "Brecon Beacons": 8×8 windows, eigenarrays

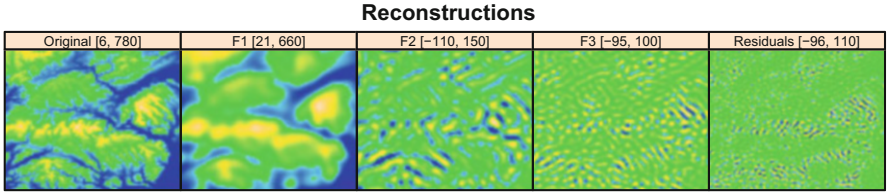

Fig. 5.17 "Brecon Beacons": 8×8 window, reconstructions $(\widetilde{\mathbb{X}}_k)$

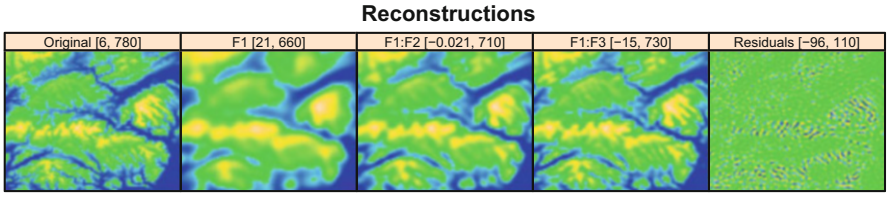

Fig. 5.18 "Brecon Beacons": 8×8 window, cumulative reconstructions $(\widetilde{\mathbb{Y}}_k)$

Next, we reconstruct the image by grouping the components with numbers from $I_1 = \{1, \ldots, 3\}$, $I_2 = \{4, \ldots, 8\}$, and $I_3 = \{9, \ldots, 17\}$. The grouping was chosen based on eigenarrays to gather frequencies of similar scale in the same components.

We take cumulative sums of the reconstructed components $\widetilde{\mathbb{Y}}_1 = \widetilde{\mathbb{X}}_1$, $\widetilde{\mathbb{Y}}_1 = \widetilde{\mathbb{X}}_1 + \widetilde{\mathbb{X}}_2$, and $\widetilde{\mathbb{Y}}_1 = \widetilde{\mathbb{X}}_1 + \widetilde{\mathbb{X}}_2 + \widetilde{\mathbb{X}}_3$ (this is a convenient way to compute reconstructed components for sets $J_1 = I_1$, $J_2 = I_1 \cup I_2$, and $J_3 = I_1 \cup I_2 \cup I_3$). The reconstructed components are plotted in Fig. 5.17. The cumulative components $\widetilde{\mathbb{Y}}_k$ are shown in Fig. 5.18 using the parameter `cumsum` of the plotting function of Rssa.

Fragment 5.4.3 ("Brecon Beacons": Reconstruction)

```
> r.brecon <- reconstruct(s.brecon,
+                         groups = list(1:3, 4:8, 9:17))
> plot(r.brecon, cuts = 255, layout = c(5, 1),
+      par.strip.text = list(cex = 0.75),
+      col = topo.colors(1000))
> plot(r.brecon, cuts = 255, layout = c(5, 1),
+      par.strip.text = list(cex = 0.75),
+      type = "cumsum", at = "free",
+      col = topo.colors(1000))
```

From observing Figs. 5.17 and 5.18 we can conclude that the reconstructed components capture morphological features at different scales (Golyandina et al. 2007). The cumulative reconstructions represent smoothing of the original DTM of different resolution.

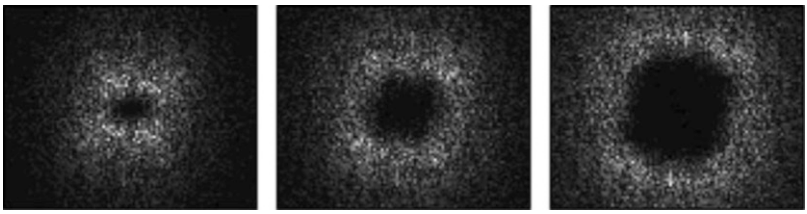

Fig. 5.19 "Brecon Beacons": 8×8 window, absolute values of the DFT of $\mathbb{X} - \widetilde{\mathbb{Y}}_k$, $k = 1, 2, 3$

To illustrate the behavior of smoothing, we plot the absolute values of the centered discrete Fourier transforms (DFT) of $\mathbb{X} - \widetilde{\mathbb{Y}}_k$ (residuals for the cumulative reconstructions, see Fig. 5.19). The corresponding code can be found in Fragment 5.4.5. We also use the function `plot2d`, which is defined in Fragment 5.2.1, and introduce a code for computing their centered DFTs in Fragment 5.4.4.

Fragment 5.4.4 (2D-SSA: Centered DFT)

```
> centered.mod.fft <- function(x) {
+   N <- dim(x)
+   shift.exp <- exp(2i*pi * floor(N/2) / N)
+   shift1 <- shift.exp[1]^(0:(N[1] - 1))
+   shift2 <- shift.exp[2]^(0:(N[2] - 1))
+   Mod(t(mvfft(t(mvfft(outer(shift1, shift2) * x)))))
+ }
```

Fragment 5.4.5 ("Brecon Beacons": DFT of Cumulative Reconstructions)

```
> plot2d(centered.mod.fft(brecon - r.brecon$F1))
> plot2d(centered.mod.fft(brecon - r.brecon$F1 - r.brecon$F2))
> plot2d(centered.mod.fft(brecon - r.brecon$F1 -
+                         r.brecon$F2 - r.brecon$F3))
```

In Fig. 5.19, it is clearly seen that 2D-SSA reconstruction by the leading components acts as a filter that preserves dominating frequencies (in this case, a low-pass filter).

5.4.3 Analysis of Data Given on a Cylinder

Let us consider an example of data for expression of gene activity measured on the embryo of the drosophila (fruit fly) (Shlemov and Golyandina 2014). The aim of the analysis is to decompose the data into a sum of a pattern and noise and then measure the signal/noise ratio. The form of the embryo is similar to an ellipsoid and therefore the cylindrical projection is adequate for the middle part of the embryo. The data are cropped and then are interpolated to a regular grid; in this way, we obtain a digital image, which can be processed by 2D-SSA. After that, the values of the smoothed expression are interpolated onto the nuclei centers.

As an input, we take the data `v5_s11643-28no06-04.pca`, gene "Krüppel." The data was downloaded from the BDTNP archive (Berkeley Drosophila Transcription Network Project 2014), and then was preprocessed to obtain a digital image by means of BioSSA (Shlemov et al. 2014). The conventional technique of SSA analysis of such kind of 2D data is a planar non-shaped 2D-SSA (Holloway et al. 2011; Golyandina et al. 2012b). However, this technique does not take into consideration the fact that the top edge of this image continues the bottom edge in the considered orientation; the data is in fact cylindrical.

Fragment 5.4.6 demonstrates that the circular version of 2D-SSA can perform smoothing without making the artificial transformation of the cylinder to a rectangle. Thus the result of processing does not depend on the technique of this transformation and has no extra edge effects. We consider the cylindrical topology ($T_x = N_x$ and $T_y = \infty$). Since the axis x with circular topology corresponds to rows of the image, this can be set by `circular = c(TRUE, FALSE)`. We use a mask corresponding to the middle part from 20 to 80% of the embryo lengths to remove corruptions caused by the cylindrical projection of an ellipsoid. That is, in fact, we use the shaped version of 2D-SSA.

Fragment 5.4.6 ("Krüppel": Analysis of Data Given on a Cylinder)

```
> data("kruppel", package = "ssabook")
> mask <- matrix(0, ncol = 200, nrow = 200)
> mask[, 40:160] <- 1
> s <- ssa(kruppel, L = c(20, 20), mask = mask,
+          circular = c(TRUE, FALSE))
> plot(s, type = "vectors", vectors = "factor",
+      idx = 1:12, aspect = 1.4, col = topo.colors(1000))
> rec6 <- reconstruct(s, list("C1-6" = 1:6))
> p <- plot(rec6, aspect = 1.4, col = topo.colors(1000))
> p$condlevels[[1]] <-
+   sub("Residuals", "Resd.", p$condlevels[[1]], fixed = TRUE)
> plot(p)
```

The result of the decomposition (in the form of the factor vectors from 12 leading eigentriples) is depicted in Fig. 5.20, where contributions of the corresponding eigentriples are indicated in the headers.

The components 1–6 grouped together provide an adequate smoothing; the residual oscillates around zero, see Fig. 5.21 (the maximal and minimal values of the reconstructed components are shown in the headers). Note that the top edge of the reconstruction continues its bottom edge by construction; that is, there is no edge effect.

Several examples of application of the circular and shaped versions of 2D-SSA to similar data can be found in Shlemov et al. (2015b).

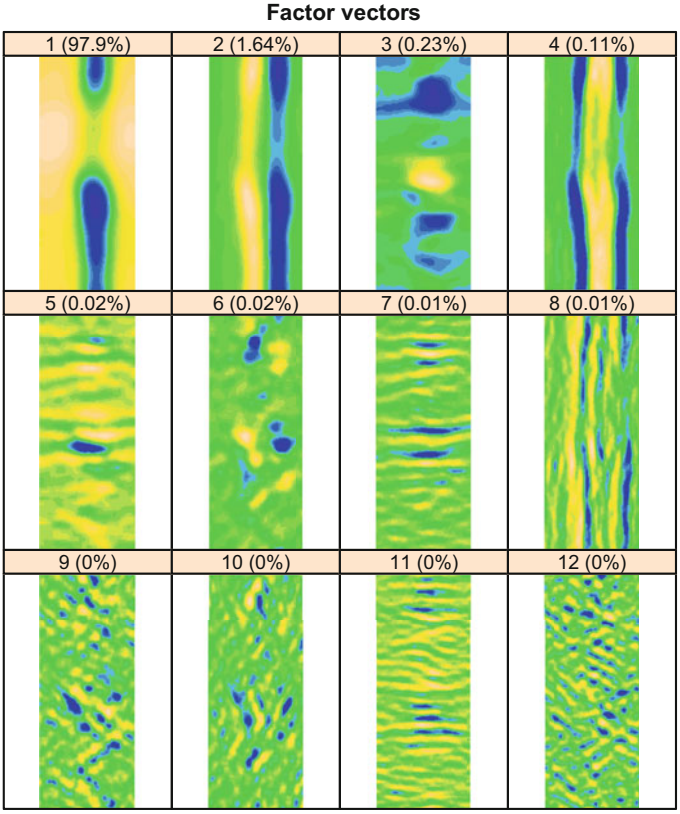

Fig. 5.20 "Krüppel": Factor vectors

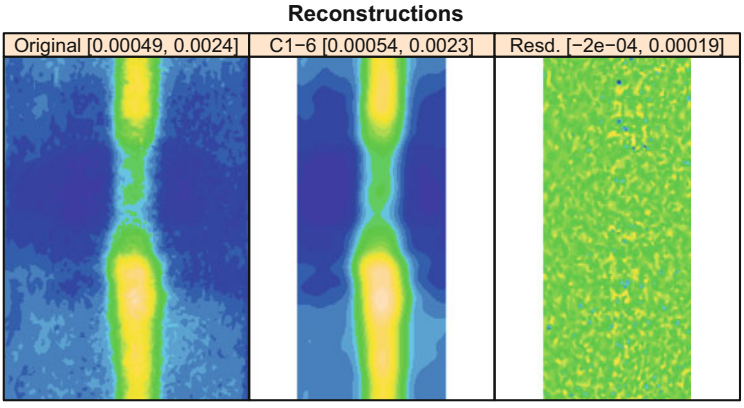

Fig. 5.21 "Krüppel": Original image, reconstruction and residual

5.4.4 Analysis of nD Objects: Decomposition of a Color Image

As multidimensional data is very difficult to visualize, we demonstrate possibilities of RSSA to perform nD-SSA on an example of a decomposition of a color image. Since color images can be given by three channels corresponding to the red, green, and blue colors, they can be considered as 3D objects. The window for 3D objects should also be 3D; that is, be given by 3 sizes (L_x, L_y, L_z). Since the size of the third image dimension is equal to 3, there are not many choices for L_z. We take $L_z = 1$, which corresponds to Multivariate 2D-SSA (M-2D-SSA), since with $L_z = 1$ the window has in fact 2 dimensions. Fragment 5.4.7 contains an example of the code for performing M-2D-SSA.

Fragment 5.4.7 ("Monet": Decomposition by Multivariate 2D-SSA)

```
> data("monet", package = "ssabook")
> plot(c(0, 1), c(0, 1), type = "n", xlab = "", ylab = "",
+      axes = FALSE)
> rasterImage(monet, 0, 0, 1, 1, interpolate = FALSE)
> ss <- ssa(monet, L = c(30, 30, 1))
> plot(ss, type = "vectors", idx = 1:20, slice = list(k = 1),
+      plot.contrib = FALSE)
> rec <- reconstruct(ss, groups = list(smooth = 1:6))
> rec$smooth[rec$smooth > 1] <- 1
> rec$smooth[rec$smooth < 0] <- 0
> plot(c(0, 1), c(0, 1), type = "n", xlab = "", ylab = "",
+      axes = FALSE)
> rasterImage(rec$smooth, 0, 0, 1, 1,
+             interpolate = FALSE)
> p1 <- plot(rec, slice = list(k = 1), main = "Red",
+            col = c("#000000", "#FF7070"), layout = c(3, 1))
> p2 <- plot(rec, slice = list(k = 2), main = "Green",
+            col = c("#000000", "#70FF70"), layout = c(3, 1))
> p3 <- plot(rec, slice = list(k = 3), main = "Blue",
+            col = c("#000000", "#7070FF"), layout = c(3, 1))
> plot(p1, split = c(1, 1, 1, 3), more = TRUE)
> plot(p2, split = c(1, 2, 1, 3), more = TRUE)
> plot(p3, split = c(1, 3, 1, 3), more = FALSE)
```

We have selected a small color image 263×400 of the painting of Claude Monet called "Study of a Figure Outdoors: Woman with a Parasol, facing left," 1886, (Fig. 5.22 (left), the file was taken from the public domain via Wikimedia Commons) and apply M-2D-SSA with the window of sizes $(50, 50, 1)$. For the component identification, we can examine eigenarrays of the same size as the window has. For depicting multidimensional objects, 2D slices can be used, where the values in all the dimensions except two are fixed. Since the window is in fact 2D, there is only one slice in the third dimension (`slice = list(k = 1)`). Figure 5.23 contains images of 12 leading eigenarrays. White contours separate positive and negative values.

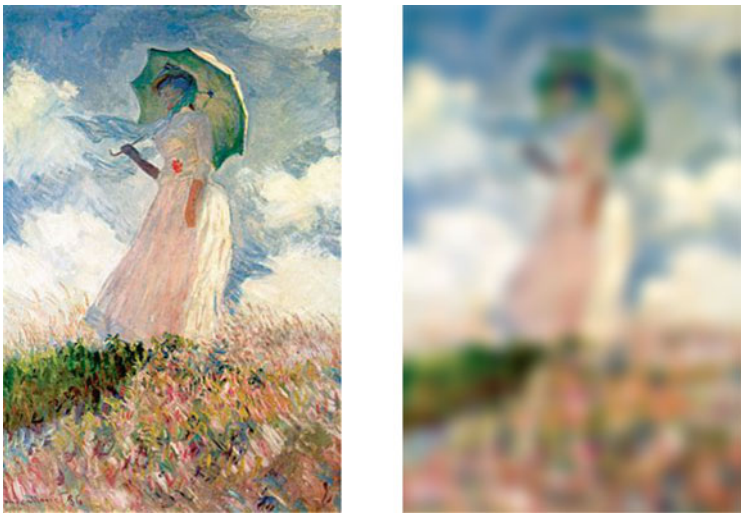

Fig. 5.22 "Monet": Initial (left) and smoothed (right) images

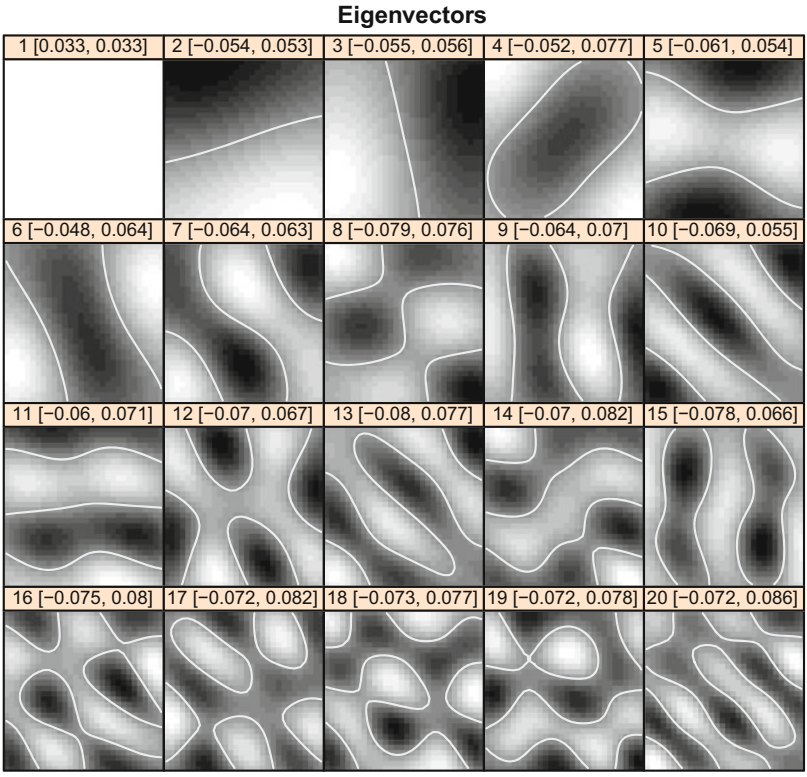

Fig. 5.23 "Monet": Eigenarrays

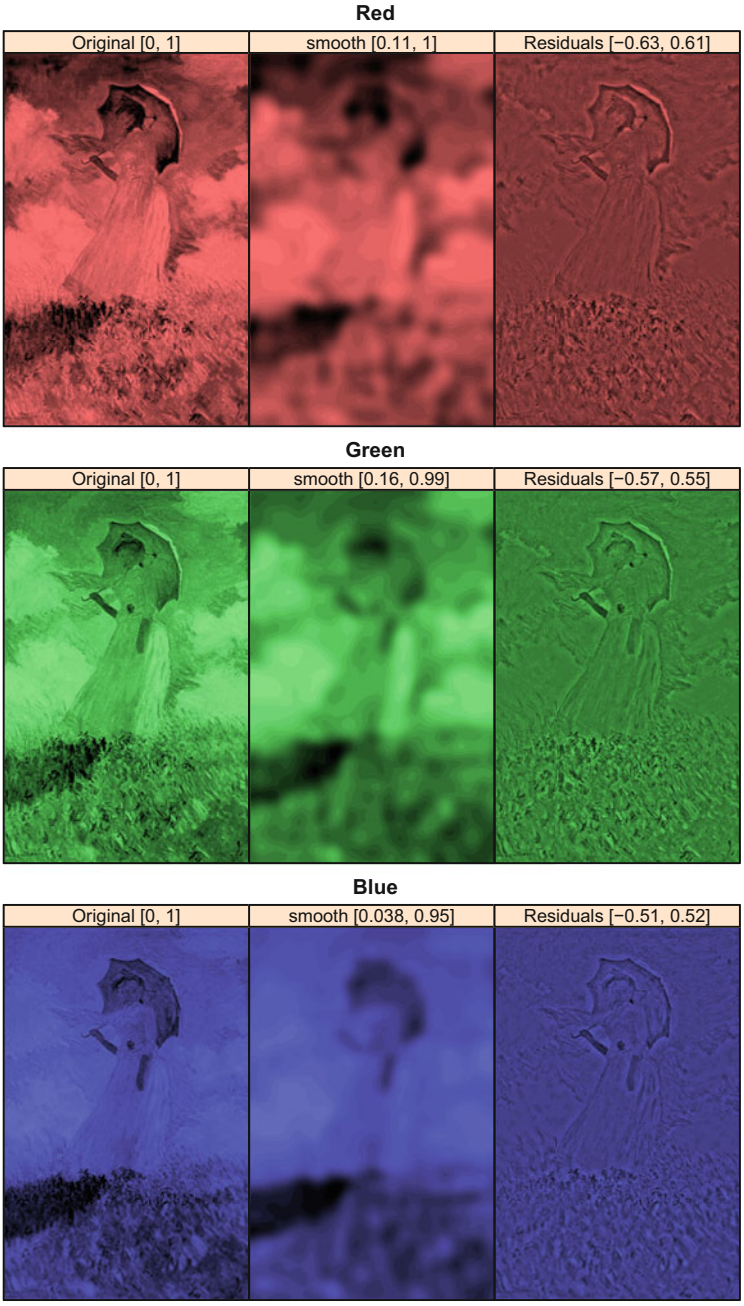

Fig. 5.24 "Monet": Three channels of the reconstructed and residual images

Assume that we want to look at fine details of the image. Then we should smooth the image by taking several leading components for reconstruction, say, 3, and look at the residual.

Figure 5.22 (right) contains a smoothed color image. Figure 5.24 contains three layers (channels) of the reconstructed and residual images in the b/w scale. The channels were extracted by means of indicating three splices by `k=1`, `k=2` and `k=3` in the third dimension.

In this example we have decomposed a beautiful painting of Monet. Decompositions of paintings can be useful to those studying different styles of painting and comparing or verifying painters. This can be done similarly to the face verification by 2D-SSA; this problem was discussed in Rodríguez-Aragón and Zhigljavsky (2010). The nD-SSA methods can be used, similarly to the 1D- and 2D-cases, for imputation of gaps in multidimensional objects and performing clustering, recognition, verification, etc.; all applications that need decompositions of objects, which may have complex geometry. All this can be done within the framework of nD-SSA by Rssa.

References

Ade F (1983) Characterization of textures by 'eigenfilters'. Signal Process 5(5):451–457

Berkeley Drosophila Transcription Network Project (2014) URL http://bdtnp.lbl.gov/Fly-Net/

Danilov D, Zhigljavsky A (eds) (1997) Principal components of time series: the "Caterpillar" method. St. Petersburg Press (in Russian)

Golyandina N, Usevich K (2009) An algebraic view on finite rank in 2D-SSA. In: Proceedings of the 6th St.Petersburg workshop on simulation, June 28–July 4. St. Petersburg, Russia, pp 308–313

Golyandina N, Usevich K (2010) 2D-extension of singular spectrum analysis: algorithm and elements of theory. In: Olshevsky V, Tyrtyshnikov E (eds) Matrix methods: theory, algorithms and applications. World Scientific, pp 449–473

Golyandina N, Nekrutkin V, Zhigljavsky A (2001) Analysis of time series structure: SSA and telated techniques. Chapman&Hall/CRC

Golyandina N, Florinsky I, Usevich K (2007) Filtering of digital terrain models by 2D singular spectrum analysis. Int J Ecol Dev 8(F07):81–94

Golyandina N, Korobeynikov A, Shlemov A, Usevich K (2015) Multivariate and 2D extensions of singular spectrum analysis with the Rssa package. J Stat Softw 67(2):1–78

Golyandina NE, Holloway DM, Lopes FJ, Spirov AV, Spirova EN, Usevich KD (2012b) Measuring gene expression noise in early *Drosophila* embryos: Nucleus-to-nucleus variability. In: Procedia Comput Sci, vol 9, pp 373–382

Hannachi A, Jolliffe IT, Stephenson DB (2007) Empirical orthogonal functions and related techniques in atmospheric science: A review. Int J Climatol 27(9):1119–1152

Hijmans RJ (2016) RASTER: Geographic Data Analysis and Modeling. URL http://CRAN.R-project.org/package=raster, R package version 2.5–8

Holloway DM, Lopes FJP, da Fontoura Costa L, Travençolo BAN, Golyandina N, Usevich K, Spirov AV (2011) Gene expression noise in spatial patterning: *hunchback* promoter structure affects noise amplitude and distribution in *Drosophila* segmentation. PLoS Comput Bioly 7(2):e1001,069

Korobeynikov A (2010) Computation- and space-efficient implementation of SSA. Stat Interface 3(3):357–368

Kurakin V, Kuzmin A, Mikhalev A, Nechaev A (1995) Linear recurring sequences over rings and modules. J Math Sci 76(6):2793–2915

Larsen RM (1998) Efficient algorithms for helioseismic inversion. PhD thesis, University of Aarhus, Denmark

Monadjemi A (2004) Towards efficient texture classification and abnormality detection. PhD thesis, University of Bristol

Mourrain B, Pan VY (2000) Multivariate polynomials, duality, and structured matrices. J Complexity 16(1):110–180

Oropeza V (2010) The singular spectrum analysis method and its application to seismic data denoising and reconstruction. Master Thesis in University of Alberta

Rodríguez-Aragón L, Zhigljavsky A (2010) Singular spectrum analysis for image processing. Stat Interface 3(3):419–426

Rouquette S, Najim M (2001) Estimation of frequencies and damping factors by two-dimensional ESPRIT type methods. IEEE Trans Signal Process 49(1):237–245

Roy R, Kailath T (1989) ESPRIT: estimation of signal parameters via rotational invariance techniques. IEEE Trans Acoust 37:984–995

Shlemov A, Golyandina N (2014) Shaped extensions of Singular Spectrum Analysis. In: 21st international symposium on mathematical theory of networks and systems, July 7–11, 2014. Groningen, The Netherlands, pp 1813–1820

Shlemov A, Golyandina N, Korobeynikov A, Zvonarev N, Spirov A (2014) BioSSA. URL http://biossa.github.io/

Shlemov A, Golyandina N, Holloway D, Spirov A (2015a) Shaped 3D singular spectrum analysis for quantifying gene expression, with application to the early *Drosophila* embryo. BioMed Res Int 2015(Article ID 986436):1–18

Shlemov A, Golyandina N, Holloway D, Spirov A (2015b) Shaped singular spectrum analysis for quantifying gene expression, with application to the early *Drosophila* embryo. BioMed Res Int 2015(Article ID 689745)

Trickett S (2008) F-xy Cadzow noise suppression. In: 78th annual international meeting, SEG, Expanded Abstracts, pp 2586–2590

Trickett SR (2003) F-xy eigenimage noise suppression. Geophysics 68(2):751–759

Wang Y, Chan JW, Liu Z (2005) Comments on 'Estimation of frequencies and damping factors by two-dimensional ESPRIT type methods'. IEEE Trans Sig Proc 53(8):3348–3349

Yamazaki I, Bai Z, Simon H, Wang LW, Wu K (2008) Adaptive projection subspace dimension for the thick-restart Lanczos method. Tech. rep., Lawrence Berkeley National Laboratory, University of California, One Cyclotron road, Berkeley, California 94720

Index

N. Golyandina et al., *Singular Spectrum Analysis with R*, Use R!,
https://doi.org/10.1007/978-3-662-57380-8