Journal of Statistical Software
October 2015, Volume 67, Issue 2. doi: 10.18637/jss.v067.i02

Multivariate and 2D Extensions of Singular Spectrum Analysis with the Rssa Package

Nina Golyandina
St. Petersburg State University

Anton Korobeynikov
St. Petersburg State University

Alex Shlemov
St. Petersburg State University

Konstantin Usevich
GIPSA-lab, CNRS

Abstract

Implementation of multivariate and 2D extensions of singular spectrum analysis (SSA) by means of the R package **Rssa** is considered. The extensions include MSSA for simultaneous analysis and forecasting of several time series and 2D-SSA for analysis of digital images. A new extension of 2D-SSA analysis called shaped 2D-SSA is introduced for analysis of images of arbitrary shape, not necessary rectangular. It is shown that implementation of shaped 2D-SSA can serve as a basis for implementation of MSSA and other generalizations. Efficient implementation of operations with Hankel and Hankel-block-Hankel matrices through the fast Fourier transform is suggested. Examples with code fragments in R, which explain the methodology and demonstrate the proper use of **Rssa**, are presented.

Keywords: singular spectrum analysis, time series, image processing, analysis, forecasting, decomposition, R package.

1. Introduction

Singular spectrum analysis as a method of time series analysis has a well-elaborated theory and solves various problems: e.g., time series decomposition, trend extraction, periodicity detection and extraction, signal extraction, denoising, filtering, forecasting, missing data imputation, change point detection, spectral analysis, among others(see examples and references in Vautard and Ghil 1989; Golyandina, Nekrutkin, and Zhigljavsky 2001; Ghil, Allen, Dettinger, Ide, Kondrashov, Mann, Robertson, Saunders, Tian, Varadi, and Yiou 2002; Golyandina and Zhigljavsky 2013). Since the method does not need a model given a priori, it is called nonparametric and is well suited for exploratory analysis of time series.

Additionally, SSA allows the construction of a model during or after exploratory analysis. The underlying parametric model of the signal is the sum of products of polynomial, exponential and sine-wave functions. This is a linear model in the following sense: such series constitute the class of solutions of linear differential equations. In the case of discrete time, such time series satisfy linear recurrent relations (LRRs). There is a class of so called subspace-based methods (Veen, Deprettere, and Swindlehurst 1993), which are related to estimation of parameters in the mentioned parametric model, in particular, to estimation of frequencies.

Although some problems like frequency estimation do need the model, some problems like smoothing do not need a model at all. For forecasting in SSA, the time series may satisfy the model approximately and locally, since the forecasting methods in SSA are based on estimation of the signal subspace and not on estimation of the parameters of the model. Depending on the quality of approximation of the series by the model, long or short horizon forecasts can be constructed.

SSA software

The aforementioned possibilities are implemented in different software packages based on different methodologies of SSA, see Golyandina and Korobeynikov (2014) for references. The sources of the methodology used in this paper are the books Golyandina *et al.* (2001); Golyandina and Zhigljavsky (2013) and the software from Gistat Group (2013), where SSA analysis and forecasting of one-dimensional and multivariate time series are implemented in an interactive manner. The same methodology is used and developed in the **Rssa** package (Korobeynikov, Shlemov, Usevich, and Golyandina 2015) for the R system for statistical computing (R Core Team 2015); see Golyandina and Korobeynikov (2014), where analysis, forecasting and parameter estimation by means of **Rssa** for one-dimensional series are described.

At the present time, package **Rssa** is extensively developed and includes SSA-processing of one-dimensional time series (basic SSA, or simply SSA), systems of series (multivariate or multichannel SSA, shortened to MSSA) and of digital images (2D-SSA). The aim of this paper is to describe this multidimensional part of the **Rssa** package as of version 0.10. Note that the effective implementation of the algorithms of multidimensional SSA extensions is very important, since the algorithms are much more time-consuming than in the one-dimensional case. Therefore, we pay attention to both the guidelines for the proper use of the package and the efficient implementation techniques, which are based on the methods described in Korobeynikov (2010). In addition, a new method called shaped 2D-SSA (ShSSA for short) is introduced. This method can be applied to the images of non-rectangular form, e.g., circular images, images with gaps and so on. Also, it is shown that the implementation of shaped 2D-SSA can serve as a common basis for implementation of many SSA extensions.

General scheme of SSA

Let us introduce a general scheme of SSA-like algorithms, including MSSA and 2D-SSA. The SSA-like algorithms decompose the given data $\mathbb{X}$ into a sum of different components:

$$\mathbb{X} = \mathbb{X}_1 + \cdots + \mathbb{X}_m. \tag{1}$$

A typical SSA decomposition of a time series is the decomposition into slowly-varying trend, seasonal components and residual or the decomposition into some pattern, regular oscillations and noise. The input data can be a time series, a multivariate time series, or a digital image.

The algorithm is divided into four steps. The first step is generation of a multivariate object on the basis of the initial object (time series or image) by moving a *window* of some form and taking the elements from the window. For one-dimensional time series, this window is an interval that produces subseries of length L of the time series, where L is called window length (in SSA). For multivariate time series (a system of s one-dimensional series), the window also produces subseries of length L, but we apply this window to all time series of the system (in MSSA). For images, the window can be a 2D rectangle (in 2D-SSA) or some other 2D shape (in ShSSA). Then all the subobjects (subseries or vectorized 2D shapes) obtained by applying the window are stacked as columns into the *trajectory* matrix. The trajectory matrix has a specific structure: Hankel, stacked Hankel, Hankel-block-Hankel or quasi-Hankel.

The second step consists in decomposition of the trajectory matrix into a sum of elementary matrices of rank 1. The most frequently used decomposition, which has a lot of optimal approximation properties, is the singular value decomposition (SVD).

The third step is grouping of the decomposition components. At the grouping step, the elementary rank-one matrices are grouped and summed within groups.

The last and forth step converts the grouped matrix decomposition back to the decomposition of the initial time series or image decomposition. The elements of each component of the grouped decomposition are obtained by averaging the entries of the grouped matrices that correspond to the same element in the initial object.

Thus, the result of the algorithm is the decomposition (1) of the initial object into the sum of objects. We assume that the initial object is a sum of some identifiable components, for example, trend and seasonality or signal and noise and that we observe only the sum. Then the aim of the SSA-like methods is to reconstruct these components. The possibility to reconstruct the object components is called *separability* of the components.

Complex SSA is the same as the basic SSA but it is applied to complex-valued one-dimensional time series with complex-valued SVD and therefore fits well into the described scheme. Certainly, multivariate complex SSA and 2D complex SSA can be considered in similar manner, but it is out the scope of this paper. Note that complex SSA can be applied to a system of two real-valued time series considered as real and imaginary parts of a complex series.

We mentioned that a model of the series can be constructed during the SSA processing. The simplified form of the signal model is $s_n = \sum_{k=1}^{r} A_k \mu_k^n$, where $\mu_k = \rho_k e^{\mathrm{i}2\pi\omega_k}$ (possible polynomial modulation is omitted). The general approach for estimation of the modulations ρ_k and frequencies ω_k in this model is to use the so-called signal subspace, which can be estimated at the third step of the SSA algorithm. One of the subspace-based methods for parameter estimation is ESPRIT (Roy and Kailath 1989). This approach has a 2D extension named 2D-ESPRIT for estimation of parameters in the model $s_{ln} = \sum_{k=1}^{r} A_k \mu_k^l \nu_k^n$.

Rssa *and related packages*

The range of problems solved by SSA and its multidimensional extensions is very wide, as can be seen from the brief review of SSA capabilities given above. That is why we do not consider in the scope of the paper the comparison of **Rssa** with other packages implementing decomposition, filtering, regression, frequency estimation, etc. We refer the reader to the CRAN (Comprehensive R Archive Network) task view on "Time Series Analysis" (Hyndman

2015) for a review of packages for processing of time series and to the CRAN task view on "Analysis of Spatial Data" (Bivand 2015) for a review of packages for processing of spatial data. As an implementation of the SSA method, **Rssa** is the only package currently available from CRAN, where it is accessible at `http://CRAN.R-project.org/package=Rssa`.

Let us put attention to several issues related to the difference between SSA and many other methods and algorithms implemented in various R packages. SSA-like methods can be applied as model-free approaches that do not need to know parametric models and the periods of possible periodic components in advance. This distinguishes them from methods like seasonal decomposition and parametric regression. Also, an important point is that in SSA-like methods it is not essential for data to have an additive or multiplicative structure. In addition, note that although the model of series governed by linear recurrence relations may seem similar to the autoregressive models, they have nothing in common.

The present version of the **Rssa** package deals with equidistant series and spatial data given on a rectangular grid. For processing of data given on an irregular grid, two approaches can be used. Certainly, interpolation to a rectangular grid can be performed (it is not implemented in the current version of the package). For univariate and multivariate series, one can formally deal with non-equidistant data as with equidistant measurements, ignoring the irregularity.

For processing of time series or of systems of time series, **Rssa** can deal with vectors, lists of vectors, matrices and also with input data of classes '`ts`', '`mts`', '`zooreg`' (Zeileis and Grothendieck 2005) – also, '`zoo`' data can be used, but the contents of the index attribute of a '`zoo`' object is simply ignored. The package tries to preserve attributes of input data, making operations with the decomposition results convenient: no conversion to the original class is required.

For processing of multidimensional data like digital images, **Rssa** takes only matrices as input data. In theory, 2D-SSA can be used with data of different nature: spatial, spatio-temporal or arbitrary digital images. However, the R implementation of objects of these types usually carries with itself a lot of additional information. It seems to be error-prone to try to preserve such information in a generic way, and it also would introduce many spurious dependencies of **Rssa** on other packages. Thus, it is considered as a job of the user to convert the results of the decomposition back to the appropriate form (see Fragment 23 for an example with objects from the **raster** package; Hijmans 2015).

Most of the plotting capabilities of the package are implemented via package **lattice** (Sarkar 2008). Therefore, the user can either plot **lattice** objects returned by `plot` functions directly, or use the capabilities of **lattice** to manipulate these plot objects in a convenient way. Also, since the input time series classes may vary, **Rssa**, where possible, allows one to fall back to native plotting functions of such classes.

Finally we dwell on the issues related to the efficient implementation of SSA-related methods in the **Rssa** package. The two main points that underlie the implementation are (1) the use of fast methods for the singular value decomposition and (2) the use of fast multiplication of Hankel-related matrices and matrix hankelization by means of the fast Fourier transform (FFT), see Korobeynikov (2010). However, the direct usage of the algorithms possessing good theoretical complexity will not automatically yield an efficient implementation per se. The package also implements many practical considerations which make the core SSA steps really fast. Besides this, many of the functions in the package employ on-demand calculation of the necessary objects and memoization. These points are very important for large window sizes and long time series, and are crucial for 2D-SSA and ShSSA.

Contribution of the paper

On the theoretical/algorithmic side, this paper has several new contributions. First, we describe a fast algorithm of the vector SSA forecasting and provide its implementation. Second, we introduce a new extension of 2D-SSA, shaped 2D-SSA, which should extend the application area of the method in digital image processing. Then, fast FFT-based algorithms are described and implemented for each considered SSA-like method. Finally, we show that all the considered extensions of SSA (and several related extensions) are special cases of shaped 2D-SSA.

Structure of the paper

The structure of the paper is as follows. Section 2 contains an overview of the SSA approach, including the common scheme of the algorithms, general notions, and a summary of implementation details. The multivariate version of SSA for analysis and forecasting of systems of series is described in Section 3. 2D-SSA for image processing together with 2D-ESPRIT for parameter estimation is considered in Section 4. Section 5 is devoted to ShSSA for analysis of images of non-rectangular form. Section 6 contains details on the fast algorithms, including FFT-based matrix multiplication, hankelization and the fast algorithm of the vector forecasting. Theoretical details of MSSA and 2D-SSA are put in Appendix A and B

Each of Sections 3–5 contains the implemented algorithms, description of the **Rssa** functionality with typical R code, simulated and real-life examples with the corresponding fragments of R code accompanied by guidelines for the proper use. In Section 5.3, it is shown that all the considered SSA extensions are special cases of ShSSA. Thus the fast FFT-based algorithms in Section 6 are provided only for shaped 2D-SSA.

This paper contains a comprehensive description of multivariate extensions of SSA. Although it is impossible to consider all applications of multivariate extensions, we provide examples for the most common ones, with code fragments and references to the literature. We also demonstrate plotting capabilities of the package which are necessary for the proper use of the SSA methodology and representation of the results.

2. Common structure of algorithms and general notions

Before introducing the details of the SSA-like algorithms and the corresponding notions, we present a general scheme of the algorithms in Figure 1.

The algorithms consist of four steps. The input data are the object $\mathbb{X}$ and the mapping $\mathcal{T}$. The result of the algorithms is a decomposition of $\mathbb{X}$ into sum of additive components. The intermediate outcomes of the steps can be used for solving additional problems like forecasting and parameter estimation. Table 1 contains a summary of SSA extensions considered in the paper. Further we discuss the steps of the algorithms of the SSA-like methods in detail.

2.1. Details of the algorithms

Step 1. Embedding. The first step consists in the construction of the so-called *trajectory matrix* $\mathbf{X} = \mathcal{T}(\mathbb{X})$ by means of a map $\mathcal{T}$.

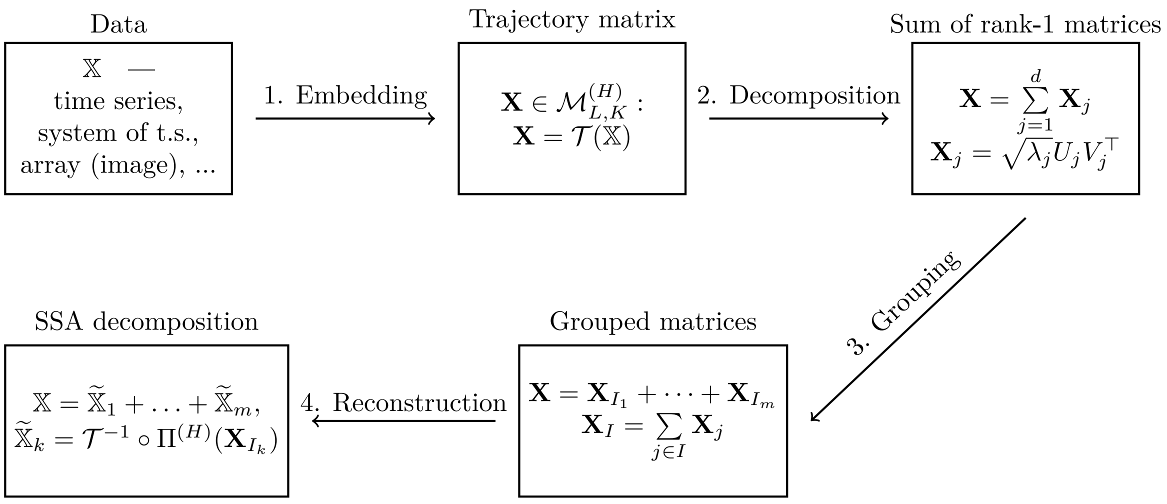

Data
$\mathbb{X}$ — time series, system of t.s., array (image), ...
1. Embedding
Trajectory matrix
$\mathbf{X} \in \mathcal{M}^{(H)}_{L,K}$: $\mathbf{X} = \mathcal{T}(\mathbb{X})$
2. Decomposition
Sum of rank-1 matrices
$\mathbf{X} = \sum_{j=1}^{d} \mathbf{X}_j$
$\mathbf{X}_j = \sqrt{\lambda_j} U_j V_j^\top$
3. Grouping
Grouped matrices
$\mathbf{X} = \mathbf{X}_{I_1} + \dots + \mathbf{X}_{I_m}$
$\mathbf{X}_I = \sum_{j \in I} \mathbf{X}_j$
4. Reconstruction
SSA decomposition
$\mathbb{X} = \widetilde{\mathbb{X}}_1 + \ldots + \widetilde{\mathbb{X}}_m,$
$\widetilde{\mathbb{X}}_k = \mathcal{T}^{-1} \circ \Pi^{(H)}(\mathbf{X}_{I_k})$

Figure 1: Scheme of the SSA-like algorithms.

Method	Data	Notation	Trajectory matrix	Section
SSA	time series	$\mathbb{X} = (x_1, \ldots, x_N)$	Hankel	2.1
MSSA	system of time series	$\mathbb{X}^{(p)}$, $p = 1, \ldots, s$	Stacked Hankel	3
CSSA	complex time series	$\mathbb{X}^{(1)} + \mathrm{i}\, \mathbb{X}^{(2)}$	Complex Hankel	3
2D-SSA	rectangular image	$\mathbb{X} = (x_{ij})_{i,j=1}^{N_x,N_y}$	Hankel-block-Hankel	4
ShSSA	shaped image	$\mathbb{X} = (x_{(i,j)})_{(i,j)\in\mathfrak{N}}$	Quasi-Hankel	5

Table 1: Kinds of SSA-like multivariate extensions.

In the basic SSA algorithm applied to a one-dimensional time series $\mathbb{X} = (x_1, \ldots, x_N)$ of length N, $\mathcal{T}$ maps R^N to the space of Hankel matrices $L \times K$ with equal values on anti-diagonals:

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

where L is the window length and $K = N - L + 1$.

Let us describe the scheme of this step in a more formal manner for the general case of SSA-like methods. Denote $\mathcal{M}_{p,q}$ the set of $p \times q$ real matrices. Let $\mathcal{M}$ be the linear space of all possible $\mathbb{X}$, where $\mathbb{X}$ is an ordered set of real values representing initial data (i.e., a time series or image). Then the *embedding* $\mathcal{T}$ is a one-to-one mapping from $\mathcal{M}$ to $\mathcal{M}^{(H)}_{L,K} \subset \mathcal{M}_{L,K}$, where $\mathcal{M}^{(H)}_{L,K}$ is the set of matrices with a Hankel-like structure, which is determined together with L and K by the method parameters.

The mapping $\mathcal{T}$ puts elements of $\mathbb{X}$ in some places of the trajectory matrix $\mathbf{X} = \mathcal{T}(\mathbb{X})$. Let x_k denote the kth element of $\mathbb{X}$. (The index k may be two-dimensional, see e.g., Table 1.) Then the mapping $\mathcal{T}$ determines the set of indices $\mathcal{A}_k$ such that $(\mathbf{X})_{ij} = x_k$ for any $(i, j) \in \mathcal{A}_k$.

Formally, let $E_k \in \mathcal{M}$ be the object with the kth element equal to 1 and all the other elements equal to zero. Then the set $\mathcal{A}_k$ corresponds to the places of "1" in the matrix $\mathcal{T}(E_k)$.

For example, in the basic SSA algorithm applied to a one-dimensional time series $\mathbb{X}$, $\mathcal{M} = \mathsf{R}^N$, $\mathcal{M}^{(H)}_{L,K}$ is the set of Hankel matrices and the mapping $\mathcal{T}$ is given in (2). The objects E_k, $k = 1, \ldots, N$, are the standard unit vectors in R^N, and $\mathcal{A}_k$ consists of the positions of the elements on the kth anti-diagonal of $\mathbf{X}$.

Step 2. Singular value decomposition. Let $\mathbf{S} = \mathbf{X}\mathbf{X}^\top$, $\lambda_1 \geq \ldots \geq \lambda_L \geq 0$ be *eigenvalues* of the matrix $\mathbf{S}$, $d = \max\{j : \lambda_j > 0\}$, $U_1, \ldots, U_d$ be the corresponding *eigenvectors*, and $V_j = \mathbf{X}^\top U_j / \sqrt{\lambda_j}$, $j = 1, \ldots, d$, be the *factor vectors*. Denote $\mathbf{X}_j = \sqrt{\lambda_j} U_j V_j^\top$. Then the SVD of the trajectory matrix $\mathbf{X}$ can be written as

$$\mathbf{X} = \mathbf{X}_1 + \ldots + \mathbf{X}_d. \tag{3}$$

The values $\sqrt{\lambda_j}$ are exactly the singular values of $\mathbf{X}$. The vectors U_j (respectively, V_j) are exactly the left (respectively, right) singular vectors of $\mathbf{X}$. The triple $(\sqrt{\lambda_j}, U_j, V_j)$ is called the *jth eigentriple* (or ETj for short).

Step 3. Grouping. Once the expansion (3) has been obtained, the grouping procedure partitions the set of indices $\{1, \ldots, d\}$ into m disjoint subsets $I_1, \ldots, I_m$. For a subset $I = \{i_1, \ldots, i_p\}$, the matrix $\mathbf{X}_I$ corresponding to the group I is defined as $\mathbf{X}_I = \mathbf{X}_{i_1} + \ldots + \mathbf{X}_{i_p}$. Thus, we have the *grouped matrix decomposition*

$$\mathbf{X} = \mathbf{X}_{I_1} + \ldots + \mathbf{X}_{I_m}. \tag{4}$$

For example, grouping can be performed on the basis of the form of the eigenvectors, which reflect the properties of initial data components. The grouping with $I_j = \{j\}$ is called *elementary*.

Step 4. Reconstruction. At this final step, each matrix of the decomposition (4) is transferred back to the form of the input object $\mathbb{X}$. It is performed optimally in the following sense: for a matrix $\mathbf{Y} \in \mathcal{M}_{L,K}$ we seek for the object $\widetilde{\mathbb{Y}} \in \mathcal{M}$ that provides the minimum to $\|\mathbf{Y} - \mathcal{T}(\widetilde{\mathbb{Y}})\|_{\mathcal{F}}$, where $\|\mathbf{Z}\|_{\mathcal{F}} = \sqrt{\sum_{i,j=1}^{L,K} (\mathbf{Z})^2_{ij}}$ is the Frobenius norm.

Denote $\Pi^{(H)} : \mathcal{M}_{L,K} \to \mathcal{M}^{(H)}_{L,K}$ the orthogonal projection on $\mathcal{M}^{(H)}_{L,K}$ in Frobenius norm. Then $\widetilde{\mathbb{Y}} = \mathcal{T}^{-1}(\Pi^{(H)}\mathbf{Y})$. By the embedding nature of $\mathcal{T}$, the projection $\Pi^{(H)}$ is the averaging of the entries along the sets $\mathcal{A}_k$, that is, the elements of $\widetilde{\mathbb{Y}}$ are equal to $\widetilde{y}_k = \sum_{(i,j)\in\mathcal{A}_k} (\mathbf{Y})_{ij} / |\mathcal{A}_k|$, where $|\mathcal{A}_k|$ is the number of elements in $\mathcal{A}_k$. The same averaging can be rewritten as $\widetilde{y}_k = \langle \mathbf{Y}, \mathcal{T}(E_k)\rangle_{\mathcal{F}} / \|\mathcal{T}(E_k)\|^2_{\mathcal{F}}$, where $\langle \mathbf{Y}, \mathbf{Z}\rangle_{\mathcal{F}} = \sum_{i,j=1}^{L,K} (\mathbf{Y})_{ij} (\mathbf{Z})_{ij}$ is the Frobenius inner product.

In basic SSA, the set $\mathcal{A}_k = \{(i,j) : 1 \leq i \leq L, 1 \leq j \leq K, i+j = k+1\}$ corresponds to the kth anti-diagonal, and the composite mapping $\mathcal{T}^{-1} \circ \Pi^{(H)}$ is the averaging along anti-diagonals. Hence, in the SSA literature the reconstruction step is often called *diagonal averaging*.

Thus, denote $\widehat{\mathbf{X}}_k = \mathbf{X}_{I_k}$ the reconstructed matrices, $\widetilde{\mathbf{X}}_k = \Pi^{(H)} \widehat{\mathbf{X}}_k$ the trajectory matrices of the reconstructed data and $\widetilde{\mathbb{X}}_k = \mathcal{T}^{-1}(\widetilde{\mathbf{X}}_k)$ the reconstructed data themselves. Then the resultant decomposition of the initial data has the form

$$\mathbb{X} = \widetilde{\mathbb{X}}_1 + \ldots + \widetilde{\mathbb{X}}_m. \tag{5}$$

If the grouping is elementary, then $m = d$, $\mathbf{X}_{I_k} = \mathbf{X}_k$, and the reconstructed objects $\widetilde{\mathbb{X}}_k = \mathcal{T}^{-1} \circ \Pi^{(H)}\mathbf{X}_k$ are called *elementary components.*

2.2. General notions

Separability

A very important notion in SSA is separability. Let $\mathbb{X} = \mathbb{X}_1 + \mathbb{X}_2$. (Approximate) separability means that there exists such a grouping that the reconstructed series $\widetilde{\mathbb{X}}_1$ is (approximately) equal to $\mathbb{X}_1$. Properties of the SVD yield (approximate) orthogonality of columns and orthogonality of rows of trajectory matrices $\mathbf{X}_1$ and $\mathbf{X}_2$ of $\mathbb{X}_1$ and $\mathbb{X}_2$ as the separability condition. There is a well-elaborated theory of separability of one-dimensional time series (Golyandina *et al.* 2001, Sections 1.5 and 6.1). It appears that many important decomposition problems, from noise reduction and smoothing to trend, periodicity or signal extraction, can be solved by SSA. Certainly, the embedding operator $\mathcal{T}$ determines separability conditions. It could be said that the success of SSA in separability is determined by the Hankel structure of the trajectory matrix and optimality features of the SVD.

Information for grouping

The theory of SSA provides various ways to detect the SVD components related to the series component in order to perform proper grouping in conditions of separability. One of the rules is that the eigenvector produced by a data component repeats the properties of this component. For example, in 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 frequencies, and so on. These properties help to perform the grouping by visual inspection of eigenvectors and also by some automatic procedures (see Alexandrov 2009 and Golyandina and Zhigljavsky 2013, Section 2.4.5).

To check separability of the reconstructed components $\widetilde{\mathbb{X}}_1$ and $\widetilde{\mathbb{X}}_2$, we should check the 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 $\langle \widetilde{\mathbf{X}}_1, \widetilde{\mathbf{X}}_2 \rangle_{\mathcal{F}}$. The normalized measure of orthogonality is $\rho(\widetilde{\mathbf{X}}_1, \widetilde{\mathbf{X}}_2) = \langle \widetilde{\mathbf{X}}_1, \widetilde{\mathbf{X}}_2 \rangle_{\mathcal{F}} / (\|\widetilde{\mathbf{X}}_1\|_{\mathcal{F}} \|\widetilde{\mathbf{X}}_2\|_{\mathcal{F}})$.

Since the trajectory matrix consists of $w_k = |\mathcal{A}_k| = \|\mathcal{T}(E_k)\|^2_{\mathcal{F}}$ entries corresponding to the kth element x_k of the initial object, we can introduce the *weighted inner product* in the space $\mathcal{M}$: $(\mathbb{Y}, \mathbb{Z})_{\mathbf{w}} = \sum_k w_k y_k z_k$, which induces the *weighted norm* $\|\mathbb{Y}\|_{\mathbf{w}} = \sqrt{(\mathbb{Y}, \mathbb{Y})_{\mathbf{w}}}$. Then

$$\rho_{\mathbf{w}}(\widetilde{\mathbb{X}}_1, \widetilde{\mathbb{X}}_2) = \rho(\widetilde{\mathbf{X}}_1, \widetilde{\mathbf{X}}_2) = \frac{(\widetilde{\mathbb{X}}_1, \widetilde{\mathbb{X}}_2)_{\mathbf{w}}}{\|\widetilde{\mathbb{X}}_1\|_{\mathbf{w}} \|\widetilde{\mathbb{X}}_2\|_{\mathbf{w}}} \tag{6}$$

is called $\mathbf{w}$ *correlation* by statistical analogy. Note that in this definition 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 $\rho^{(\mathbf{w})}_{ij} = \rho_{\mathbf{w}}(\widetilde{\mathbb{X}}_i, \widetilde{\mathbb{X}}_j)$ is called $\mathbf{w}$ *correlation matrix.*

The weighted norm $\|\cdot\|_{\mathbf{w}}$ serves as a measure of contribution of components to the decomposition (5): the contribution of $\widetilde{\mathbb{X}}_j$ is defined as $\|\widetilde{\mathbb{X}}_j\|^2_{\mathbf{w}} / \|\mathbb{X}\|^2_{\mathbf{w}}$.

Trajectory spaces and signal subspace

Let us introduce several notions related to subspaces generated by the data. For the data $\mathbb{X}$ the column (row) subspace of its trajectory matrix $\mathbf{X}$ is called *column (row) trajectory space*. 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 data in applications the trajectory spaces coincide with the corresponding Euclidean spaces, since they are produced by a signal corrupted by noise. However, if the signal has a rank-deficient trajectory matrix, then the signal trajectory space can be called "signal subspace". Column and row signal subspaces can be considered. Note that the dimensions of row and column subspaces coincide.

Objects of finite rank

The class of objects that suit the SSA method are the so-called objects of finite rank. We say that the object (time series or image) has rank r if the rank of its trajectory matrix is equal to $r < \min(L, K)$, that is, the trajectory matrix is rank-deficient. If the 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 has finite rank (rank does not tend to infinity as the size of the object tends to infinity), see Golyandina *et al.* (2001, Chapter 5) and Golyandina and Usevich (2010) for rigorous definitions.

Since the trajectory matrices considered in SSA methods are Hankel or consist of Hankel blocks, the rank-deficient Hankel matrices are closely related to objects satisfying some linear relations. These linear relations can be taken as a basis for forecasting algorithms. In the one-dimensional case, rank-deficient Hankel matrices are closely related to linear recurrent relations $x_n = \sum_{i=1}^{r} a_i x_{n-i}$ and therefore to time series which can be expressed as a sum of products of exponentials, polynomials and sinusoids.

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 group the corresponding SVD components (whose number is equal to the rank value). For example, to reconstruct the exponential trend in the one-dimensional case, we need to group only one SVD component (the exponential has rank 1), while to reconstruct a sine wave we generally need to group two SVD components (the rank equals 2).

Surely, most of real-life time series or images are not of finite rank. For example, a time series can be a sum of a signal of rank r and noise. Then, due to approximate separability, we can use SSA to extract the signal and then apply methods designed for series of finite rank.

2.3. Typical code

The general structure of the **Rssa** package is described in Golyandina and Korobeynikov (2014) and holds for the multivariate extensions. Therefore, let us briefly discuss the base **Rssa** functions for analysis and forecasting of one-dimensional time series. Implementation of SSA analysis and forecasting is mostly contained in the functions `ssa` (decomposition), `reconstruct` (reconstruction), `predict`, `rforecast` and `vforecast` (forecasting), `plot` (plotting), `wcor` (weighted correlations for grouping).

For demonstration, we consider the series of sales of fortified wines (shortly FORT) taken from the dataset "Monthly Australian wine sales: thousands of litres. By wine makers in bottles $\leq$

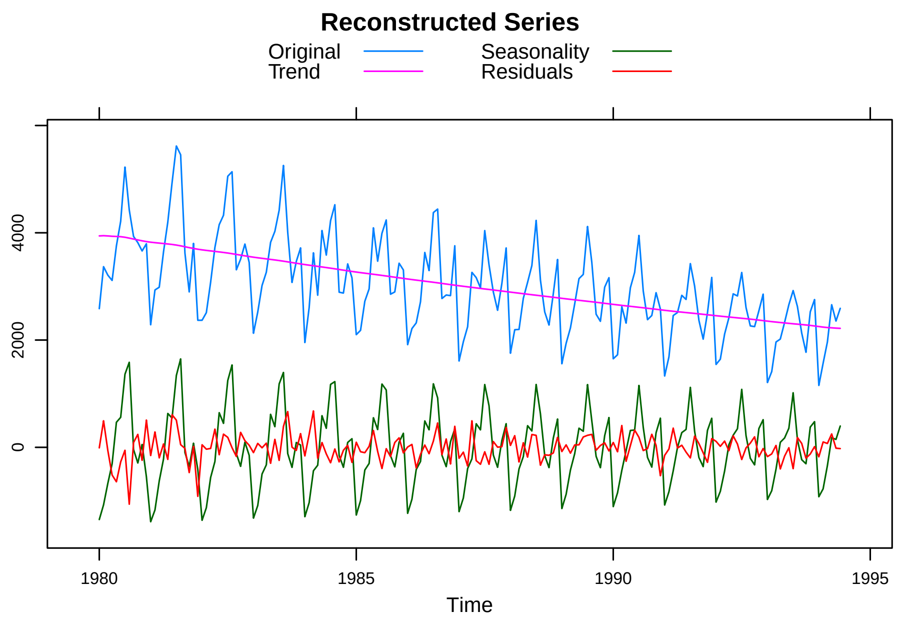


Figure 2: FORT: Decomposition.

1 litre" (Hyndman 2013). The full dataset contain sales from January, 1980, to July, 1995 (187 points). However, the data after June, 1994 have missing values. Therefore, we begin with the first 174 points.

Fragment 1 contains the standard code for loading the package **Rssa** and input of the data included into the package.

Fragment 1 (Australian wines: Input)**.**

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

Fragment 2 contains a typical code for extraction of the trend and seasonality. The resultant decomposition is depicted in Figure 2.

Fragment 2 (FORT: Reconstruction)**.**

```
R> fort <- wine[, "Fortified"]
R> s.fort <- ssa(fort, L = 84, kind = "1d-ssa")
R> r.fort <- reconstruct(s.fort, groups = list(Trend = 1,
+    Seasonality = 2:11))
R> 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 2014), `ssa` performs steps 1 and 2 of the algorithm described in Section 2.1, 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. Note that the function `plot` for the reconstruction object implements different special kinds of its plotting. In Fragment 2, the last two parameters of `plot` are the parameters of the function `xyplot` from the package **lattice**.

The grouping for reconstruction was made on the basis of the following information obtained from the '`ssa`' object:

1. one-dimensional (1D) figures of eigenvectors U_i (Figure 3),
2. two-dimensional (2D) figures of eigenvectors (U_i, U_{i+1}) (Figure 4), and,
3. matrix of **w** correlations $\rho_{\mathbf{w}}$ between elementary reconstructed series (functions `wcor` and `plot`, Figure 5).

The following fragment shows the code that reproduces Figures 3–5.

Fragment 3 (FORT: Identification)**.**

```
R> plot(s.fort, type = "vectors", idx = 1:8)
R> plot(s.fort, type = "paired", idx = 2:11, plot.contrib = FALSE)
R> parestimate(s.fort, groups = list(2:3, 4:5), method = "esprit-ls")

$F1
   period      rate  |    Mod     Arg  |     Re        Im
   12.003  -0.006572 |  0.99345   0.52 |  0.86042   0.49661
  -12.003  -0.006572 |  0.99345  -0.52 |  0.86042  -0.49661
$F2
   period      rate  |    Mod     Arg  |     Re        Im
    4.005   0.000037 |  1.00004   1.57 |  0.00189   1.00003
   -4.005   0.000037 |  1.00004  -1.57 |  0.00189  -1.00003

R> plot(wcor(s.fort, groups = 1:30), scales = list(at = c(10, 20, 30)))
R> plot(reconstruct(s.fort, add.residuals = FALSE, add.original = FALSE,
+    groups = list(G12 = 2:3, G4 = 4:5, G6 = 6:7, G2.4 = 8:9)))
```

Let us explain how the figures obtained by means of Fragment 3 can help to perform the grouping. Figure 3 shows that the first eigenvector is slowly-varying and therefore the eigentriple (abbreviated as ET) ET1 should be included in the trend group. Figure 4 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 are similar to regular 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 Figure 4, 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` and are described in Golyandina *et al.* (2001, Sections 2.4.2.4. and 3.8.2) and Golyandina and Korobeynikov (2014)

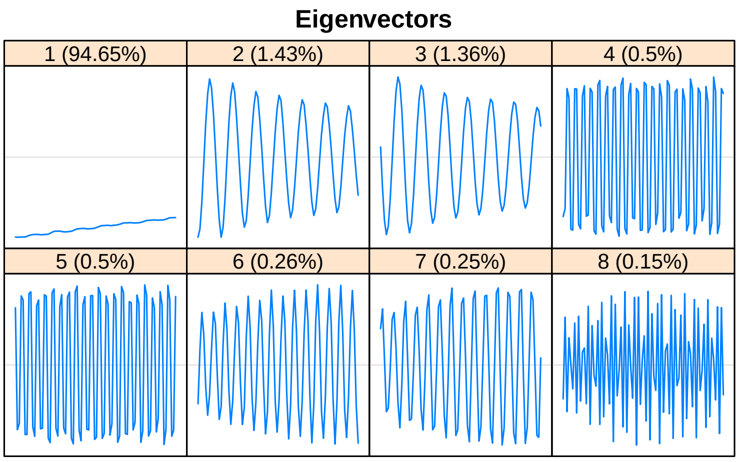


Figure 3: FORT: 1D graphs of eigenvectors.

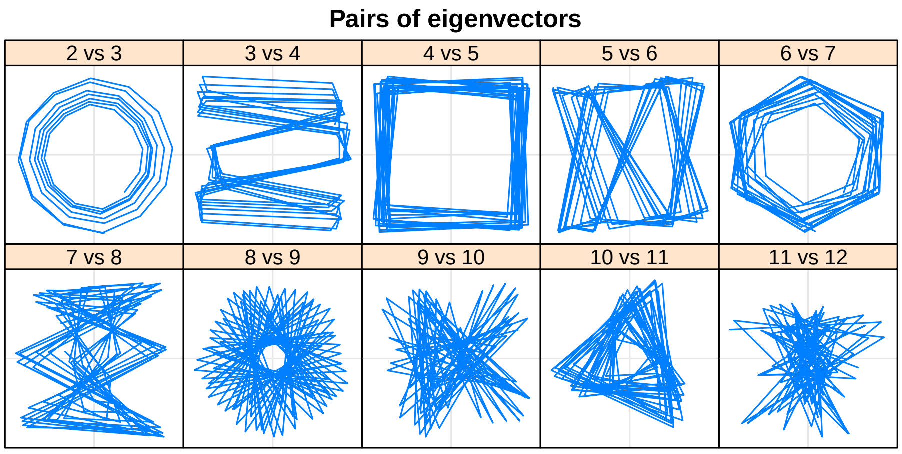


Figure 4: FORT: 2D scatterplots of eigenvectors.

for one-dimensional time series. The periods calculated by the automatic `parestimate` method in Fragment 3 agree with the numbers of vertices in Figure 4 for the five pairs listed.

The matrix of absolute values of **w** correlations in Figure 5 is depicted in gray scale (white color corresponds to zero values, while black color corresponds to the absolute values equal to 1). Figure 5 confirms that the indicated pairs are separated between themselves and also from the trend component, since the correlations between the pairs are small, while correlations between the components from one pair are very large. The block of 12–84 components is “gray”, therefore we can expect that these components are mixed and are produced by noise.

Figure 6 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 Figure 3. In Figure 2, we grouped the modulated sine waves and obtained the seasonal component with varying form.

Finally, Fragment 4 contains an example of forecasting the series components (trend and signal), and the result is depicted in Figure 7. Two forecasting methods are implemented for one-dimensional time series: recurrent (function `rforecast`) and vector forecasting (function

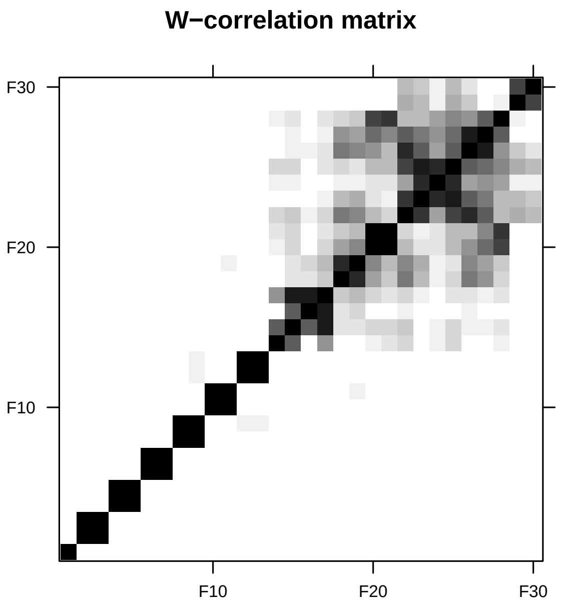


Figure 5: FORT: Weighted correlations.

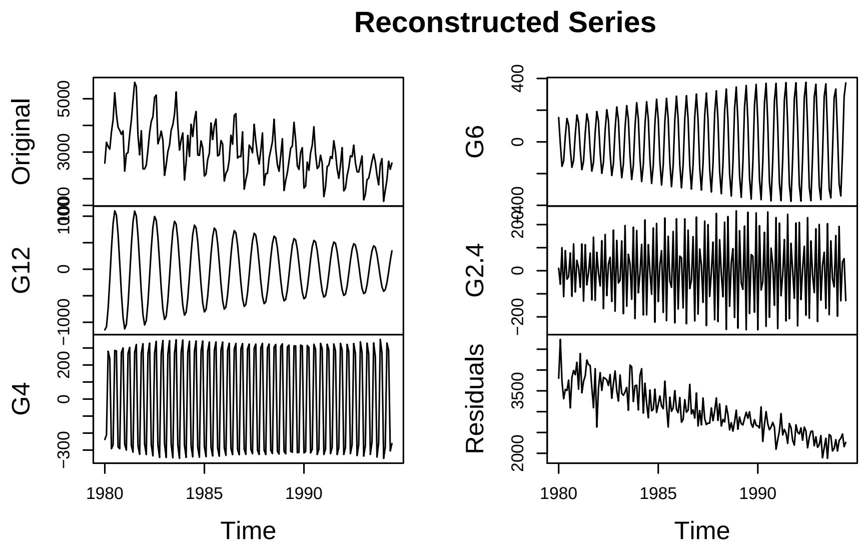


Figure 6: FORT: Reconstructed sine waves.

`vforecast`). Both forecasting methods are based on estimating linear recurrent relations that govern time series, and are described in detail in Section 3.3.

In Fragment 4, the function `vforecast` is used. Alternatively, one can use the all-in-one `predict` wrapper.

Fragment 4 (FORT: Forecast)**.**

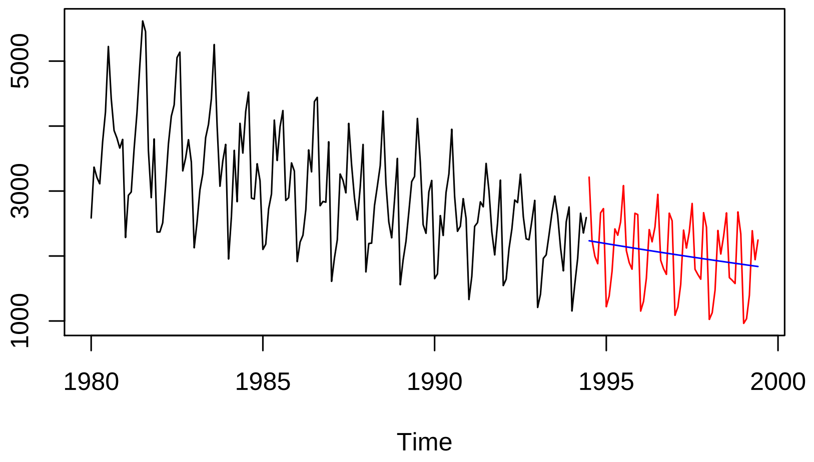


Figure 7: FORT: Forecasts of trend and signal.

```
R> f.fort <- vforecast(s.fort, groups = list(Trend = 1, Signal = 1:11),
+    len = 60, only.new = TRUE)
R> plot(cbind(fort, f.fort$Signal, f.fort$Trend), plot.type = "single",
+    col = c("black", "red", "blue"), ylab = NULL)
```

2.4. Comments on Rssa

The detailed description of the **Rssa** package structure and the principles of implementation is contained in Golyandina and Korobeynikov (2014). It is related to the use of **Rssa** for the SSA processing of one-dimensional time series, but most of the given information is also valid for other considered types of data. The typical code demonstrates the main logic of the SSA processing by **Rssa** and thereby the structure of functions from the package. Below we briefly comment on the most important issues and main differences with the one-dimensional case. More details are contained in the corresponding sections as comments to the typical code or the package.

Formats of input and output data

The inputs of SSA can be quite different depending on the kind of SSA used. For example, the inputs can be vectors, '`ts`' objects, matrices, data frames, lists of vector-like objects of different lengths, images in the form of matrices. For shaped 2D-SSA, the images can contain missing `NA` values. For MSSA the missing values can be used to construct the series of different lengths.

The routines in the packages are designed in a such way that they preserve all the attributes (shape, time scale, etc.) of the input object. So, the result of the reconstruction step is exactly of the same type as the input object was. This can be seen in Figure 6, where the `plot` method for '`ts`' objects (corresponding to the default `plot.method = "native"`) was used to draw the result of the reconstruction step.

All the forecasting routines try to use the attributes of the input object for the resulting object (in particular, they try to add the time scale to the result). Unfortunately, this cannot be done in a class-neutral way, as it is done in the reconstruction case, and needs to be handled separately for each possible type of the time series classes. The forecasting routines know how to impute the time indices for some standard time series classes like ‘`ts`’ or ‘`mts`’.

Plotting specifics

The package implements most of its own plotting capabilities with the help of the **lattice** package (Sarkar 2008); thus, the majority of the plotting specifics comes from **lattice**. In particular, the results of plotting functions of **Rssa** are proper ‘`trellis`’ objects and can be modified, combined and otherwise altered using the standard **lattice** functionality.

A very convenient way of plotting the reconstructed series is via the `"xyplot"` method of the `plot` method for SSA reconstruction objects. There are powerful specializations of `xyplot` for ‘`ts`’ and ‘`zoo`’ objects and **Rssa** provides a stub implementation of `xyplot` for bare matrices.

SVD method

Rssa allows one to use several SVD implementations: either full decompositions implemented via R functions `eigen` and `svd`, or truncated Lanczos SVDs from the package **svd** (Korobeynikov 2014). These implementations differ in terms of speed, memory consumption and numerical stability, see Golyandina and Korobeynikov (2014) for discussion concerning these properties. Here we note that the use of fast SVD methods is the key point in the SSA processing of images.

By default, the `ssa` routine tries to select the best SVD implementation given the series length, window length and the desired number of eigentriples. This corresponds to the selection of `"auto"` for the `svd.method` argument. However, one can override this default setting and select the desired SVD implementation, if necessary. This differs from the behavior of previous versions of **Rssa**, when the package tried to “fix” the SVD method settings, when it thought it would be a bad idea to proceed. The present and next package versions (as of **Rssa** 0.10 and later on) will always tolerate explicit user choice.

Efficient implementation

Complementary to the use of efficient SVD methods, a considerable speed-up is achieved by means of special methods of actions with Hankel-related matrices, see Section 6 for the algorithms’ description. This is important, since multiplication by a Hankel matrix and hankelization of matrices, which can be very large, take a substantial part of the time needed by the SSA algorithms. The efficient implementation of these routines relies on the possibility to compute the fast Fourier transform of a vector of arbitrary length with the optimal $O(N \log N)$ complexity. In order to achieve this, **Rssa** uses the **FFTW** library (Frigo and Johnson 2005). While we strongly encourage to always complement **Rssa** installation with **FFTW**, the package will fallback to R’s FFT implementation if the package was compiled without **FFTW**. Pre-built Windows and Mac packages on CRAN are statically linked with **FFTW**; for other platforms it is usually possible to install the library using the standard package manager. (Note that a `-dev` version of the **FFTW** package is usually required.) If **FFTW** is not installed, only inherently one-dimensional analysis (1D-SSA, Toeplitz SSA and complex SSA) will be available. The computational speed will be slower in this case too.

Following Korobeynikov (2010), let us briefly describe the algorithm complexity for the one-dimensional case to show the order of speed-up. The direct implementation of the SSA algorithms has $O(N^3)$ computational and space complexity in the worst case $L \sim K \sim N/2$ (this case is standard for SSA), where N is the series length. Therefore, it is important to provide efficient implementations which makes non-trivial cases feasible.

- The methods of an efficient Hankel matrix-vector multiplication by the means of the fast Fourier transform (see Section 6.1) and the usage of fast Lanczos-based truncated SVD implementations drop the complexity from $O(N^3)$ down to $O(kN \log N + k^2N)$, where k is the number of calculated eigentriples.

- Second, one can represent the computation of the elementary series, which is *rank 1 hankelization*, as a special form of convolution. In this way, the efficient implementation of the hankelization (diagonal averaging) procedure is again possible via the fast Fourier transform and has similar improvement of complexity.

- Third, it is possible to implement the vector forecast much more efficiently than using the direct implementation. The details can be found in Section 6.3.

Note that the principle of automatic calculation of necessary objects is used in the implementation of the package. For example, if 10 eigentriples were calculated while decomposing, then the user could still perform reconstruction by the first 15 components, since the decomposition will be automatically continued to calculate 11–15 eigentriples. Also, the routines reuse the results of the previous calculations as much as possible in order to save time (hence the argument `cache` of many routines). For example, the elementary series once calculated are stored inside the SSA object, so next time the function `reconstruct` might not need to calculate the resulting series from scratch.

All these speed-up improvements make it possible to perform in reasonable time the tasks of analysis of long series, image processing, tracking procedures and batch processing.

3. Multivariate singular spectrum analysis

Let us consider the problem of simultaneous decomposition, reconstruction and forecasting for a collection of time series from the viewpoint of SSA. The method is called multichannel SSA or multivariate SSA, shortened to MSSA. The main idea of the algorithm is the same as in the basic SSA, the difference consists in the way how the trajectory matrix is constructed.

In a sense, MSSA is a straightforward extension of SSA. However, the algorithm of MSSA was published even earlier than the algorithm of SSA; see Weare and Nasstrom (1982), where the MSSA algorithm was named extended empirical orthogonal function (EEOF) analysis. Formally, the algorithm of MSSA in the framework of SSA was formulated in Broomhead and King (1986b).

Here we consider the algorithm of MSSA for 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. In this section, we also describe the complex-valued version of SSA (called CSSA), since it can be considered as a multidimensional version of SSA for analysis and forecasting of a system of two time series.

The theory studying the ranks of the multivariate time series and the separability of their components for MSSA and CSSA is very similar to that of SSA and is briefly described in Appendix A. Let us start with the algorithm description.

3.1. MSSA and CSSA algorithms

CSSA analysis

Let the system of series consist of two series, that is, $s = 2$. Then we can consider one-dimensional complex-valued series $\mathbb{X} = \mathbb{X}^{(1)} + \mathrm{i}\mathbb{X}^{(2)}$ and apply the complex version of SSA to this one-dimensional series. Since the general algorithm in Section 2 is written down in real-valued form, there is the difference in the form of the SVD performed in the complex-valued space, where the transposition should be Hermitian.

Also, there is a difference regarding 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 a lot of suitable constants c.

MSSA analysis

Consider a multivariate time series, that is, a collection $\{\mathbb{X}^{(p)} = (x_j^{(p)})_{j=1}^{N_p},\ \ p = 1, \ldots, s\}$ 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. Since the general scheme of the algorithm described in Section 2 holds for MSSA, we need to define the embedding operator $\mathcal{T}_{\mathrm{MSSA}}(\mathbb{X}) = \mathbf{X}$ only.

Let L be an integer called window length, $1 < L < \min(N_p, p = 1, \ldots, s)$. For each time series $\mathbb{X}^{(p)}$, the embedding procedure forms $K_p = N_p - L + 1$ L-lagged vectors $X_j^{(p)} = (x_j^{(p)}, \ldots, x_{j+L-1}^{(p)})^\top$, $1 \le j \le K_p$. Denote $K = \sum_{p=1}^{s} K_p$. The *trajectory matrix* of the multidimensional series $\mathbb{X}$ is the $L \times K$ matrix of the form

$$\mathcal{T}_{\mathrm{MSSA}}(\mathbb{X}) = \mathbf{X} = [X_1^{(1)} : \ldots : X_{K_1}^{(1)} : \ldots : X_1^{(s)} : \ldots : X_{K_s}^{(s)}] = [\mathbf{X}^{(1)} : \ldots : \mathbf{X}^{(s)}], \tag{7}$$

where $\mathbf{X}^{(p)} = \mathcal{T}_{\mathrm{SSA}}(\mathbb{X}^{(p)})$ is the trajectory matrix of the one-dimensional series $\mathbb{X}^{(p)}$ defined in (2). Thus, the trajectory matrix of a system of time series has *stacked Hankel* structure.

The eigenvectors $\{U_i\}$ in the SVD (3) of $\mathbf{X}$ form the common basis of the column trajectory spaces of all time series from the system. Factor vectors V_i (named EEOF in climatology applications) consist of parts related to each time series separately, that is,

$$V_i = \begin{pmatrix} V_i^{(1)} \\ \vdots \\ V_i^{(s)} \end{pmatrix}, \tag{8}$$

where $V_i^{(p)} \in \mathsf{R}^{K_p}$ and belong to the row trajectory spaces of the pth series.

The eigenvectors U_i reflect the common features of time series, while the factor subvectors $V_i^{(1)}$ 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.

Analogously to the one-dimensional case, the main result of the application of MSSA is the decomposition (5) of the multivariate time series $\mathbb{X}$ into a sum of m multivariate series.

Remarks

1. Note that the indexing of time points $1, \ldots, N_p$ $(p = 1, \ldots, s)$ starting from 1 does not mean that the series start at the same time and can finish at different times if lengths of time series are different. The resultant decomposition obtained by the MSSA algorithm does not depend on the shift between series and therefore this numeration is just formal. Even more, decompositions of two series measured at the same time and in disjoint time intervals do not differ.

2. The original time ranges of series $\mathbb{X}^{(p)}$ can be useful for depicting and interpreting them. Certainly, the reconstructed series have the same time ranges as the original ones. 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 the SSA analysis of one time series, it makes sense to consider window lengths $2 \le L \le \lfloor (N+1)/2 \rfloor$, 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 $2 \le L \le \min(N_p - 1, p = 1, \ldots s)$ are generally different.

4. For simultaneous analysis of several time series, it is recommended to transfer them onto the same scale. Otherwise, the structure of one time series will overweigh the results. To balance the time series, they can be either standardized (centered and normalized; in additive models) or only normalized (in multiplicative models). On the other hand, the scale of series can be used instead of their weights in the common decomposition if, e.g., one of the series is more important or has smaller noise level.

5. In a sense, the most detailed decomposition can be obtained if the trajectory matrix $\mathbf{X}$ has maximal rank. In the general case of arbitrary time series, this corresponds to the case of square matrix. Thus, for a system of s time series of length N the window length providing the square (or the closest to square) trajectory matrix $\mathbf{X}$ is approximately $s(N+1)/(s+1)$ for MSSA. In the case of two time series, this corresponds to $2(N+1)/3$ for MSSA, while CSSA gives $(N+1)/2$.

6. The MSSA algorithm might be modified in the same ways as the SSA one; for example, Toeplitz MSSA, MSSA with centering can be considered. However, these options are not implemented in the described version of **Rssa**.

3.2. Comments on the algorithms

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 MSSA considers the eigendecomposition of $\mathbf{S} = \mathbf{X}\mathbf{X}^\top = \mathbf{F}\mathbf{F}^\top + \mathbf{G}\mathbf{G}^\top$, that is, MSSA analyzes the averaging structure of two time series.

Since the SVD of a matrix coincides with the SVD of its transpose, we can consider the decomposition of the time series trajectory matrices on the basis of right singular vectors or, equivalently, to transpose $\mathbf{X}$ and consider eigenvectors of

$$\mathbf{S} = \begin{pmatrix} \mathbf{F}^\top\mathbf{F} & \mathbf{F}^\top\mathbf{G} \\ \mathbf{G}^\top\mathbf{F} & \mathbf{G}^\top\mathbf{G} \end{pmatrix}$$

called EEOFs. The last formula demonstrates more clearly that MSSA takes into consideration cross-covariances of time series (more precisely, cross-covariances are considered if centering is used).

Since the eigendecomposition of a complex-valued matrix $\mathbf{A} + \mathbf{i}\mathbf{B}$ 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 CSSA we in fact analyze eigenvectors of the matrix

$$\mathbf{S} = \begin{pmatrix} \mathbf{F}^\top\mathbf{F} & \mathbf{F}^\top\mathbf{G} \\ \mathbf{G}^\top\mathbf{F} & \mathbf{G}^\top\mathbf{G} \end{pmatrix} + \begin{pmatrix} \mathbf{G}^\top\mathbf{G} & -\mathbf{G}^\top\mathbf{F} \\ -\mathbf{F}^\top\mathbf{G} & \mathbf{F}^\top\mathbf{F} \end{pmatrix},$$

that is, structures of the time series are mixed in greater degree in comparison with MSSA.

Matching of series

Simultaneous analysis of several time series is usually performed to identify their relation and extract the common structure. Recall that the structure in SSA means that the trajectory matrix is rank-deficient. Certainly, for real series in applications, the trajectory matrix is of full rank, since at least noise has no structure. Therefore, in what follows we say about rank applies to the signal or its components.

Consider the system of series with rank-deficient trajectory matrix. The structure of the series is reflected by its trajectory space. Therefore, we can say that two time series have the same structure if their trajectory spaces coincide. For example, the trajectory spaces of two sine waves with equal periods coincide irrespective of the amplitudes and phases. This fact is evident, since the trajectory space is the span of subseries of length L of the initial series. To the contrary, sine waves with different frequencies have totally different structure and their combined trajectory space is the direct sum of the trajectory spaces of individual time series.

If two time series are fully matched, then the trajectory space of one time series can be used for reconstruction or forecasting of the second series. If the series are totally different, then the first series is useless for the analysis of the second one.

For MSSA, the shift between time series, for example, the difference between phases of two matched sine waves, is of no consequence. Therefore, we cannot say anything about direction of causality (if any; see Hassani, Heravi, and Zhigljavsky 2013, where the attempt to detect causality is performed). Moreover, asymmetry of influence of one time series to another series can be caused by different levels of noise.

Relation between SSA, MSSA and CSSA ranks

For MSSA and CSSA, the notions of time series of finite rank and of time series satisfying linear recurrence relations are analogous to these notions in SSA. However ranks of the same time series can differ depending on the applied method. Therefore, we can think about SSA, MSSA and CSSA ranks.

If the individual time series have the same structure (and therefore the same SSA ranks), then the MSSA rank is equal to the SSA rank of each time series. As for CSSA rank, it can be even less than all the individual SSA ranks for some special cases.

Consider a collection $\mathbb{H}^{(p)} = (h_j^{(p)})_{j=1}^N$, $p = 1, \ldots, s$, of s signals of length N. Let r_p denote the SSA rank of $\mathbb{H}^{(p)}$ (i.e., dimension of the trajectory spaces 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_p, $p = 1, \ldots, s$, is considered in Appendix A. In particular, it is shown that $r_{\mathrm{min}} \le r \le r_{\mathrm{max}}$, where $r_{\mathrm{min}} = \max\{r_p, p = 1, \ldots, s\}$ and $r_{\mathrm{max}} = \sum_{p=1}^s r_p$. The case $r = r_{\mathrm{max}}$ is the least favorable for MSSA and means that different time series do not have matched components. The case $r < r_{\mathrm{max}}$ indicates the presence of matched components and can lead to advantages of simultaneous processing of the time series system.

In terms of matching, if all the series have the same characteristic roots (see Appendix A for definition), then this means that the time series $\mathbb{X}^{(p)}$, $p = 1, \ldots, s$, consist of additive components of the same SSA structure. Such time series are fully matched. For fully matched time series the MSSA rank is much smaller than the sum of the SSA ranks of the separate time series from the system. On the contrary, if the sets of characteristic roots do not intersect, then the time series have no common structure. In this case, the MSSA rank is exactly equal to the sum of the SSA ranks of the separate time series from the system. For the real-world time series, we are usually between these extreme cases.

Separability

The notion of separability for multidimensional time series is analogous to that for one-dimensional series briefly commented on in Section 2.2 and thoroughly described in Golyandina *et al.* (2001, Sections 1.5 and 6.1). Appendix A contains several results on separability of multidimensional series together with definitions of weak and strong separability. Generally, conditions of separability of multidimensional time series are more restrictive than that for one-dimensional series. In particular, the sufficient condition for separability of two series systems is the separability of each series from one collection with each series from the second one. However, for matched signals, their (weak) separability from the 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 time series components. It appears (see Example 2 in Appendix A) that the time series $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ can produce different eigenvalues in SSA, MSSA and CSSA. Therefore, the application of an appropriate multidimensional modification of SSA can improve strong separability. Again, matching of time series diminishes the number of eigenvalues related to the signal and thereby weakens the chance of mixing of the signal with the residual.

Choice of window length

Recommendations on the choice of window length for one-dimensional SSA analysis can be found, e.g., in Golyandina *et al.* (2001, Section 1.6) and Golyandina (2010). However, the problem of the choice of window length in MSSA is more complicated than that for SSA. To the best of the authors' knowledge, there is no appropriate investigation of the choice of optimal window length for analysis and, to a greater extent, forecasting of multidimensional time series. Moreover, the choice of the best window length for MSSA forecasting differs for different types of forecasting methods, see the numerical comparison in Section 3.5, which, in particular, contains an extension of the numerical investigation done in Golyandina and Stepanov (2005).

By analogy to the one-dimensional case, we can formulate some principles for the choice of L. The main principle is the same as for SSA and states that the choice of L should provide (approximate) separability of series. However, the MSSA case has additional features. For example, while in SSA analysis it makes no sense to take $L > (N+1)/2$, in MSSA analysis large L with small $K_p = N_p - L + 1$ can be taken instead of small L for trend extraction and smoothing.

Various special techniques can be transferred from SSA to MSSA, such as sequential SSA. Sequential SSA is based on successive application of SSA with different window lengths, see Golyandina and Zhigljavsky (2013, Section 2.5.5) for more details. For example, if trends are of complex or of different structure, a smaller window length can be applied to achieve the similarity of eigenvectors and to improve separability. Then the residual can be decomposed with a larger window length. For multidimensional time series, sequential MSSA can be applied by analogy to sequential SSA.

Two alternatives to sequential SSA are described in Golyandina and Shlemov (2015) and are implemented for one-dimensional time series in version 0.11 of the **Rssa** package. However, these approaches can be extended to multivariate and multidimensional cases.

Since L in SSA does not exceed the half of the time series length, the divisibility of $L = \min(L, K)$ on possible periods of oscillations is recommended in SSA. In MSSA, $\min(L, K_p)$ is not necessary equal to L and therefore one puts attention on values of K_p.

Numerical investigations show that the choice $L = \lfloor sN/(s+1) \rfloor$ is appropriate for the decomposition of several (a few) time series (see simulation results in Section 3.5), but evidently cannot be applied for the system of many short series (K_p becomes too small for separability). In general, the choice $L = \lfloor N/2 \rfloor$ is still appropriate even for multivariate SSA.

3.3. Multivariate SSA forecasting

Forecasting in SSA is performed for a time series component which can be separated by SSA and is described (maybe, approximately) by a linear recurrent relation (LRR). For brevity, we will refer to this as forecasting of a signal. If the forecasted series component (signal) is governed by an LRR, then it can be forecasted by this or extended LRR.

Without loss of generality, we assume that the first component $\widetilde{\mathbb{X}}_1$ of the decomposition (5) is forecasted. In order to simplify the notation, we denote the reconstructed series by $\widetilde{\mathbb{X}} = \widetilde{\mathbb{X}}_1 = (\widetilde{\mathbb{X}}^{(1)}, \ldots, \widetilde{\mathbb{X}}^{(s)})$, where $\widetilde{\mathbb{X}}^{(p)} = (\tilde{x}_j^{(p)})_{j=1}^{N_p}$, $p = 1, \ldots, s$. The methods of SSA forecasting aim at obtaining the forecasted points $\tilde{x}_j^{(p)}$ for $j > N_p$, based on the reconstructed time series $\widetilde{\mathbb{X}}$ and the corresponding reconstructed matrix $\widehat{\mathbf{X}} = \widehat{\mathbf{X}}_1$. The forecasting is called *M-step ahead* if the points $(\tilde{x}_{N_p+1}^{(p)}, \ldots, \tilde{x}_{N_p+M}^{(p)})$, $p = 1, \ldots, s$ are being obtained.

There are two main methods of SSA forecasting: recurrent and vector. In *recurrent* forecasting, an estimated LRR is applied to the the reconstructed series $\widetilde{\mathbb{X}}$ in order to obtain a 1-step ahead forecast. The M-step ahead forecast is obtained by recurrence (by applying M times the 1-step ahead forecasting with the same LRR), which explains the name of the method. In *vector* forecasting, the reconstructed matrix $\widehat{\mathbf{X}}$ is first extended, by continuation of the reconstructed lagged vectors in a given subspace (row or column space of $\widehat{\mathbf{X}}$). The forecasted points are then obtained by diagonal averaging of the extended reconstructed matrix.

Recurrent and vector methods of one-dimensional SSA forecasting ($s = 1$) are fully described in Golyandina *et al.* (2001, Chapter 2). For CSSA, the forecasting algorithms are straightforward extensions of SSA forecasting algorithms to the complex-valued case; therefore we do not discuss them here. The methods of MSSA forecasting, however, need special attention.

As in SSA, methods of MSSA forecasting can be subdivided into recurrent and vector forecasting. In contrast with 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, depending on which space is used (row or column space respectively). In total, there are four variants of MSSA forecasting: recurrent column forecasting, recurrent row forecasting, vector column forecasting and vector row forecasting.

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. Next, we describe all the variants of MSSA forecasting in detail.

Common notation

First, we introduce some common notation used for description of all the variants of MSSA forecasting.

For a vector $A \in \mathsf{R}^Q$, we denote by $\overline{A} \in \mathsf{R}^{Q-1}$ the vector of the last $Q-1$ coordinates and by $\underline{A} \in \mathsf{R}^{Q-1}$ the vectors of the first $Q-1$ coordinates. The line on the top (respectively, on the bottom) indicates that the the first (respectively, the last) coordinate is removed from the vector A. Also, denote by $\pi(A)$ the last coordinate of the vector. For a matrix $\mathbf{A} = [A_1 : \ldots : A_r]$ denote $\overline{\mathbf{A}} = [\overline{A}_1 : \ldots : \overline{A}_r]$ and $\underline{\mathbf{A}} = [\underline{A}_1 : \ldots : \underline{A}_r]$ and let $\boldsymbol{\pi}(\mathbf{A}) = (\pi(A_1), \ldots, \pi(A_r))^\top$ be the last row of the matrix $\mathbf{A}$.

For a vector $B \in \mathsf{R}^K$, where $K = \sum_{p=1}^s K_p$, we use the following notation:

$$B = \begin{pmatrix} B^{(1)} \\ B^{(2)} \\ \vdots \\ B^{(s)} \end{pmatrix}, \quad \underline{\underline{B}} = \begin{pmatrix} \underline{B}^{(1)} \\ \underline{B}^{(2)} \\ \vdots \\ \underline{B}^{(s)} \end{pmatrix}, \quad \overline{\overline{B}} = \begin{pmatrix} \overline{B}^{(1)} \\ \overline{B}^{(2)} \\ \vdots \\ \overline{B}^{(s)} \end{pmatrix}, \quad \boldsymbol{\mu}(B) = \begin{pmatrix} \pi(B^{(1)}) \\ \pi(B^{(2)}) \\ \vdots \\ \pi(B^{(s)}) \end{pmatrix}. \qquad (9)$$

where $B^{(p)} \in \mathsf{R}^{K_p}$. For $\mathbf{B} = [B_1 : \ldots : B_r]$, let $\underline{\underline{\mathbf{B}}} = [\underline{\underline{B}}_1 : \ldots : \underline{\underline{B}}_r]$ and $\mathbf{B}^{(p)} = [B_1^{(p)} : \ldots : B_r^{(p)}]$.

For simplicity we assume that the set $I = I_1$ corresponding to the forecasted component is given by the set of the leading components; that is, $I = I_1 = \{1, \ldots, r\}$. (This is made just for simplification of 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, and denote $\mathbf{U} = [U_1 : \ldots : U_r]$, $\mathbf{V} = [V_1 : \ldots : V_r]$.

The reconstructed series $\widetilde{\mathbb{X}}$, its trajectory matrix $\widetilde{\mathbf{X}}$ and the reconstructed matrix $\widehat{\mathbf{X}}$ are defined in Section 2.1. Define $\mathcal{L}^{\text{col}} = \text{span}(U_i, i \in I)$, $\mathcal{L}^{\text{row}} = \text{span}(V_i, i \in I)$. The reconstructed matrix $\widehat{\mathbf{X}} = [\widehat{X}_1^{(1)} : \ldots : \widehat{X}_{K_1}^{(1)} : \ldots : \widehat{X}_1^{(s)} : \ldots : \widehat{X}_{K_s}^{(s)}]$ consists of column vectors that are projections of column vectors of the trajectory matrix (7) on the chosen subspace $\mathcal{L}^{\text{col}}$. For convenience, we also denote by $\widehat{Y}_i$ the ith row of the matrix $\widehat{\mathbf{X}}$, such that $\widehat{\mathbf{X}}^\top = [\widehat{Y}_1 : \ldots : \widehat{Y}_L]$.

Recurrent MSSA forecast

Denote by $R_N = (\tilde{x}_{N_1+1}^{(1)}, \tilde{x}_{N_2+1}^{(2)}, \ldots, \tilde{x}_{N_s+1}^{(s)})^\top$ the vector of forecasted signal values for each time series (1-step ahead forecast). 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 can write out 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 1-step ahead forecasting formulas can be applied for M-term ahead forecast by 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, one 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, LRRs are different, but initial data for them are the same.

Column forecast. Denote by $\mathbf{Z}$ the matrix consisting of the last $L-1$ values of the reconstructed signals:

$$\mathbf{Z} = \begin{pmatrix} \tilde{x}_{N_1-L+2}^{(1)} & \ldots & \tilde{x}_{N_1}^{(1)} \\ \tilde{x}_{N_2-L+2}^{(2)} & \ldots & \tilde{x}_{N_2}^{(2)} \\ \vdots & \vdots & \vdots \\ \tilde{x}_{N_s-L+2}^{(s)} & \ldots & \tilde{x}_{N_s}^{(s)} \end{pmatrix},$$

$\nu^2 = \sum_{j=1}^{r} \pi(U_j)^2$. If $\nu^2 < 1$, then the recurrent column forecast is uniquely defined and can be calculated by the formula

$$R_N = \mathbf{Z}\mathcal{R}_L, \qquad \text{where} \qquad \mathcal{R}_L = \frac{1}{1-\nu^2} \sum_{j=1}^{r} \pi(U_j)\underline{U_j} \quad \in \mathsf{R}^{L-1}. \tag{10}$$

Note that (10) implies that the forecasting of all individual series is made using the same LRR (with coefficients $\mathcal{R}_L$), which is generated by the whole system of series.

Row forecast. Introduce the vectors of the last $K_p - 1$ values of the reconstructed signals

$$Z^{(p)} = (\tilde{x}_{N-K_p+2}^{(p)}, \ldots, \tilde{x}_{N_p}^{(p)})^\top, \quad p = 1, \ldots, s,$$

and define a vector $Z \in \mathsf{R}^{K-s}$ and an $s \times r$ matrix $\mathbf{S}$ as

$$Z = \begin{pmatrix} Z^{(1)} \\ Z^{(2)} \\ \vdots \\ Z^{(s)} \end{pmatrix}, \qquad \mathbf{S} = [\boldsymbol{\mu}(V_1) : \ldots : \boldsymbol{\mu}(V_r)].$$

If the inverse matrix $(\mathbf{I}_s - \mathbf{S}\mathbf{S}^\top)^{-1}$ exists and $r \leq K - s$, then the recurrent row forecast exists and can be calculated by the formula

$$R_N = \boldsymbol{\mathcal{R}}_K Z, \tag{11}$$

where $\boldsymbol{\mathcal{R}}_K = (\mathbf{I}_s - \mathbf{S}\mathbf{S}^\top)^{-1}\, \mathbf{S}\underline{\underline{\mathbf{V}}}^\top$. Note that (11) implies that the forecasting of the individual signals is made using the linear relations which are different for different series. The forecasting value generally depends on the last values of all the time series in the system of series.

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 $\underline{\underline{\mathcal{L}}}^{\mathrm{row}}$.

The rows of $\widehat{\mathbf{X}}$ are the projections of rows of the trajectory matrix $\mathbf{X}$ on $\mathcal{L}^{\mathrm{row}}$, while the columns of $\widehat{\mathbf{X}}$ are the projections of columns of the trajectory matrix $\mathbf{X}$ on $\mathcal{L}^{\mathrm{col}}$.

The explicit form of the matrices of the column and row projectors can be found in Golyandina and Osipov (2007, Formula (4)). However, the calculation based on this formula is time-consuming. The fast algorithms for vector forecasting are presented in Section 6.3.

Column forecast. We mentioned 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

$$\mathcal{P}^{\mathrm{col}}_{\mathrm{Vec}} Z = \begin{pmatrix} \Pi^{\mathrm{col}} \overline{Z} \\ \mathcal{R}_L^\top \overline{Z} \end{pmatrix}. \tag{12}$$

Let us formulate the *vector forecasting algorithm* for jth series.

1. In the notation above, define the vectors $Z_j \in \mathsf{R}^L$ as follows:

$$Z_j = \begin{cases} \widehat{X}_j^{(p)} & \text{for } j = 1, \ldots, K_p, \\ \mathcal{P}^{\mathrm{col}}_{\mathrm{Vec}} Z_{j-1} & \text{for } j = K_p + 1, \ldots, K_p + M + L - 1. \end{cases} \tag{13}$$

2. By constructing the matrix $\mathbf{Z} = [Z_1 : \ldots : Z_{K_p+M+L-1}]$ and making its diagonal averaging we obtain the series $z_1, \ldots, z_{N_p+M+L-1}$.

3. The values $(\tilde{x}^{(p)}_{N_p+1}, \ldots, \tilde{x}^{(p)}_{N_p+M}) = (z_{N_p+1}, \ldots, z_{N_p+M})$ form the M terms of the forecast.

Row forecast. Define the linear operator $\mathcal{P}^{\mathrm{row}}_{\mathrm{Vec}} : \mathsf{R}^K \mapsto \mathcal{L}^{\mathrm{row}}$ by the formula

$$\mathcal{P}^{\mathrm{row}}_{\mathrm{Vec}} Z = A, \tag{14}$$

such that $\underline{\underline{A}} = \Pi^{\mathrm{row}} \overline{\overline{Z}}$ and $\boldsymbol{\mu}(A) = \boldsymbol{\mathfrak{R}}_K \overline{\overline{Z}}$.

Let us formulate the *vector forecasting algorithm.*

1. In the notation above, define the vectors $Z_i \in \mathsf{R}^K$ as follows:

$$Z_i = \begin{cases} \widehat{Y}_i & \text{for } i = 1, \ldots, L, \\ \mathcal{P}^{\mathrm{row}}_{\mathrm{Vec}} Z_{i-1} & \text{for } i = L+1, \ldots, L+M+K^*-1, \end{cases} \tag{15}$$

   where $K^* = \max(K_p, p = 1, \ldots, s)$.

2. By constructing the matrix $\mathbf{Z} = [Z_1 : \ldots : Z_{L+M+K^*-1}]^\top$ and making the MSSA reconstruction step we obtain the series $z_1^{(p)}, \ldots, z_{N_p+M+K^*-1}^{(p)}$, $p = 1, \ldots, s$.

3. The values $(\tilde{x}_{N_p+1}^{(p)}, \ldots, \tilde{x}_{N_p+M}^{(p)}) = (z_{N_p+1}^{(p)}, \ldots, z_{N_p+M}^{(p)})$ form the M terms of the forecast.

Remark 1. *For the M-step ahead vector forecast, $M + K^* - 1$ new lagged vectors for row forecasting and $M + L - 1$ ones for column forecasting are constructed. The reason for this is to make the M-step forecast inheriting the $(M-1)$-step forecast with no redrawing. This characteristic of the vector forecasting provides its stability and accuracy if the accurately extracted component of finite rank is forecasted, that is, if long-term forecast is appropriate. Otherwise, long-term vector forecasting can be wrong and even the result of short-term vector forecasting can be also wrong for large K^* or L correspondingly.*

3.4. Package

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 for one-dimensional time series by **Rssa** are thoroughly described in Golyandina and Korobeynikov (2014), we put more attention on the difference.

In Section 2.3, we decomposed the one-dimensional series FORT (sales of fortified wines). Here we add one more series, sales of dry wines (shortly DRY), for simultaneous analysis.

For loading the data we use the code from Fragment 1.

Fragment 5 (FORT and DRY: Reconstruction)**.**

```
R> wineFortDry <- wine[, c("Fortified", "Drywhite")]
R> L <- 84
R> s.wineFortDry <- ssa(wineFortDry, L = L, kind = "mssa")
R> r.wineFortDry <- reconstruct(s.wineFortDry,
+    groups = list(Trend = c(1, 6), Seasonality = c(2:5, 7:12)))
R> plot(r.wineFortDry, add.residuals = FALSE, plot.method = "xyplot",
+    superpose = TRUE, auto.key = list(columns = 3))
```

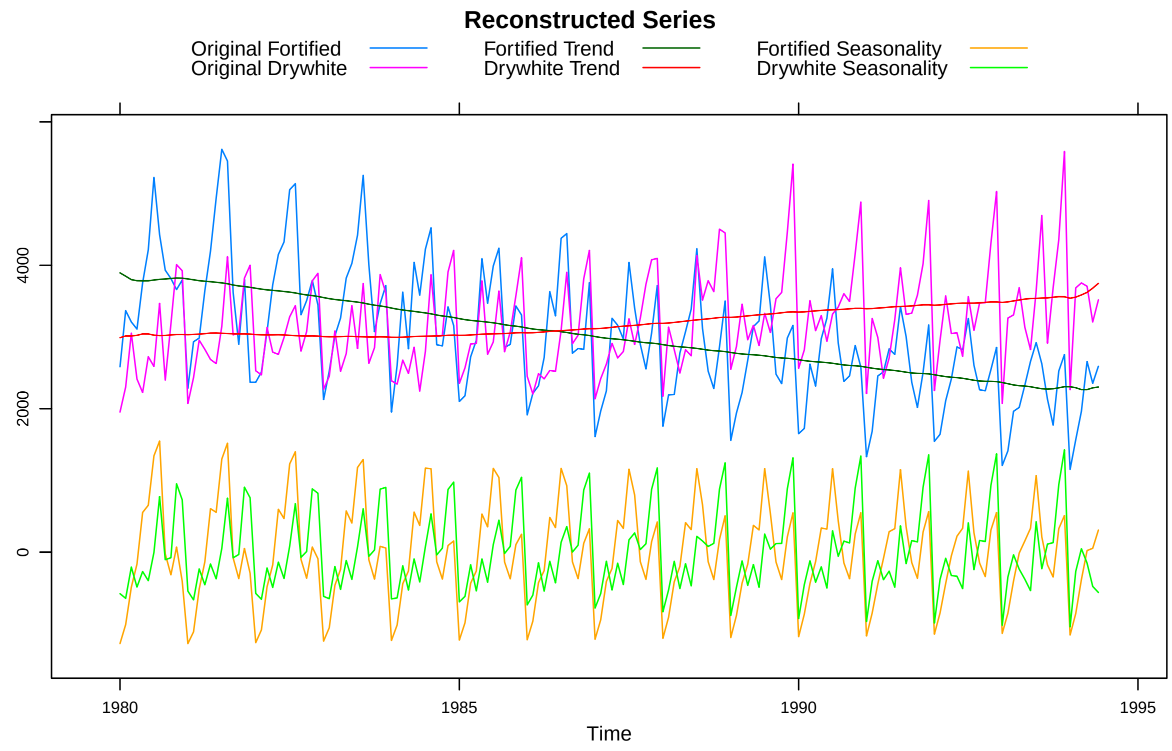


Figure 8: FORT and DRY: Reconstructed trend and seasonality.

Fragment 5 contains a typical code for simultaneous extraction of the trend and seasonality (compare with Fragment 2). An evident difference is in the indicated value of parameter `kind` in the `ssa` function. A more significant difference is related to plotting of the results. For multivariate series there is in a sense a matrix of series, where one index is the number of the series in the system, while the second index corresponds to the number of the component in the decomposition. The `plot` function for the reconstruction object allows to indicate which subset (which slice) of this matrix one wants to depict by means of the parameter `slice`. The parameter `slice` consists of the list of numbers of series and numbers of decomposition components.

The code for component identification in MSSA is very similar to that in SSA, compare Fragments 6 and 3. The difference consists in the structure of the factor vectors; however, the factor vectors are not necessary for identification. Figure 9 (compare Figure 3) shows that the trend is described by ET1 and ET6, which is slightly mixed with seasonality. Figure 10 (compare Figure 4) demonstrates the pairs of ETs that are related to seasonality.

Since the implemented methods of parameter estimation are based on eigenvectors only, they can be applied to eigenvectors in MSSA in exactly the same way as in the one-dimensional case.

Fragment 6 (FORT and DRY: Identification).

```
R> plot(s.wineFortDry, type = "vectors", idx = 1:8)
R> plot(s.wineFortDry, type = "paired", idx = 2:11, plot.contrib = FALSE)
R> parestimate(s.wineFortDry, groups = list(2:3, 4:5), method = "esprit-ls")
```

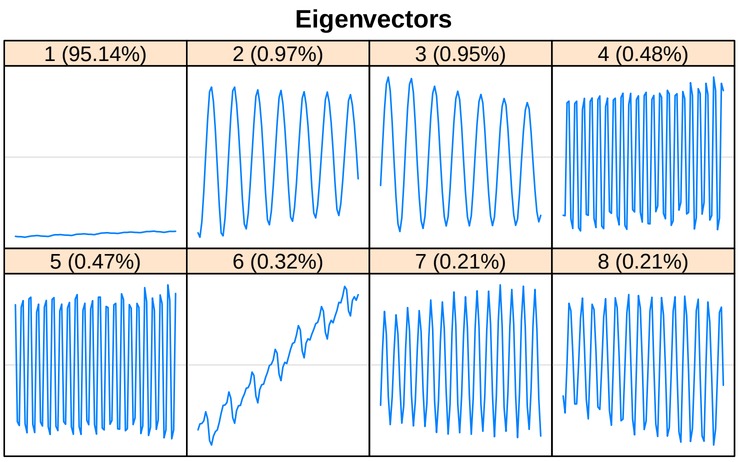


Figure 9: FORT and DRY: 1D graphs of eigenvectors.

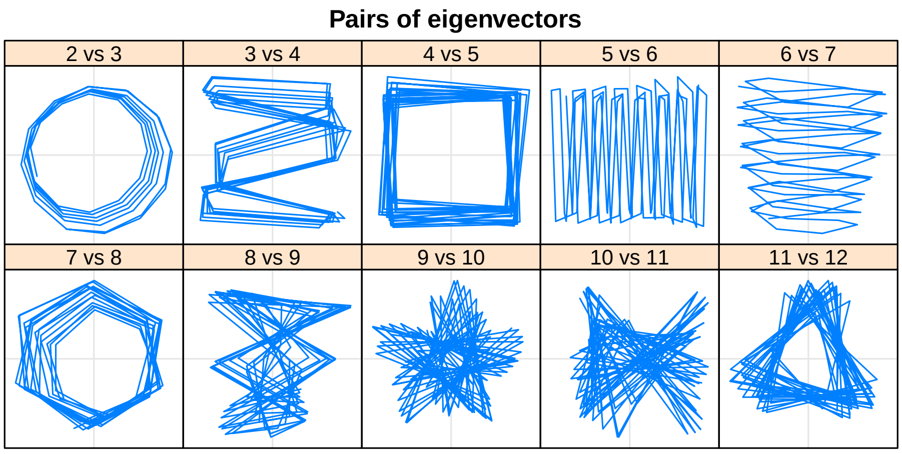


Figure 10: FORT and DRY: 2D scatterplots of eigenvectors.

```
$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
```

```
R> plot(wcor(s.wineFortDry, groups = 1:30),
+    scales = list(at = c(10, 20, 30)))
```

The code for forecasting is also very similar to that in SSA, compare Fragments 7 and 4.

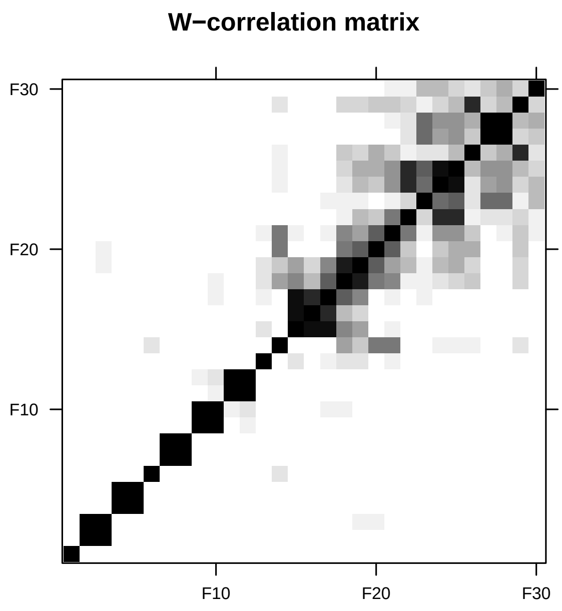


Figure 11: FORT and DRY: Weighted correlations.

Fragment 7 (FORT and DRY: Forecast)**.**

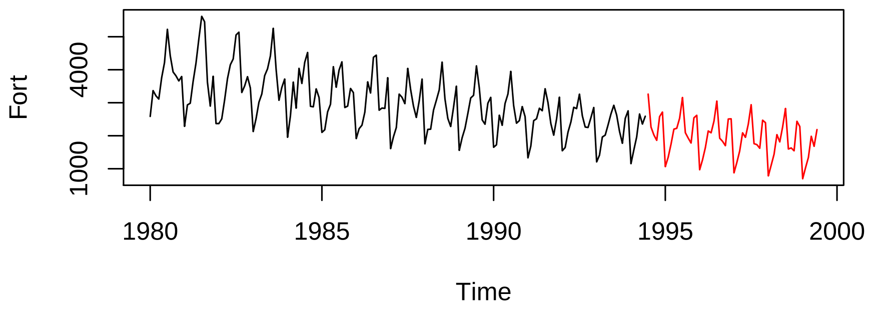
```
R> f.wineFortDry <- rforecast(s.wineFortDry, groups = list(1, 1:12),
+    len = 60, only.new = TRUE)
R> par(mfrow = c(2, 1))
R> plot(cbind(wineFortDry[, "Fortified"], f.wineFortDry$F2[, "Fortified"]),
+    plot.type = "single", col = c("black", "red"), ylab = "Fort")
R> plot(cbind(wineFortDry[, "Drywhite"], f.wineFortDry$F2[, "Drywhite"]),
+    plot.type = "single", col = c("black", "red"), ylab = "Dry")
```

The results of MSSA analysis are similar to the results of SSA analysis. However, the separability is slightly worse (compare **w** correlations between signal components in Figures 5 and 11). Therefore, the methods for improvement of separability can be very useful (Golyandina and Shlemov 2015). Note that the mixture of the signal components is not important for signal forecasting.

Comments

Formats of input and output data. While the representation of a one-dimensional time series in R is pretty obvious, there are multiple possible ways of defining the multivariate time series. Let us outline some common choices.

- A matrix with separate series in the columns. Optionally, 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,



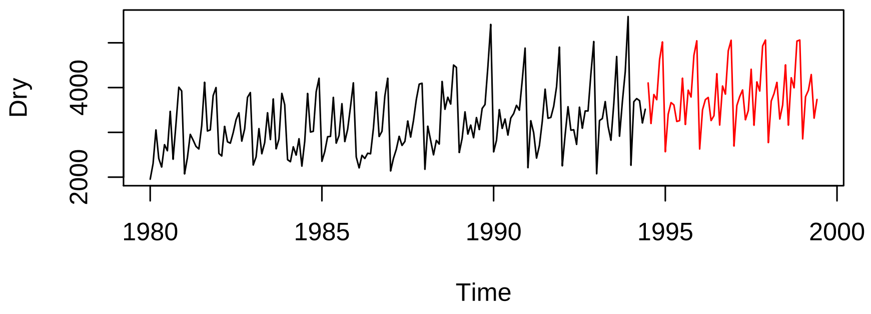


Figure 12: FORT and DRY: Forecast of the signal.

`data.frame` would be the result of reading the series in 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. Or, everything can be mixed all together.

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 the Fragments 5 and 14 provide an overview of the possible input series formats.

Plotting specifics. 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 might totally be unusable in case of `data.frame` objects, because it will just call the `pairs` function on the resulting data frame in the end.

Summary for `ssa` object. The `summary` for SSA objects (see Fragment 8) in the MSSA case coincides with the one for SSA up to slight differences.

Fragment 8 (FORT and DRY: Summary)**.**

```
R> summary(s.wineFortDry)

Call:
ssa(x = wineFortDry, L = L, kind = "mssa")
Series length: 174, 174,       Window length: 84,       SVD method: eigen
Special triples:  0
Computed:
Eigenvalues: 50,       Eigenvectors: 50,       Factor vectors: 0
Precached: 0 elementary series (0 MiB)
Overall memory consumption (estimate): 0.03809 MiB
```

Here one can see the individual series lengths (excluding the `NA` padding), the window length, the selected default SVD decomposition method and the number of computed eigentriples. No factor vectors are computed, they will be recomputed on the fly when necessary.

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 the sake of simplicity, assume that all the series have equal lengths N), the worst case corresponds to $L \sim K \sim s(N+1)/(s+1)$, that is, the order of complexity is the same, $O(N^3)$, but the constant can be considerably larger. Therefore, the achieved speed-up can be much higher than that in the one-dimensional case.

Note that the multichannel SSA can be viewed as a special case of shaped 2D-SSA (see Section 5.3) and the current implementation in the package uses this under the hood.

3.5. Examples

Factor vectors

Factor vectors in MSSA have length K and consist of stacked vectors of length K_p, $p = 1, \ldots, s$, related to each series. Therefore, it is natural to depict them as a system of s vectors. Factor vectors are not necessarily contained in the '`ssa`' object. In particular, the `s.wineFortDry` object created in Fragment 5 has 0 factor vectors, which can be checked using `summary(s.wineFortDry)`. Note that if one uses `svd.method = "svd"` (and `"propack"`), then factor vectors will be calculated automatically. In order to get factor vectors corresponding to the eigenvectors contained in '`ssa`', the user can call the `calc.v` function.

Fragment 9 shows the difference between eigenvectors and factor vectors for small window length. The result for $L = 24$ is depicted in Figure 13. The eigenvector in Figure 13 captures the common behavior (which is almost constant) in the timescale of two years, while the factor vector is divided into parts reflecting individual features of the series, compared with trends in Figure 8 which are repeated in Figure 13. Note that signs of the calculated eigenvectors are random. For example, the first eigenvector is negative here. Thereby, the factor vectors are similar to the reconstructed series with opposite sign.

The choice of large window length $L = 163$ also yields the trajectory matrix of rank 24, since then $K_p = 176 - 163 + 1 = 12$ and therefore $K = 2 \cdot 12 = 24$; however, the common structure is captured by two eigentriples. Note that in climatology the SVD of the transposed trajectory matrix is traditionally considered (Hannachi, Jolliffe, and Stephenson 2007). Therefore, the eigenvectors U_i correspond to normalized extended principal components in Hannachi *et al.* (2007), while the factor vectors V_i are called EEOFs.

Fragment 9 (FORT and DRY: Use of factor vectors in MSSA)**.**

```
R> library("lattice")
R> L <- 24
R> s.wineFortDrya <- ssa(wineFortDry, L = L, kind = "mssa")
R> r.wineFortDrya <- reconstruct(s.wineFortDrya, groups = list(Trend = 1))
R> tp1 <- plot(r.wineFortDrya, add.residuals = FALSE, add.original = TRUE,
+    plot.method = "xyplot", aspect = 0.3, superpose = TRUE,
+    scales = list(y = list(draw = FALSE)), auto.key = "", xlab = "",
+    col = c("blue", "violet", "blue", "violet"))
R> tp2 <- plot(s.wineFortDrya, type = "vectors", vectors = "factor",
+    idx = 1, aspect = 0.5, superpose = TRUE,
+    scales = list(x = list(draw = TRUE), y = list(draw = FALSE)),
+    auto.key = list(columns = 2))
R> tp3 <- plot(s.wineFortDrya, type = "vectors", vectors = "eigen",
+    idx = 1, aspect = 1,
+    scales = list(x = list(draw = TRUE), y = list (draw = FALSE)))
R> plot(tp3, split = c(1, 1, 1, 3), more = TRUE)
R> plot(tp2, split = c(1, 2, 1, 3), more = TRUE)
R> plot(tp1, split = c(1, 3, 1, 3), more = FALSE)
```

Preprocessing (normalization)

Let the time series in the system be measured in different scales. In statistics, this problem is typically resolved by standardizing the data. In SSA, centering may not be an appropriate preprocessing. Therefore, two types of preprocessing can be applied, conventional standardization and normalization, that is, division by the square root of mean sum of squares. The normalization can be even more appropriate for positive series, since it changes only the scale of data.

Let us consider Fortified and Rosé wine sales. Sales of Fortified wines are of the order of thousands while sales of Rosé wines are of the order of tens and hundreds. Fragment 10 shows how the scale influences the reconstruction result.

Fragment 10 (FORT and ROSE: Influence of series scales)**.**

```
R> wineFortRose <- wine[, c("Fortified", "Rose")]
R> summary(wineFortRose)
```

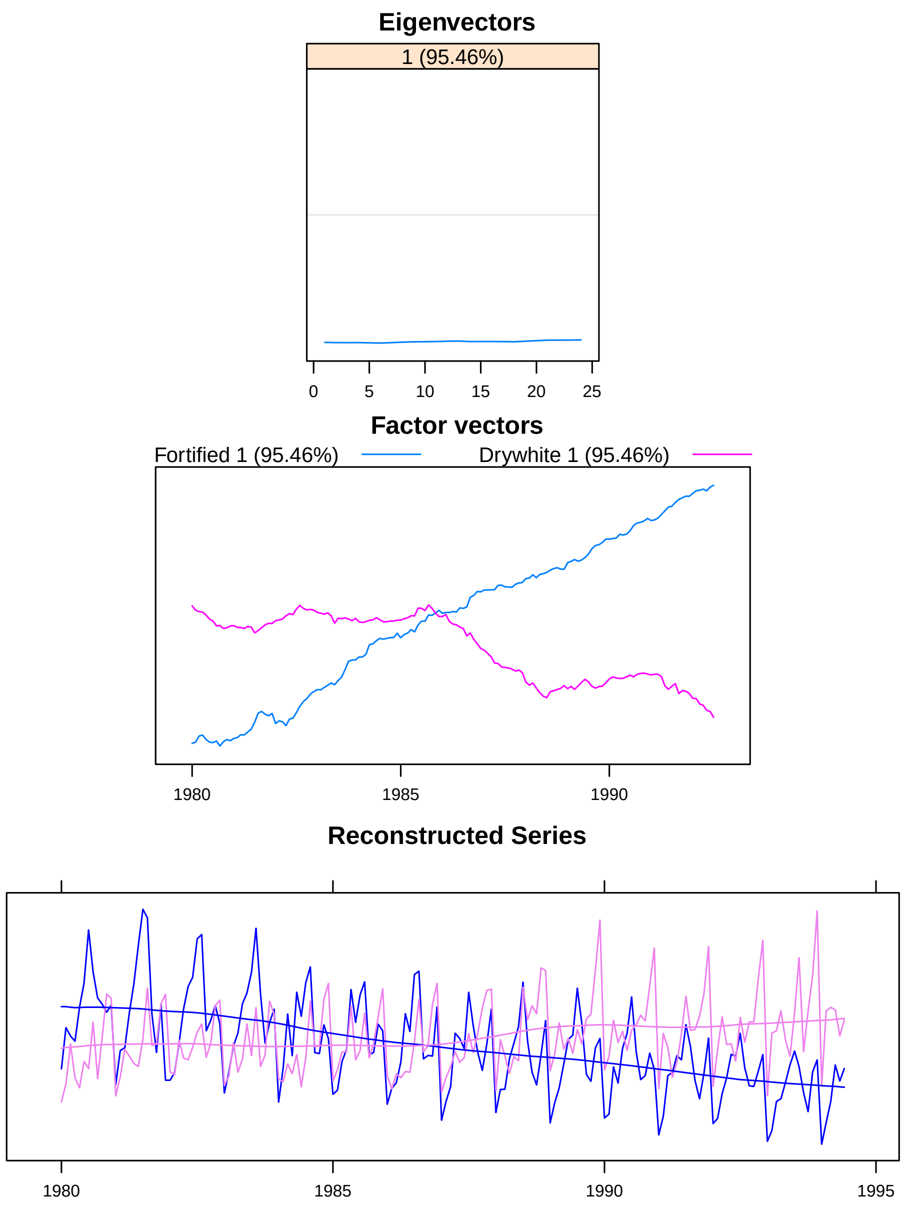


Figure 13: FORT and DRY: $L = 24$, relation between eigen- and factor vectors and reconstructed series.

```
   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
```

```
R> norm.wineFortRosen <- sqrt(colMeans(wineFortRose^2))
```

```
R> wineFortRosen <- sweep(wineFortRose, 2, norm.wineFortRosen, "/")
R> L <- 84
R> s.wineFortRosen <- ssa(wineFortRosen, L = L, kind = "mssa")
R> r.wineFortRosen <- reconstruct(s.wineFortRosen,
+    groups = list(Trend = c(1, 12, 14), Seasonality = c(2:11, 13)))
R> s.wineFortRose <- ssa(wineFortRose, L = L, kind = "mssa")
R> r.wineFortRose <- reconstruct(s.wineFortRose,
+    groups = list(Trend = 1, Seasonality = 2:11))
R> 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 = "", ...)
R> trel1 <- wrap.plot(r.wineFortRosen, series = 2, ylab = "Rose, norm")
R> trel2 <- wrap.plot(r.wineFortRosen, series = 1, ylab = "Fort, norm")
R> trel3 <- wrap.plot(r.wineFortRose, series = 2, ylab = "Rose")
R> trel4 <- wrap.plot(r.wineFortRose, series = 1, ylab = "Fort")
R> plot(trel1, split = c(1, 1, 2, 2), more = TRUE)
R> plot(trel2, split = c(1, 2, 2, 2), more = TRUE)
R> plot(trel3, split = c(2, 1, 2, 2), more = TRUE)
R> plot(trel4, split = c(2, 2, 2, 2))
```

Figure 14 demonstrates the result of a trend reconstruction, where the trend was detected in the same way as before, that is, by means of form 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 SSA processing with no normalization is worse for analysis of the series ROSE with smaller scale.

Forecasting of series with different lengths as filling in

Typical code in Section 3.4 was demonstrated for series of equal lengths. However, the same code can be applied to series of different lengths. The full Australian wine data are incomplete, there are no data for two months (point 175 and 176) for sales of Rosé wines and there are no data for the last 11 months of Total sales.

Let us perform the following actions: (A) fill in the missing data in ROSE, (B) calculate the sum of sales of the present wines and (C) process this sum together with the Total series and fill in the missing data in the Total series by the simultaneous forecasting.

To use the analysis performed before, let us forecast the ROSE series together with the FORT series for (A). Fragment 11 implements (A). Certainly, this can be done by a separate analysis of the ROSE series and the SSA methods of filling in missing data, but Figure 15 shows that the result of the used method is quite accurate.

Fragment 11 (FORT and ROSE: Forecast of the missing data in ROSE)**.**

```
R> f.wineFortRosen <- rforecast(s.wineFortRosen, groups = list(1:14),
```

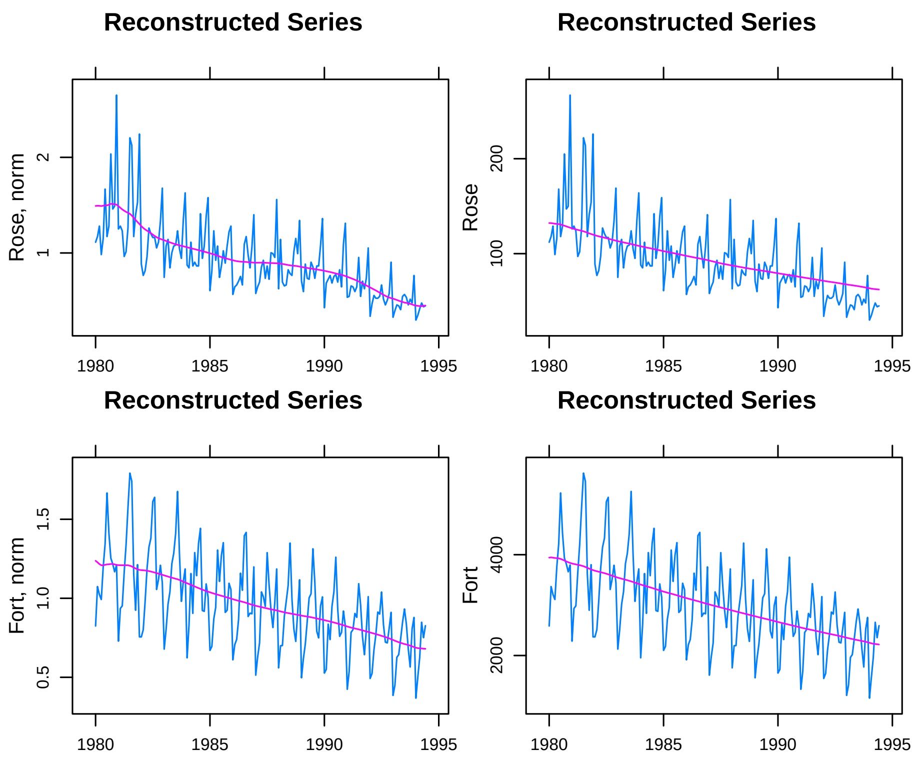


Figure 14: FORT and ROSE: Trends with normalization (ET1, 12, 14) and without (ET1).

```
+    len = 13, only.new = TRUE)[, "Rose"]
R> f.wineFortRosen_long <- c(rep(NA, 174),
+    norm.wineFortRosen["Rose"] * f.wineFortRosen)
R> xyplot(AustralianWine[100:187, "Rose"] + f.wineFortRosen_long[100:187] ~
+    time(AustralianWine)[100:187], type = "l", xlab = "Time",
+    ylab = "Rose", lty = c(1, 2))
```

Fragment 12 implements (B) and (C). Also, we compare the forecast of Total sales separately (TOTAL) and together with the available sum of sales of main wines (MAINSALES). We consider two simultaneous forecasts together with changing the weight of the series MAINSALES from 1 to 100. Figure 16 clearly demonstrates that for small contributions of MAINSALES the simultaneous forecast is close to the separate forecast of TOTAL, while for large weights of MAINSALES the simultaneous forecast of TOTAL tends to have the form similar to the form of MAINSALES. An additional investigation is needed to choose the better version.

Fragment 12 (TOTAL: Different ways of forecasting)**.**

```
R> FilledRoseAustralianWine <- AustralianWine
R> FilledRoseAustralianWine[175:176, "Rose"] <- f.wineFortRosen_long[175:176]
R> mainsales <- ts(rowSums(FilledRoseAustralianWine[, -1]))
R> total <- FilledRoseAustralianWine[, "Total"]
R> L <- 84
R> s.totalmain1 <- ssa(list(mainsales[12:187], total[1:176]),
```

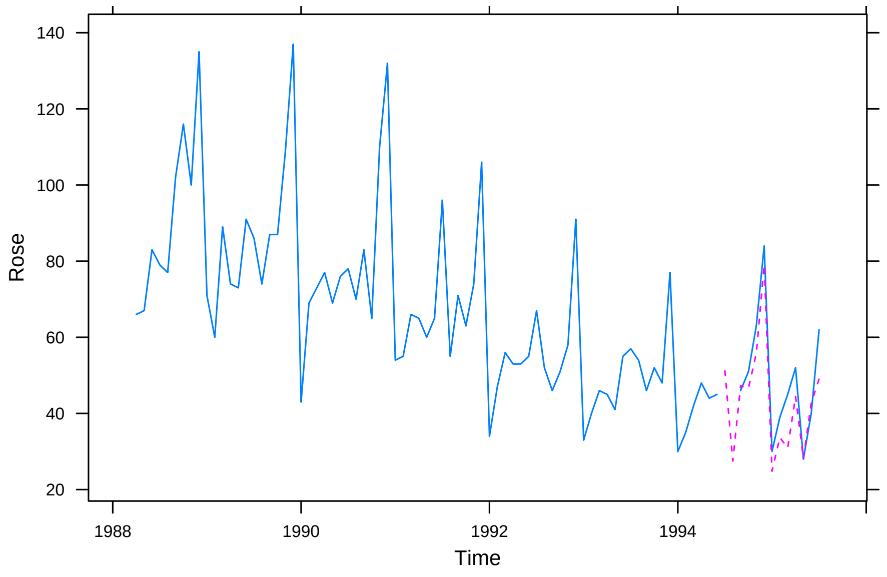


Figure 15: FORT and ROSE: Forecast of ROSE in comparison with the existing data.

```
+    L = L, kind = "mssa")
R> f.totalmain1 <- rforecast(s.totalmain1, groups = list(1:14),
+    len = 11, only.new = TRUE)
R> s.totalmain2 <- ssa(list(100 * mainsales[12:187], total[1:176]),
+    L = L, kind = "mssa")
R> f.totalmain2 <- rforecast(s.totalmain2, groups = list(1:14),
+    len = 11, only.new = TRUE)
R> s.total <- ssa(total[1:176], L = L, kind = "1d-ssa")
R> f.total <- rforecast(s.total, groups = list(1:14),
+    len = 11, only.new = TRUE)
R> xtime <- time(AustralianWine)[177:187]
R> xtime.labels <- paste(month.abb[round(xtime * 12) %% 12 + 1],
+    floor(xtime), sep = ", ")
R> xyplot(f.total + f.totalmain1[[2]] + f.totalmain2[[2]] +
+    mainsales[177:187] ~ xtime, type = "l", xlab = "Time", ylab = "Total",
+    scales = list(x = list(labels = xtime.labels)),
+    auto.key = list(text = c("Separate forecast of 'Total'",
+    "Forecast of 'Total' using 'Main'",
+    "Forecast of 'Total' using 'Main' with weight 100",
+    "Known `Main' sales"), lines = TRUE, points = FALSE))
```

Simultaneous decomposition of many series

In this example, we consider a system of many time series and show that the decomposition by MSSA helps to look at similar patterns of the series. We will use the methodology described in the previous sections.

Let us consider the collection of $s = 6$ series from the Australian wine dataset, which includes

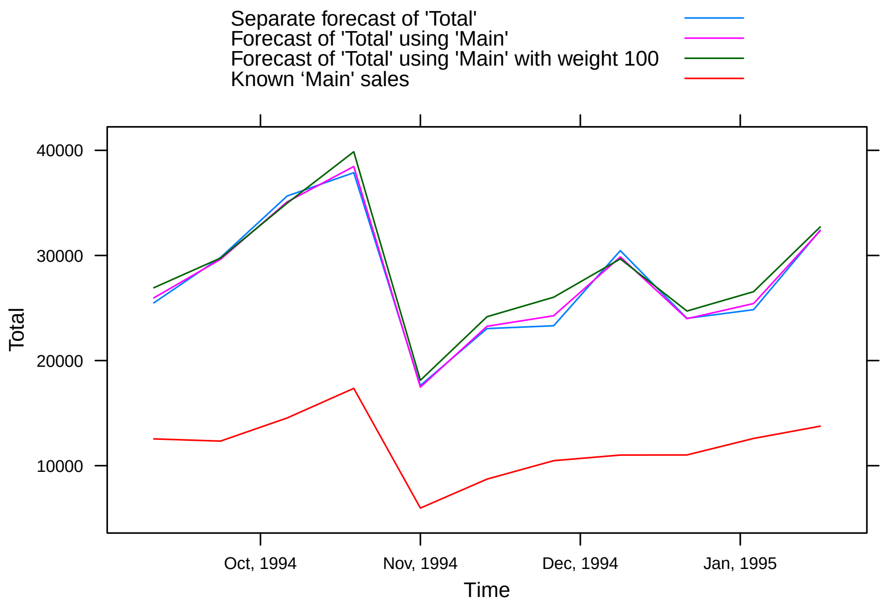


Figure 16: TOTAL: Forecast of total sales, separate and with the available information.

the series of wine sales considered in the typical code from Section 3.4. A considerable part of this multivariate series can be described as seasonality. Therefore, MSSA can have an advantage over conventional SSA.

Since the time series have different scales, namely, the Rosé wines sales have the order of 100 thousand of litres in a month, while the sales of fortified wines are about 30 times larger, the time series should be transformed onto the same scale. We choose the window length $L = 163$, then $K_p = 12$ and $K = 84$ and therefore the number of elementary components is equal to $84 = \min(163, 84)$. For the choice of L that makes the number of elementary components larger, there can be elementary components with approximately equal contribution (there can be a lack of strong separability). It appears that the decomposition with the choice $L = 163$ is better than, say, a more detailed decomposition with $L = 151$, $K_p = 24$, since the choice $L = 163$ helps to avoid mixture of components.

The identification of the trend (ET1, 2, 5) and the seasonality (ET3, 4, 6–12) is performed on the basis of eigenvectors and uses the principles described in the typical code from Section 3.4. Fragment 13 contains the code to get the reconstruction shown in Figures 17 and 18.

Fragment 13 (Wine sales: Simultaneous decomposition by MSSA)**.**

```
R> L <- 163
R> norm.wine <- sqrt(colMeans(wine[, -1]^2))
R> winen <- sweep(wine[, -1], 2, norm.wine, "/")
R> s.winen <- ssa(winen, L = L, kind = "mssa")
R> r.winen <- reconstruct(s.winen,
+    groups = list(Trend = c(1, 2, 5), Seasonality = c(3:4, 6:12)))
R> 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"),
```

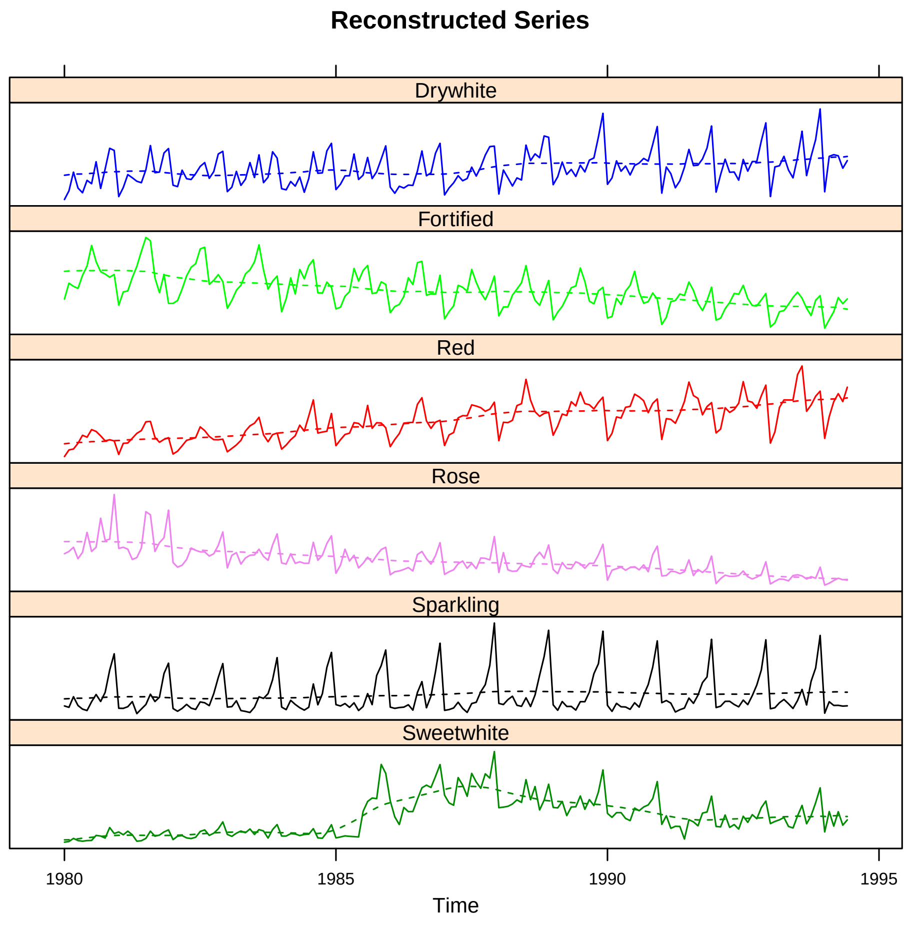


Figure 17: Wine sales: Extraction of trends.

```
+     lty = rep(c(1, 2), each = 6), scales = list(y = list(draw = FALSE)),
+     layout = c(1, 6))
R> 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, since we obtain similar time series components all at once. In particular, it is clearly seen from Figure 18 that the sale volumes of fortified wines are maximal in June–July (that are winter months in Australia), while the sale volumes of sparkling wines have a peak in December.

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 repeat the results from Golyandina and Stepanov (2005) and supplement them with new comparisons. In particular,

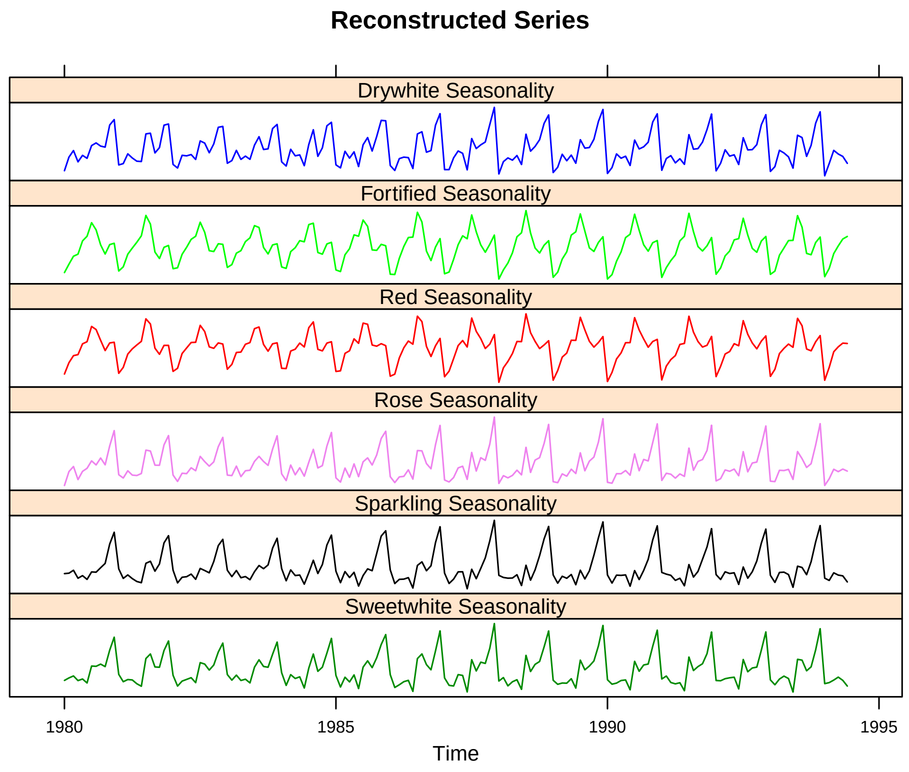


Figure 18: Wine sales: Extraction of seasonality.

the comparison results explain the choice of the default forecasting method.

In the study below, we consider the case $s = 2$ and hence include CSSA onto the range of SSA methods we compare. 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 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 and $\mathbb{N}^{(1)}$ and $\mathbb{N}^{(2)}$ are realizations of independent Gaussian white noises. Then we can use the standard simulation procedure to obtain estimates of mean square errors (MSE) for reconstruction and forecasting of $(\mathbb{H}^{(1)}, \mathbb{H}^{(2)})$ by the considered above SSA methods. Note that the resultant MSE is calculated as the mean of $\mathrm{MSE}^{(1)}$ and $\mathrm{MSE}^{(2)}$ for $\mathbb{H}^{(1)}$ and $\mathbb{H}^{(2)}$, respectively.

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 not equal to $\pi/2$):

$$h_k^{(1)} = 30\cos(2\pi k/12), \quad h_k^{(2)} = 20\cos(2\pi k/12 + \pi/4), \quad k = 1, \ldots, N.$$

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 A	Example B	Example C
MSSA	**2**	2	4
SSA	2	2	**2**
CSSA	2	**1**	4

Table 2: Dimension of the signal trajectory space.

Example A	$L = 12$	$L = 24$	$L = 36$	$L = 48$	$L = 60$
MSSA	3.18	1.83	1.59	***1.47***	2.00
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
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
SSA	3.23	***2.01***	***2.00***	***2.01***	3.23
CSSA	6.98	4.06	**3.82**	4.06	6.98

Table 3: MSE of signal reconstruction.

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 fact that the dimensions of the signal trajectory spaces (i.e., ranks) are different for different extensions of SSA methods, see Table 2. For each example the rank in bold font corresponds to the method with the best accuracy for this example.

The results of investigating different window lengths L are summarized in Tables 3 and 4. 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 italics.

Comparison of Tables 3 and 4 with Table 2 clearly demonstrates the relation between the accuracy of the signal reconstruction/forecast and the dimension of the signal trajectory space.

Note that the reconstruction by SSA and CSSA is 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 SSA-trajectory matrix has rank equal to $\min(L, N - L + 1)$ and the rank is maximal for $L \approx (N + 1)/2$. 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 $L \approx s(N + 1)/(s + 1)$. Although the maximality of the rank does not guarantee the minimality of errors, this consideration means that the window length L for better separability might be larger than $(N + 1)/2$. The 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

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
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
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
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
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

Table 4: MSE of signal forecast.

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 include matched components.
- Optimal window lengths for analysis and forecasting can differ. The accuracy of forecast is related to the accuracy of reconstruction; however, 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 fulfilled when comparing the accuracy of methods for the same window length. Note that this is probably valid for forecasting of well-separated signals of finite rank only, see Remark 1 for an explanation.
- The recommendations for the choice of window length (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 row and column forecasting (SSA and CSSA forecasting methods are particular cases of column forecasting) the recommendations are also opposite. It is not surprising, since L and K are swapped.

Fragment 14 demonstrates how the **Rssa** package allows for estimation of the reconstruction and forecast accuracy on the example of MSSA and CSSA analysis and vector forecasting applied to Example A for $R = 10$. The full code used to obtain the numbers in Tables 3 and 4 can be found in the replication materials. Note that $R = 10000$ was used there and therefore the running time is quite large.

Fragment 14 (Simulation for reconstruction and forecasting accuracy estimation)**.**

```
R> N <- 71
R> sigma <- 5
R> Ls <- c(12, 24, 36, 48, 60)
R> len <- 24
R> signal1 <- 30 * cos(2 * pi * (1:(N + len)) / 12)
R> signal2 <- 30 * cos(2 * pi * (1:(N + len)) / 12 + pi / 4)
R> signal <- cbind(signal1, signal2)
R> R <- 10
R> 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)
+  }
R> mres <- replicate(R, mssa.errors(Ls))
R> err.rec <- rowMeans(simplify2array(mres["Reconstruction", ]))
R> err.for <- rowMeans(simplify2array(mres["Forecast", ]))
R> err.rec

      12       24       36       48       60
2.869683 1.587789 1.248881 1.153730 1.855115

R> err.for
```

```
      12       24       36       48       60
2.671251 2.578059 1.501565 2.595378 4.564218

R> signal <- signal1 + 1i*signal2
R> 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)
+  }
R> cres <- replicate(R, cssa.errors(Ls))
R> err.rec <- rowMeans(simplify2array(cres["Reconstruction", ]))
R> err.for <- rowMeans(simplify2array(cres["Forecast", ]))
R> err.rec

      12       24       36       48       60
7.349316 4.298144 4.101666 4.298144 7.349316

R> err.for

      12       24       36       48       60
24.67425 13.60116 14.54819 11.72135 15.86380
```

4. 2D-singular spectrum analysis

In this section, we consider the extension of the 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 SSA in Danilov and Zhigljavsky (1997), and was further developed in Golyandina and Usevich (2010) and Rodríguez-Aragón and Zhigljavsky (2010). However, its first usage can be traced back to the work of Ade (1983) on texture

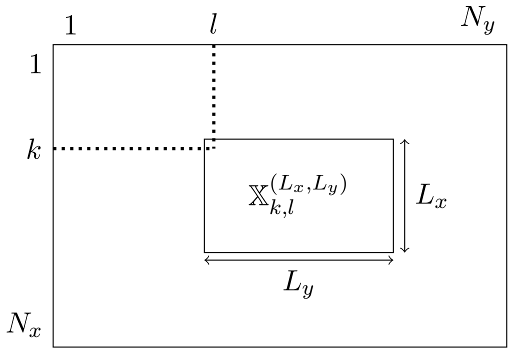


Figure 19: Moving 2D windows.

analysis (this work was also continued recently, see Monadjemi 2004). Related decompositions can be found in methods for processing of seismological data (Trickett 2008). Finally, as with SSA-like methods for time series, the 2D-SSA decomposition is the basis of subspace-based parameter estimation methods for sums of two-dimensional complex exponentials (see, e.g., Rouquette and Najim 2001).

A 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 overcomes this deficiency.

4.1. 2D-SSA algorithm

For a matrix $\mathrm{A} \in \mathsf{R}^{M\times N}$ (or $\mathsf{C}^{M\times N}$), we denote by $\mathrm{vec}(\mathrm{A}) \in \mathsf{R}^{MN}$ (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 $\mathrm{vec}_M^{-1}(A) = \mathrm{B} \in \mathsf{R}^{M\times N}$ (or $\mathsf{C}^{M\times N}$) that satisfies $\mathrm{vec}(\mathrm{B}) = A$. We mostly use the notation in Golyandina and Usevich (2010).

The embedding operator

Since the general scheme of the 2D-SSA algorithm is described in Section 2, we need to define only the embedding operator $\mathcal{T}_{\text{2D-SSA}}(\mathbb{X}) = \mathbf{X}$.

The parameters of the method are the two-dimensional *window sizes* (L_x, L_y), which are bounded as $1 \le L_x \le N_x$, $1 \le L_y \le N_y$ and $1 < L_xL_y < N_xN_y$. 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 $L = L_xL_y$ (the number of rows of $\mathbf{X}$) and $K = K_xK_y$ (the number of columns of $\mathbf{X}$).

Consider all possible $L_x \times L_y$ submatrices of $\mathbb{X}$ (2D sliding windows). For $k = 1, \dots, K_x$ and $l = 1, \dots, K_y$, we define by $\mathbb{X}_{k,l}^{(L_x,L_y)} = (x_{i,j})_{i=k,j=l}^{L_x+k-1,L_y+l-1}$ the $L_x \times L_y$ submatrix shown in Figure 19. 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 because it is consistent with the standard mathematical indexing of matrices (Golyandina and Usevich 2010).

Then the trajectory matrix is defined as

$$\mathcal{T}_{\text{2D-SSA}}(\mathbb{X}) = \mathbf{X} = [X_1 : \ldots : X_{K_xK_y}], \tag{16}$$

where the columns are vectorizations of $L_x \times L_y$ submatrices:

$$X_{k+(l-1)K_x} = \mathrm{vec}(\mathbb{X}_{k,l}^{(L_x,L_y)})$$

Hankel-block-Hankel structure

The trajectory matrix (16) has the following structure (Golyandina and Usevich 2010):

$$\mathbf{X} = \mathcal{T}_{\text{2D-SSA}}(\mathbb{X}) = \begin{pmatrix} \mathbf{H}_1 & \mathbf{H}_2 & \mathbf{H}_3 & \dots & \mathbf{H}_{K_y} \\ \mathbf{H}_2 & \mathbf{H}_3 & \mathbf{H}_4 & \dots & \mathbf{H}_{K_y+1} \\ \mathbf{H}_3 & \mathbf{H}_4 & \iddots & \iddots & \vdots \\ \vdots & \vdots & \iddots & \iddots & \vdots \\ \mathbf{H}_{L_y} & \mathbf{H}_{L_y+1} & \dots & \dots & \mathbf{H}_{N_y} \end{pmatrix}, \tag{17}$$

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, $\mathbf{H}_j = \mathcal{T}_{\text{SSA}}(\mathbb{X}_{:,j})$, where $\mathcal{T}_{\text{SSA}}$ is defined in (2). The matrix (16) is called *Hankel-block-Hankel* (shortened to *HbH*), since it is block-Hankel with Hankel blocks.

Trajectory space

From (16) we have 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 also can be viewed as vectorized $L_x \times L_y$ arrays. Their devectorizations are denoted by $\Psi_i = \mathrm{vec}_{L_x}^{-1}(U_i)$. Similarly, the rows of $\mathbf{X}$ are vectorizations of the (K_x, K_y) submatrices

$$\mathbf{X} = [X^1 : \ldots : X^{L_x L_y}]^\top, \quad X^{k+(l-1)L_x} = \mathrm{vec}(\mathbb{X}_{k,l}^{(K_x,K_y)}), \tag{18}$$

where X^j is the jth row of the matrix $\mathbb{X}$. Therefore, the factor vectors V_i also can be viewed as $K_x \times K_y$ arrays. Their devectorizations are denoted by $\Phi_i = \mathrm{vec}_{K_x}^{-1}(U_i)$.

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 idea will be extended later on in Section 5.

2. The arrays of finite rank in 2D-SSA (i.e., the arrays such that $\mathcal{T}_{\text{2D-SSA}}$ has finite rank) are sums of products of polynomials, exponentials and cosines, similarly to the one-dimensional case. More details can be found in Appendix B.

4.2. Package

Typical code

Here we demonstrate a typical decomposition of 2D images with the package **Rssa**. We follow the example in Section 2.3, but stress on the differences that appear in the 2D case.

As an example, we use the image of Mars from Buil (2010b) (Source: Pierre Thierry), a tutorial to the free **IRIS** software (see the data description in the package; Buil 2010a). The image is of size 258×275, 8-bit gray scale, values from 0 to 255. The input code for this image can be found in Fragment 15 (the image is included in the package **Rssa**).

Fragment 15 (Mars: Input)**.**

```
R> library("Rssa")
R> data("Mars", package = "Rssa")
```

We would like to decompose this image with a 25×25 window (see the example in Golyandina and Usevich 2010). Easy calculations show that even these small window sizes would give rise to a 625×58734 trajectory matrix. In Fragment 16 with `svd.method = "svd"`, we intentionally commented out the call to the SSA function, because we do not recommend to use it, unless the trajectory matrix is very small.

Fragment 16 (Mars: Decomposition with `svd.method = "svd"`)**.**

```
R> # ssa(Mars, kind = "2d-ssa", L = c(25, 25), svd.method = "svd")
```

A remedy for this could be the calculation of just the matrix $\mathbf{X}\mathbf{X}^\top$, and computing its eigendecomposition (this approach was taken in Golyandina and Usevich 2010 and other papers). In the package **Rssa**, this is implemented in `svd.method = "eigen"`, see Fragment 17.

Fragment 17 (Mars: Decomposition with `svd.method = "eigen"`)**.**

```
R> system.time(ssa(Mars, kind = "2d-ssa", L = c(25, 25),
+    svd.method = "eigen"))

   user  system elapsed
  5.118   0.051   5.134
```

However, 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 by Fragment 18. Note that for the 2D-SSA decomposition, `kind = "2d-ssa"` should be used.

Fragment 18 (Mars: Decomposition)**.**

```
R> system.time(s.Mars.25 <- ssa(Mars, kind = "2d-ssa", L = c(25, 25)))

   user  system elapsed
  0.701   0.006   0.708
```

Fragment 19 shows a typical reconstruction code for 2D-SSA.

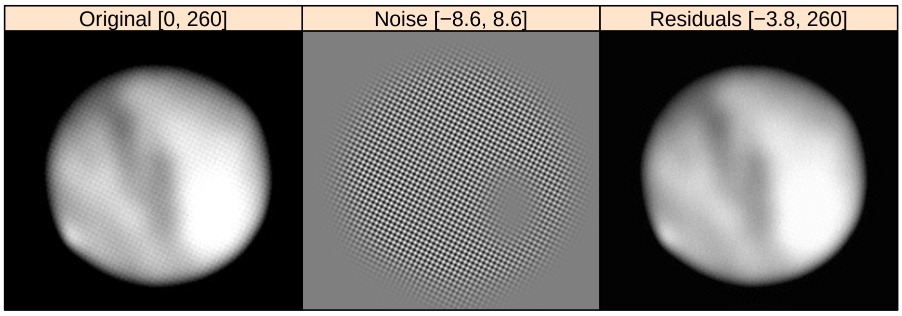


Figure 20: Mars: Separated periodic noise, $(L_x, L_y) = (25, 25)$.

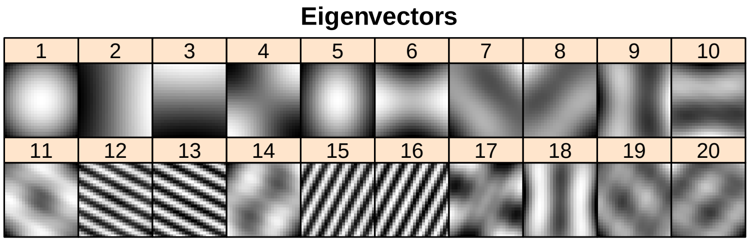


Figure 21: Mars: Eigenarrays, $(L_x, L_y) = (25, 25)$.

Fragment 19 (Mars: Reconstruction)**.**

```
R> r.Mars.25 <- reconstruct(s.Mars.25,
+    groups = list(Noise = c(12, 13, 15, 16)))
R> plot(r.Mars.25, cuts = 255, layout = c(3, 1))
```

The reconstruction results are shown in Figure 20.

The grouping for this decomposition was made, as in Golyandina and Usevich (2010), based on the following information:

- eigenarrays Ψ_i (see Figure 21), and,
- the matrix of **w** correlations (see Figure 22).

Fragment 20 shows the corresponding code.

Fragment 20 (Mars: Identification)**.**

```
R> plot(s.Mars.25, type = "vectors", idx = 1:20,
+    cuts = 255, layout = c(10, 2), plot.contrib = FALSE)
R> plot(wcor(s.Mars.25, groups = 1:30), scales = list(at = c(10, 20, 30)))
```

Next, we try more challenging window sizes $(L_x, L_y) = (160, 80)$, where the trajectory matrix is of size 12800×19404. In this case, `svd.method = "eigen"` would take a very long time. However, with the default method `svd.method = "nutrlan"` the computation of the 50 first

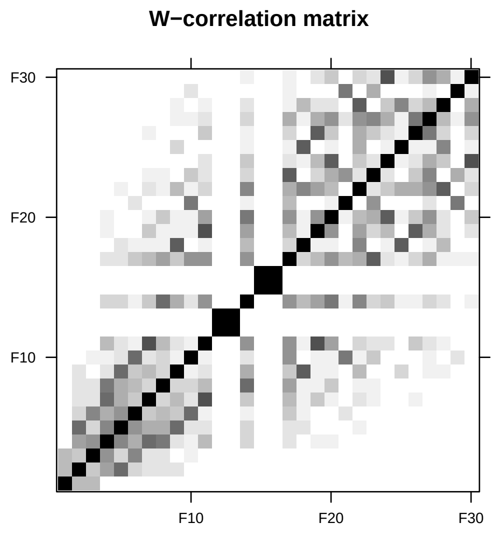


Figure 22: Mars: $\mathbf{w}$ correlations, $(L_x, L_y) = (25, 25)$.

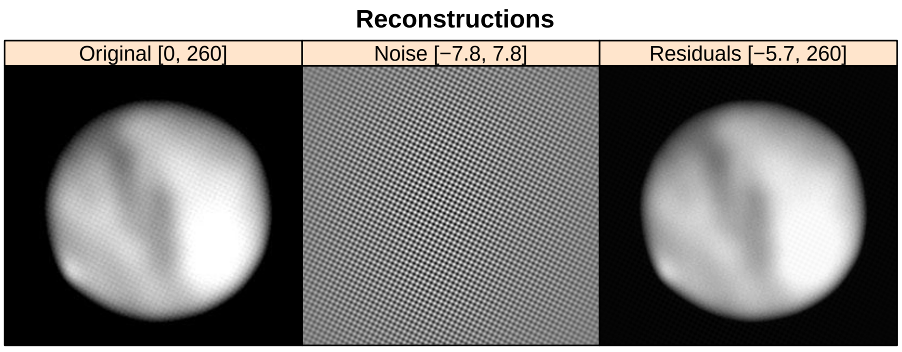


Figure 23: Mars: Reconstruction, $(L_x, L_y) = (160, 80)$.

eigenvectors can be done in about a second, see Fragment 21. The results of the reconstruction are shown in Figure 23.

Fragment 21 (Mars: Reconstruction)**.**

```
R> system.time(s.Mars.160.80 <-
+    ssa(Mars, kind = "2d-ssa", L = c(160, 80)))

   user  system elapsed
  0.694   0.012   0.671

R> r.Mars.160.80.groups <- list(Noise = c(36, 37, 42, 43))
R> r.Mars.160.80 <- reconstruct(s.Mars.160.80, groups = r.Mars.160.80.groups)
R> plot(r.Mars.160.80, cuts = 255, layout = c(3, 1))
```

From Figure 23 we see that in the case of large window sizes, the extracted periodic noise is not modulated (compare to Figure 20). This can be interpreted as follows. We choose 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 (see Golyandina 2010). On the other hand, if we choose smaller window sizes (for example, 25×25), then the 2D-SSA decomposition acts more like smoothing.

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. All the plotting routines by default use the raster representation (via the `useRaster = TRUE` argument 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 the 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 prohibitively time consuming even for moderate image and window sizes. (Recall that $L = L_x L_y$ and $K = K_x K_y$.) The ideas presented in Section 6.2 coupled with Lanczos-based truncated SVD implementations (Larsen 1998; Yamazaki, Bai, Simon, Wang, and Wu 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 SSA and MSSA cases.

Note that the Lanczos-based methods have significant overhead for small trajectory matrices, so that in this case other SVD methods should be used. For `svd.method = "eigen"`, the matrix $\mathbf{X}\mathbf{X}^\top$ is computed in $O(LN \log N)$ flops using the fast matrix-vector multiplication from Section 6.2. 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.

4.3. Examples

Adaptive smoothing

2D-SSA can also be used for adaptive smoothing of two-dimensional data. It was in Golyandina, Florinsky, and Usevich (2007) for smoothing digital terrain models (DTM) and in Holloway, Lopes, Costa, Travençolo, Golyandina, Usevich, and Spirov (2011) and Golyandina, Holloway, Lopes, Spirov, Spirova, and Usevich (2012) 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 2015). 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) lies in a neighborhood of Newport, and $(1, N_y)$ is near Whitney-on-Wye. In Fragment 22, we decompose the image with a small window, and plot the eigenvectors and the matrix of **w** correlations in Figure 24.

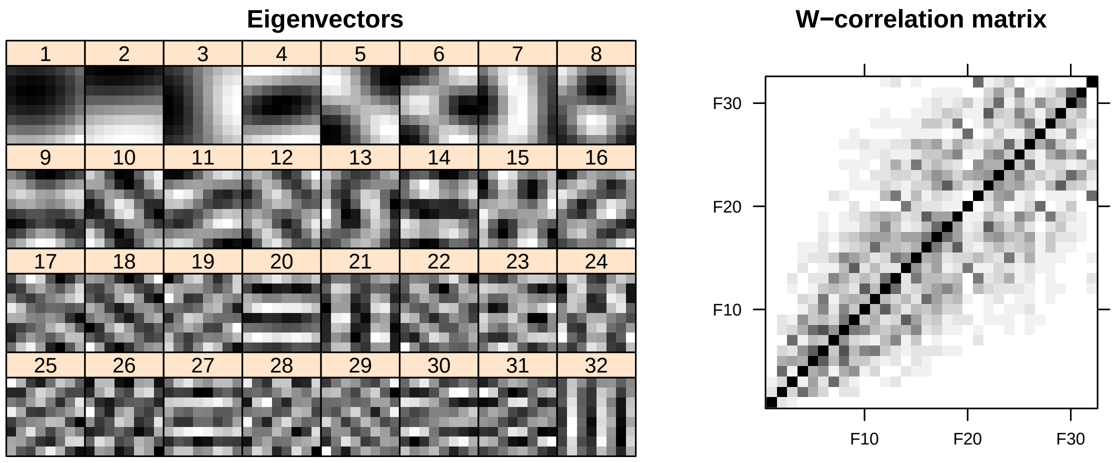


Figure 24: Brecon Beacons: 8×8 windows, eigenarrays and $\mathbf{w}$ correlations.

Fragment 22 (Brecon Beacons: Decomposition)**.**

```
R> library("raster")
R> library("Rssa")
R> UK <- getData("alt", country = "GB", mask = TRUE)
R> brecon <- crop(UK, extent(UK, 1040, 1119, 590, 689))
R> m.brecon <- as.matrix(brecon)
R> s.brecon <- ssa(m.brecon, kind = "2d-ssa", L = c(8, 8),
+    svd.method = "eigen")
R> plot(s.brecon, type = "vectors", idx = 1:32,
+    cuts = 255, layout = c(8, 4), plot.contrib = FALSE)
R> plot(wcor(s.brecon, groups = 1:32), scales = list(at = c(10, 20, 30)))
```

Next, we reconstruct the image with components with $I_1 = \{1, \ldots, 3\}$, $I_2 = \{4, \ldots, 8\}$, 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 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 $J_3 = I_1 \cup I_2 \cup I_3$). We plot the reconstructed components in Figure 25. The cumulative components $\widetilde{\mathbb{Y}}_k$ are shown in Figure 26 using the type `"cumsum"` of the plotting function of **Rssa**.

Fragment 23 (Brecon Beacons: Reconstruction)**.**

```
R> r.brecon <- reconstruct(s.brecon, groups = list(1:3, 4:8, 9:17))
R> plot(r.brecon, cuts = 255, layout = c(5, 1),
+    par.strip.text = list(cex = 0.75))
R> plot(r.brecon, cuts = 255, layout = c(5, 1),
+    par.strip.text = list(cex = 0.75), type = "cumsum", at = "free")
R> brecon.F1 <- raster(r.brecon$F1, template = brecon)
R> brecon.F2 <- raster(r.brecon$F2, template = brecon)
```

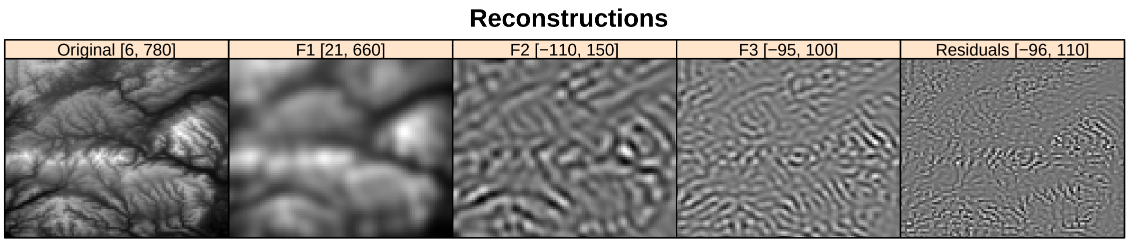


Figure 25: Brecon Beacons: 8×8 window, reconstructions ($\widetilde{\mathbb{X}}_k$).

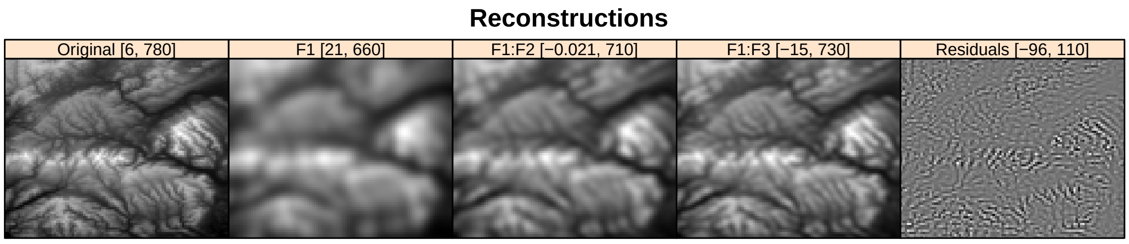


Figure 26: Brecon Beacons: 8×8 window, cumulative reconstructions ($\widetilde{\mathbb{Y}}_k$).

From Figures 25 and 26 it can be seen that the reconstructed components capture morphological features of different scale (Golyandina *et al.* 2007). Cumulative reconstructions represent smoothing of the original DTM of different resolution.

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 cumulative reconstructions, see Figure 27). The corresponding code can be found in Fragment 25. We also introduce some code for plotting the arrays and computing their centered DFTs in Fragment 24.

Fragment 24 (2D-SSA: Plotting functions)**.**

```
R> 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 = "")
+  }
R> 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 25 (Brecon Beacons: DFT of cumulative reconstructions)**.**

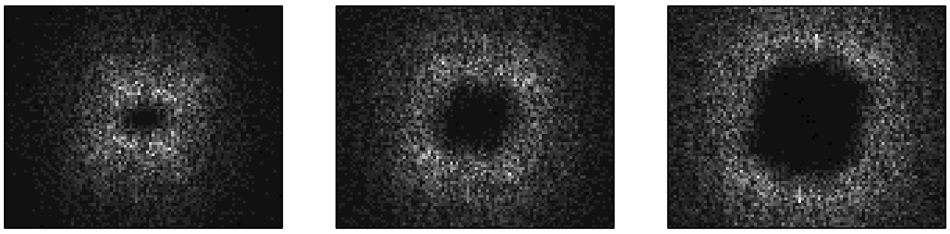


Figure 27: Brecon Beacons: 8×8 window, absolute values of the DFT of $\mathbb{X} - \widetilde{\mathbb{Y}}_k$, $k = 1, \ldots, 3$.

```
R> library("lattice")
R> plot2d(centered.mod.fft(m.brecon - r.brecon$F1))
R> plot2d(centered.mod.fft(m.brecon - r.brecon$F1 - r.brecon$F2))
R> plot2d(centered.mod.fft(m.brecon - r.brecon$F1 - r.brecon$F2-r.brecon$F3))
```

In Figure 27, it is clearly seen that 2D-SSA reconstruction by leading components acts as filter that preserves dominating frequencies (in this case, a low-pass filter).

Parameter estimation

2D-SSA decomposition is also used in subspace-based methods of parameter estimation. Let $\mathbb{X} = \mathbb{S} + \mathbb{R}$, where $\mathbb{S}$ is an array of finite rank and $\mathbb{R}$ is the residual. If the signal and noise are (approximately) separable, then the matrix $\widehat{U} \in \mathsf{R}^{L_x L_y \times r}$ of the basis eigenvectors approximates the original signal subspace of $\mathbb{S}$.

Some of the methods of 2D-ESPRIT type are implemented in **Rssa**. The methods are based on computing a pair of shift matrices for x and y directions and their joint diagonalization (Rouquette and Najim 2001). Currently, two methods of joint diagonalization are implemented in **Rssa** 2D-ESPRIT (from Rouquette and Najim 2001) and 2D-MEMP with improved pairing step (see Rouquette and Najim 2001; Wang, Chan, and Liu 2005).

We continue the Mars example from Fragment 21. This example demonstrates advantages of **Rssa** because of the possibility to choose large window sizes, which was always considered to be problematic ESPRIT-type methods. Fragment 26 shows the corresponding code that outputs the estimated exponentials. In Figure 28, the estimated pairs of complex exponentials (λ_k, μ_k), are shown on separate complex plane plots and the points for λ_k and μ_k have the same color (for the same k).

Fragment 26 (Mars: Parameter estimation with 2D-ESPRIT)**.**

```
R> pe.Mars.160.80 <- parestimate(s.Mars.160.80,
+    groups = r.Mars.160.80.groups)
R> 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
```

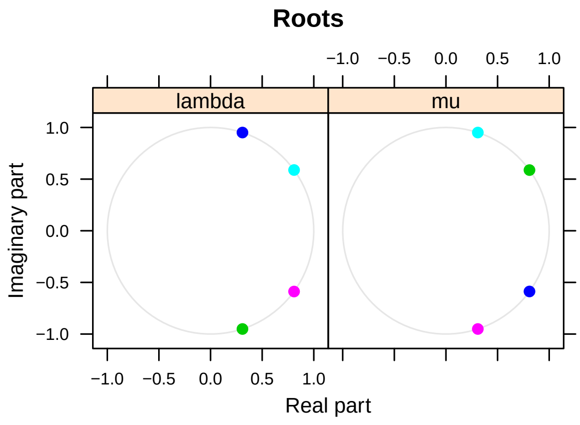


Figure 28: Mars: Parameter estimation with 2D-ESPRIT, $(L_x, L_y) = (160, 80)$.

```
R> 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

R> 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

R> plot(pe.Mars.160.80, col = c(11, 12, 13, 14))
R> 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 26 and Figure 28, it can be seen that each pair (λ_k, μ_k) has its conjugate counterpart $(\lambda_{k'}, \mu_{k'}) = \mathrm{conj}((\lambda_k, \mu_k))$ (where $\mathrm{conj}(\cdot)$ denotes the complex conjugation). Indeed, it is the case for $k = 1, k' = 2$, and for $k = 3, k' = 4$. Therefore, the periodic noise is a sum of two planar sines, as explained in Appendix B. This is also confirmed by the plots of eigenarrays in Figure 29.

5. Shaped 2D-singular spectrum analysis

Shaped 2D-SSA (or *ShSSA* for short) is a generalization of 2D-SSA, which allows an arbitrary shape of the input array and window. This feature considerably extends the range of real-life applications, since it makes it possible to decompose parts of the image with different

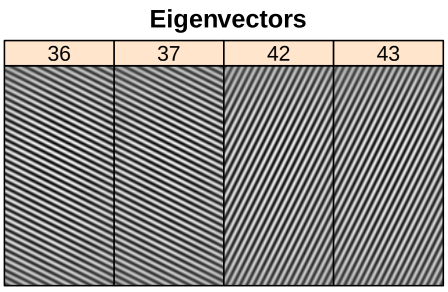


Figure 29: Mars: Eigenarrays corresponding to periodic noise, $(L_x, L_y) = (160, 80)$.

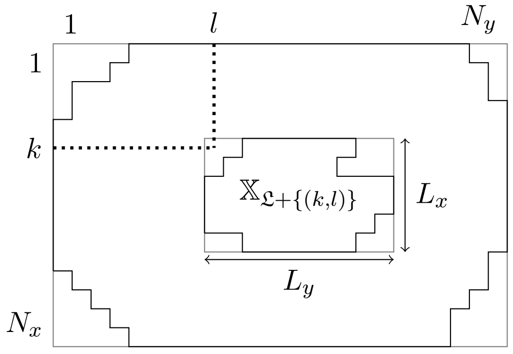


Figure 30: Moving shaped windows.

structures separately, to exclude the areas with corrupted data, to analyze images with gaps, to decompose non-rectangular images, etc. In ShSSA, not all the values of the rectangular image need to be specified, and the sliding window is not necessarily rectangular (see Figure 30).

Formally speaking, a *shape* $\mathfrak{B}$ is a bounded subset of N^2 (a set of two-dimensional natural indices). A $\mathfrak{B}$-*shaped array* is a partially indexed array $\mathbb{X} = \mathbb{X}_{\mathfrak{B}} = (x_{(i,j)})_{(i,j)\in\mathfrak{B}}$.

For two-dimensional indices $\ell = (\ell_x, \ell_y)$ and $\kappa = (\kappa_x, \kappa_y)$ we define a shifted sum

$$\ell +\!\!\text{-}\, \kappa = (\ell_x + \kappa_x - 1, \ell_y + \kappa_y - 1).$$

We also define a shifted Minkowski sum of two shapes $\mathfrak{A}$ and $\mathfrak{B}$ as

$$\mathfrak{A} +\!\!\text{-}\, \mathfrak{B} = \{\alpha +\!\!\text{-}\, \beta \mid \alpha \in \mathfrak{A}, \beta \in \mathfrak{B}\}.$$

5.1. Construction of the trajectory matrix

The input data for the algorithm is an $\mathfrak{N}$-shaped array $\mathbb{X} = \mathbb{X}_{\mathfrak{N}} = (x_\alpha)_{\alpha\in\mathfrak{N}}$, where $\mathfrak{N} \subseteq \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$. The parameter of the algorithm is a *window shape* $\mathfrak{L} \subseteq \{1, \ldots, L_x\} \times \{1, \ldots, L_y\}$, given as $\mathfrak{L} = \{\ell_1, \ldots, \ell_L\}$, where $\ell_i \in \mathsf{N}^2$ are ordered in lexicographical order (i.e., the order in which the elements x_α would appear in the vectorized rectangular array $\mathrm{vec}(\mathbb{X}_{\{1,\ldots,N_x\}\times\{1,\ldots,N_y\}})$).

The embedding operator

For $\kappa \in \mathsf{N}^2$ we define a shifted $\mathfrak{L}$-shaped subarray as $\mathbb{X}_{\mathfrak{L}+\!\text{-}\{\kappa\}} = (x_\alpha)_{\alpha\in\mathfrak{L}+\!\text{-}\{\kappa\}}$ (see Figure 30).

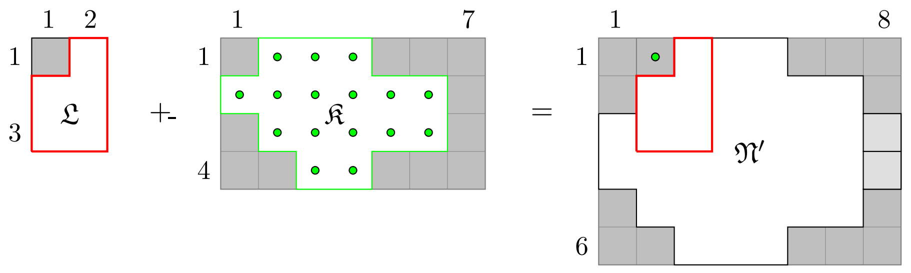


Figure 31: Construction of $\mathfrak{K}$.

The index κ is a position of the origin for the window. Consider the set of all K possible origin positions for $\mathfrak{L}$-shaped windows:

$$\mathfrak{K} = \{\kappa \in \mathsf{N}^2 \,|\, \mathfrak{L} +\!\!\!- \{\kappa\} \subseteq \mathfrak{N}\}. \tag{19}$$

We assume that $\mathfrak{K} = \{\kappa_1, \ldots, \kappa_K\} \subset \mathsf{N}^2$, where κ_j are ordered in lexicographical order. Then the trajectory matrix is constructed as follows:

$$\mathcal{T}_{\text{ShSSA}}(\mathbb{X}) := \mathbf{X} = [X_1 : \ldots : X_K], \tag{20}$$

where the columns

$$X_j = (x_{\ell_i +\!\!\!- \kappa_j})_{i=1}^{l}$$

are vectorizations of the shaped submatrices $\mathbb{X}_{\mathfrak{L}+\!\!\!-\{\kappa_j\}}$.

Quasi-Hankel matrix

The trajectory matrix is exactly the *quasi-Hankel* matrix (Mourrain and Pan 2000) constructed from the sets $\mathfrak{L}, \mathfrak{K} \subset \mathsf{N}^2$:

$$\mathbf{X} = \mathcal{T}_{\text{ShSSA}}(\mathbb{X}) = \begin{pmatrix} x_{\ell_1+\!\!\!-\kappa_1} & x_{\ell_1+\!\!\!-\kappa_2} & \ldots & x_{\ell_1+\!\!\!-\kappa_K} \\ x_{\ell_2+\!\!\!-\kappa_1} & x_{\ell_2+\!\!\!-\kappa_2} & \ldots & x_{\ell_2+\!\!\!-\kappa_K} \\ \vdots & \vdots & & \vdots \\ x_{\ell_L+\!\!\!-\kappa_1} & x_{\ell_L+\!\!\!-\kappa_2} & \ldots & x_{\ell_L+\!\!\!-\kappa_K} \end{pmatrix}. \tag{21}$$

Note that Mourrain and Pan (2000) use the conventional sum of indices instead of the shifted sum $+\!\!\!-$, because their definition of natural numbers N includes 0.

Comments

If $\mathfrak{N} = \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$ and $\mathfrak{L} = \{1, \ldots, L_x\} \times \{1, \ldots, L_y\}$ (hence, $\mathfrak{K} = \{1, \ldots, K_x\} \times \{1, \ldots, K_y\}$), then the matrix $\mathcal{T}_{\text{ShSSA}}(\mathbb{X})$ coincides with $\mathcal{T}_{\text{2D-SSA}}(\mathbb{X})$. Therefore, the ShSSA decomposition with the rectangular window and input array coincides with the 2D-SSA decomposition.

In Figure 31, we demonstrate the case of general shapes. The set $\mathfrak{K}$ corresponds to all possible positions of the left upper corner of the bounding box around the shaped window (depicted by green dots in Figure 31). It also corresponds to the maximal set $\mathfrak{K}$ such that $\mathfrak{L} +\!\!\!- \mathfrak{K} \subseteq \mathfrak{N}$. We describe an algorithm for computation of the set $\mathfrak{K}$ in Section 6.

It may happen that some of the elements of the original shaped array $\mathbb{X}_{\mathfrak{N}}$ do not enter in the trajectory matrix (for example, the elements $(3, 8)$ and $(4, 8)$, shown in gray in Figure 31). In this case, the ShSSA analysis applies only to the $\mathfrak{N}'$-shaped subarray $\mathbb{X}_{\mathfrak{N}'}$, where $\mathfrak{N}'$ is the set of all indices that enter in the trajectory matrix. (In fact, $\mathfrak{N}' = \mathfrak{K} \,\underline{+}\, \mathfrak{L}$.)

Column and row spaces

From the construction of the trajectory matrix, the trajectory space is the linear space spanned by all $\mathfrak{L}$-shaped subarrays and the eigenvectors U_i also can be viewed as $\mathfrak{L}$-shaped subarrays. These arrays, as in 2D-SSA, are called eigenarrays.

Analogously, each row X^i of $\mathbf{X}$ is a vectorized (in lexicographical order) $\mathfrak{K}$-shaped array $\mathbb{X}_{\ell_i+\kappa-1}$. This is a $\mathfrak{K}$-shaped subarray of $\mathbb{X}$ starting from the element ℓ_i. Therefore, the factor vectors V_i also can be viewed as $\mathfrak{K}$-shaped subarrays. These arrays are called factor arrays.

5.2. Package

Typical code

We repeat the experiment from Section 4.2 (noise removal from the image of Mars), but using the shaped 2D-SSA. Therefore, the code for loading the image is the same as in Fragment 15.

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` — 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 27.

Fragment 27 (Mars: Mask specification and decomposition)**.**

```
R> mask.Mars.0 <- (Mars != 0)
R> mask.Mars.1 <- (Mars != 255)
R> is.na(Mars[!mask.Mars.0]) <- TRUE
R> system.time(s.Mars.shaped <-
+    ssa(Mars, kind = "2d-ssa", mask = mask.Mars.1, wmask = circle(15)))

   user  system elapsed
  0.776   0.010   0.784

R> mask.Mars.res <- (s.Mars.shaped$weights > 0)
R> plot2d(mask.Mars.0)
R> plot2d(mask.Mars.1)
R> plot2d(mask.Mars.res)
```

In Figure 32 one can see both types of masks and the combined mask. Fragment 28 shows a typical reconstruction code for ShSSA.

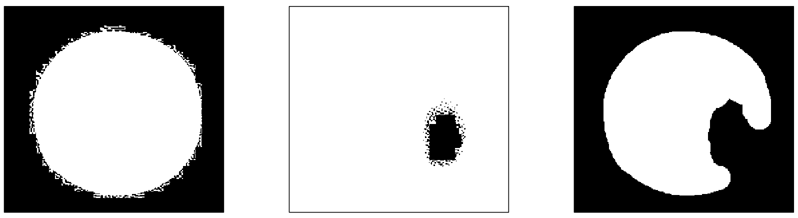


Figure 32: Mars masks specification. Left: specified by `NA`, center: the parameter `mask`, right: resulting mask. White squares — `TRUE`, black squares — `FALSE`.

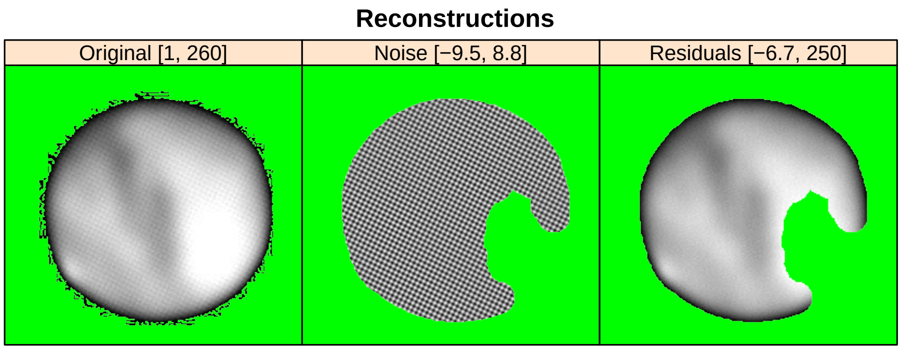


Figure 33: Mars: Reconstruction, ShSSA.

Fragment 28 (Mars: Reconstruction)**.**

```
R> r.Mars.shaped <- reconstruct(s.Mars.shaped,
+    groups = list(Noise = c(7, 8, 9, 10)))
R> plot(r.Mars.shaped, cuts = 255, layout = c(3, 1), fill.color = "green")
```

The reconstruction results are shown in Figure 33. In Figure 33 we can see that the elements are reconstructed only inside the resulting mask, however 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:

- eigenarrays (see Figure 34), and,
- the matrix of matrix of **w** correlations (see Figure 35).

Fragment 29 shows the code that reproduces Figures 34 and 35.

Fragment 29 (Mars: Identification)**.**

```
R> plot(s.Mars.shaped, type = "vectors", idx = 1:20, fill.color = "green",
+    cuts = 255, layout = c(10, 2), plot.contrib = FALSE)
R> plot(wcor(s.Mars.shaped, groups = 1:30),
+    scales = list(at = c(10, 20, 30)))
```

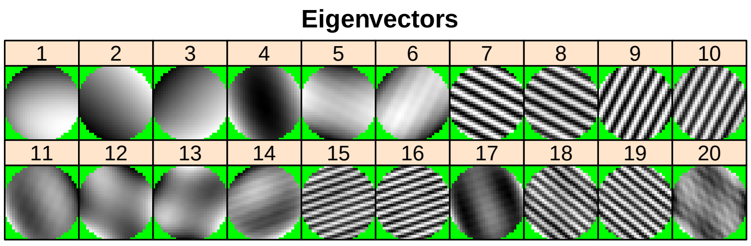


Figure 34: Mars: Eigenarrays, ShSSA.

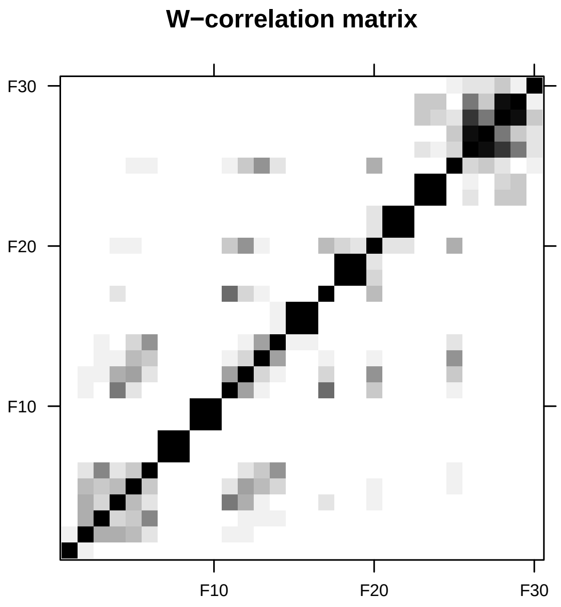


Figure 35: Mars: **w** correlations, ShSSA.

The quality of the texture extraction and therefore of the image recovery by shaped SSA (Figure 33) is considerably better than that performed by 2D-SSA (Figure 20). The improvement of reconstruction accuracy is explained by an edge effect that is caused by a sharp drop of intensity near the boundary of Mars. In Figure 36, we compare magnified reconstructed images for 2D-SSA and ShSSA. In the left subfigure, a green shadow is shown for the background area in order to indicate the Mars boundary. In the right subfigure, light green color corresponds to `NA`. The code that reproduces Figure 36 is shown in Fragment 30.

Fragment 30 (Mars: Magnified reconstructions by 2D-SSA and ShSSA)**.**

```
R> Mars.sh <- r.Mars.shaped$Noise
R> Mars.rect.sh <- Mars.rect <- r.Mars.25$Noise
R> is.na(Mars.rect.sh[is.na(Mars.sh)]) <- TRUE
R> library("latticeExtra")
R> p.part.rect <- plot2d(Mars.rect[60:110, 200:250]) +
+    layer(panel.fill(col = "green", alpha = 0.2), under = FALSE) +
+    plot2d(Mars.rect.sh[60:110, 200:250])
```

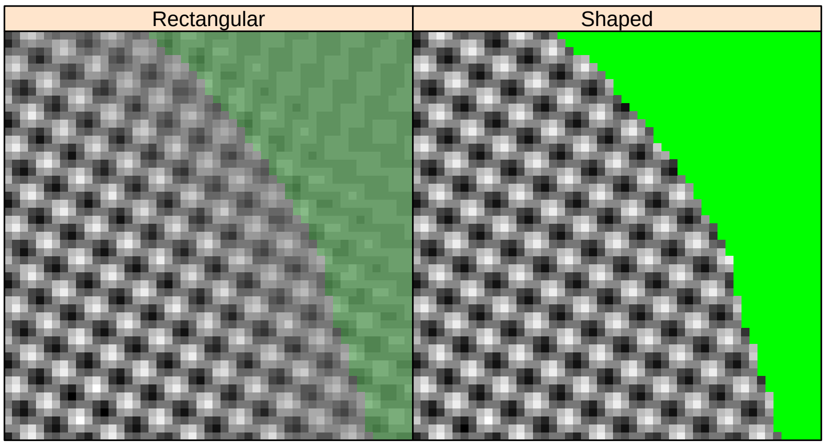


Figure 36: Mars: Comparison of texture reconstructions by 2D-SSA and ShSSA.

```
R> p.part.shaped <- plot2d(r.Mars.shaped[[1]][60:110, 200:250]) +
+    layer(panel.fill(col = "green"), under = TRUE)
R> plot(c(Rectangular = p.part.rect, Shaped = p.part.shaped))
```

Comments

Efficient implementation. In **Rssa**, the shaped 2D-SSA decomposition shares `kind = "2d-ssa"` with the 2D-SSA decomposition. However, the use of input arrays with non-trivial shapes would incur additional projection operation during the computations (see Section 6.2). Ordinary 2D-SSA can be considered as a special case of shaped 2D-SSA with rectangular window; in this case, the mask covers the whole image. The package optimizes for this common case and no projections are performed when it is known that they are trivial and thus we have ordinary 2D-SSA. Therefore, the computational complexity of the method is the same regardless how the full mask was specified: providing `NULL` to `wmask` and `mask` arguments or making the masks trivial. This is a convenient behavior which simplifies, e.g., batch processing of the images with shapes automatically induced by the input images.

Special window shapes. The package provides a convenient interface for setting several special forms of the window. This is implemented via special expressions which can be passed to the `wmask` argument:

- `wmask = circle(R)` specifies a circular mask of radius `R`.
- `wmask = triangle(side)` specifies the mask in the form of isosceles right-angled triangle with cathetus `side`. The right angle lays on the top left corner of the bounding box of the mask.

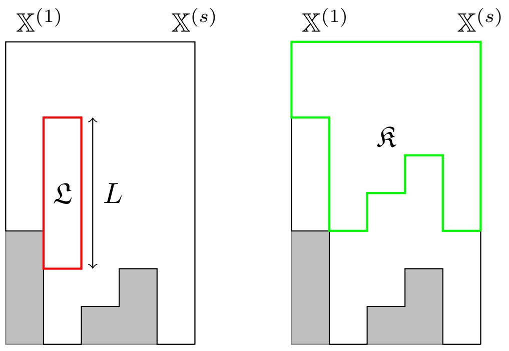


Figure 37: MSSA: 2D packing.

5.3. Special cases of shaped 2D-SSA

In this section, we show that all the considered variants of SSA are in fact special cases of ShSSA, for carefully chosen array and mask.

First of all, SSA can be easily embedded in ShSSA. Consider an $N \times 1$ array and $L \times 1$ window (or, alternatively $1 \times N$ array and $1 \times L$ window). Next, as discussed in the previous section, 2D-SSA is a special case of ShSSA. Finally, in Golyandina and Usevich (2010) it was mentioned that MSSA with equal time series lengths is a special case of 2D-SSA. In what follows, we extend this construction to time series with different lengths.

MSSA: 2D packing

Consider a multivariate time series, that is, a collection $\{\mathbb{X}^{(p)} = (x_j^{(p)})_{j=1}^{N_p},\ \ p = 1, \dots, s\}$ of s time series of length N_p, $p = 1, \dots, s$. We construct the shaped array $\mathbb{X}_{\mathfrak{N}}$ such that

$$\mathfrak{N} = \{1, \dots, N_1\} \times \{1\} \cup \{1, \dots, N_s\} \times \{s\} \subset \{1, \dots, \max_p N_p\} \times \{1, \dots, s\}.$$

The window is taken to be $\mathfrak{L} = \{1, \dots, L\} \times \{1\}$. The construction of this array is shown in Figure 37, also with the window $\mathfrak{L}$ and the shape $\mathfrak{K}$. It is easy to verify that $\mathbf{X} = \mathcal{T}_{\text{ShSSA}}(\mathbb{X})$ coincides with $\mathcal{T}_{\text{MSSA}}(\mathbb{X})$ defined in (7). Therefore, the rows of $\mathbf{X}$ are vectorizations of the $\mathfrak{K}$-shaped subarrays (see Figure 37).

MSSA: 1D packing

Now we consider an alternative packing of MSSA. From the same set of series we construct a $N' \times 1$ (or $1 \times N'$) array $\mathbb{X}$, where $N' = N + (s-1)$ (recall that $N = \sum\limits_{p=1}^{s} N_p$). The array consists of the time series plus "separators" between them that are not included in the array shape. The window is taken to be $L \times 1$ (or $1 \times L$), depending on the arrangement chosen. In Figure 38 we show the horizontal variant of packing.

Mosaic Hankel matrices

A mosaic Hankel matrix (Heinig 1995) is a block matrix with Hankel blocks. It can be considered the most general one-dimensional (i.e., with one-dimensional displacement) generalization of Hankel matrices.

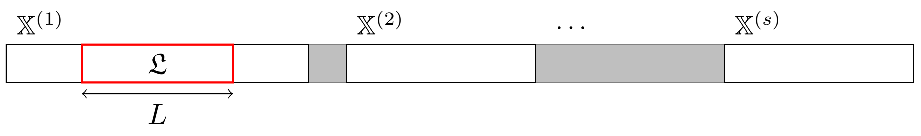


Figure 38: MSSA: 1D packing.

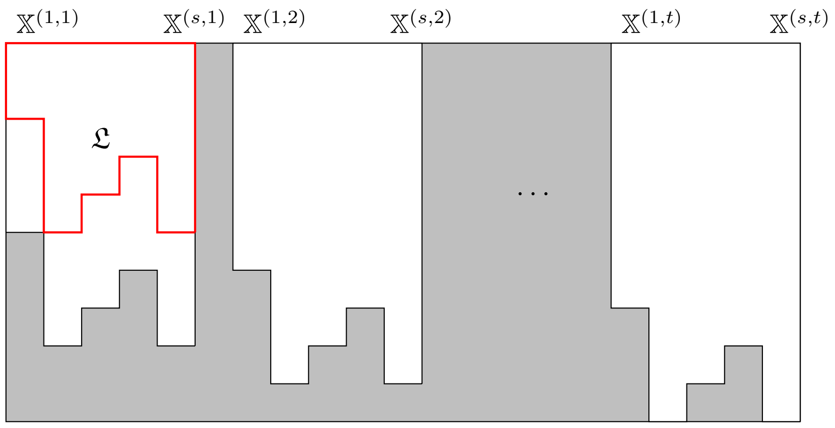


Figure 39: Shaped construction for mosaic Hankel matrices.

Let $L_1, \dots, L_s$ and $K_1, \dots, K_t$ be integer vectors, and $\mathbb{X}^{(i,j)} \in \mathsf{R}^{L_i+K_j-1}$ be time series. Then the mosaic Hankel matrix is constructed as follows:

$$\begin{pmatrix} H_{L_1,K_1}(\mathbb{X}^{(1,1)}) & \cdots & H_{L_1,K_t}(\mathbb{X}^{(1,t)}) \\ \vdots & \cdots & \vdots \\ H_{L_s,K_1}(\mathbb{X}^{(s,1)}) & \cdots & H_{L_s,K_t}(\mathbb{X}^{(p,t)}) \end{pmatrix}.$$

Note that the sizes of the blocks may be different. The only requirement is that they should match as a "mosaic". The case of mosaic Hankel matrices corresponds to several collections of multidimensional time series (Markovsky and Usevich 2014).

It is easy to construct mosaic Hankel matrices, based on 2D embedding of MSSA. A jth block column is a transposed matrix $\mathcal{T}_{\text{MSSA}}$ for the collection of time series $(\mathbb{X}^{(1,j)}, \dots, \mathbb{X}^{(s,j)})$ and window length $L = K_j$. Therefore, the mosaic Hankel matrix can be constructed by stacking shapes (with separators) from Figure 37 and replacing $\mathfrak{L}$ with $\mathfrak{K}$ due to transposition. The resulting construction of the shaped array is shown in Figure 39.

M-2D-SSA

Suppose that we have s arrays $\mathbb{X}^{(1)}, \dots, \mathbb{X}^{(s)} \in \mathsf{R}^{N_x \times N_y}$ and we would like to consider a variant of 2D-SSA where the trajectory matrix is stacked from s trajectory matrices (as in MSSA):

$$\mathcal{T}_{\text{M-2D-SSA}}(\mathbb{X}^{(1)}, \dots, \mathbb{X}^{(s)}) = \left[\mathcal{T}_{\text{2D}-\text{SSA}}(\mathbb{X}^{(1)}) : \dots : \mathcal{T}_{\text{2D}-\text{SSA}}(\mathbb{X}^{(s)})\right]. \tag{22}$$

In this case, the 2D-SSA-like decomposition will have the common basis of eigenvectors, as in MSSA. The trajectory matrices of the form (22) are used in 2D-SSA-based for comparison of images (Rodríguez-Aragón and Zhigljavsky 2010). These matrices are also used in recent

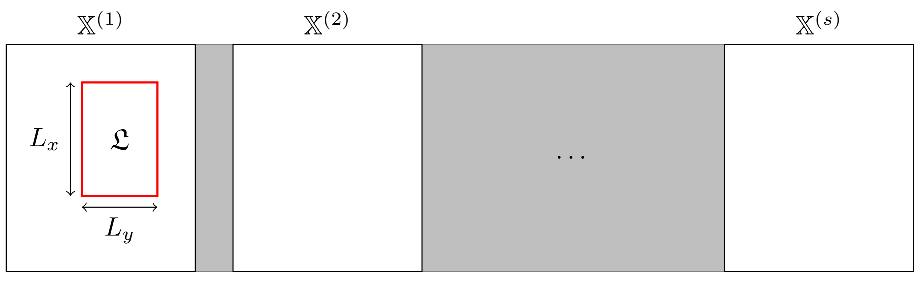


Figure 40: Shaped construction for M-2D-SSA.

methods of parallel magnetic resonance imaging (Uecker, Lai, Murphy, Virtue, Elad, Pauly, Vasanawala, and Lustig 2013).

The packing for the M-2D-SSA can be constructed in a similar way to the case of mosaic Hankel matrices. An array $\mathbb{X}'$ of size $N_x \times (sN_y + s - 1)$ is constructed from the arrays with one-element separators. The resulting array is shown in Figure 40. In general, a similar construction can handle arrays of different sizes, shapes, and shaped windows. Also note that in the extended array the original arrays (both for Figures 39 and 40) may be arranged arbitrarily (for example in a table-like planar arrangement). The only requirement is that they should be separated.

6. Implementation

This section contains details of the core algorithms implemented in the **Rssa** package. The algorithms discussed in this section are either absent or scarcely described in the literature. Although the implementation does not influence the interface of the package, this section is important for understanding why the package works correctly and effectively.

6.1. Fast computations with Hankel matrices

We start with a summary of the algorithms for multiplication of Hankel matrices by vectors and rank-one hankelization. Although algorithms for these operations were already proposed in Korobeynikov (2010), we provide here an alternative description that uses another type of circulants. The alternative description simplifies the algorithms compared to Korobeynikov (2010) and also lays a foundation for Section 6.2 on computations with quasi-Hankel matrices.

Preliminaries

In what follows, $A \odot B$ denotes the elementwise product of two vectors $A, B \in \mathsf{C}^N$, $\mathrm{rev}(A)$ denotes the reversion of the vector A, and $\mathrm{conj}(A)$ denotes the elementwise conjugation. We also denote by $E_j^{(N)}$ the standard jth unit vector in R^N.

For a complex vector $X \in \mathsf{C}^N$ we define its discrete Fourier transform as

$$\mathcal{F}_N(X) = \mathbf{F}_N X, \tag{23}$$

where $\mathbf{F}_N \in \mathsf{C}^{N\times N}$ is the Fourier matrix with elements $(\mathbf{F}_N)_{k,l} = e^{-2\pi \mathrm{i}(k-1)(l-1)/N}$ (see Korobeynikov 2010). The inverse Fourier transform is given by

$$\mathcal{F}_N^{-1}(X) = \frac{1}{N}\mathbf{F}_N^* X,$$

where $\mathbf{F}_N^*$ is the Hermitian transpose of $\mathbf{F}_N$.

The *Toeplitz circulant* constructed from the vector $X = (x_k)_{k=1}^N$ is, by definition,

$$\mathbf{C}_\mathrm{T}(X) = \begin{pmatrix} x_1 & x_N & x_{N-1} & \cdots & x_3 & x_2 \\ x_2 & x_1 & x_N & \cdots & x_4 & x_3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_{N-1} & x_{N-2} & x_{N-3} & \cdots & x_1 & x_N \\ x_N & x_{N-1} & x_{N-2} & \cdots & x_2 & x_1 \end{pmatrix}.$$

The *Hankel circulant* constructed from the vector X is, by definition,

$$\mathbf{C}_\mathrm{H}(X) = \begin{pmatrix} x_1 & x_2 & \cdots & x_{N-2} & x_{N-1} & x_N \\ x_2 & x_3 & \cdots & x_{N-1} & x_N & x_1 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ x_{N-1} & x_N & \cdots & x_{N-4} & x_{N-3} & x_{N-2} \\ x_N & x_1 & \cdots & x_{N-3} & x_{N-2} & x_{N-1} \end{pmatrix}.$$

We will need the following relation between Hankel and Toeplitz circulants.

Lemma 1. *For any $j = 1, \ldots, N$ and $\mathrm{A} \in \mathsf{C}^N$ we have that*

$$A^\top \mathbf{C}_\mathrm{H}(E_j^{(N)}) = (E_j^{(N)})^\top \mathbf{C}_\mathrm{T}(A).$$

Proof. The vector $A^\top \mathbf{C}_\mathrm{H}(E_j^{(N)}) = (\mathbf{C}_\mathrm{H}(E_j^{(N)})A)^\top$ is the jth forward circular shift of the vector $\mathrm{rev}(A)^\top$. Therefore, it coincides with the jth row of $\mathbf{C}_\mathrm{T}(A)$, which is equal to $(E_j^{(N)})^\top \mathbf{C}_\mathrm{T}(A)$. □

Matrix-vector multiplication

It is well known (Korobeynikov 2010) that multiplication of the Toeplitz circulant $\mathbf{C}_\mathrm{T}(X)$ by a vector $A \in \mathsf{C}^N$ can be computed using discrete Fourier transform as

$$\mathbf{C}_\mathrm{T}(X)A = \mathcal{F}_N^{-1}(\mathcal{F}_N(X) \odot \mathcal{F}_N(A)). \tag{24}$$

A similar property holds for the $\mathbf{C}_\mathrm{H}(X)$. Since it is less common, we provide a short proof.

Lemma 2. *For $X \in \mathsf{C}^N$ and $A \in \mathsf{R}^N$ we have that*

$$\mathbf{C}_\mathrm{H}(X)A = \mathcal{F}_N^{-1}\left(\mathcal{F}_N(X) \odot \mathrm{conj}(\mathcal{F}_N(A))\right). \tag{25}$$

Proof. The circulant $C_2 = \mathbf{C}_{\mathrm{T}}((x_N, x_1, \dots, x_{N-1}))$ can be obtained from $\mathbf{C}_{\mathrm{H}}(X)$ by reversion of all rows. Therefore, the product of $\mathbf{C}_{\mathrm{H}}(X)$ by A is equal to

$$\begin{aligned}\mathbf{C}_{\mathrm{H}}(X)A &= C_2 \operatorname{rev}(A) = \mathcal{F}_N\left((x_N, x_1, \dots, x_{N-1})^\top\right) \odot \mathcal{F}_N(\operatorname{rev}(A)) \\ &= (\mathbf{F}_N X) \odot (e^{2\pi \mathrm{i}/N} \mathbf{F}_N \operatorname{rev}(A)) = (\mathbf{F}_N X) \odot \operatorname{conj}(\mathbf{F}_N A).\end{aligned}$$

$\square$

For $\mathbb{X} \in \mathsf{R}^N$ and $K, L : N = K + L - 1$, the Hankel matrix $\mathcal{T}_{\mathrm{SSA}}(\mathbb{X})$ is an $L \times K$ submatrix of $\mathbf{C}_{\mathrm{H}}(\mathbb{X})$. Therefore, multiplication by $\mathcal{T}_{\mathrm{SSA}}(\mathbb{X})$ can be performed using the following algorithm.

Algorithm 1. *Input:* $V \in \mathsf{R}^K$, $\mathbb{X} \in \mathsf{R}^N$. *Output:* $U = \mathcal{T}_{\mathrm{SSA}}(\mathbb{X})V \in \mathsf{R}^L$.

```
1. $V' \leftarrow \begin{pmatrix} V \\ 0_{L-1} \end{pmatrix}$;
2. $\widehat{V'} \leftarrow \mathcal{F}_N(V')$;
3. $\widehat{X} \leftarrow \mathcal{F}_N(\mathbb{X})$;
4. $U' \leftarrow \mathcal{F}_N^{-1}(\widehat{X} \odot \operatorname{conj}(\widehat{V'}))$;
5. $U \leftarrow (u'_1, \dots, u'_L)^\top$.
```

In Korobeynikov (2010), two different algorithms were used for multiplication by $\mathcal{T}_{\mathrm{SSA}}(\mathbb{X})$ and its transpose. But Algorithm 1 also suits for multiplication by $\mathcal{T}^\top_{\mathrm{SSA}}(\mathbb{X})$, because sizes of the matrix are used only in the first step (padding of V by zeros) and the last step (truncating the result). Also, the DFT $\mathbf{F}_N\mathbb{X}$ (step 3) can be precomputed. But, in contrast to the approach of Korobeynikov (2010), the precomputed object does not depend on L.

Rank-one hankelization

Next, we describe the algorithm for hankelization, which coincides with that from Korobeynikov (2010). But we describe the algorithm in a different way, that lays a foundation for quasi-hankelization in Section 6.2.

The hankelization operation computes for a given $\mathbf{X} \in \mathsf{R}^{L \times K}$ the vector $\widetilde{\mathbb{X}} \in \mathsf{R}^N$ that minimizes the distance $\|\mathcal{T}_{\mathrm{SSA}}(\widetilde{\mathbb{X}}) - \mathbf{X}\|^2_{\mathcal{F}}$. It is well-known that the hankelization of $\mathbf{X}$ is the vector $\widetilde{\mathbb{X}} = (\widetilde{x}_j)_{j=1}^N$ of diagonal averages

$$\widetilde{x}_j = \frac{\sum\limits_{k+l=j} (\mathbf{X})_{k,l}}{w_j} = \frac{\langle \mathbf{X}, \mathcal{T}_{\mathrm{SSA}}(E_j^{(N)})\rangle_{\mathcal{F}}}{\|\mathcal{T}_{\mathrm{SSA}}(E_j^{(N)})\|^2_{\mathcal{F}}}. \tag{26}$$

The denominators $w_j = \|\mathcal{T}_{\mathrm{SSA}}(E_j^{(N)})\|^2_{\mathcal{F}}$ are exactly the weights in $\mathbf{w}$ correlations. For rank-one matrices $\mathbf{X} = UV^\top \in \mathsf{R}^{L\times K}$, the numerator in (26) can be expressed more compactly:

$$\langle UV^\top, \mathcal{T}_{\mathrm{SSA}}(E_j^{(N)})\rangle_{\mathcal{F}} = U^\top \mathcal{T}_{\mathrm{SSA}}(E_j^{(N)})V = (U')^\top \mathbf{C}_{\mathrm{H}}(E_j^{(N)})V' = (E_j^{(N)})^\top \mathbf{C}_{\mathrm{T}}(U')V',$$

where $U' = \begin{pmatrix} U \\ 0_{K-1} \end{pmatrix}$, $V' = \begin{pmatrix} V \\ 0_{L-1} \end{pmatrix}$, and the last equality follows from Lemma 1.

Then, the hankelization can be computed using the following algorithm.

Algorithm 2. *Input: $U \in \mathsf{R}^L$, $V \in \mathsf{R}^K$. Output: hankelization $\widetilde{\mathbb{X}} \in \mathsf{R}^N$.*

```
1. $L^* \leftarrow \min(L, K)$;
2. $W \leftarrow (1, 2, \ldots, L^*, \ldots, L^*, \ldots, 2, 1)^\top \in \mathsf{R}^N$ (weights for $\mathbf{w}$ correlations);
3. $U' \leftarrow \begin{pmatrix} U \\ 0_{K-1} \end{pmatrix}$, $V' \leftarrow \begin{pmatrix} V \\ 0_{L-1} \end{pmatrix}$;
4. $\widehat{U'} \leftarrow \mathcal{F}_N(U')$, $\widehat{V'} \leftarrow \mathcal{F}_N(V')$;
5. $\widetilde{\mathbb{X}}' \leftarrow \mathcal{F}_N^{-1}(\widehat{U'} \odot \widehat{V'})$;
6. $\widetilde{x}_k \leftarrow \widetilde{x}'_k / w_k$.
```

Note that in Algorithm 2, we calculate the weights explicitly, as in Korobeynikov (2010). However, writing the weights in the expanded form (the denominator in (26)) helps to understand how the weights are computed for quasi-hankelization in Section 6.2.

6.2. Fast computations with quasi-Hankel matrices

Preliminaries

For two matrices $\mathrm{A}, \mathrm{B} \in \mathsf{C}^{N_x \times N_y}$ we define by $\mathrm{A} \odot \mathrm{B}$ their elementwise product and by $\mathrm{conj}(\mathrm{A})$ the elementwise complex conjugation. For $k = 1, \ldots, N_x$ and $l = 1, \ldots, N_y$, we define elementary arrays as $\mathbf{E}_{k,l} = E_k^{(N_x)} (E_l^{(N_y)})^\top \in \mathsf{R}^{N_x \times N_y}$, where $E_j^{(N)} \in \mathsf{R}^N$ is the jth unit vector. For two matrices A and C their Kronecker product is denoted as $\mathrm{A} \otimes \mathrm{C}$.
For a complex matrix $\mathrm{X} = (x_{k,l})_{k,l=1}^{N_x,N_y}$ its discrete Fourier transform is defined as

$$\mathcal{F}_{N_x,N_y}(\mathrm{X}) = \mathbf{F}_{N_x} \mathrm{X} \mathbf{F}_{N_y}^\top,$$

where $\mathbf{F}_N$ is the matrix of the discrete Fourier transform (23). By the properties of the vectorization operator, we have that

$$\mathrm{vec}(\mathcal{F}_{N_x,N_y}(\mathrm{X})) = (\mathbf{F}_{N_y} \otimes \mathbf{F}_{N_x}) \,\mathrm{vec}(\mathrm{X}).$$

The *TbT (Toeplitz-block-Toeplitz) circulant* constructed from the matrix X is, by definition,

$$\mathbf{C}_{\mathrm{TbT}}(\mathrm{X}) = \begin{pmatrix} \mathbf{C}_{\mathrm{T},1} & \mathbf{C}_{\mathrm{T},N_y} & \mathbf{C}_{\mathrm{T},N_y-1} & \cdots & \mathbf{C}_{\mathrm{T},3} & \mathbf{C}_{\mathrm{T},2} \\ \mathbf{C}_{\mathrm{T},2} & \mathbf{C}_{\mathrm{T},1} & \mathbf{C}_{\mathrm{T},N_y} & \cdots & \mathbf{C}_{\mathrm{T},4} & \mathbf{C}_{\mathrm{T},3} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \mathbf{C}_{\mathrm{T},N_y-1} & \mathbf{C}_{\mathrm{T},N_y-2} & \mathbf{C}_{\mathrm{T},N_y-3} & \cdots & \mathbf{C}_{\mathrm{T},1} & \mathbf{C}_{\mathrm{T},N_y} \\ \mathbf{C}_{\mathrm{T},N_y} & \mathbf{C}_{\mathrm{T},N_y-1} & \mathbf{C}_{\mathrm{T},N_y-2} & \cdots & \mathbf{C}_{\mathrm{T},2} & \mathbf{C}_{\mathrm{T},1} \end{pmatrix},$$

where $\mathbf{C}_{\mathrm{T},j} = \mathbf{C}_{\mathrm{T}}(\mathrm{X}_{:,j})$, and $\mathrm{X}_{:,j}$ denotes the jth column of X. The *HbH circulant* constructed from the matrix X is, by definition,

$$\mathbf{C}_{\mathrm{HbH}}(\mathrm{X}) = \begin{pmatrix} \mathbf{C}_{\mathrm{H},1} & \mathbf{C}_{\mathrm{H},2} & \cdots & \mathbf{C}_{\mathrm{H},N_y-2} & \mathbf{C}_{\mathrm{H},N_y-1} & \mathbf{C}_{\mathrm{H},N_y} \\ \mathbf{C}_{\mathrm{H},2} & \mathbf{C}_{\mathrm{H},3} & \cdots & \mathbf{C}_{\mathrm{H},N_y-1} & \mathbf{C}_{\mathrm{H},N_y} & \mathbf{C}_{\mathrm{H},1} \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ \mathbf{C}_{\mathrm{H},N_y-1} & \mathbf{C}_{\mathrm{H},N_y} & \cdots & \mathbf{C}_{\mathrm{H},N_y-4} & \mathbf{C}_{\mathrm{H},N_y-3} & \mathbf{C}_{\mathrm{H},N_y-2} \\ \mathbf{C}_{\mathrm{H},N_y} & \mathbf{C}_{\mathrm{H},1} & \cdots & \mathbf{C}_{\mathrm{H},N_y-3} & \mathbf{C}_{\mathrm{H},N_y-2} & \mathbf{C}_{\mathrm{H},N_y-1} \end{pmatrix},$$

where $\mathbf{C}_{\mathrm{H},j} = \mathbf{C}_{\mathrm{H}}(\mathrm{X}_{:,j})$. Note that for a rank-one array $\mathrm{X} = X_x X_y^\top$ we have that

$$\mathbf{C}_{\mathrm{HbH}}(\mathrm{X}) = \mathbf{C}_{\mathrm{H}}(X_y) \otimes \mathbf{C}_{\mathrm{H}}(X_x) \quad \text{and} \quad \mathbf{C}_{\mathrm{TbT}}(\mathrm{X}) = \mathbf{C}_{\mathrm{T}}(X_y) \otimes \mathbf{C}_{\mathrm{T}}(X_x). \tag{27}$$

We will also need the following relation between HbH and TbT circulants.

Lemma 3. *For $k = 1, \ldots, N_x$, $l = 1, \ldots, N_y$ and $\mathrm{A} \in \mathsf{R}^{N_x \times N_y}$*

$$\mathrm{vec}(\mathrm{X})^\top \mathbf{C}_{\mathrm{HbH}}(\mathbf{E}_{k,l}) = \mathrm{vec}(\mathbf{E}_{k,l})^\top \mathbf{C}_{\mathrm{TbT}}(\mathrm{X}).$$

Proof. Due to linearity of the equation, we need to prove it only for rank-one matrices $\mathrm{X} = X_x X_y^\top$. Then, by (27) we have that

$$\begin{aligned}\mathrm{vec}(\mathrm{X})^\top \mathbf{C}_{\mathrm{HbH}}(\mathbf{E}_{k,l}) &= (X_y \otimes X_x)^\top \left(\mathbf{C}_{\mathrm{H}}(E_l^{(N_y)}) \otimes \mathbf{C}_{\mathrm{H}}(E_k^{(N_x)})\right) \\ &= \left(X_y^\top \mathbf{C}_{\mathrm{H}}(E_l^{(N_y)})\right) \otimes \left(X_x^\top \mathbf{C}_{\mathrm{H}}(E_k^{(N_x)})\right) = \left((E_l^{(N_y)})^\top \mathbf{C}_{\mathrm{T}}(X_y)\right) \otimes \left((E_k^{(N_x)})^\top \mathbf{C}_{\mathrm{T}}(X_x)\right) \\ &= \left(E_l^{(N_y)} \otimes E_k^{(N_x)}\right)^\top \mathbf{C}_{\mathrm{T}}(X_y) \otimes \mathbf{C}_{\mathrm{T}}(X_x) = \mathrm{vec}(\mathbf{E}_{k,l})^\top \mathbf{C}_{\mathrm{TbT}}(\mathrm{X}).\end{aligned}$$

□

Multiplication by a quasi-Hankel matrix

Analogously to (24) and (25), the following equalities hold true.

Lemma 4. *For $\mathrm{X}, \mathrm{A} \in \mathsf{R}^{N_x \times N_y}$ we have that*

$$\begin{aligned}\mathbf{C}_{\mathrm{TbT}}(\mathrm{X})\,\mathrm{vec}\,(\mathrm{A}) &= \mathrm{vec}\left(\mathcal{F}^{-1}_{N_x,N_y}\left(\mathcal{F}_{N_x,N_y}(\mathrm{X}) \odot \mathcal{F}_{N_x,N_y}(\mathrm{A})\right)\right), \\ \mathbf{C}_{\mathrm{HbH}}(\mathrm{X})\,\mathrm{vec}\,(\mathrm{A}) &= \mathrm{vec}\left(\mathcal{F}^{-1}_{N_x,N_y}\left(\mathcal{F}_{N_x,N_y}(\mathrm{X}) \odot \mathrm{conj}(\mathcal{F}_{N_x,N_y}(\mathrm{A}))\right)\right).\end{aligned}$$

Proof. Due to linearity of the Fourier transform and circulant matrices we can prove the statements only for matrices $\mathrm{X} = X_x X_y^\top$ and $\mathrm{A} = A_x A_y^\top$. In this case, by (27) we have that

$$\begin{aligned}&\mathrm{vec}\left(\mathcal{F}^{-1}_{N_x,N_y}\left(\mathcal{F}_{N_x,N_y}(\mathrm{X}) \odot \mathcal{F}_{N_x,N_y}(\mathrm{A})\right)\right) \\ &\qquad = \mathrm{vec}\left(\mathcal{F}^{-1}_{N_x,N_y}\left((\mathbf{F}_{N_x}X_x)(\mathbf{F}_{N_y}X_y)^\top \odot (\mathbf{F}_{N_x}A_x)(\mathbf{F}_{N_y}A_y)^\top\right)\right) \\ &\qquad = \mathrm{vec}\left(\mathbf{F}^{-1}_{N_x}\left((\mathbf{F}_{N_x}X_x) \odot (\mathbf{F}_{N_x}A_x)\right)\left((\mathbf{F}_{N_y}X_y) \odot (\mathbf{F}_{N_y}A_y)\right)^\top (\mathbf{F}^{-1}_{N_y})^\top\right) \\ &\qquad = \mathrm{vec}((\mathbf{C}_{\mathrm{T}}(X_x)A_x)(\mathbf{C}_{\mathrm{T}}(X_y)A_y)^\top) = \mathbf{C}_{\mathrm{TbT}}(\mathrm{X})\,\mathrm{vec}\,\mathrm{A}.\end{aligned}$$

The proof of the second statement is analogous. □

Now we consider multiplication by a quasi-Hankel matrix. Let $\mathfrak{L}, \mathfrak{K}, \mathfrak{N} \subseteq \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$ be such that $\mathfrak{L} \mathbin{+\!\!\!-} \mathfrak{K} = \mathfrak{N}$. Then the quasi-Hankel matrix $\mathcal{T}_{\mathrm{ShSSA}}(\mathbb{X})$ (defined in (21)) is a submatrix of the HbH circulant $\mathbf{C}_{\mathrm{HbH}}(\mathbb{X})$. Now let us write this formally and provide an algorithm for the calculation of the matrix-vector product.

For a set of indices $\mathfrak{A} = \{(k_1, l_1), \ldots, (k_N, l_N)\}$, which is ordered lexicographically, we define the projection matrix

$$P_{\mathfrak{A}} = [\mathrm{vec}(\mathbf{E}_{k_1,l_1}) : \ldots : \mathrm{vec}(\mathbf{E}_{k_N,l_N})].$$

Then we have that

$$\mathcal{T}_{\mathrm{ShSSA}}(\mathbb{X}) = P_{\mathfrak{L}}^{\top} \mathbf{C}_{\mathrm{HbH}}(\mathbb{X}) P_{\mathfrak{K}}.$$

Hence, the matrix-vector multiplication algorithm can be written as follows.

Algorithm 3. *Input:* $V \in \mathsf{R}^K$, $\mathbb{X} \in \mathsf{R}^{N_x \times N_y}$. *Output:* $U = \mathcal{T}_{\mathrm{ShSSA}}(\mathbb{X}) V \in \mathsf{R}^L$.

```
1. $\widehat{\mathrm{X}} \leftarrow \mathcal{F}_{N_x,N_y}(\mathbb{X})$;
2. $\mathrm{U}' \leftarrow \mathrm{vec}_{K_x}^{-1}(P_{\mathfrak{K}} U)$;
3. $\widehat{\mathrm{U}'} \leftarrow \mathcal{F}_{N_x,N_y}(\mathrm{U}')$;
4. $\mathrm{V}' \leftarrow \mathcal{F}_{N_x,N_y}^{-1}(\widehat{\mathrm{X}} \odot \mathrm{conj}(\widehat{\mathrm{U}'}))$;
5. $V \leftarrow P_{\mathfrak{L}}^{\top} \mathrm{vec}(\mathrm{V}')$.
```

Rank-one quasi-hankelization

Let $\mathfrak{L}, \mathfrak{K}, \mathfrak{N} \subseteq \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$ be such that $\mathfrak{L} \mathbin{+\!\!\!-} \mathfrak{K} = \mathfrak{N}$. The *quasi-hankelization* operator, by definition, computes for a given $\mathbf{X} \in \mathsf{R}^{L \times K}$ the shaped array $\widetilde{\mathbb{X}}_{\mathfrak{N}} = (\widetilde{x}_{k,l})_{(k,l) \in \mathfrak{N}}$ that minimizes the distance $\|\mathcal{T}(\widetilde{\mathbb{X}}_{\mathfrak{N}}) - \mathbf{X}\|_{\mathcal{F}}^2$.

As shown in Section 2.1, quasi-hankelization can be expressed as averaging. More precisely,

$$\widetilde{x}_{k,l} = \frac{\langle \mathbf{X}, \mathcal{T}(\mathbf{E}_{k,l}) \rangle_{\mathcal{F}}}{\|\mathcal{T}(\mathbf{E}_{k,l})\|_{\mathcal{F}}^2}, \quad \text{for } (k,l) \in \mathfrak{N}.$$

The numerator represents summation over the set of positions in $\mathbf{X}$ that correspond to the (k,l)th element of the array. The denominator is equal to the number of such elements.

In this section, we assume that $\mathbf{X}$ is a rank-one matrix, i.e., $\mathbf{X} = UV^{\top}$. Then

$$\widetilde{x}_{k,l} = \frac{\langle UV^{\top}, \mathcal{T}(\mathbf{E}_{k,l}) \rangle_{\mathcal{F}}}{\|\mathcal{T}(\mathbf{E}_{k,l})\|_{\mathcal{F}}^2} = \frac{\langle UV^{\top}, \mathcal{T}(\mathbf{E}_{k,l}) \rangle_{\mathcal{F}}}{\langle 1_L 1_K^{\top}, \mathcal{T}(\mathbf{E}_{k,l}) \rangle_{\mathcal{F}}},$$

where $1_Q = (1, \ldots, 1)^{\top} \in \mathsf{R}^Q$. If we define by

$$\mathrm{diagsums}(U, V) = (x_{k,l})_{k,l=1}^{N_x, N_y}, \quad \text{where } x_{k,l} = \langle UV^{\top}, \mathcal{T}(\mathbf{E}_{k,l}) \rangle_{\mathcal{F}},$$

then quasi-hankelization is given by the following algorithm.

Algorithm 4.
Input: $U \in \mathsf{R}^L$, $V \in \mathsf{R}^K$, *shapes* $\mathfrak{L}$, $\mathfrak{K}$. *Output: shape* $\mathfrak{N}' = \mathfrak{L} \mathbin{+\!\!\!-} \mathfrak{K}$, *quasi-hankelization* $\widetilde{\mathbb{X}}_{\mathfrak{N}}$.

```
1. $\mathbb{W} \leftarrow \mathrm{diagsums}(1_L, 1_K)$ (array of weights for $\mathbf{w}$ correlations);
2. $\mathfrak{N}' \leftarrow \{(k,l) \mid w_{k,l} \neq 0\}$;
3. $\widetilde{\mathbb{X}'} \leftarrow \mathrm{diagsums}(U, V)$;
4. Compute $\widetilde{x}_{k,l} = \widetilde{x}'_{k,l} / w_{k,l}$.
```

We note that the weights $\mathbb{W}$ can be precomputed, as well as the shape $\mathfrak{N}'$.

The only missing part in Algorithm 4 is computation of diagsums(U, V) for given $U \in \mathsf{R}^L$ and $V \in \mathsf{R}^K$. For this we note that

$$\begin{aligned}\langle UV^\top, \mathcal{T}_{\mathrm{ShSSA}}(\mathbf{E}_{k,l})\rangle_{\mathcal{F}} &= \operatorname{trace}(VU^\top P_{\mathfrak{L}}^\top \mathcal{T}_{\mathrm{ShSSA}}(\mathbf{E}_{k,l}) P_{\mathfrak{K}}) \\ &= \operatorname{vec}(\mathrm{U}')^\top \mathbf{C}_{\mathrm{HbH}}(\mathbf{E}_{k,l}) \operatorname{vec}(\mathrm{V}') = \operatorname{vec}(\mathbf{E}_{k,l})^\top \mathbf{C}_{\mathrm{TbT}}(\mathrm{U}') \operatorname{vec}(\mathrm{V}'),\end{aligned}$$

where $\mathrm{U}' = \operatorname{vec}^{-1}_{L_x}(P_{\mathfrak{L}} U)$, $\mathrm{V}' = \operatorname{vec}^{-1}_{K_x}(P_{\mathfrak{K}} V)$, and the last equality holds by Lemma 3. Thus, diagsums(U, V) can be computed by Algorithm 5.

Algorithm 5.

Input: $U \in \mathsf{R}^L$, $V \in \mathsf{R}^K$. *Output:* $\mathbb{X} = \operatorname{diagsums}(U, V) \in \mathsf{R}^{N_x \times N_y}$.

```
1. $\mathrm{U}' \leftarrow \operatorname{vec}^{-1}_{L_x}(P_{\mathfrak{L}} U)$;
2. $\mathrm{V}' \leftarrow \operatorname{vec}^{-1}_{K_x}(P_{\mathfrak{K}} V)$;
3. $\widehat{\mathrm{U}'} \leftarrow \mathcal{F}_{N_x,N_y}(\mathrm{U}')$;
4. $\widehat{\mathrm{V}'} \leftarrow \mathcal{F}_{\mathrm{N_x,N_y}}(\mathrm{V}')$;
5. $\mathbb{X} \leftarrow \mathcal{F}^{-1}_{N_x,N_y}(\widehat{\mathrm{U}'} \odot \widehat{\mathrm{V}'})$.
```

Calculation of shapes

Assume that we have $\mathfrak{L} \subseteq \{1, \ldots, L_x\} \times \{1, \ldots, L_y\}$ and $\mathfrak{N} \subseteq \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$. We would like to find the maximal $\mathfrak{K}$ such that $\mathfrak{L} \mathbin{+\!\!\!-} \mathfrak{K} \subseteq \mathfrak{N}$.

Let $I^{(\mathfrak{N})} \in \mathcal{R}^{N_x \times N_y}$ be the indicator array of $\mathfrak{N}$, i.e.,

$$i^{(\mathfrak{N})}_{k,l} = \begin{cases} 1, & (k,l) \in \mathfrak{N}, \\ 0, & (k,l) \notin \mathfrak{N}. \end{cases}$$

Then the shape $\mathfrak{K}$ is equal to the maximal $\mathfrak{A}$ such that all the elements of the $\mathfrak{L} \times \mathfrak{A}$ submatrix of $\mathcal{T}_{\mathrm{2DSSA}}(I^{(\mathfrak{N})})$ are equal to 1. Algorithm 6 finds such a maximal shape from the product of $\mathcal{T}^\top_{\mathrm{2DSSA}}(I^{(\mathfrak{N})})$ and 1_L. The elements of $\mathfrak{K}$ correspond to the elements of the resulting vector that have the value L.

Algorithm 6.

Input: $\mathfrak{L} \subseteq \{1, \ldots, L_x\} \times \{1, \ldots, L_y\}$, $\mathfrak{N} \subseteq \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$. *Output: maximal* $\mathfrak{K}$ *such that* $\mathfrak{L} \mathbin{+\!\!\!-} \mathfrak{K} \subseteq \mathfrak{N}$.

```
1. $V \leftarrow \mathcal{T}^\top_{\mathrm{2DSSA}}(I^{(\mathfrak{N})}) 1_L$;
2. $\mathrm{V}' \leftarrow \operatorname{vec}^{-1}_{K_x}(V)$;
3. $\mathfrak{K} \leftarrow \{(k,l) \mid v'_{k,l} = L\}$.
```

6.3. Fast vector forecasting algorithm

It is mentioned in Golyandina and Zhigljavsky (2013, p. 76) that the vector forecasting is time-consuming, while recurrent forecasting is fast. However, this is only so if one implements the algorithm from Section 3.3 directly. It appears that it is possible to considerably accelerate the vector forecasting. Moreover, in the current implementation in **Rssa** the vector forecasting is slightly faster than the recurrent one.

In this section, we will use the notation from Section 3.3. We also denote by † the pseudo-inversion of a matrix.

Column MSSA forecast

As shown in Section 3.3, column vector MSSA forecasting is reduced to performing s 1D vector forecasts in the same subspace $\mathcal{L}^{\mathrm{col}}$. Next, we describe the algorithm for fast vector forecasting of 1D time series in a given subspace $\mathcal{L}^{\mathrm{col}}$ (vector forecast in basic SSA).

Consider the forecasting in the subspace $\mathcal{L}^{\mathrm{col}}$ given by a basis $\{P_1, \dots, P_r\}$. Denote $\mathbf{P} = [P_1 : \dots : P_r]$. Each reconstructed vector $\widehat{X}_k$ of the 1D time series belongs to $\mathcal{L}^{\mathrm{col}}$; hence, there exist coefficients $W_k \in \mathsf{R}^r$ such that $\widehat{X}_k = \mathbf{P}W_k$. Denote $\mathbf{W} = [W_1 : \dots : W_K]$. In fact, the input for the algorithm is the minimal decomposition of $\widehat{\mathbf{X}}$ into the sum of elementary matrices of rank 1 in the form $\widehat{\mathbf{X}} = \mathbf{P}\mathbf{Q}^\top$ and $\mathbf{W} = \mathbf{Q}^\top$.

Note that if we have a singular value decomposition of $\widehat{\mathbf{X}} = [\widehat{X}_1 : \dots : \widehat{X}_K]$, then the left singular vectors provide the basis of the subspace, while the W_k are determined by the right singular vectors and singular values: $\mathbf{P} = [U_1 : \dots : U_r]$ and $\mathbf{Q} = [\sqrt{\lambda_1}V_1 : \dots : \sqrt{\lambda_r}V_r]$.

In vector forecasting, we extend the reconstructed matrix as $\mathbf{Z} = [\widehat{\mathbf{X}} : Z_{K+1} : \dots : Z_{K+M+L-1}]$, where the vectors Z_k, $k \ge K+1$, are obtained as $Z_k = \mathcal{P}^{\mathrm{col}}_{\mathrm{Vec}} Z_{k-1}$ (and $Z_k = \widehat{X}_k$ for $k = 1, \dots, K$). Since all Z_k belong to $\mathcal{L}^{\mathrm{col}}$, there exist $W_k \in \mathsf{R}^r$, $k \ge K+1$ such that

$$\mathbf{Z} = \mathbf{P}[W_1 : \dots : W_{K+M+L-1}].$$

Thus, there exists a matrix $\mathbf{D}$ such that $W_k = \mathbf{D}W_{k-1}$ for $k \ge K+1$. This observation leads to the following algorithm for vector forecasting.

Algorithm 7. *Input:* $\mathbf{P}$, $\mathbf{W}$. *Output: the forecasted values* $z_{N+1}, \dots, z_{N+M}$.

1. *Compute the matrix* $\mathbf{D} = \underline{\mathbf{P}}^\dagger \overline{\mathbf{P}}$ *using the QR-decomposition (Golub and Loan 1996).*
2. *For* $k = K+1, \dots, K+M+L-1$ *compute* $W_k = \mathbf{D}W_{k-1}$.
3. *Perform the fast rank-one hankelization algorithm (Section 6.1) for the matrix* $\mathbf{Z} = \mathbf{P}[\mathbf{W} : W_{K+1} : \dots : W_{K+M-L-1}]$, *which is explicitly expressed as a sum of rank-one matrices, and obtain the series* $z_1, \dots, z_{N+M+L-1}$.
4. *The numbers* $z_1, \dots, z_{N+M}$ *form updated reconstructed and forecasted series.*

In order to prove the correctness of the algorithm, it remains to show that the formula $\mathbf{D} = \underline{\mathbf{P}}^\dagger \overline{\mathbf{P}}$ is correct, which will be discussed at the end of this section. Finally, there is also a further improvement on the computation of $\mathbf{D}$ possible.

Remark 2. *The following two things can be noted:*

1. *Item 1 in Algorithm 7 is exactly the shift matrix from the LS-ESPRIT method for frequency estimation (Roy and Kailath 1989).*
2. *If $\{P_i\}$ is an orthonormal system, then $\mathbf{D}$ can be computed without the QR-decomposition as:*
$$\mathbf{D} = \left(\mathbf{I}_r - \frac{1}{1-\boldsymbol{\pi}^\top\boldsymbol{\pi}}\boldsymbol{\pi}\boldsymbol{\pi}^\top\right)\underline{\mathbf{P}}^\top\overline{\mathbf{P}},$$
where $\boldsymbol{\pi} = \boldsymbol{\pi}(\mathbf{P})$, $\mathbf{I}_r$ is the $r \times r$ identity matrix.

Row MSSA forecast

Row vector forecast is slightly different from the column one, but the idea is the same. We deal with forecasting in the row subspace $\mathcal{L}^{\mathrm{row}} = \mathrm{span}(Q_1, \ldots, Q_r)$ and continue the sequence of the row vectors $\widehat{Y}_k$ of $\widehat{\mathbf{X}}$. If the row vectors are equal to $\widehat{Y}_k = \mathbf{Q}W_k$, $k = 1, \ldots, L$, then $\widehat{\mathbf{X}}^\top = \mathbf{Q}\mathbf{W}$, where $\mathbf{W} = [W_1 : \ldots : W_L]$. In the following algorithm, as in column forecasting, the vectors W_k are continued instead of the vectors Y_k.

Algorithm 8. *Input: $\mathbf{Q}$, $\mathbf{W}$. Output: the forecasted values $z^{(p)}_{N_p+1}, \ldots, z^{(p)}_{N_p+M}$, $p = 1, \ldots, s$.*

1. *Compute the matrix $\mathbf{D} = \underline{\underline{\mathbf{Q}}}^\dagger\overline{\overline{\mathbf{Q}}}$ using the QR-decomposition.*
2. *For $k = L+1, \ldots, L+M+\max_{p=1,\ldots,s} K_p - 1$ compute $W_k = \mathbf{D}W_{k-1}$.*
3. *Perform the fast rank-one hankelization algorithm (Section 6.1) for each of the s matrices $\mathbf{Z}^{(p)} = \mathbf{Q}^{(p)}[\mathbf{W} : W_{L+1} : \ldots : W_{L+M+K_p-1}]$ for $p = 1, \ldots, s$ and obtain s series $z^{(p)}_1, \ldots z^{(p)}_{N_p+M+L-1}$.*
4. *The numbers $z^{(p)}_1, \ldots, z^{(p)}_{N_p+M}$ form updated reconstructed and forecasted series.*

Remark 3. *If $\{Q_i\}$ is an orthonormal system, then $\mathbf{D}$ from Algorithm 8 can be expressed as*
$$\mathbf{D} = \left(\mathbf{I}_r - \mathbf{S}^\top(\mathbf{I}_s - \mathbf{S}\mathbf{S}^\top)^{-1}\mathbf{S}\right)\underline{\underline{\mathbf{Q}}}^\top\overline{\overline{\mathbf{Q}}},$$
where $\mathbf{S} = [\boldsymbol{\mu}(Q_1) : \ldots : \boldsymbol{\mu}(Q_r)]$, $\mathbf{I}_r$ and $\mathbf{I}_s$ are the $r \times r$ and $s \times s$ identity matrices.

Proof of the algorithms' correctness

For simplicity, we will consider the one-dimensional case, that is, SSA vector forecast, which coincides with the column MSSA forecast for one-dimensional series. The proof for row forecasting is analogous.

We need to prove that $Z_{k+1} = \mathbf{P}W_{k+1}$, where $W_{k+1} = \mathbf{D}W_k$, $\mathbf{D} = \underline{\mathbf{P}}^\dagger\overline{\mathbf{P}}$. It is sufficient to prove $\underline{Z}_{k+1} = \underline{\mathbf{P}}W_{k+1}$, since the last coordinate of the vector is uniquely defined. In the standard formulation of vector forecasting algorithm in Golyandina *et al.* (2001, Section 2.3) and in Section 3.3, $\underline{Z}_{k+1}$ is the projection on the column space of $\underline{\mathbf{P}}$, that is, $\underline{Z}_{k+1} = \underline{\mathbf{P}}\underline{\mathbf{P}}^\dagger\overline{Z}_k$. Since $\overline{\mathbf{P}}W_k$ is exactly $\overline{Z}_k$ by definition of W_k, we have that $\underline{Z}_{k+1} = \underline{\mathbf{P}}(\underline{\mathbf{P}}^\dagger\overline{\mathbf{P}})W_k = \underline{\mathbf{P}}W_{k+1}$, and the equivalence of the standard and fast vector forecasting algorithms is proved.

Remark 4. *The speed-up of the algorithms' implementation is explained by two reasons:*

1. *multiplication by matrices of small size $r \times r$ at each step instead of multiplication by matrices of much larger size, and,*

2. *the form of the matrix to be hankelized is suitable for application of the fast rank-one hankelization algorithm.*

7. Conclusion

The paper contains an extended guide to the use of the **Rssa** package for analysis of multivariate and multidimensional objects by singular spectrum analysis (SSA). The following extensions of SSA are considered: MSSA for multidimensional time series, 2D-SSA for two-dimensional arrays (images), and a new method shaped 2D-SSA (ShSSA) for arrays of arbitrary shape.

Numerous examples for each SSA extension are included in the paper. The examples cover typical tasks that occur in SSA analysis and show how these tasks can be performed with the **Rssa** package. The examples also demonstrate the plotting capabilities of **Rssa**. Together with practical and implementation issues, the paper contains a summary of theoretical and methodological aspects of SSA that assists in the proper use of the package.

In the paper, we stress on a common form of theory, algorithms, package interface and implementation. The algorithms for SSA extensions are presented as particular cases of the general SSA scheme, which is specialized in each case with the help of an appropriate embedding operator. The examples with typical code demonstrate the common structure of the algorithms, which is reflected in the common structure of **Rssa** interface for all SSA extensions. In the implementation of **Rssa**, a unified approach is proposed, based on the shaped 2D-SSA algorithm.

We hope that our general approach to the variety of SSA versions will help users to apply **Rssa** properly and effectively.

Acknowledgments

The authors would like to thank the anonymous referees for their patience and thoughtful suggestions and comments that led to the significantly improved presentation of the paper.

The paper was partly supported by the project NG13-083 of the Dynasty Foundation in the area of computer science. Konstantin Usevich was supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC Grant agreements no. 258581 and 320594.

References

Ade F (1983). "Characterization of Textures by 'Eigenfilters'." *Signal Processing*, **5**(5), 451–457. doi:10.1016/0165-1684(83)90008-7.

Alexandrov T (2009). "A Method of Trend Extraction Using Singular Spectrum Analysis." *REVSTAT*, **7**(1), 1–22.

Bivand R (2015). "CRAN Task View: Analysis of Spatial Data." Version 2015-06-09, URL http://CRAN.R-project.org/view=Spatial.

Broomhead DS, King GP (1986b). "On the Qualitative Analysis of Experimental Dynamical Systems." In S Sarkar (ed.), *Nonlinear Phenomena and Chaos*, pp. 113–144. Adam Hilger, Bristol.

Buil C (2010a). ***IRIS**: An Astronomical Images Processing Software.* Version 5.59, URL http://www.astrosurf.com/buil/us/iris/iris.htm.

Buil C (2010b). ***IRIS** Tutorial: Cosmetic Corrections.* URL http://www.astrosurf.com/buil/iris/tutorial8/doc23_us.htm.

Danilov D, Zhigljavsky A (eds.) (1997). *Principal Components of Time Series: The 'Caterpillar' Method*. St. Petersburg University Press. (in Russian).

Frigo M, Johnson SG (2005). "The Design and Implementation of FFTW3." *Proceedings of the IEEE*, **93**(2), 216–231. doi:10.1109/jproc.2004.840301.

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." *Reviews of Geophysics*, **40**(1), 1–41. doi:10.1029/2000rg000092.

Gistat Group (2013). "**CaterpillarSSA**." (1996–2013), URL http://gistatgroup.com/.

Golub G, Loan C (1996). *Matrix Computations*. 3rd edition. Johns Hopkins University Press, Baltimore, MD, USA.

Golyandina N (2010). "On the Choice of Parameters in Singular Spectrum Analysis and Related Subspace-Based Methods." *Statistics and Its Interface*, **3**(3), 259–279. doi:10.4310/sii.2010.v3.n3.a2.

Golyandina N, Florinsky I, Usevich K (2007). "Filtering of Digital Terrain Models by 2D Singular Spectrum Analysis." *International Journal of Ecology & Development*, **8**(F07), 81–94.

Golyandina N, Holloway D, Lopes F, Spirov A, Spirova E, Usevich K (2012). "Measuring Gene Expression Noise in Early Drosophila Embryos: Nucleus-to-Nucleus Variability." In *Procedia Computer Science*, volume 9, pp. 373–382.

Golyandina N, Korobeynikov A (2014). "Basic Singular Spectrum Analysis and Forecasting with R." *Computational Statistics & Data Analysis*, **71**, 934–954. doi:10.1016/j.csda.2013.04.009.

Golyandina N, Nekrutkin V, Stepanov D (2003). "Variants of the 'Caterpillar'-SSA Method for Analysis of Multidimensional Time Series." In *Proceedings of the II International Conference on System Identification and Control Problems (SICPRO 03)*, pp. 2139–2168. V.A. Trapeznikov Institute of Control Sciences, Moscow. (In Russian), URL http://gistatgroup.com/cat/mssa_analysis_ru.pdf.

Golyandina N, Nekrutkin V, Zhigljavsky A (2001). *Analysis of Time Series Structure: SSA and Related Techniques*. Chapman & Hall/CRC. doi:10.1201/9781420035841.

Golyandina N, Osipov E (2007). "The "Caterpillar"-SSA Method for Analysis of Time Series with Missing Values." *Journal of Statistical Planning and Inference*, **137**(8), 2642–2653. doi:10.1016/j.jspi.2006.05.014.

Golyandina N, Shlemov A (2015). "Variations of Singular Spectrum Analysis for Separability Improvement: Non-Orthogonal Decompositions of Time Series." *Statistics and Its Interface*, **8**(3), 277–294. doi:10.4310/sii.2015.v8.n3.a3.

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. URL http://www.gistatgroup.com/gus/mssa2.pdf.

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 V Olshevsky, E Tyrtyshnikov (eds.), *Matrix Methods: Theory, Algorithms and Applications*, pp. 449–473. World Scientific Publishing.

Golyandina N, Zhigljavsky A (2013). *Singular Spectrum Analysis for Time Series*. Springer Briefs in Statistics. Springer-Verlag. doi:10.1007/978-3-642-34913-3.

Hannachi A, Jolliffe I, Stephenson D (2007). "Empirical Orthogonal Functions and Related Techniques in Atmospheric Science: A Review." *International Journal of Climatology*, **27**(9), 1119–1152. doi:10.1002/joc.1499.

Hassani H, Heravi S, Zhigljavsky A (2013). "Forecasting UK Industrial Production with Multivariate Singular Spectrum Analysis." *Journal of Forecasting*, **32**(5), 395–408. doi:10.1002/for.2244.

Heinig G (1995). "Generalized Inverses of Hankel and Toeplitz Mosaic Matrices." *Linear Algebra and Its Applications*, **216**, 43–59. doi:10.1016/0024-3795(93)00097-j.

Hijmans R (2015). ***raster**: Geographic Data Analysis and Modeling*. R package version 2.4-15, URL http://CRAN.R-project.org/package=raster.

Holloway D, Lopes F, Costa L, Travençolo B, Golyandina N, Usevich K, Spirov A (2011). "Gene Expression Noise in Spatial Patterning: *hunchback* Promoter Structure Affects Noise Amplitude and Distribution in *Drosophila* Segmentation." *PLoS Computational Biology*, **7**(2), e1001069. doi:10.1371/journal.pcbi.1001069.

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

Hyndman R (2015). "CRAN Task View: Time Series Analysis." Version 2015-07-18, URL http://CRAN.R-project.org/view=TimeSeries.

Korobeynikov A (2010). "Computation- And Space-Efficient Implementation of SSA." *Statistics and Its Interface*, **3**(3), 357–368. doi:10.4310/sii.2010.v3.n3.a9.

Korobeynikov A (2014). ***svd**: Interfaces to Various State-of-Art SVDs and Eigensolvers.* R package version 0.3.3-2, URL http://CRAN.R-project.org/package=svd.

Korobeynikov A, Shlemov A, Usevich K, Golyandina N (2015). ***Rssa**: A Collection of Methods for Singular Spectrum Analysis.* R package version 0.13-1, URL http://CRAN.R-project.org/package=Rssa.

Larsen R (1998). *Efficient Algorithms for Helioseismic Inversion.* Ph.D. thesis, University of Aarhus, Denmark.

Markovsky I, Usevich K (2014). "Software for Weighted Structured Low-Rank Approximation." *Journal of Computational and Applied Mathematics*, **256**, 278–292. doi:10.1016/j.cam.2013.07.048.

Monadjemi A (2004). *Towards Efficient Texture Classification and Abnormality Detection.* Ph.D. thesis, University of Bristol.

Mourrain B, Pan V (2000). "Multivariate Polynomials, Duality, and Structured Matrices." *Journal of Complexity*, **16**(1), 110–180. doi:10.1006/jcom.1999.0530.

R Core Team (2015). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Rodríguez-Aragón L, Zhigljavsky A (2010). "Singular Spectrum Analysis for Image Processing." *Statistics and Its Interface*, **3**(3), 419–426. doi:10.4310/sii.2010.v3.n3.a14.

Rouquette S, Najim M (2001). "Estimation of Frequencies and Damping Factors by Two-Dimensional ESPRIT Type Methods." *IEEE Transactions on Signal Processing*, **49**(1), 237–245. doi:10.1109/78.890367.

Roy R, Kailath T (1989). "ESPRIT: Estimation of Signal Parameters via Rotational Invariance Techniques." *IEEE Transactions on Acoustics, Speech and Signal Processing*, **37**(7), 984–995. doi:10.1109/29.32276.

Sarkar D (2008). ***lattice**: Multivariate Data Visualization with R.* Springer-Verlag, New York. URL http://lmdvr.R-Forge.R-project.org/.

Stepanov D, Golyandina N (2005). "Variants of the 'Caterpillar'-SSA Method for Forecast of Multidimensional Time Series." In *Proceedings of IV International Conference on System Identification and Control Problems (SICPRO 05)*, pp. 1831–1848. Moscow: V.A.Trapeznikov institute of Control Sciences. (In Russian), URL http://gistatgroup.com/cat/mssa_forecast_ru.pdf.

Trickett S (2008). "F-Xy Cadzow Noise Suppression." In *78th Annual International Meeting, SEG, Expanded Abstracts*, pp. 2586–2590.

Uecker M, Lai P, Murphy M, Virtue P, Elad M, Pauly J, Vasanawala S, Lustig M (2013). "ESPIRiT — An Eigenvalue Approach to Autocalibrating Parallel MRI: Where SENSE Meets GRAPPA." *Magnetic Resonance in Medicine.* doi:10.1002/mrm.24751.

Vautard M, Ghil M (1989). "Singular Spectrum Analysis in Nonlinear Dynamics, with Applications to Paleoclimatic Time Series." *Physica D*, **35**(3), 395–424. doi:10.1016/0167-2789(89)90077-8.

Veen AJ, Deprettere E, Swindlehurst A (1993). "Subspace-Based Signal Analysis Using Singular Value Decomposition." *Proceedings of the IEEE*, **81**(9), 1277–1308. doi:10.1109/5.237536.

Wang Y, Chan JW, Liu Z (2005). "Comments on 'Estimation of Frequencies and Damping Factors by Two-Dimensional ESPRIT Type Methods'." *IEEE Transactions on Signal Processing*, **53**(8), 3348–3349. doi:10.1109/tsp.2005.851184.

Weare B, Nasstrom J (1982). "Examples of Extended Empirical Orthogonal Function Analyses." *Monthly Weather Review*, **110**(6), 481–485. doi:10.1175/1520-0493(1982)110<0481:eoeeof>2.0.co;2.

Yamazaki I, Bai Z, Simon H, Wang LW, Wu K (2008). "Adaptive Projection Subspace Dimension for the Thick-Restart Lanczos Method." *Technical report*, Lawrence Berkeley National Laboratory, University of California, One Cyclotron Road, Berkeley, California 94720.

Zeileis A, Grothendieck G (2005). "**zoo**: S3 Infrastructure for Regular and Irregular Time Series." *Journal of Statistical Software*, **14**(6), 1–27. doi:10.18637/jss.v014.i06.

A. Elements of MSSA theory

We put into this section several propositions related to the MSSA theory taken from Golyandina, Nekrutkin, and Stepanov (2003), Stepanov and Golyandina (2005, in Russian).

A.1. Separability

Separability is the key notion in the SSA theory, since separability of series means the ability of the method to extract them from the given sum-total series. Notion of separability for multidimensional time series is analogous to that for one-dimensional series, which is briefly commented in Section 2.2 and is thoroughly described in Golyandina *et al.* (2001, Sections 1.5 and 6.1). There is weak separability, which means orthogonality of the trajectory spaces, and strong separability, that means empty intersection of the sets of singular values produced by the separated series.

Generally, 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.

Proposition 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 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 and 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 by CSSA.*

Proposition 1 can be extended to an analogous result for asymptotic separability ($N_i \to \infty$) and therefore for approximate separability for fixed N_i.

Example 1. *Consider the example of four harmonic real-valued time series $\mathbb{F}^{(1)}$, $\mathbb{F}^{(2)}$, $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$ of length N*

$$f_k^{(1)} = A_1 \cos(2\pi\omega_1 k + \varphi_1), \qquad f_k^{(2)} = B_1 \cos(2\pi\omega_1 k + \varphi_2),$$
$$g_k^{(1)} = A_2 \cos(2\pi k\omega_2 k + \phi_1), \qquad g_k^{(2)} = B_2 \cos(2\pi k\omega_2 k + \phi_2),$$

$\omega_1 \neq \omega_2$, $k = 0, \ldots, N-1$, $A_1, A_2, B_1, B_2 \neq 0$. If $L\omega_i$ and $K\omega_i$, $i = 1, 2$, are integer, then $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ and $(\mathbb{G}^{(1)}, \mathbb{G}^{(2)})$ are L-separable by MSSA and 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 by CSSA.

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 each time series $(\mathbb{F}^{(i)}, \mathbb{G}^{(i)})$ can produce different eigenvalues in SSA, MSSA and CSSA. Therefore, the application of an appropriate multidimensional modification of SSA can improve the strong separability.

Note that if $L\omega_i$ or $K\omega_i$ is not integer, then the series become approximately separable.

Example 2. *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 integer, then $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ produces two equal eigenvalues in MSSA: $\lambda_1 = \lambda_2 = (A^2 + B^2)LK/4$, while $\mathbb{F}^{(1)} + \mathrm{i}\mathbb{F}^{(1)}$ produces two nonequal eigenvalues in CSSA:

$$\begin{aligned}\lambda_1 &= (A^2 + B^2 + 2AB\sin\varphi)LK/4,\\ \lambda_2 &= (A^2 + B^2 - 2AB\sin\varphi)LK/4,\end{aligned}$$

where $\varphi = \varphi_1 - \varphi_2$ and $|\varphi| \neq \pi/2 \bmod \pi$ (otherwise, the series produces non-zero eigenvalues equal to $(A^2 + B^2)LK/2$). Note that the series $\mathbb{F}^{(1)}$ itself produces two eigenvalues equal to $A^2LK/4$.

A.2. Multi-dimensional time series and LRRs

Consider a system of infinite time series $\mathbb{X}^{(1)}, \mathbb{X}^{(2)}, \ldots, \mathbb{X}^{(s)}$, choose the window length L and denote $\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(s)}$ the column trajectory spaces of the series (subspaces spanned by the L-lagged vectors of the series). Let $\mathcal{X} = \mathrm{span}(\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(s)})$ be the column trajectory space of the collection of time series $(\mathbb{X}^{(1)}, \mathbb{X}^{(2)}, \ldots, \mathbb{X}^{(s)})$. As well as for one-dimensional time series, we call the dimension of the trajectory space (equal to the rank of the trajectory matrix of the series collection) the rank of the series collection, see Section 2.2 for short description of general notions.

Denote the ranks of $\mathbb{X}^{(l)}$ by $r_l = \dim \mathcal{X}^{(l)} \leq L$, $l = 1, \ldots, s$. For each time series $\mathbb{X}^{(l)}$ we can write out the minimal LRR governing the time series:

$$x^{(l)}_{j+r_l} = \sum_{k=1}^{r_l} a^{(l)}_k x^{(l)}_{j+r_l-k}, \quad \text{where} \quad a^{(l)}_{r_l} \neq 0, \quad l = 1, \ldots, s. \tag{28}$$

The corresponding characteristic polynomials of the LRR (28) are

$$P^{(l)}_{r_l}(\mu) = \mu^{r_l} - \sum_{k=1}^{r_l} a^{(l)}_k \mu^{r_l-k}, \quad l = 1, \ldots, s. \tag{29}$$

The roots of the characteristic polynomial of the minimal LRR governing the series are called *characteristic roots*.

Let

$p^{(l)}$	be	the number of different roots of the polynomial $P^{(l)}_{r_l}(\lambda)$,
$\mu^{(l)}_m$	be	the mth root of the polynomial $P^{(l)}_{r_l}(\lambda)$,
$k^{(l)}_m$	be	the multiplicity of the root $\mu^{(l)}_m$.

Then, from standard theory of time series of finite rank (Golyandina *et al.* 2001), we have that

$$k^{(l)}_1 + \ldots + k^{(l)}_{p^{(l)}} = r_l, \quad l = 1, \ldots, s.$$

The characteristic roots determine the series behavior. For example, if $k^{(l)}_m = 1$, then the time series has the form

$$x^{(l)}_n = \sum_{j=1}^{r_l} C^{(l)}_j \left(\mu^{(l)}_j\right)^n.$$

Also let

$\mu_1, \ldots, \mu_p$	be	the pooled set of roots of all the 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 maximal multiplicity of the corresponding root across all the polynomials.

Since the roots are determined by the structure of the trajectory space, the following proposition can be proved.

Proposition 2. *Rank of the infinite multi-dimensional time series* $(\mathbb{X}^{(1)}, \mathbb{X}^{(2)}, \ldots, \mathbb{X}^{(s)})$ *is equal to* $r = \sum_{i=1}^{p} k_i$*, for* $L > r$*.*

Consider a simple example.

Example 3. *Let* $\mathbb{F} = (f_1, \ldots, f_N)$ *and* $\mathbb{G} = (g_1, \ldots, g_N)$ *with*

$$f_k^{(1)} = A\cos(2\pi\omega_1 k + \varphi_1), \quad f_k^{(2)} = B\cos(2\pi\omega_2 k + \varphi_2), \tag{30}$$

where $0 < \omega < 1/2$*,* $0 \le \varphi_1, \varphi_2 < 2\pi$ *and* $A, B \neq 0$*. Let us fix the window length* L *and find the SSA rank of the time series* $\mathbb{F}$*, the MSSA rank of* $(\mathbb{F}^{(1)}, \mathbb{F}^{(2)})$ *and the CSSA rank of* $\mathbb{F}^{(1)} + \mathrm{i}\mathbb{F}^{(2)}$*:*

1. *For* $\omega_1 = \omega_2$ *the SSA and the MSSA ranks of the sinusoid (30) is equal to 2. The CSSA rank is equal to 1 if* $A = B$ *and* $|\varphi_1 - \varphi_2| = \pi/2 \bmod \pi$ *and is equal to 2 otherwise.*
2. *For* $\omega_1 \neq \omega_2$ *the SSA, MSSA and CSSA ranks equal 4.*

B. Elements of 2D-SSA theory

B.1. Arrays of finite rank

The theory of arrays of finite rank is mainly contained in Golyandina and Usevich (2009). We present here a short summary.

In this section we consider a class of infinite arrays $\mathbb{X} = (x_{m,n})_{m,n=0}^{\infty,\infty}$ given in a parametric form:

$$x_{m,n} = \sum_{k=1}^{r} c_k \lambda_k^m \mu_k^n, \tag{31}$$

where $(\lambda_k, \mu_k) \in \mathsf{C}^2$ are distinct pairs of complex numbers. 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 (31) are *arrays of finite rank*). Note that the exponentials can be also represented in the form

$$\lambda_k = \rho_{x,k}\exp(2\pi\mathrm{i}\omega_{x,k} + \phi_{x,k}), \quad \mu_k = \rho_{y,k}\exp(2\pi\mathrm{i}\omega_{y,k} + \phi_{y,k}).$$

We should note that the class of arrays of finite rank also contains bivariate polynomials and their products with the functions from (31). An algebraic characterization of the arrays of finite rank can be found in Golyandina and Usevich (2009).

B.2. Real arrays

Now let us summarize what happens in the case of real arrays $\mathbb{X}$. If (31) is real then for any pair $(\lambda, \mu) \in \mathsf{C}^2$ its complex conjugate $(\overline{\lambda}, \overline{\mu})$ also should be present in (31). We order the roots such that (λ_k, μ_k) are real for $1 \le k \le d$, and other $2s$ roots are at least partially complex (there are complex conjugate pairs), and are arranged as $(\lambda_k, \mu_k) = (\mathrm{conj}(\lambda_{k-s}), \mathrm{conj}(\mu_{k-s}))$ for $d+s+1 \le k \le d+2s$. Then the roots have a representation

$$(\lambda_k, \mu_k) = \begin{cases} (\rho_{x,k}, \rho_{y,k}) = (\rho_{y,k} \exp(2\pi\mathrm{i}\omega_{x,k}), \rho_{y,k} \exp(2\pi\mathrm{i}\omega_{y,k}))\,, & 1 \le k \le d, \\ (\rho_{x,k} \exp(2\pi\mathrm{i}\omega_{x,k}), \rho_{y,k} \exp(2\pi\mathrm{i}\omega_{y,k}))\,, & d+1 \le k \le d+s, \\ (\rho_{x,k-s} \exp(-2\pi\mathrm{i}\omega_{x,k-s}), \rho_{y,k} \exp(-2\pi\mathrm{i}\omega_{y,k-s}))\,, & d+s+1 \le k \le r, \end{cases}$$

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 (31) becomes a sum of $d+s$ planar modulated sinewaves:

$$x_{m,n} = \sum_{k=1}^{d+s} b_k \rho_{x,k}^m \rho_{y,k}^n \cos\left(2\pi(\omega_{x,k} m + \omega_{y,k} n) + \phi_k\right),$$

where b_k and $\phi_k \in [0; 2\pi)$ are unique real coefficients obtained from c_k.

Example 4. *The product of sines can be uniquely represented as a sum of two planar sines:*

$$\begin{aligned} &2\cos(2\pi\omega_x m + \phi_1)\cos(2\pi\omega_y n + \phi_2) \\ &\qquad = \cos(2\pi(\omega_x m + \omega_y n) + \phi_1 + \phi_2) + \cos(2\pi(\omega_x m - \omega_y n) + \phi_1 - \phi_2) \end{aligned}$$

Affiliation:

Nina Golyandina, Anton Korobeynikov, Alex Shlemov
Department of Statistical Modelling
Faculty of Mathematics and Mechanics
St. Petersburg State University
Universitetskij pr 28, 198504, St.Petersburg, Russia
E-mail: nina@gistatgroup.com, anton@korobeynikov.info, shlemovalex@gmail.com

Konstantin Usevich
GIPSA-lab, CNRS UMR5216
11 rue des Mathématiques
Grenoble Campus, BP 46
F-38402 St. Martin d'Hères cedex, France
E-mail: konstantin.usevich@gipsa-lab.grenoble-inp.fr

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
October 2015, Volume 67, Issue 2 *Submitted:* 2013-09-18
doi:10.18637/jss.v067.i02 *Accepted:* 2014-11-13