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 base for implementation of MSSA and other generalizations. Efficient implementation of operations with Hankel and Hankel-block-Hankel matrices through FFT 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 (SSA) as a method of time series analysis has well-elaborated theory and solves various problems: time series decomposition, trend extraction, periodicity detection and extraction, signal extraction, denoising, filtering, forecasting, missing data imputation, change point detection, spectral analysis among them (see examples and references in Vautard and Ghil (1989); Golyandina et al. (2001); Ghil et al. (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 the 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 (Van Der Veen et al. 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 is 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.

The aforementioned possibilities are implemented in different software based on different methodologies of SSA, see Golyandina and Korobeynikov (2013) 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 Giset 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; see Golyandina and Korobeynikov (2013), where analysis, forecasting and parameter estimation by means of Rssa for one-dimensional series are described.

At the present time, Rssa is extensively developed and includes SSA-processing of systems of series (Multivariate or Multichannel SSA, shortly 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., circle images, images with gaps and so on. Also, it is shown that the implementation of Shaped 2D-SSA can serve as a common base for implementation of many SSA extensions.

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

$$ X = X_1 + \cdots + X_m. $$

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, a digital image. The algorithm is divided into four steps. The first step is the generation of some multivariate object on the base 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 the time series of length $L$, 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 into a matrix as columns. Let us call this matrix trajectory. The trajectory matrix has a specific structure: it is a Hankel, stacked Hankel, Hankel-block-Hankel or quasi-Hankel matrix.

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

The last forth step converts the grouped matrix decomposition back to the decomposition of the initial time series or image decomposition. The elements in 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 objects 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 these 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 the model of the series can be constructed during the SSA processing. The simplified (possible polynomial modulation is omitted) model of the signal has the form $s_n = \sum_{k=1}^{r} A_k \mu_k^n$, where $\mu_k = \rho_k e^{i2\pi \omega_k}$. The general approach for estimation of the modulations $\rho_k$ and frequencies $\omega_k$ 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 is ESPRIT (Roy and Kailath 1989). This approach has an 2D extension named 2D-ESPRIT for estimation of parameters in the model $(S)_m = \sum_{k=1}^{r} A_k \mu_k^n \nu_k^n$.

Finally we dwell on the issues related to efficient implementation. 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 usage of the algorithms possessing good theoretical complexity will not automatically yield the 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.

What is more, the package supports several input data types, tries not to constraint the user and carefully preserves attributes of input data and has various useful plotting capabilities.

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.

The structure of the paper is as follows. Section 2 contains the general description of the SSA approach to show common character of the considered extensions and introduce the necessary notions. 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 are considered in Section 4. Section 5 devoted to a new shaped 2D version of SSA for analysis of images of non-rectangular form.

Each of Sections 3–5 contains the implemented algorithms, description of the \texttt{Rssa} functionality with a typical \texttt{R} code, simulated and real-life examples with the corresponding fragments of \texttt{R} code accompanied by guidelines for the proper use. Since implementation of multivariate extensions is based on that of Shaped 2D-SSA, the description of implementations in Section 6 is provided in the inverse order; for example, MSSA is considered as a particular case of Shaped 2D-SSA.

Section 6 contains details of the used fast methods for matrix multiplication and hankelization and also the fast algorithm of the vector forecasting. Theoretical details of the MSSA and 2D-SSA algorithms are put into Appendix (Section 7).

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

Let us describe the common features of the algorithms of the SSA extensions and the corresponding notions. Before introduction of the algorithm details, we present a general scheme of the SSA-like algorithms in Fig. 1.

![Scheme of the SSA-like algorithms](image)

The algorithm consists of four steps. The input data are the object $X$ and the mapping $T$. The result of the algorithm is a decomposition of $X$ into sum of additive components. The intermediate outcome of the steps can be used for solving additional problems like forecasting.
Table 1: Kinds of SSA-like multivariate extensions

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

and parameter estimation.

Tabl. 2 contains the list of extensions considered in the paper with some details.

Further we discuss the steps of the algorithms of these extensions in detail.

2.1. Details of the algorithms

Step 1. Embedding. The first step consists in the construction of a so called trajectory matrix $X = T(X)$ by means of a map $T$.

For example, in the Basic SSA algorithm for analysis of an one-dimensional time series $X = (x_1, \ldots, x_N)$, $T$ maps $\mathbb{R}^N$ to the space of Hankel matrices $L \times K$ with equal values on anti-diagonals, where $N$ is the series length, $L$ is the window length and $K = N - L + 1$:

$$
T_{SSA}(X) = \begin{pmatrix}
    x_1 & x_2 & x_3 & \cdots & x_K \\
    x_2 & x_3 & x_4 & \cdots & x_{K+1} \\
    x_3 & x_4 & x_5 & \cdots & x_{K+2} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    x_L & x_{L+1} & x_{L+2} & \cdots & x_N
\end{pmatrix}.
$$

Let us describe the scheme of this step in a more formal manner. Denote $\mathcal{M}_{p,q}$ the set of $p \times q$ matrices. Let an ordered set of real values $X$ be the initial data, i.e., series or image, and $\mathcal{M}$ be the linear space consisting of possible $X$. Let $T$ be a one-to-one mapping from $\mathcal{M}$ to $\mathcal{M}^{(H)}_{L,K} \subset \mathcal{M}_{L,K}$, where $\mathcal{M}^{(H)}_{L,K}$ consists of matrices of some Hankel-like structure, $L$ and $K$ are determined by the method parameters. The mapping $T$ is called embedding and put elements of $X$ on some places of $X$. Therefore, for each element $x$ of $X$ the mapping $T(X) = X = \{(X)_{ij}\}$ determines the set of indices $\mathcal{A}$ such that $(X)_{ij} = x$ for any $(i,j) \in \mathcal{A}$. Formally, let $E_i \in \mathcal{M}$ be the object with 1 on the $i$-th entry and other entries are zeros. Then $\mathcal{A} = \mathcal{A}_i$ corresponds to places of ‘1’ in $T(E_i)$.

For example, in the Basic SSA algorithm applied to an one-dimensional time series $X$, $\mathcal{M} = \mathbb{R}^N$, $\mathcal{M}^{(H)}_{L,K}$ is the set of Hankel matrices with equal values on anti-diagonals $\mathcal{A}$, $T$ is given in (2).

Step 2. Singular Value Decomposition.

Let $S = XX^T$, $\lambda_1 \geq \ldots \geq \lambda_L \geq 0$ be eigenvalues of the matrix $S$, $d = \max\{j : \lambda_j > 0\}$, $U_1, \ldots, U_d$ be the corresponding eigenvectors, and $V_j = X^TU_j/\sqrt{\lambda_j}$, $j = 1, \ldots, d$, be factor vectors. Denote $X_j = \sqrt{\lambda_j}U_jV_j^T$. Then the SVD of the trajectory matrix $X$ can be written...
Then the resultant matrix $X$ inferred back to the form of the input object is

$$X = X_1 + \ldots + X_d. \quad (3)$$

The triple $(\sqrt{\lambda}_j, U_j, V_j)$ consisting of the singular value $\sqrt{\lambda}_j$, the left singular vector $U_j$ and the right singular vector $V_j$ of $X$ is called $j$-th eigentriple.

**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 $I_1, \ldots, I_m$. Let $I = \{i_1, \ldots, i_p\}$. Then the resultant matrix $X_I$ corresponding to the group $I$ is defined as $X_I = X_{i_1} + \ldots + X_{i_p}$. Thus, we have the **grouped matrix decomposition**

$$X = X_{I_1} + \ldots + X_{I_m}. \quad (4)$$

For example, grouping can be performed on the base of the form of eigenvectors, which reflects 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 $X$. It is performed optimally in the following sense: for a matrix $Y$ we seek for the object $\tilde{Y} \in M$ that provides the minimum to $\|Y - T(\tilde{Y})\|_F$, where $\|Z\|_F = \sum_{i,j} z_{ij}^2$ is the Frobenius norm.

Denote $\Pi^{(H)} : M_{L,K} \to M^{(H)}_{L,K}$ the orthogonal projection on $M^{(H)}_{L,K}$ in Frobenius norm. Then $\tilde{Y} = T^{-1}(\Pi^{(H)}Y)$. Note that by the embedding nature of $T$ the projection $\Pi^{(H)}$ is the averaging of the entries along the sets $A$. For example, in SSA the composite mapping $T^{-1} \circ \Pi^{(H)}$ is the averaging along anti-diagonals, that is, $x_k = \sum_{(i,j) \in A_k} (X)_{ij} / |A_k|$, where $A = A_k = \{(i, j) : 1 \leq i \leq L, 1 \leq j \leq K, i + j = k - 1\}$. The same averaging can be written down in a general form: $x_k = \langle X, T(E_k) \rangle / \|T(E_k)\|^2_F$.

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

$$\tilde{X} = \tilde{X}_1 + \ldots + \tilde{X}_m. \quad (5)$$

If the grouping is elementary, then $m = d$ and the reconstructed objects in (5) are called **elementary components**.

### 2.2. General notions

**Separability** Very important notion in SSA is separability. Let $X = X_1 + X_2$. (Approximate) separability means that there exist such grouping that the reconstructed series $\tilde{X}_1$ is (approximately) equal to $X_1$. Properties of the SVD yields the (approximate) orthogonality of columns and of rows of trajectory matrices $X_1$ and $X_2$ of $\tilde{X}_1$ and $X_2$ as the separability condition. There is 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 $T$ determines separability conditions. It could be said that the success of SSA in separability is determined by Hankel structure of the trajectory matrix and optimality features of the SVD.
Information for grouping  The theory of SSA provides the ways how to detect the SVD components related to the series component 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 \( \tilde{X}_1 \) and \( \tilde{X}_2 \), we should check the orthogonality of their reconstructed trajectory matrices \( X_1 \) and \( X_2 \). The convenient measure of their orthogonality is the Frobenius inner product \( \langle X_1, X_2 \rangle_F = \sum_{i,j=1}^{L,K} y_{ij} z_{ij} \).

Since the trajectory matrix consists of \( w_i = |A_i| = \| T(E_i) \|_F^2 \) entries of \( i \)th element \( x_i \) of the initial ordered object, we can introduce the weighted inner product in the space \( \mathcal{M} \):
\[
\langle Y, Z \rangle_w = \sum_i w_i y_i z_i.
\]

Then
\[
\rho_w(\tilde{X}_1, \tilde{X}_2) = \rho(\tilde{X}_1, \tilde{X}_2) = \frac{\langle \tilde{X}_1, \tilde{X}_2 \rangle_w}{\| \tilde{X}_1 \|_w \| \tilde{X}_2 \|_w} \tag{6}
\]
called \( w \)-correlation by statistical analogy. Note that in this definition means are not subtracted.

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

The norm \( \| \cdot \|_w \) is called the weighted norm and serves as a measure of contribution of components to the decomposition (5): the share of \( \tilde{X}_k \) is defined as \( \| \tilde{X}_k \|_w^2 / \| X \|_w^2 \).

Trajectory spaces and signal subspace  Let us introduce several notions related to subspaces generated by the data. For the data \( X \) the column (row) subspace of its trajectory matrix \( X \) is called column (row) trajectory space. The term ‘trajectory space’ usually means ‘column trajectory space’. The column trajectory space is a subspace of \( \mathbb{R}^L \), while the row trajectory space is a subspace of \( \mathbb{R}^K \). In general, for real-world data the trajectory spaces coincide with the corresponding Euclidean spaces, since they are produced by a signal corrupted by noise. However, if the signal has 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 consists of Hankel blocks, then by well-known properties, rank-deficient Hankel matrices are closely related to
objects satisfying to 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 (LRRs) \( x_n = \sum_{i=1}^{r} a_i x_{n-i} \) 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 indicate the SVD components (whose number is equal to the rank value) for the component reconstruction. 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, SSA can extract the signal and then apply the methods that are designed for series of finite rank.

2.3. Typical code

The general structure of the \texttt{Rssa} package is described in Golyandina and Korobeynikov (2013) and holds for the multivariate extensions. Therefore let us discuss shortly the base \texttt{Rssa} functions for analysis and forecasting of one-dimensional time series. Implementation of SSA analysis and forecasting is mostly contained in the functions \texttt{ssa} (decomposition), \texttt{reconstruct} (reconstruction), \texttt{rforecast} and \texttt{vforecast} (forecasting), \texttt{plot} (plotting), \texttt{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 \(< 1 \text{ 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 \texttt{Rssa} and input of the data included into the package.

\textbf{Fragment 1} (Australian Wines: input).

\begin{verbatim}
> library(Rssa)
> data(AustralianWine)
> wine <- window(AustralianWine, end = time(AustralianWine)[174])
\end{verbatim}

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

\textbf{Fragment 2} (FORT: reconstruction).

\begin{verbatim}
> fort = wine[, "Fortified"]
> s.fort <- ssa(fort, L = 84, kind = "1d-ssa")
> r.fort <- reconstruct(s.fort, groups = list(Trend = 1,
+ Seasonality = 2:11))
> plot(r.fort, add.residuals = TRUE, add.original = TRUE,
+ plot.method = "xyplot",
+ superpose = TRUE, auto.key = list(columns = 2))
\end{verbatim}
Roughly speaking (see details in Golyandina and Korobeynikov (2013)), ssa performs steps 1 and 2, while reconstruct perform steps 3 and 4 of the algorithm described in Section 2.1. The 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 standard package lattice.

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

1. 1D figures of eigenvectors $U_i$ (Fig. 3),
2. 2D figures of eigenvectors $(U_i, U_{i+1})$ (Fig. 4),
3. matrix of w-correlations $\rho_w$ between elementary reconstructed series (functions wcor and plot, Fig. 5).

The following fragment shows the corresponding code.

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

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

| 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 |
Multivariate and 2D Extensions of SSA with \texttt{Rssa}

\begin{verbatim}
$F2$

| period | rate  | Mod    | Arg    | Re     | Im     |
|--------|-------|--------|--------|--------|--------|
| 4.005  | 0.00037 | 1.00004 | 1.57 | 0.00189 | 1.00003 |
| -4.005 | 0.00037 | 1.00004 | -1.57 | 0.00189 | -1.00003 |

> plot(wcor(s.fort, groups = 1:30),
+ scales = list(at = c(10, 20, 30)))
> 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 for the grouping. Fig. 3 shows that the first eigenvector is slowly-varying and therefore the eigentriple (abbreviated as ET) ET1 should be included to the trend group. Fig. 4 shows that the pairs 2–3, 4–5, 6–7, 8–9, 10–11 are produced by 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 orthogonality of eigenvectors.

Their periods are 12, 4, 6, 2.4, 3, what can be determined by calculation of the number of polygon vertexes. Automatic methods of frequency calculation including LS-ESPRIT and TLS-ESPRIT Roy and Kailath (1989) are implemented in \texttt{Rssa} in the function \texttt{parestimate} and are described in (Golyandina et al. 2001, Sections 2.4.2.4. and 3.8.2) and Golyandina and Korobeynikov (2013) for one-dimensional time series.

The matrix of absolute values of $w$-correlations in Fig. 5 is depicted in white-black scale (white color corresponds to zero values, while black color reflects values equal to 1). Fig. 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.

Fig. 6 contains four reconstructed modulated sine waves and shows that several sine waves has increasing amplitudes, while others are decreasing; the same is seen in Fig. 3. In Fig. 2, we grouped the modulated sine waves and obtained the seasonal component with varying form.

Finally, two methods of forecasting are implemented for one-dimensional time series: recurrent (function \texttt{rforecast}) and vector forecasting (function \texttt{vforecast}).

Fragment 4 shows how to forecast the series components (trend and signal) and depicts the result (Fig. 7). A proper decomposition is the necessary condition of the forecast accuracy.

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

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

|   | 1 (94.65%) | 2 (1.43%) | 3 (1.36%) | 4 (0.5%) |
|---|------------|-----------|-----------|----------|
|   | [Graph 1](#) | [Graph 2](#) | [Graph 3](#) | [Graph 4](#) |
|   | 5 (0.5%)   | 6 (0.26%) | 7 (0.25%) | 8 (0.15%) |

Figure 3: FORT: 1d graphs of eigenvectors

Pairs of eigenvectors

|   | 2 vs 3 | 3 vs 4 | 4 vs 5 | 5 vs 6 | 6 vs 7 |
|---|--------|--------|--------|--------|--------|
|   | [Graph 5](#) | [Graph 6](#) | [Graph 7](#) | [Graph 8](#) | [Graph 9](#) |
|   | 7 vs 8   | 8 vs 9   | 9 vs 10 | 10 vs 11 | 11 vs 12 |

Figure 4: FORT: 2d scatterplots of eigenvectors

2.4. Comments on Rssa

The detailed description of the Rssa package structure and the principles of implementation is contained in Golyandina and Korobeynikov (2013). 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. Note that the 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 Fig. 6, where `plot.ts` (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 to the result the time scale). 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`, `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 `xyplot` method of `plot` for SSA reconstruction objects. There are powerful specializations of `xyplot` for `ts` and `zoo` objects and `Rssa` provides 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`. These implementations differ in terms of speed, memory consumption and numerical stability, see (Golyandina and Korobeynikov 2013) 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 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 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, the 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 and hankelization of matrices, which can be very large, take a substantial part of 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 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. 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
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, shortly MSSA. The main idea of the algorithm is the same as in the Basic SSA, the difference consists in the way of construction of the trajectory matrix.

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 the analysis and forecasting of multivariate time series following the approach described in Golyandina et al. (2001, Chapter 2) for one-dimensional series and in Golyandina and Stepanov (2005) for multidimensional ones. In this section we also describe the complex-valued version of SSA (called CSSA), since it can be considered as a multidimensional version 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 Section 7.1. Let us start with the algorithm description.

3.1. MSSA and CSSA algorithms

**MSSA analysis**

Consider a multivariate time series, that is, a collection \( \{X^{(p)} = (x_j^{(p)})_{j=1}^{N_p}, \ p = 1, \ldots, s\} \) of \( s \) time series of length \( N_p, p = 1, \ldots, s \).

Denote \( X = [X^{(1)} : \ldots : X^{(s)}] \) 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 \( T_{MSSA}(X) = X \) only.

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

\[
T_{MSSA}(X) = X = [X^{(1)}_1 : \ldots : X^{(1)}_{K_1} : \ldots : X^{(s)}_1 : \ldots : X^{(s)}_{K_s}] = [X^{(1)} : \ldots : X^{(s)}], \tag{7}
\]

where \( X^{(p)} = T_{SSA}(X^{(p)}) \) is the trajectory matrix of the one-dimensional series \( X^{(p)} \) 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 \( X \) 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,

\[
V_i = \begin{pmatrix}
    V_i^{(1)} \\
    \vdots \\
    V_i^{(s)}
\end{pmatrix},
\]

where \( V_i^{(p)} \in \mathbb{R}^{K_p} \) and belong to the row trajectory spaces of the \( p \)th 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 \textit{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 one-dimensional case, the main result of application of MSSA is a decomposition of the multivariate time series into a sum of \( m \) multivariate series; the parameters are the window length \( L \) and the way of grouping. For the frequently used case \( m = 2 \), we denote by \( \tilde{X}^{(k)} = (\tilde{x}_j^{(k)})_{j=1}^N \), \( k = 1, \ldots, s \), the reconstructed series (usually, the signal) corresponding to the first group of eigentriples \( I_1 \).

\textbf{CSSA analysis}

Let the system of series consist of two series, that is, \( s = 2 \). Then we can consider one-dimensional complex-valued series \( X = X^{(1)} + iX^{(2)} \) 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-values space, where the transposition should be Hermitian.

Also, there is a generality of the uniqueness feature 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 \( c \).

\textbf{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 \( X^{(p)} \) 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
\( 2 \leq L \leq [(N + 1)/2] \), since the SVD expansions for window lengths \( L \) and \( N - L + 1 \)
coincide. For the MSSA-analysis of more than one time series the expansions for all
possible window lengths \( 2 \leq L \leq \min_i N_i - 1 \) are generally different.

4. For simultaneous analysis of several time series, it is recommended to transfer them into
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 \( X \)
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 trajectory matrix \( X \) is approximately \( sN/(s + 1) \) for MSSA. For
the case of two time series this corresponds to \( 2L/3 \) for MSSA, while CSSA gives \( N/2 \).

6. The MSSA algorithm might be modified by 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 \texttt{Rssa}.

3.2. Comments to the algorithms

\textit{Covariance structure}

Consider in more detail the case of two time series \( X = (F, G) \) and let \( F \) and \( G \) be the
trajectory matrices of \( F \) and \( G \) correspondingly. Then MSSA considers the eigendecomposition
of \( S = XX^\top = FF^\top + GG^\top \), 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,
the same, to transpose \( X \) and consider eigenvectors of

\[
S = \begin{pmatrix}
F^\top F & F^\top G \\
G^\top F & G^\top G
\end{pmatrix}
\]

called EEOFs. The last formula more clearly demonstrates that MSSA takes into considera-
tion cross-covariances of time series (more exactly, cross-covariances if centering is used).

Since the eigendecomposition of a complex-valued matrix \( A + iB \) can be reduced to the
eigendecomposition of the real-valued matrix

\[
D = \begin{pmatrix}
A & -B \\
B & A
\end{pmatrix},
\]

in the case of CSSA we in fact analyze eigenvectors of the matrix

\[
S = \begin{pmatrix}
F^\top F & F^\top G \\
G^\top F & G^\top G
\end{pmatrix} + \begin{pmatrix}
G^\top G & -G^\top F \\
-F^\top G & F^\top F
\end{pmatrix},
\]
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-world series, the trajectory matrix is of full rank, since at least noise has no structure. Therefore, in what follows we say about rank of 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, for two sine waves with equal periods their trajectory spaces 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 series trajectory spaces.

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 et al. (2013), where the attempt to detect causality is performed). Moreover, asymmetry of influence of one time series to the 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 time series has 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 SSA-ranks for some special cases.

Consider a collection $\mathbb{H}^{(k)} = (h_j^{(k)})_{j=1}^N$, $k = 1, \ldots, s$, of $s$ signals of length $N$. Let $r_k$ denote the SSA rank of $\mathbb{H}^{(k)}$ (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 $r_k$, $k = 1, \ldots, s$, is considered in Section 7.1. In particular, it is shown that $r_{\min} \leq r \leq r_{\max}$, where $r_{\min} = \max\{r_k, k = 1, \ldots, s\}$ and $r_{\max} = \sum_{k=1}^s r_k$. The case $r = r_{\max}$ is the least favorable for MSSA and means that different time series do not have matched components. The case $r < r_{\max}$ indicates the presence of matched components and 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 Section 7.1 for definition), then this means that the time series $X^{(m)}$, $m = 1, \ldots, s$, 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 in Section 2.2 and thoroughly described in (Golyandina et al. 2001, Sections 1.5 and 6.1). Section 7.1 contains several results on separability of multidimensional series. 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 Section 7.1) 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 the 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. As far as is known, 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. The attempt of numerical investigation is done in Golyandina and Stepanov (2005) and extended in Section 3.5.

Analogy with one-dimensional case can help to 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. Also, different special techniques can be transferred from SSA to MSSA. For example, Sequential MSSA can be applied by analogy with Sequential SSA. 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. While for SSA analysis it makes no sense to take \(L > (N + 1)/2\), for MSSA analysis large \(L\) with small \(K_i = N_i - L + 1\) can be taken instead of small \(L\) for trend extraction and smoothing.

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, \(\min(L, K_i)\) is not necessary equals \(L\) and therefore one puts attention on values of \(K_i\).

Numerical investigations show that the formula \(L = sN/(s + 1)\) 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 ($K_i$ become too small for separability). Generally, the choice $L = N/2$ is still appropriate even for multivariate SSA.

### 3.3. Multivariate SSA forecasting

Forecasting in SSA is performed for a series component which can be separated by SSA and is described, maybe, approximately, by an LRR. For brevity, we’ll say about forecasting of signal. If the forecasted series component (signal) is governed by an LRR, then it can be forecasted by this or extended LRR. The forecasting method is called **recurrent** if it consists in the continuation of the estimated signal by the estimated LRR.

Since the series is transformed to the sequence of $L$-dimensional reconstructed vectors laying in the signal subspace, then the forecast can be constructed by continuation of vectors in the given subspace. The forecasting SSA method is called **vector** if it is based on the continuation of the reconstructed vectors in the given subspace. The estimated signal subspace is also used in the recurrent forecasting for construction of the LRR.

CSSA forecasting algorithm is the straightforward extension of SSA forecasting to the complex-valued case, therefore we will not describe the algorithm.

Methods of MSSA forecasting of series systems are similar to that of SSA forecasting of one-dimensional series. Since rows and columns of the trajectory matrix of a system of time series has different structure in contrast with SSA, there are two kinds of MSSA forecasting, row and column, depending on what space is used, row or column. Note that the column forecasting is exactly the SSA forecasting of each series in the system separately, in a given common subspace. Methods of one-dimensional SSA forecasting in a given subspace are fully described in (Golyandina et al. 2001, Chapter 2).

**Notation**

Let us introduce necessary notation.

Denote by $\overline{A} \in \mathbb{R}^{Q-1}$ the vectors of the last $Q-1$ coordinates of $A \in \mathbb{R}^Q$, that is, with the removed first coordinate, what is indicated by the line on the top of the vector denotation. Also, denote by $\underline{A} \in \mathbb{R}^{Q-1}$ the vectors of the first $Q-1$ coordinates of $A$, by $\pi(A)$ the last coordinate of the vector. For matrix $A = [A_1 : \ldots : A_r]$ denote $\overline{A} = [\overline{A}_1 : \ldots : \overline{A}_r]$ and $\underline{A} = [\underline{A}_1 : \ldots : \underline{A}_r]$ and let $\pi(A) = (\pi(A_1), \ldots, \pi(A_r))^\top$ be the last row of the matrix $A$.

Consider the following form of $B \in \mathbb{R}^K$, where $K = \sum_{i=1}^s K_i$, induced by the structure of the row trajectory space:

$$B = \begin{pmatrix} B^{(1)} \\ B^{(2)} \\ \vdots \\ B^{(s)} \end{pmatrix}, \quad \overline{B} = \begin{pmatrix} \overline{B}^{(1)} \\ \overline{B}^{(2)} \\ \vdots \\ \overline{B}^{(s)} \end{pmatrix}, \quad \underline{B} = \begin{pmatrix} \underline{B}^{(1)} \\ \underline{B}^{(2)} \\ \vdots \\ \underline{B}^{(s)} \end{pmatrix},$$

where $B^{(j)} \in \mathbb{R}^{K_j}$, and let $\mu(B) = (\pi(B^{(1)}), \ldots, \pi(B^{(s)}))^\top$. Also, for $B = [B_1 : \ldots : B_r]$ let $\overline{B} = [\overline{B}_1 : \ldots : \overline{B}_r]$ and $\underline{B}^{(j)} = [\underline{B}_1^{(j)} : \ldots : \underline{B}_r^{(j)}]$.

For simplicity we assume that the set $I$ corresponding to the forecasted component is given by the set of the leading components; 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 \( r \) so that \( I = I_1 = \{1, \ldots, r\}, U = [U_1 : \ldots : U_r], V = [V_1 : \ldots : V_r] \). The reconstructed series \( \hat{X} \), its trajectory matrix \( \hat{X} \) and the reconstructed matrix \( \hat{X} \) are defined in Section 2.1. Define \( L^\text{col} = \text{span}(U_i, i \in I), L^\text{row} = \text{span}(V_i, i \in I) \). The reconstructed matrix \( \hat{X} = [\hat{X}_1 : \ldots : \hat{X}_K] \) consists of column vectors that are projections of column vectors of the trajectory matrix on the chosen subspace \( L^\text{col} \).

To avoid repeating the transpose sign, denote \( \tilde{Y} = [\tilde{Y}_1 : \ldots : \tilde{Y}_L] = \hat{X}^\top, \hat{Y} = [\hat{Y}_1 : \ldots : \hat{Y}_L] = \hat{X}^\top, \hat{Y}_k = \hat{X}_k^\top \).

**Recurrent MSSA forecast**

Denote by \( R_N = (\tilde{x}^{(1)}_{N_1+1}, \tilde{x}^{(2)}_{N_2+1}, \ldots, \tilde{x}^{(s)}_{N_s+1})^\top \) the vector of forecasted signal values for each time series. 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 one-term 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 \( Z \) the matrix consisting of the last \( L - 1 \) values of the reconstructed signals:

\[
Z = \begin{pmatrix}
\tilde{x}^{(1)}_{N_1-L+2} & \cdots & \tilde{x}^{(1)}_{N_1} \\
\tilde{x}^{(2)}_{N_2-L+2} & \cdots & \tilde{x}^{(2)}_{N_2} \\
\vdots & \ddots & \vdots \\
\tilde{x}^{(s)}_{N_s-L+2} & \cdots & \tilde{x}^{(s)}_{N_s}
\end{pmatrix},
\]

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

\[
R_N = Z \mathcal{R}_L, \quad \text{where} \quad \mathcal{R}_L = \frac{1}{1 - \nu^2} \sum_{j=1}^r \pi(U_j) U_j \in \mathbb{R}^{L-1}.
\]

Note that the formula (10) implies that the forecasting of all individual signals is made using the same linear recurrent formula which is generated by the whole system.

**Row forecast**

Introduce the vectors of the last \( K_m - 1 \) values of the reconstructed signals

\[
Z^{(m)} = (\tilde{x}^{(m)}_{N-K_m+2}, \ldots, \tilde{x}^{(m)}_{N_m})^\top, \quad m = 1, \ldots, s,
\]
and denote,

\[ Z = \begin{pmatrix} Z^{(1)} \\ Z^{(2)} \\ \vdots \\ Z^{(s)} \end{pmatrix} = \bar{Y}_L, \quad S = [\mu(V_1) : \ldots : \mu(V_r)]. \]

If the inverse matrix \((I_s - SS^T)^{-1}\) exists and \(r \leq K - s\), then the column MSSA recurrent forecast exists and can be calculated by the formula

\[ R_N = R_K Z, \quad (11) \]

where \(R_K = (I_s - SS^T)^{-1} SV^T\). Note that formula (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 of the series system.

**Vector MSSA forecasting**

Denote \(L^{\text{col}} = \text{span}(U_1, \ldots, U_r)\) and \(L^{\text{row}} = \text{span}(V_1, \ldots, V_r)\). Let \(\Pi^{\text{col}}\) be the orthogonal projector of \(R^{K-s}\) on \(L^{\text{col}}\) and \(\Pi^{\text{row}}\) be the orthogonal projector of \(R^{K-s}\) on \(L^{\text{row}}\).

Columns of \(\hat{Y}\) are projections of row vectors of the trajectory matrix \(X\) on \(L^{\text{row}}\), while columns of \(\hat{X}\) are the projections of column vectors of the trajectory matrix \(X\) on \(L^{\text{col}}\).

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 by this formula is time-consuming. The fast algorithm of calculation is presented in Section 6.3.

**Column forecast**

We mentioned that for a given subspace \((L^{\text{col}}\) in our case\) the column forecast is performed independently for each time series. Define the linear operator \(\psi^{\text{col}}_\text{Vec} : R^L \mapsto L^{\text{col}}\) by the formula

\[ \psi^{\text{col}}_\text{Vec} Z = \begin{pmatrix} \Pi^{\text{col}} Z \\ R_L Z \end{pmatrix}. \quad (12) \]

Let us formulate the vector forecasting algorithm for \(j\)-th series.

1. In the notation above, define the vectors \(Z_i\) as follows:

\[ Z_i = \begin{cases} \hat{X}^{(j)}_i & \text{for } i = 1, \ldots, K_j, \\ \psi^{\text{col}}_\text{Vec} \bar{Z}_{i-1} & \text{for } i = K_j + 1, \ldots, K_j + M + L - 1. \end{cases} \quad (13) \]

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

3. The numbers \(z_{N_j+1}, \ldots, z_{N_j+M}\) form the \(M\) terms of the vector forecast.
Row forecast  Define the linear operator \( P_{\text{row}} \) by the formula
\[
P_{\text{row}} \text{Vec} Z = A, \tag{14}\]
such that \( A = \Pi_{\text{row}} \vec{Z} \) and \( \mu(A) = \mathcal{R}_{K} \vec{Z} \).

Let us formulate the vector forecasting algorithm.

1. In the notation above, define the vectors \( Z_i \) as follows:
\[
Z_i = \begin{cases} 
\hat{Y}_i & \text{for } i = 1, \ldots, L, \\
P_{\text{row}} \text{Vec} Z_{i-1} & \text{for } i = L + 1, \ldots, L + M + K^* - 1,
\end{cases} \tag{15}\]
where \( K^* = \max(K_i, i = 1, \ldots, s) \).

2. By constructing the matrix \( Z = [Z_1 : \ldots : Z_{L+M+K^*-1}] \) and making Reconstruction step we obtain the series \( z_{N_j+M+K^*-1}^{(j)} \), \( j = 1, \ldots, s \).

3. The numbers \( z_{N_j+1}^{(j)}, \ldots, z_{N_j+M}^{(j)} \) form the \( M \) terms of the vector 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 specific 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 (2013), 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")]
> L <- 84
> s.wineFortDry <- ssa(wineFortDry, L = L, kind = "mssa")
> r.wineFortDry <- reconstruct(s.wineFortDry,
+   groups = list(Trend = c(1, 6),
+                 Seasonality = c(2:5, 7:12)))
> plot(r.wineFortDry, add.residuals = FALSE,
+      plot.method = "xyplot",
+      superpose = TRUE, auto.key = list(columns = 3))
```
Fragment 5 contains the 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 \texttt{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 \texttt{plot} function for the reconstruction object allows to indicate, what subset (what slice) of this matrix one wants to depict by means of the parameter \texttt{slice}. The parameter \texttt{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 Fragment 6 and Fragment 3. The difference consists in the structure of the factor vectors; however, they are not necessary for identification. Fig. 9 (cf. Fig. 3) shows that the trend is described by ET1 and ET6, which is slightly mixed with seasonality. Fig. 10 (cf. Fig. 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 multidimensional case in exactly the same way as in the one-dimensional case.

\textbf{Fragment 6} (FORT and DRY: identification).

\begin{verbatim}
> plot(s.wineFortDry, type = "vectors", idx = 1:8)
> plot(s.wineFortDry, type = "paired", idx = 2:11, plot.contrib = FALSE)
\end{verbatim}
> parestimate(s.wineFortDry, groups = list(2:3, 4:5), method = "esprit-ls")

$F1$

| period | rate | Mod  | Arg  | Re   | Im   |
|--------|------|------|------|------|------|
| 12.128 | -0.004789 | 0.99522 | 0.52 | 0.86463 | 0.49283 |
| -12.128 | -0.004789 | 0.99522 | -0.52 | 0.86463 | -0.49283 |

$F2$

| period | rate | Mod  | Arg  | Re   | Im   |
|--------|------|------|------|------|------|
| 4.007  | -0.001226 | 0.99877 | 1.57 | 0.00279 | 0.99877 |
| -4.007 | -0.001226 | 0.99877 | -1.57 | 0.00279 | -0.99877 |

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

### Eigenvectors

| 1 (95.14%) | 2 (0.97%) | 3 (0.95%) | 4 (0.48%) |
|------------|----------|-----------|-----------|
| ![Eigenvector 1](image1) | ![Eigenvector 2](image2) | ![Eigenvector 3](image3) | ![Eigenvector 4](image4) |
| 5 (0.47%) | 6 (0.32%) | 7 (0.21%) | 8 (0.21%) |
| ![Eigenvector 5](image5) | ![Eigenvector 6](image6) | ![Eigenvector 7](image7) | ![Eigenvector 8](image8) |

Figure 9: FORT and DRY: 1d graphs of eigenvectors

### Pairs of eigenvectors

| 2 vs 3 | 3 vs 4 | 4 vs 5 | 5 vs 6 | 6 vs 7 |
|--------|--------|--------|--------|--------|
| ![Pair 1](image9) | ![Pair 2](image10) | ![Pair 3](image11) | ![Pair 4](image12) | ![Pair 5](image13) |
| 7 vs 8 | 8 vs 9 | 9 vs 10 | 10 vs 11 | 11 vs 12 |
| ![Pair 6](image14) | ![Pair 7](image15) | ![Pair 8](image16) | ![Pair 9](image17) | ![Pair 10](image18) |

Figure 10: FORT and DRY: 2d scatterplots of eigenvectors

The code for forecasting is also very similar to that in SSA, compare Fragment 7 and Fragment 4.
Multivariate and 2D Extensions of SSA with \textit{Rssa}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{W-correlation_matrix}
\caption{FORT and DRY: weighted correlations}
\end{figure}

\textbf{Fragment 7} (FORT and DRY: forecast).

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

The results of MSSA analysis are similar to the results of SSA analysis. However, the separability is slightly worse. Therefore, the methods for improvement of separability can be very useful (Golyandina and Shlemov 2013). Note that the mixture of the signal components is not important for signal forecasting.

\textit{Comments}

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

\begin{itemize}
  \item Matrix with separate series in the columns. Optionally, additional time structure like in \texttt{mts} objects, can be embedded.
\end{itemize}
Matrix-like (e.g. a \texttt{data.frame}) object with series in the columns. In particular, \texttt{data.frame} would be a result of reading the series from the file via \texttt{read.table} function.

- List of separate time series objects (e.g. a \texttt{list} of \texttt{ts} or \texttt{zoo} objects).

Also, the time scales of the individual time series can be normalized via head or tail padding with \texttt{NA} (for example, as a result of the \texttt{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, \texttt{NA} padding, etc. is carefully preserved. For forecasted series, the time scale attributes for several known time series objects (e.g. \texttt{ts}) are inferred automatically where possible.

The examples in the Fragments 5, 14 provide an overview of the possible input series formats.

\textbf{Plotting specifics} Keep in mind that the default (\texttt{’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 \texttt{ts} and \texttt{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 object (see Fragment 8) in MSSA case coincides with the one for 1D-SSA with the slight specifics of MSSA.

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
Computed:
  Eigenvalues: 50,
  Eigenvectors: 50,
  Factor vectors: 0
Precached: 0 elementary series (0 MiB)
Overall memory consumption (estimate): 0.03734 MiB
```

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

**Efficient implementation**  All ideas from the one-dimensional case can be extended to the multivariate case. In 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 \( L \sim K \sim sN/(s+1) \), 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 stronger 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 consists of stacked vectors of length \( K_i, i = 1, \ldots, s \), 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, what can be checked by the `summary(s.wineFortDry)` function. Note that if to use `svd.method="svd"` (and `propack`), then factor vectors will be calculated automatically. To get the factor vector corresponding to 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 Fig. 13. The eigenvector in Fig. 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, compare with trends in Fig. 8 which are repeated in Fig. 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 $K_i = 176 - 163 + 1 = 12$ 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 et al. 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 Extended Empirical Orthogonal Functions (EEOFs).

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

```r
> library("lattice")
> L <- 24
> s.wineFortDrya <- ssa(wineFortDry, L = L, kind = "mssa")
> r.wineFortDrya <- reconstruct(s.wineFortDrya, groups = list(Trend = 1))
> 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"))
> 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))
> tp3 <- plot(s.wineFortDrya, type = "vectors", vectors = "eigen", idx = 1,
+ aspect = 1,
+ scales = list(x = list(draw = TRUE), y = list(draw = FALSE)))
> plot(tp3, split = c(1, 1, 1, 3), more = TRUE)
> plot(tp2, split = c(1, 2, 1, 3), more = TRUE)
> 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 decided by standardization of data. In Singular Spectrum Analysis, centering may not be an appropriate preprocessing. Therefore, two types of preprocessing can be applied, conventional standardization and normalization, that is, division on 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 Róse wine sales. Sales of Fortified wines are measured in thousands while that of Róse wines is measured in tens and hundred. Fragment 10 shows how the scale influences the reconstruction result.

**Fragment 10 (FORT and ROSE: influence of series scales).**
Figure 13: FORT and DRY: \( L = 24 \), relation between eigen- and factor vectors and reconstructed series

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

Fortified      Rose
Min. :1154   Min. : 30.00
1st Qu.:2372  1st Qu.: 66.00
Median :2898  Median : 87.00
Mean :3010    Mean : 93.01
3rd Qu.:3565  3rd Qu.:114.25
```

- **Fortified**
  - Min: 1154
  - 1st Qu: 2372
  - Median: 2898
  - Mean: 3010
  - 3rd Qu: 3565

- **Rose**
  - Min: 30.00
  - 1st Qu: 66.00
  - Median: 87.00
  - Mean: 93.01
  - 3rd Qu: 114.25
Max. : 5618  Max. : 267.00
> norm.wineFortRosen <- sqrt(colMeans(wineFortRose^2))
> wineFortRosen <- sweep(wineFortRose, 2, norm.wineFortRosen, "/")
> L <- 84
> s.wineFortRosen <- ssa(wineFortRosen, L = L, kind = "mssa")
> r.wineFortRosen <- reconstruct(s.wineFortRosen,
+   groups = list(Trend = c(1, 12, 14),
+                  Seasonality = c(2:11, 13)))
> s.wineFortRose <- ssa(wineFortRose, L = L, kind = "mssa")
> r.wineFortRose <- reconstruct(s.wineFortRose,
+   groups = list(Trend = c(1),
+                  Seasonality = 2:11))
> wrap.plot <- function(rec, component = 1, series, xlab = "," "ylab, ...
+   plot(rec, add.residuals = FALSE, add.original = TRUE,
+   +   plot.method = "xyplot", superpose = TRUE,
+   +   scales = list(y = list(tick.number = 3)),
+   +   slice = list(component = component, series = series),
+   +   xlab = xlab, ylab = ylab, auto.key = "", ...
+   trel1 <- wrap.plot(r.wineFortRosen, series = 2, ylab = "Rose, norm")
> trel2 <- wrap.plot(r.wineFortRosen, series = 1, ylab = "Fort, norm")
> trel3 <- wrap.plot(r.wineFortRose, series = 2, ylab = "Rose")
> trel4 <- wrap.plot(r.wineFortRose, series = 1, ylab = "Fort")
> plot(trel1, split = c(1, 1, 2, 2), more = TRUE)
> plot(trel2, split = c(1, 2, 2, 2), more = TRUE)
> plot(trel3, split = c(2, 1, 2, 2), more = TRUE)
> plot(trel4, split = c(2, 2, 2, 2), more = TRUE)

Fig. 14 demonstrates the result of a trend reconstruction, where the trend was detected by 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 AustralianWine data are incomplete, there are no data for two months (point 175 and 176) for sales of Rose 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 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 separate analysis of the ROSE series and the SSA methods of filling in missing data, but Fig. 15 shows that the result of the used method is quite accurate.
Figure 14: FORT and ROSE: trends with normalization (ET1,12,14) and without (ET1)

Fragment 11 (FORT and ROSE: forecast the missing data in ROSE).

```r
> f.wineFortRosen <- rforecast(s.wineFortRosen, groups = list(1:14),
+   len = 13, only.new = TRUE)[, "Rose"]
> f.wineFortRosen_long <- c(rep(NA, 174),
+   norm.wineFortRosen["Rose"] * f.wineFortRosen)
> 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. Fig. 16 clearly demonstrates that for small contribution of MAINSALES the simultaneous forecast is close to separate forecast of Total, while for large weight of MAINSALES the simultaneous forecast of TOTAL tend 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
> FilledRoseAustralianWine[175:176, "Rose"] <- f.wineFortRosen_long[175:176]
```
Figure 15: FORT and ROSE: Forecast of ROSE in comparison with the existing data

```r
> mainsales <- ts(rowSums(FilledRoseAustralianWine[, -1]))
> # Sum of sales of main wines
> total <- FilledRoseAustralianWine[, "Total"]
> L = 84
> s.totalmain1 <- ssa(list(mainsales[12:187], total[1:176]),
+   L = L, kind = "mssa")
> f.totalmain1 <- rforecast(s.totalmain1, groups = list(1:14),
+   len = 11, only.new = TRUE)
> s.totalmain2 <- ssa(list(100 * mainsales[12:187], total[1:176]),
+   L = L, kind = "mssa")
> f.totalmain2 <- rforecast(s.totalmain2, groups = list(1:14),
+   len = 11, only.new = TRUE)
> s.total <- ssa(total[1:176], L = L, kind = "1d-ssa")
> f.total <- rforecast(s.total, groups = list(1:14),
+   len = 11, only.new = TRUE)
> xtime <- time(AustralianWine)[177:187]
> xtime.labels <- paste(month.abb[round(xtime * 12) %% 12 + 1],
+   floor(xtime), sep = ", ")
> 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")),
```

Multivariate and 2D Extensions of SSA with \texttt{Rssa}

\begin{verbatim}
+ lines = TRUE, points = FALSE))
\end{verbatim}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure16}
\caption{TOTAL: Forecast of total sales, separate and with the available information}
\end{figure}

**Simultaneous decomposition of many series**

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

Let us consider the collection of $s = 6$ series from the ‘AustralianWine’ dataset, which includes 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 ‘Rôse wines’ sales have the order of 100 thousand of litres in month, while the sales of ‘Fortified wines’ are about 30 times larger, the time series should be transformed into the same scale. We choose the window length $L = 163$, then $K_i = 12$ and $K = 84$. This choice does not correspond to the maximal rank of the trajectory matrix; however, this helps to avoid the mixture of components.

The identification of the trend (ET1,2,5) and the seasonality (ET3,4, 6–12) is performed on the base 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 Fig. 17 and 18.

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

\begin{verbatim}
> L <- 163
> norm.wine <- sqrt(colMeans(wine[, -1]^2))
> winen <- sweep(wine[, -1], 2, norm.wine, "/")
\end{verbatim}
> s.winen <- ssa(winen, L = L, kind = "mssa")
> r.winen <- reconstruct(s.winen,
+   groups = list(Trend = c(1, 2, 5),
+                  Seasonality = c(3:4, 6:12)))
> plot(r.winen, add.residuals = FALSE,
+      plot.method = "xyplot",
+      slice = list(component = 1), screens = list(colnames(winen)),
+      col = c("blue", "green", "red", "violet", "black", "green4"),
+      lty = rep(c(1, 2), each = 6),
+      scales = list(y = list(draw = FALSE)),
+      layout = c(1, 6))
> plot(r.winen, plot.method = "xyplot", add.original = FALSE,
+      add.residuals = FALSE, slice = list(component = 2),
+      col = c("blue", "green", "red", "violet", "black", "green4"),
+      scales = list(y = list(draw = FALSE)),
+      layout = c(1, 6))

Reconstructed Series

Figure 17: Wine sales: extraction of trends
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 Fig. 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, the comparison results explain the choice of default forecasting method.

In the study below, we consider the case \( s = 2 \) and hence include CSSA into 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 \( (X^{(1)}, X^{(2)}) = (H^{(1)}, H^{(2)}) + (N^{(1)}, N^{(2)}) \), where \((H^{(1)}, H^{(2)})\) is a two-dimensional signal consisting of two harmonic time series, \(N^{(1)}\) and \(N^{(2)}\) 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 \((H^{(1)}, H^{(2)})\) by the
considered above SSA methods. Note that the resultant MSE is calculated as the mean of $\text{MSE}^{(1)}$ and $\text{MSE}^{(2)}$ for $\mathbb{H}^{(1)}$ and $\mathbb{H}^{(2)}$ correspondingly.

We take the following parameters for the simulation of the time series: $N = 71$, the variance of all noise components is $\sigma^2 = 25$, the number of replications is 10000. We consider the following three versions of the signal ($\mathbb{H}^{(1)}, \mathbb{H}^{(2)}$).

**Example A** (the same periods, the difference between the phases is 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 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.

| Table 2: Dimension of the signal trajectory space |
|--------------------------------------------------|
|         | Example A | Example B | Example C |
| SSA     | 2         | 2         | 2         |
| MSSA    | 2         | 2         | 4         |
| CSSA    | 2         | 1         | 4         |

The results of investigation for different window lengths $L$ are summarized in Tables 3 and 4. The smallest value in each row is marked by typing it in bold. The 24 term-ahead forecast was performed.

| Table 3: MSE of signal reconstruction |
|--------------------------------------|
| Method (Ex.) | $L = 12$ | $L = 24$ | $L = 36$ | $L = 48$ | $L = 60$ |
| SSA (Ex.A,B,C) | 3.22  | **2.00** | 2.00  | 2.00  | 3.22  |
| MSSA (Ex.A,B) | 3.17  | 1.82   | 1.58  | **1.46** | 1.97  |
| MSSA (Ex.C) | 6.90  | 3.77   | 3.07  | **2.87** | 3.84  |
| CSSA (Ex.A) | 3.23  | **2.01** | **2.01** | **2.01** | 3.23  |
| CSSA (Ex.B) | 1.57  | **1.00** | **1.00** | **1.00** | 1.57  |
| CSSA (Ex.C) | 6.97  | 4.05   | **3.81** | 4.05  | 6.97  |

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.
Table 4: MSE of signal forecast

| Example A | $L = 12$ | $L = 24$ | $L = 36$ | $L = 48$ | $L = 60$ |
|-----------|---------|---------|---------|---------|---------|
| **Recurrent** |         |         |         |         |         |
| MSSA-column | 5.34    | 3.60    | 3.64    | 3.66    | 4.38    |
| MSSA-row   | 6.01    | 4.20    | 3.75    | **3.27** | 3.93    |
| SSA        | 7.18    | 5.55    | 6.23    | 6.04    | 8.00    |
| CSSA       | 7.40    | 5.53    | 6.22    | 6.41    | 7.80    |
| **Vector** |         |         |         |         |         |
| MSSA-column | 5.94    | 3.72    | 3.56    | **3.08** | 3.64    |
| MSSA-row   | 3.98    | **2.98** | 3.24    | 3.12    | 4.20    |
| SSA        | 7.62    | 5.37    | 5.82    | **5.12** | 6.53    |
| CSSA       | 7.83    | 5.42    | 5.89    | **5.14** | 7.02    |
| Example C  |         |         |         |         |         |
| **Recurrent** |         |         |         |         |         |
| MSSA-column | 25.42   | **7.36** | 7.47    | 7.41    | 9.02    |
| MSSA-row   | 19.76   | 8.43    | 7.93    | **6.54** | 8.28    |
| SSA        | 7.18    | 5.55    | 6.23    | 6.04    | 8.00    |
| CSSA       | 38.29   | **11.19** | 13.35   | 13.08   | 25.03   |
| **Vector** |         |         |         |         |         |
| MSSA-column | 25.02   | 7.50    | 7.50    | **6.15** | 7.64    |
| MSSA-row   | 57.15   | **6.01** | 6.96    | 6.25    | 8.73    |
| SSA        | 7.62    | 5.37    | 5.82    | **5.12** | 6.53    |
| CSSA       | 38.43   | 10.80   | 13.45   | **10.19** | 70.17   |

| Example B  | $L = 12$ | $L = 24$ | $L = 36$ | $L = 48$ | $L = 60$ |
|------------|---------|---------|---------|---------|---------|
| CSSA recurrent | 3.45    | **2.78** | 3.15    | 3.13    | 3.99    |
| CSSA vector  | 3.86    | 2.69    | 2.93    | **2.55** | 3.23    |

space. For each example, the cells corresponding to the method with the best reconstruction accuracy are shown in bold and the overall minimum is in blue color.

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 $L \approx N/2$. 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 $L \approx sN/(s + 1)$. 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 $N/2$. 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 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 if to compare accuracies for the same window length. Note that this is probably valid for forecasting of well-separated signal of finite rank only, see Remark 1 for 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 surprised, since $L$ and $K$ are swapped.

The next fragment demonstrates how the numbers from the tables can be obtain by means of \texttt{Rssa} for MSSA and CSSA analysis and vector forecasting applied to Example A.

\textbf{Fragment 14} (Simulation for reconstruction and forecasting accuracy estimation).

```r
> N <- 71
> sigma <- 5
> Ls <- c(12, 24, 36, 48, 60)
> len <- 24
> signal1 <- 30 * cos(2*pi * (1:(N + len)) / 12)
> signal2 <- 30 * cos(2*pi * (1:(N + len)) / 12 + pi / 4)
> signal <- cbind(signal1, signal2)
> R <- 10  # 10000 in simulations for the paper
> 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)
  +}
```

Nina Golyandina, Anton Korobeynikov, Alex Shlemov, Konstantin Usevich
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 \(X\) 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 \(X = X_{N_x,N_y} = (x_{ij})_{i,j=1}^{N_x,N_y}\). 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 Golyandina and Usevich (2010); Rodríguez-Aragón and Zhigljavsky (2010). However, its first usage can traced back to the work of Ade (1983) on texture 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 base 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.

4.1. 2D-SSA algorithm

For a matrix \(A \in \mathbb{R}^{M \times N}\) (or \(\mathbb{C}^{M \times N}\)), we denote by \(\text{vec}(A) \in \mathbb{R}^{MN}\) (or \(\mathbb{C}^{MN}\)) its column-major vectorization. For a vector \(A \in \mathbb{R}^{MN}\) (or \(\mathbb{C}^{MN}\)), we define its \(M\)-devectorization as the matrix \(\text{vec}^{-1}_M(A) \defeq A \in \mathbb{R}^{M \times N}\) (or \(\mathbb{C}^{M \times N}\)) that satisfies \(\text{vec}(A) = A\). We mostly use the notation from the paper Golyandina and Usevich (2010).

The embedding operator

Since the general scheme of the 2D-SSA algorithm is described in Section 2, we need to describe only the embedding operator \(T_{2D-SSA}(X) = X\).

The parameters of the method are the two-dimensional window sizes \((L_x, L_y)\), which are bounded as \(1 \leq L_x \leq N_x\), \(1 \leq L_y \leq N_y\) and \(1 < L_x L_y < N_x N_y\). For convenience, we also denote \(K_x = N_x - L_x + 1\), \(K_y = N_y - L_y + 1\). As in the general scheme of the algorithms, we define \(L = L_x L_y\) (the number of rows of the trajectory matrix) and \(K = K_x K_y\) (the number of columns of the trajectory matrix).

Consider all possible \(L_x \times L_y\) submatrices of \(X\) (2D sliding windows). For \(k = 1, \ldots, K_x\) and \(l = 1, \ldots, K_y\), we define by \(X_{(L_x,L_y)}^{(k,l)} = (x_{ij})_{i=k-l, j=l}^{L_x+k-1, L_y+l-1}\) a \(L_x \times L_y\) submatrix, which is shown in Fig. 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

\[
T_{2D-SSA}(X) := X = [X_1 : \ldots : X_{K_x K_y}],
\]

where the columns

\[
X_{k+(l-1)K_x} = \text{vec}(X_{(L_x,L_y)}^{(k,l)})
\]

are vectorizations of \(L_x \times L_y\) submatrices.

Hankel-block-Hankel structure

It can be shown (see Golyandina and Usevich (2010)) that the trajectory matrix (16) has the
following structure:

$$
X = T_{2D-SSA}(X) = \begin{pmatrix}
H_1 & H_2 & H_3 & \cdots & H_{K_y} \\
H_2 & H_3 & H_4 & \cdots & H_{K_y+1} \\
& H_3 & H_4 & \cdots & \vdots \\
& & \ddots & \ddots & \vdots \\
& & & H_{L_y} & H_{L_y+1} & \cdots & \cdots & H_{N_y}
\end{pmatrix}, \quad (17)
$$

where each block $H_j$ is an $L_x \times L_y$ Hankel matrix constructed from the $X_{.,j}$ (the $j$-th column of the 2D-array $X$). More precisely, $H_j = T_{SSA}(X_{.,j})$, where $T_{SSA}$ is defined in (2). Therefore, in the literature the matrix (16) is called Hankel-block-Hankel (or HbH for short) because 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 $X$. Therefore, the eigenvectors $U_i$ also can be viewed as vectorized $L_x \times L_y$ arrays. These devectorizations are denoted by $\Psi_i \overset{def}{=} \text{vec}_{L_x}^{-1}(U_i)$.

Similarly, the rows of $X$ are vectorizations of the $(K_x, K_y)$ submatrices

$$
X = [X^1 : \ldots : X^{L_x L_y}]^\top, X^{k+(l-1)L_x} = \text{vec}(X_{k,l})^{(K_x,K_y)}, \quad (18)
$$

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

**Comments**

1. The algorithm of 2D-SSA coincides with the algorithm of MSSA if to choose the window sizes $(L_x, 1)$ or $(1, L_y)$. 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 $T_{2D-SSA}$ has finite rank) are sums of products of polynomials, exponentials and cosines, similarly to the one-dimensional case. More details can be found in Section 7.2.
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 2D case.

As an example, we use the image of Mars by Pierre Thierry, from the tutorial\(^1\) of the free IRIS software (see the data description in the package). The image is of size 258 × 275, 8-bit grayscale, values from 0 to 255. The input code for this image can be found in Fragment 15 (the image is included in the package *Rssa*).

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

```
> library("Rssa")
> data(Mars)
```

We would like to decompose this image with a 25 × 25 window (the example in the paper 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 comment 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"`).

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

A remedy for this could be calculation of just the matrix \(XX^\top\) (covariation matrix), 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"`).

```
> print(system.time(ssa(Mars, kind = "2d-ssa", L = c(25, 25),
+  svd.method = "eigen")))
    user  system elapsed
   4.089   0.070   4.052
```

However, for larger window sizes this approach quickly becomes impractical, because the complexity of the full eigendecomposition grows at least as \(O(L^3 + KL^2)\). Therefore, in the *Rssa* package the method `nutrlan` is used by default. This gives a considerable speedup 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).

\(^1\)http://www.astrosurf.com/buil/iris/tutorial8/doc23_us.htm
Multivariate and 2D Extensions of SSA with Rssa

> print(system.time(s.Mars.25 <- ssa(Mars, kind = "2d-ssa", L = c(25, 25))))
user system elapsed
0.522 0.013 0.542

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

Fragment 19 (Mars: reconstruction).

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

The reconstruction results are shown in Fig. 20.

| Reconstructions |
|-----------------|
| Original [0, 260] | Noise [−8.6, 8.6] | Residuals [−3.8, 260] |

Figure 20: Mars: separated periodic noise, \((L_x, L_y) = (25, 25)\)

The grouping for this decomposition was made based on the following information:

1. eigenarrays \(\Psi_i\) (see Fig. 21),
2. the matrix of \(w\)-correlations (see Fig. 22).

Fragment 20 shows the corresponding code.

Fragment 20 (Mars: identification).

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

Next, we try more challenging window sizes \((L_x, L_y) = (80, 160)\), where the trajectory matrix is of size 12800 \(\times\) 19404. In this case, \svd.method = "eigen" would take very long time. However, with the default method \svd.method = "nutrlan" the computation of 50 first eigenvectors can be done in about a second, see Fragment 21. The results of the reconstruction are shown in Fig. 23.
From Fig. 23 we see that in the case of large window sizes, the extracted periodic noise is not modulated (compare to Fig. 20). This can be interpreted as follows. We chose the window sizes as (80, 160) (approximately $(0.4N_x, 0.6N_y)$) for which the separability of signal and noise in the parametric model should be the best (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 useRaster = TRUE argument provided to the lattice plotting functions). It 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 $L = L_xL_y$ and $K = K_xK_y$.) The ideas presented in Section 6.2 coupled with Lanczos-based truncated SVD implementations (Larsen 1998; Yamazaki et al. 2008; Korobeynikov 2010) allows to dramatically reduce the computational complexity down to $O(kN \log N + k^2N)$, where $N = N_xN_y$ and $k$ denotes the number of desired eigentriples. Therefore, the achieved speed-up can be much higher than that in the one-dimensional (SSA) and multivariate (MSSA) cases. Also, for svd.method="eigen", the matrix $XX^\top$ is computed using the fast matrix-vector multiplication from Section 6.2. In this case, the complexity is reduced from $O(KL^2)$ to $O(LN \log N)$. 

Reconstructions

| Original [0, 260] | Noise [−7.8, 7.8] | Residuals [−5.7, 260] |
|-----------------|-----------------|-------------------|

Figure 23: Mars: reconstruction, $(L_x, L_y) = (160, 80)$
4.3. Examples

Adaptive smoothing

2D-SSA can be also used for adaptive smoothing of two-dimensional data. It was used for this purpose in Golyandina et al. (2007) for smoothing digital terrain models (DTM) and in Holloway et al. (2011); Golyandina et al. (2012) for smoothing spatial gene expression data. Similarly to the example in Golyandina et al. (2007), we consider an image extracted from SRTM database. The test DTM of a region in South Wales is extracted by the function `getData` of the package `raster`. The DTM is $80 \times 100$, and it includes the Brecon Beacons national park, Wales, UK. The point $(N_x, 1)$ lies in a neighbourhood of Port Talbot, $(N_x, N_y)$ — in a neighbourhood 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 Fig. 24.

Fragment 22 (Brecon Beacons: decomposition).

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

![Eigenvectors](image1)

![W−correlation matrix](image2)

Figure 24: Brecon Beacons: $8 \times 8$ windows, eigenarrays and w-correlations

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 \( \tilde{Y}_1 = \tilde{X}_1, \tilde{Y}_1 = \tilde{X}_1 + \tilde{X}_2, \) and \( \tilde{Y}_1 = \tilde{X}_1 + \tilde{X}_2 + \tilde{X}_3 \) (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 Fig. 25. The cumulative components \( \tilde{Y}_k \) are shown in Fig. 26 using the parameter `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))
> plot(r.brecon, cuts = 255, layout = c(5, 1),
+     par.strip.text = list(cex = 0.75))
> plot(r.brecon, cuts = 255, layout = c(5, 1),
+     par.strip.text = list(cex = 0.75), type = "cumsum", at = "free")
```

| Reconstructed Components |
|--------------------------|
| Original [6, 780]       |
| F1 [21, 660]            |
| F2 [-110, 150]          |
| F3 [-95, 100]           |
| Residuals [-96, 110]    |

**Figure 25:** Brecon Beacons: 8 × 8 window, reconstructions \( (\tilde{X}_k) \)

| Cumulative Reconstructions |
|---------------------------|
| Original [6, 780]       |
| F1 [21, 660]            |
| F1:F2 [-0.021, 710]     |
| F1:F3 [-15, 730]        |
| Residuals [-96, 110]    |

**Figure 26:** Brecon Beacons: 8 × 8 window, cumulative reconstructions \( (\tilde{Y}_k) \)

From Fig. 25 and Fig. 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 behaviour of smoothing, we plot the absolute values of the centered DFTs of \( X - \tilde{Y}_k \) (residuals for cumulative reconstructions, see Fig. 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 = ",")
+ }
> 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).

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

In Fig. 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).

![Figure 27: Brecon Beacons: 8 × 8 window, absolute values of the DFT of \( X - \tilde{Y}_k \), \( k = 1, \ldots, 3 \)](image)

Parameter estimation

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

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 et al. (2005)).

We continue the example of Mars from Fragment 21. This example demonstrates advantages of Rssa because of a possibility to choose large window sizes, which was always considered as a drawback of ESPRIT-type methods. Fragment 26 shows the corresponding code that outputs the estimated exponentials. In Fig. 28, the estimated pairs complex exponentials \((\lambda_k, \mu_k)\), are shown on separate complex plane plots and for the same \(k\) the points for \(\lambda_k\) and \(\mu_k\) have the same color, and correspondence between them is made using different marker type.

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

```r
> pe.Mars.160.80 <- parestimate(s.Mars.160.80, groups = r.Mars.160.80.groups)
> # Overview
> print(pe.Mars.160.80)
x: period rate | y: period rate
-5.000 -0.000169 | 10.003 -0.000111
 5.000 -0.000169 | -10.003 -0.000111
 9.995 0.000175 | 4.999 -0.000093
-9.995 0.000175 | -4.999 -0.000093
> # Detailed output
> print(pe.Mars.160.80[[1]])
period rate | Mod Arg | Re Im
-5.000 -0.000169 | 0.99983 -1.26 | 0.30906 -0.95087
 5.000 -0.000169 | 0.99983 1.26 | 0.30906 0.95087
 9.995 0.000175 | 1.00017 0.63 | 0.80897 0.58814
-9.995 0.000175 | 1.00017 -0.63 | 0.80897 -0.58814
> print(pe.Mars.160.80[[2]])
period rate | Mod Arg | Re Im
10.003 -0.000111 | 0.99989 0.63 | 0.80905 0.58755
-10.003 -0.000111 | 0.99989 -0.63 | 0.80905 -0.58755
 4.999 -0.000093 | 0.99991 1.26 | 0.30879 0.95103
-4.999 -0.000093 | 0.99991 -1.26 | 0.30879 -0.95103
> plot(pe.Mars.160.80, col = c(11, 12, 13, 14))
> plot(s.Mars.160.80, type = "vectors", idx = r.Mars.160.80.groups$Noise,
+     cuts = 255, layout = c(4, 1), plot.contrib = FALSE)
```

In Fragment 26 and Fig. 28, it can be seen that each pair \((\lambda_k, \mu_k)\) has its conjugate counterpart \((\lambda'_k, \mu'_k) = \text{conj}((\lambda_k, \mu_k))\) (where \(\text{conj}(\cdot)\) denotes the complex conjugation). Indeed, it is the case for 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 Section 7.2. This is also confirmed by the plots of eigenarrays in Fig. 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. In other words, not all the values of the rectangular image may be specified, and the sliding window needs not be rectangular (see Fig. 30).

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

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

\[
\ell + \kappa = (\ell_x + \kappa_x - 1, \ell_y + \kappa_y - 1).
\]

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

\[
\mathfrak{A} + \mathfrak{B} = \{ \alpha + \beta | \alpha \in \mathfrak{A}, \beta \in \mathfrak{B} \}.
\]

5.1. Construction of the trajectory matrix

The input data for the algorithm is an \( \mathfrak{R} \)-shaped array \( \mathfrak{X} = \mathfrak{X}_\mathfrak{R} = (x_{\alpha})_{\alpha \in \mathfrak{R}} \), where \( \mathfrak{R} \subseteq \{1, \ldots, N_x\} \times \{1, \ldots, N_y\} \). 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 \( \ell_i \in \mathbb{N}^2 \) are ordered in lexicographical order.
Multivariate and 2D Extensions of SSA with \textbf{Rssa}

The embedding operator

For $\kappa \in \mathbb{N}^2$ we define a shifted $\mathcal{L}$-shaped subarray as $X_{\mathcal{L}+\{\kappa\}} = (x_\alpha)_{\alpha \in \mathcal{L}+\{\kappa\}}$ (see Fig. 30). The index $\kappa$ is a position of the origin for the window. Consider the set of all possible origin positions for $\mathcal{L}$-shaped windows:

$$\mathcal{K} = \{ \kappa \in \mathbb{N}^2 \mid \mathcal{L} + \{ \kappa \} \subseteq \mathcal{N} \}. \quad (19)$$

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

$$T_{\text{ShSSA}}(X) := X = [X_1 : \ldots : X_K], \quad (20)$$

where the columns

$$X_j = (x_{\ell_1 + \kappa_j})_{\ell=1}$$

are vectorizations of the shaped submatrices $X_{\mathcal{L}+\{\kappa_j\}}$.

Quasi-Hankel matrix

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

$$X = T_{\text{ShSSA}}(X) = \begin{pmatrix} x_{\ell_1 + \kappa_1} & x_{\ell_1 + \kappa_2} & \cdots & x_{\ell_1 + \kappa_K} \\ x_{\ell_2 + \kappa_1} & x_{\ell_2 + \kappa_2} & \cdots & x_{\ell_2 + \kappa_K} \\ \vdots & \vdots & \ddots & \vdots \\ x_{\ell_L + \kappa_1} & x_{\ell_L + \kappa_2} & \cdots & x_{\ell_L + \kappa_K} \end{pmatrix}. \quad (21)$$

Note that in Mourrain and Pan (2000) instead of the shifted sum $+$, the usual sum of indices is used, because their definition of natural numbers $\mathbb{N}$ includes 0.

Comments

If $\mathcal{K} = \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$ and $\mathcal{L} = \{1, \ldots, L_x\} \times \{1, \ldots, L_y\}$ (therefore, $\mathcal{R} = \{1, \ldots, L_x\} \times \{1, \ldots, K_y\}$), then the matrix $T_{\text{ShSSA}}(X)$ coincides with $T_{\text{2D-SSA}}(X)$. Therefore, the ShSSA analysis with the rectangular window and input array coincides with 2D-SSA analysis.
In Fig. 31 we demonstrate the case of general shapes. The set $\mathcal{K}$ corresponds to all possible positions of the left upper corner of the bounding box around the shaped window (depicted by green dots in Fig. 31). It also corresponds to the maximal set $\mathcal{K}$ such that $\mathcal{L} \oplus \mathcal{K} \subseteq \mathcal{N}$. We describe an algorithm for computation of the set $\mathcal{K}$ in Section 6.2.4.

It may happen that some of the elements of the original shaped array $X_N$ do not enter in the trajectory matrix (for example, the element $5, 8$, shown in gray in Fig. 31). In this case, the ShSSA analysis applies only to the $\mathcal{N}'$-shaped array $X_{N'}$, where $\mathcal{N}'$ is the set of all indices that enter in the trajectory matrix. (In fact, $\mathcal{N}' = \mathcal{K} \oplus \mathcal{L}$.)

**Column and row spaces**

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

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

**5.2. Package**

**Typical code**

In this subsection we repeat the experiment from Section 4.2 (periodic noise removal from the image of Mars), but using the Shaped 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:

(A) by passing the NAs values in the input array (these elements are excluded from the shape),

(B) by specifying the parameter mask — a logical $N_x \times N_y$ array (the indicator of $\mathcal{N}$).

If both shape specifications are present, we consider their intersection. The shape of the window is also 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)
> mask.Mars.1 <- (Mars != 255)
```
> Mars[mask.Mars.0] <- NA
> print(system.time(s.Mars.shaped <-
+ ssa(Mars, kind = "2d-ssa",
+ mask = mask.Mars.1, wmask = circle(15))))

user system elapsed
0.730 0.021 0.755

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

In Fig. 32 one can see both types of masks and the combined mask. Fragment 28 shows a

![Mars masks](image)

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

typical reconstruction code for Shaped SSA.

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

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

The reconstruction results are shown in Fig. 33. In Fig. 33 you 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:
(A) eigenarrays (see Fig. 34),
(B) the matrix of matrix of w-correlations (see Fig. 35).

Fragment 29 shows the corresponding code.

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

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

*Comments*
Reconstructions

| Original [1, 260] | Noise [−9.5, 8.8] | Residuals [−6.7, 250] |
|-------------------|------------------|---------------------|
| ![Reconstruction](image1.png) | ![Noise](image2.png) | ![Residuals](image3.png) |

Figure 33: Mars: reconstruction

Eigenvectors

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|----|
| ![Eigenvector 1](image4.png) | ![Eigenvector 2](image5.png) | ![Eigenvector 3](image6.png) | ![Eigenvector 4](image7.png) | ![Eigenvector 5](image8.png) | ![Eigenvector 6](image9.png) | ![Eigenvector 7](image10.png) | ![Eigenvector 8](image11.png) | ![Eigenvector 9](image12.png) | ![Eigenvector 10](image13.png) |

Figure 34: Mars: eigenarrays, ShSSA

**Efficient implementation** In Rssa the Shaped 2D-SSA is implemented as an additional option for `kind = "2d-ssa"`. 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 seen as a special case of Shaped 2D SSA with rectangular window and the mask which 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. So, 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 deduced from the images.

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

- `wmask = circle(R)` specifies circular mask of radius R.
5.3. Special cases of Shaped 2D-SSA

In this section we will 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). Also, in Golyandina and Usevich (2010) it was mentioned that MSSA with equal time series lengths is a special case of 2D-SSA. In the next subsection we extend this construction to time series with different lengths.

**MSSA: 2D packing**

Consider a multivariate time series, that is, a collection \( \{ \mathbf{X}(p) = (x^{(p)}_j)_{j=1}^{N_p}, \ p = 1, \ldots, s \} \) of \( s \) time series of length \( N_p, p = 1, \ldots, s \). We construct the shaped array \( \mathbf{X}_\mathfrak{R} \) such that

\[
\mathfrak{R} = \{1, \ldots, N_1\} \times \{1\} \cup \{1, \ldots, N_s\} \times \{s\} \subset \{1, \ldots, \max_j N_j\} \times \{1, \ldots, s\}.
\]

The window is taken to be \( \mathfrak{L} = \{1, \ldots, L\} \times \{1\} \). The construction of this array is shown on Fig. 36, also with the window \( \mathfrak{L} \) and the shape \( \mathfrak{R} \). It is easy to verify that \( T_{\text{ShSSA}}(\mathbf{X}) = T_{\text{MSSA}}(\mathbf{X}) \).

Consider also construction of the rows of \( \mathbf{X} = T_{\text{MSSA}}(\mathbf{X}) \). The rows of \( \mathbf{X} \) are vectorizations of the \( \mathfrak{R} \)-shaped subarrays (see Fig. 36).
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 \( X \), where \( N' = N + (s - 1) \) (recall that \( N = \sum_{p=1}^{s} N_p \)). 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 Fig. 37 we show the horizontal variant of packing.

Mosaic Hankel matrices

Mosaic Hankel matrix (Heinig 1995) is a block matrix with Hankel blocks. It can be considered as the most general one-dimensional (with one-dimensional displacement) generalization of Hankel matrices.

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

\[
\begin{pmatrix}
H_{L_1,K_1}(X^{(1,1)}) & \cdots & H_{L_1,K_t}(X^{(1,t)}) \\
\vdots & \ddots & \vdots \\
H_{L_s,K_1}(X^{(s,1)}) & \cdots & H_{L_s,K_t}(X^{(s,t)})
\end{pmatrix}
\]

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 \( j \)-th block column is a transposed matrix \( T_{MSSA} \) for the collection of time series \((X^{(1,j)}, \ldots, X^{(s,j)})\) and window length \( L = K_j \). Therefore, the mosaic Hankel matrix can be constructed by
Multivariate and 2D Extensions of SSA with Rssa

stacking shapes (with separators) from Fig. 36 (replacing $R$ by $L$) because of transposition. The resulting construction is shown in Fig. 38.

\[ L_X(1,1) \quad L_X(s,1) \quad L_X(1,2) \quad L_X(s,2) \quad L_X(1,t) \quad L_X(s,t) \]

Figure 38: Shaped construction for mosaic Hankel matrices

\textit{M-2D-SSA}

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

\[ T_{M-2D-SSA}(X^{(1)}, \ldots, X^{(s)}) = \left[ T_{2D-SSA}(X^{(1)}) : \cdots : T_{2D-SSA}(X^{(s)}) \right]. \]

In this case, in the SSA decomposition will have the common basis of eigenvectors, as in 2D-SSA. This type of matrices is used for methods of comparison of images using 2D-SSA (Rodríguez-Aragón and Zhigljavsky 2010). This is also the same matrix that is used in recent methods of parallel MRI imaging (Uecker et al. 2013).

The matrix for the M-2D-SSA can be constructed in a similar way to the case of mosaic Hankel matrices. An array $X'$ of size $N_x \times (sN_y + s - 1)$ is constructed from the arrays with one-element separators. The resulting array is shown in Fig. 39. In general, a similar

\[ L_x \]

Figure 39: Shaped construction for M-2D-SSA

construction can handle arrays of different sizes, shapes, and shaped windows. Also note that in the extended array the original arrays (both for Fig. 39 and Fig. 38) may be arranged arbitrarily (for example in a table-like planar arrangement). The only requirement is that they should be separated.
6. Implementation

6.1. Fast computations with Hankel matrices

In this section, we describe the algorithms that are used for multiplication of Hankel matrices by vectors and rank-one Hankelization. Although algorithms for these operations were already proposed in Korobeynikov (2010), we use another type of circulants which simplifies the algorithms compared to Korobeynikov (2010). This section is also 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 $A, B \in \mathbb{C}^N$, $\text{rev}(A)$ denotes the reversion of the vector $A$, and $\text{conj}(A)$ denotes the elementwise conjugation. We also denote by $E_j^{(N)}$ the $j$-th unit vector in $\mathbb{R}^N$.

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

$$F_N(X) \overset{df}{=} F_N X,$$

where $F_N \in \mathbb{C}^{N \times N}$ is the Fourier matrix with elements $F_N[k, l] = e^{-2\pi i (k-1)(l-1)/N}$ (see Korobeynikov (2010)). The inverse Fourier transform is given by

$$F_{-1}^N(X) = \frac{1}{N} F_N^* X,$$

where $F_N^*$ is the Hermitian transpose of $F_N$.

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

$$C_T(X) \overset{df}{=} \begin{pmatrix} x_1 & x_N & x_{N-1} & \cdots & x_3 & x_2 \\ x_2 & x_1 & x_N & \cdots & x_4 & x_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N-1} & x_{N-2} & x_{N-3} & \cdots & x_1 & x_N \\ x_N & x_{N-1} & x_{N-2} & \cdots & x_2 & x_1 \end{pmatrix}.$$

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

$$C_H(X) \overset{df}{=} \begin{pmatrix} x_1 & x_2 & \cdots & x_{N-2} & x_{N-1} & x_N \\ x_2 & x_3 & \cdots & x_{N-1} & x_N & x_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ x_{N-1} & x_N & \cdots & x_4 & x_3 & x_2 \\ x_N & x_1 & \cdots & x_{N-3} & x_{N-2} & x_{N-1} \end{pmatrix}.$$

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

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

$$A^\top C_H(E_j^{(N)}) = (E_j^{(N)})^\top C_T(A).$$
Proof. The vector $A^\top C_H(E_j^{(N)}) = (C_H(E_j^{(N)})A)^\top$ is the $j$-th forward circular shift of the vector $\text{rev}(A)^\top$. Therefore, it coincides with the $j$-th row of $C_T(A)$, which is equal to $(E_j^{(N)})^\top C_T(A)$. □

Matrix-vector multiplication

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

$$C_T(X)A = F_N^{-1}(F_N(X) \odot F_N(A)).$$

(23)

A similar property holds for the $C_H(X)$. Since it is less common, we provide a short proof.

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

$$C_H(X)A = F_N^{-1}(F_N(X) \odot \text{conj}(F_N(A))).$$

(24)

Proof. The circulant $C_2 \overset{\text{def}}{=} C_T((x_N, x_1, \ldots, x_{N-1}))$ can be obtained from $C_H(X)$ by reversion of all rows. Therefore, the product of $C_H(X)$ by $A$ is equal to

$$C_H(X)A = C_2 \text{rev}(A) = F_N \left((x_N, x_1, \ldots, x_{N-1})^\top \right) \odot F_N(\text{rev}(A))
= (F_N X) \odot (e^{2\pi i / N} F_N \text{rev}(A)) = (F_N X) \odot \text{conj}(F_N A).$$

(25)

□

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

Algorithm 1. Input: $V \in \mathbb{R}^K$, $X \in \mathbb{R}^N$. Output: $U = T_{SSA}(X)V \in \mathbb{R}^L$.

1. $V' \leftarrow \begin{pmatrix} V \\ 0_{L-1} \end{pmatrix}$
2. $\hat{V}' \leftarrow F_N(V')$
3. $\hat{X} \leftarrow F_N(X)$
4. $U' \leftarrow F_N^{-1}(\hat{X} \odot \text{conj}(\hat{V}'))$
5. $U \leftarrow (u'_1, \ldots, u'_L)^\top$

In Korobeynikov (2010) two different algorithms were used for multiplication by $T_{SSA}(X)$ and its transpose. But Algorithm 1 also suits for multiplication by $T_{SSA}^\top(X)$, 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 $F_NX$ (step 3) can be precomputed. But, compared to 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 is a foundation for quasi-hankelization in Section 6.2.
The hankelization operation computes for a given $X \in \mathbb{R}^{L \times K}$ the vector $\bar{X} \in \mathbb{R}^N$ that minimizes the distance $\|T_{SSA}(\bar{X}) - X\|_F^2$. It is well-known that the hankelization of $X$ is the vector $\tilde{X} = (\tilde{x}_j)_{j=1}^N$ of diagonal averages

$$\tilde{x}_j = \frac{\sum_{k+l=j} (X)_{k,l}}{w_j} = \frac{\langle X, T_{SSA}(E_j^{(N)}) \rangle_F}{\|T_{SSA}(E_j^{(N)})\|_F^2}. \tag{25}$$

The denominators $w_j = \|T_{SSA}(E_j^{(N)})\|_F^2$ are exactly the weights in w-correlations. For rank-one matrices $X = UV^\top \in \mathbb{R}^{L \times K}$, the numerator in (25) can be expressed more compactly:

$$\langle UV^\top, T_{SSA}(E_j^{(N)}) \rangle_F = U^\top T_{SSA}(E_j^{(N)}) V = (U')^\top C_H(E_j^{(N)}) V' = (E_j^{(N)})^\top C_T(U') V',$n. Then, the hankelization can be computed using the following algorithm.

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

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

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 (25)) 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 $A, B \in \mathbb{C}^{N_x \times N_y}$ we define by $A \odot B$ their elementwise product and by $\text{conj}(A)$ the elementwise complex conjugation. For $k = 1, \ldots, N_x$ and $l = 1, \ldots, N_y$, we define an elementary array $E_{k,l}^{(N_x)} = E_k^{(N_x)}(E_l^{(N_y)})^\top \in \mathbb{R}^{N_x \times N_y}$, where $E_j^{(N)} \in \mathbb{R}^N$ is the $j$-th unit vector. For two matrices $A$ and $C$ their Kronecker product is denoted as $A \otimes C$.

For a complex matrix $X = (x_{k,l})_{k,l=1}^{N_x,N_y}$ its discrete Fourier transform is defined as

$$\mathcal{F}_{N_x,N_y}(X) \overset{d.f.}{=} F_{N_x} X F_{N_y}^\top,$$
where $F_N$ is the matrix of the discrete Fourier transform (22). By the properties of the vectorization operator, we have that

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

The *TbT circulant* constructed from the matrix $X$ is, by definition

$$C_{\text{TbT}}(X) \overset{\text{def}}{=} \begin{pmatrix}
C_{T,1} & C_{T,N_y} & C_{T,N_y-1} & \cdots & C_{T,3} & C_{T,2} \\
C_{T,2} & C_{T,1} & C_{T,N_y} & \cdots & C_{T,4} & C_{T,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
C_{T,N_y-1} & C_{T,N_y-2} & C_{T,N_y-3} & \cdots & C_{T,1} & C_{T,N_y} \\
C_{T,N_y} & C_{T,N_y-1} & C_{T,N_y-2} & \cdots & C_{T,2} & C_{T,1}
\end{pmatrix},$$

where $C_{T,j} \overset{\text{def}}{=} C_T(x_{j,:})$, where $x_{j,:}$ denotes the $j$-th column of $X$. The *HbH circulant* constructed from the matrix $X$ is, by definition

$$C_{\text{HbH}}(X) \overset{\text{def}}{=} \begin{pmatrix}
C_{H,1} & C_{H,2} & \cdots & C_{H,N_y-2} & C_{H,N_y-1} & C_{H,N_y} \\
C_{H,2} & C_{H,3} & \cdots & C_{H,N_y-1} & C_{H,N_y} & C_{H,1} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
C_{H,N_y-1} & C_{H,N_y} & \cdots & C_{H,1} & C_{H,N_y-2} & C_{H,N_y-3} \\
C_{H,N_y} & C_{H,1} & \cdots & C_{H,N_y-2} & C_{H,1} & C_{H,N_y-3}
\end{pmatrix},$$

where $C_{H,j} \overset{\text{def}}{=} C_H(x_{:,j})$. Note that for a rank-one array $X = x_x x_y^\top$ we have that

$$C_{\text{HbH}}(X) = C_H(x_y) \otimes C_H(x_x) \quad \text{and} \quad C_{\text{TbT}}(X) = C_T(x_y) \otimes C_T(x_x). \tag{26}$$

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 $A \in \mathbb{R}^{N_x \times N_y}$

$$\text{vec}(X)^\top C_{\text{HbH}}(E_{k,l}) = \text{vec}(E_{k,l})^\top C_{\text{TbT}}(X).$$

**Proof.** Due to linearity of the equation, we need to prove it only for rank-one matrices $X = x_x x_y^\top$. Then, by (26) we have that

$$\text{vec}(X)^\top C_{\text{HbH}}(E_{k,l}) = (x_y \otimes x_x)^\top \left( C_H(E_{l,N_y}) \otimes C_H(E_{k,N_x}) \right)$$

$$= \left( x_y^\top C_H(E_{l,N_y}) \right) \otimes \left( x_x^\top C_H(E_{k,N_x}) \right) = \left( E_{l,N_y}^{(N_y)} \right)^\top C_T(x_y) \otimes \left( E_{k,N_x}^{(N_x)} \right)^\top C_T(x_x)$$

$$= \left( E_{l,N_y}^{(N_y)} \otimes E_{k,N_x}^{(N_x)} \right)^\top C_T(x_y) \otimes C_T(x_x) = \text{vec}(E_{k,l})^\top C_{\text{TbT}}(X). \quad \square$$

**Multiplication by a quasi-Hankel matrix**

Analogously to the (23) and (24), the following equalities hold true.

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

$$C_{\text{TbT}}(X) \text{vec}(A) = \text{vec} \left( \mathcal{F}_{N_x,N_y}^{-1} \left( \mathcal{F}_{N_x,N_y}(X) \otimes \mathcal{F}_{N_x,N_y}(A) \right) \right),$$

$$C_{\text{HbH}}(X) \text{vec}(A) = \text{vec} \left( \mathcal{F}_{N_x,N_y}^{-1} \left( \mathcal{F}_{N_x,N_y}(X) \otimes \text{conj}(\mathcal{F}_{N_x,N_y}(A)) \right) \right).$$
**Proof.** Due to linearity of the Fourier transform and circulant matrices we can prove the statements only for matrices \( X = X_x X_y^\top \) and \( A = A_x A_y^\top \). In this case, by (26) we have that

\[
\begin{align*}
&\text{vec}\left( \mathcal{F}_{N_x,N_y}^{-1} \left( \mathcal{F}_{N_x,N_y}(X) \odot \mathcal{F}_{N_x,N_y}(A) \right) \right) \\
&= \text{vec}\left( \mathcal{F}_{N_x,N_y}^{-1} \left( (\mathbf{F}_{N_x} X_x)(\mathbf{F}_{N_y} X_y)^\top \odot (\mathbf{F}_{N_x} A_x)(\mathbf{F}_{N_y} A_y)^\top \right) \right) \\
&= \text{vec}\left( \mathbf{F}_{N_x}^{-1} ((\mathbf{F}_{N_x} X_x) \odot (\mathbf{F}_{N_x} A_x)) ((\mathbf{F}_{N_y} X_y) \odot (\mathbf{F}_{N_y} A_y))^\top (\mathbf{F}_{N_y}^{-1})^\top \right) \\
&= \text{vec}( (C_T(X_x) A_x)(C_T(X_y) A_y)^\top) = C_{TbT}(X) \text{ vec } A.
\end{align*}
\]

The proof of the second statement is analogous. \( \Box \)

Now we consider multiplication by a quasi-Hankel matrix. Let \( \mathcal{L}, \mathcal{R}, \mathcal{R} \in \{1, \ldots, N_x\} \times \{1, \ldots, N_y\} \) be such that \( \mathcal{L} + \mathcal{R} = \mathcal{R} \). Then the quasi-Hankel matrix \( T_{\text{ShSSA}}(X) \) (defined in (21)) is a submatrix of the HbH circulant \( C_{\text{HbH}}(X) \). Now let us write this formally and provide an algorithm for the calculation of the matrix-vector product.

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

\[
P_{\mathcal{A}} = [\text{vec}(E_{k_1,l_1}) : \cdots : \text{vec}(E_{k_N,l_N})].
\]

Then we have that

\[
T_{\text{ShSSA}}(X) = P_{\mathcal{L}}^\top C_{\text{HbH}}(X) P_{\mathcal{R}}.
\]

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

**Algorithm 3.** *Input:* \( V \in \mathbb{R}^K \), \( X \in \mathbb{R}^{N_x \times N_y} \). *Output:* \( U = T_{\text{ShSSA}}(X) V \in \mathbb{R}^L \).

1. \( \tilde{X} \leftarrow \mathcal{F}_{N_x,N_y}(X) \)
2. \( U' \leftarrow \text{vec}^{-1}_{K_y}(P_{\mathcal{R}} U) \)
3. \( \tilde{U}' \leftarrow \mathcal{F}_{N_x,N_y}(U') \)
4. \( V' \leftarrow \mathcal{F}_{N_x,N_y}^{-1}(\tilde{X} \odot \text{conj}(\tilde{U}')) \)
5. \( V \leftarrow P_{\mathcal{L}}^\top \text{ vec } (V') \)

**Rank-one quasi-Hankelization**

Let \( \mathcal{L}, \mathcal{R}, \mathcal{R} \in \{1, \ldots, N_x\} \times \{1, \ldots, N_y\} \) be such that \( \mathcal{L} + \mathcal{R} = \mathcal{R} \). The quasi-Hankelization operator, by definition, computes for a given \( X \in \mathbb{R}^{L \times K} \) the shaped array \( \tilde{X}_{\mathcal{R}} = (\tilde{x}_{k,l})_{(k,l) \in \mathcal{R}} \) that minimizes the distance \( \| T(\tilde{X}_{\mathcal{R}}) - X \|_F^2 \).

It is well known that quasi-Hankelization can be expressed as averaging. More precisely,

\[
\tilde{x}_{k,l} = \frac{X, T(E_{k,l})}{\|T(E_{k,l})\|_F^2}, \quad \text{for } (k,l) \in \mathcal{R}.
\]

The numerator represents summation over the set of positions in \( X \) that correspond to the \((k,l)\)-th element of an array in a quasi-Hankel matrix. The denominator is equal to the number of such elements.
In this section, we assume that $X$ is a rank-one matrix, i.e. $X = UV^\top$. Then
\[
\tilde{x}_{k,l} = \frac{\langle UV^\top, T(E_{k,l}) \rangle_F}{\|T(E_{k,l})\|_F^2} = \frac{\langle UV^\top, T(E_{k,l}) \rangle_F}{(1_L^T K, T(E_{k,l}))_F}.
\]
If we define by
\[
diagsums(U, V) \overset{def}{=} (x_{k,l})_{k,l=1}^{N_x, N_y}, \quad \text{where } x_{k,l} = \langle UV^\top, T(E_{k,l}) \rangle_F,
\]
then quasi-Hankelization is given by the following algorithm.

**Algorithm 4.**

Input: $U \in \mathbb{R}^L$, $V \in \mathbb{R}^K$, shapes $\mathcal{L}$, $\mathcal{R}$. Output: shape $\mathcal{M} = \mathcal{L} + - \mathcal{R}$, quasi-Hankelization $\tilde{X}_{\mathcal{M}}$.

1. $W \leftarrow \text{diagsums}(1_L, 1_K)$ (array of weights for w-correlations)
2. $\mathcal{M} \leftarrow \{(k, l) \mid w_{k,l} \neq 0\}$
3. $\tilde{X} \leftarrow \text{diagsums}(U, V)$
4. Compute $\tilde{x}_{k,l} = \tilde{x}'_{k,l} / w_{k,l}$

We note that the weights $W$ can be precomputed, as well as the shape $\mathcal{M}'$. The only missing part in Algorithm 4 is computation of $\text{diagsums}(U, V)$ for given $U \in \mathbb{R}^L$ and $V \in \mathbb{R}$. For this we note that
\[
\langle UV^\top, T_{\text{ShSSA}}(E_{k,l}) \rangle_F = \text{trace}(VU^\top P_L^T T_{\text{ShSSA}}(E_{k,l}) P_R)
= \text{vec}(U')^\top C_{\text{HH}}(E_{k,l}) \text{vec}(V') = \text{vec}(E_{k,l})^\top C_{\text{TH}}(U') \text{vec}(V'),
\]
where $U' = \text{vec}^{-1}_{L_x}(P_L U)$ and $V' = \text{vec}^{-1}_{L_y}(P_R V)$, and the last equality holds by Lemma 3. Then $\text{diagsums}(U, V)$ can be computed using the following algorithm.

**Algorithm 5.**

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

1. $U' \leftarrow \text{vec}^{-1}_{L_x}(P_L U)$
2. $V' \leftarrow \text{vec}^{-1}_{L_y}(P_R V)$
3. $\hat{U}' \leftarrow T_{N_x, N_y}(U')$
4. $\hat{V}' \leftarrow T_{N_x, N_y}(V')$
5. $X \leftarrow T_{N_x, N_y}^{-1}(\hat{U}' \odot \hat{V}')$

**Calculation of shapes**

Assume that we have $\mathcal{L} \subseteq \{1, \ldots, L_x\} \times \{1, \ldots, L_y\}$ and $\mathcal{M} \subseteq \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$. We would like to find the maximal $\mathcal{M}'$ such that $\mathcal{L} + - \mathcal{M}' \subseteq \mathcal{M}$. 

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

\[
i_{k,l}^{(N)} = \begin{cases} 
1, & (k,l) \in \mathcal{N}, \\
0, & (k,l) \notin \mathcal{N}.
\end{cases}
\]

Then the \( \mathcal{R} \) corresponds to the maximal \( \mathcal{R} \) such that all the elements of the \( L \times \mathcal{R} \) submatrix of \( \mathcal{T}_{2DSSA}(I^{(N)}) \) are equal to 1.

**Algorithm 6.**

**Input:** \( L \subset \{1, \ldots, L_x\} \times \{1, \ldots, L_y\} \), \( \mathcal{N} \subseteq \{1, \ldots, N_x\} \times \{1, \ldots, N_y\} \). **Output:** maximal \( \mathcal{R} \) such that \( L + \mathcal{R} \subseteq \mathcal{N} \).

1. \( V \leftarrow T_{2DSSA}(I^{(N)})1_L \)
2. \( V' \leftarrow \text{vec}_K^{-1}(V) \)
3. \( \mathcal{R} \leftarrow \{(k,l) | v_{k,l} = L\} \).

### 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 so if to implement 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 \texttt{Rssa} the vector forecasting is slightly faster than the recurrent one.

In this section we will use the notation from Section 3.3.

**1d-SSA case**

Consider the forecasting in the subspace \( \mathcal{L}^{\text{col}} \) given by a basis \( \{P_1, \ldots, P_r\} \). Denote \( \mathbf{P} = [P_1 : \ldots : P_r] \). Each reconstructed vector \( \tilde{X}_k \in \mathcal{L}^{\text{col}} \) and therefore there exist coefficients \( W_k \in \mathbb{R}^r \) such that \( \tilde{X}_k = \mathbf{P}W_k \). Denote \( \mathbf{W} = [W_1 : \ldots : W_K] \). In fact, the input for the algorithm is the minimal decomposition of \( \tilde{X} \) into the sum of elementary matrices of rank 1 in the form \( \tilde{X} = \mathbf{P}Q^\top \) and \( \mathbf{W} = Q^\top \).

Note that if we have a Singular Value Decomposition of \( \tilde{X} = [\tilde{X}_1 : \ldots : \tilde{X}_K] \), then the left singular vectors provide the basis of the subspace, while \( W_k \) are determined by right singular vectors and singular values: \( \mathbf{P} = [U_1 : \ldots : U_r] \) and \( \mathbf{Q} = [\sqrt{\lambda_1}V_1 : \ldots : \sqrt{\lambda_r}V_r] \).

**Algorithm:**

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

**Remark 2.**
1. Item 1 in the algorithm is exactly the calculation of the shift matrix from the LS-ESPRIT method for frequency estimation (Roy and Kailath 1989).
2. If $\{P_i\}$ is orthonormal system, then we can compute $D$ more effectively, not using the QR decomposition:
   \[
   D = \left( I_r - \frac{1}{1 - \pi^\top \pi} \pi \pi^\top \right) P^\top P,
   \]
   where $\pi = \pi(P)$, $I_r$ is the $r \times r$ identity matrix.

**Column MSSA-forecast**

Column vector MSSA forecasting in the subspace $\mathcal{L}^{\text{col}}$ is reduced to performing $s$ 1D-vector forecasts in the same subspace $\mathcal{L}^{\text{col}}$ for each of $s$ series. Note that we can compute the shift matrix $D$ only one time, since it depends only on $P$.

**Row MSSA-forecast**

Row vector forecast slightly differs from the column one, but the idea is the same except we deal with forecasting in the row subspace $\mathcal{L}^{\text{row}} = \text{span}(Q_1, \ldots, Q_r)$ and continue the sequence of the row vectors $\hat{Y}_k$ of $\hat{X}$. Let $\hat{Y}_k = QW_k$.

**Algorithm:**

- Compute the matrix $D = Q \overline{Q}$ using the QR decomposition.
- For $k = L + 1 \ldots L + M + \max_{k=1,\ldots,s} K_k - 1$ compute $W_k = DW_{k-1}$.
- Perform Fast Hankelization Algorithm for each of $s$ matrices $Z^{(j)} = Q^{(j)}[W : W_{L+1} : \ldots : W_{L+M+K_j-1}]$ for $j = 1, \ldots, s$ and obtain $s$ series $z_1^{(j)}, \ldots, z_{N_j+M+L-1}^{(j)}$.
- The numbers $z_1^{(j)}, \ldots, z_{N_j+M}^{(j)}$ forms updated reconstructed and forecasted series.

**Remark 3.** If $\{Q_i\}$ is orthonormal system, then
   \[
   D = \left( I_r - S^\top (I_s - SS^\top)^{-1} S \right) Q^\top \overline{Q},
   \]
   where $S = [\mu(Q_1) : \ldots : \mu(Q_r)]$, $I_r$ and $I_s$ are the $r \times r$ and $s \times s$ identity matrices.

**Proof of the algorithm validity**

Let us prove that these algorithms construct the vector forecasts described in Section 3.3. For simplicity, we will consider the one-dimensional case, that is, SSA vector forecast, which coincides with the column MSSA forecasting for one-dimensional series.

It is sufficiently to prove that $Z_{k+1} = PW_{k+1}$ or even that $Z_{k+1} = PW_{k+1}$, 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, $Z_{k+1}$ is the projection
on the column space of $P$, that is, $Z_{k+1} = PP^\dagger Z_k$, where $\dagger$ means pseudo-inversion. In the fast vector forecast, $Z_{k+1} = P(P^\dagger P)W_k$. Since $PW_k$ is exactly $Z_k$ by definition of $W_k$, the equivalence of the standard and fast vector forecasting algorithms is proved.

**Remark 4.** The speed up of the algorithm implementation is explained by two reasons: (1) multiplication on matrices of small size $r \times r$ at each step instead of multiplication on matrices of much larger size and (2) the form of the matrix that should be hankelized is suitable for application of Fast Hankelization algorithm.

7. Appendix

7.1. Elements of MSSA theory

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

**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 a weak separability, which means orthogonality of the trajectory spaces. There is a 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 $F^{(1)}$ and $F^{(2)}$, $G^{(1)}$ and $G^{(2)}$, and also $G^{(1)}$ and $F^{(2)}$ are weakly $L$-separable by SSA, then the two-dimensional time series $(F^{(1)}, F^{(2)})$ and $(G^{(1)}, G^{(2)})$ are weakly $L$-separable by MSSA and the complex-valued time series $F^{(1)} + iF^{(2)}$ and $G^{(1)} + iG^{(2)}$ are weakly $L$-separable by CSSA.

Proposition 1 can be extended to the 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 $F^{(1)}$, $F^{(2)}$, $G^{(1)}$ and $G^{(2)}$ of length $N$

\[
  f^{(1)}_k = A_1 \cos(2\pi \omega_1 k + \varphi_1), \quad f^{(2)}_k = B_1 \cos(2\pi \omega_2 k + \varphi_2),
\]

\[
  g^{(1)}_k = A_2 \cos(2\pi \omega_2 k + \varphi_1), \quad g^{(2)}_k = B_2 \cos(2\pi \omega_2 k + \varphi_2),
\]

$\omega_1 \neq \omega_2$, $k = 0, \ldots, N - 1$, $A_1, A_2, B_1, B_2 \neq 0$. If $L\omega_i$ and $K\omega_i$, $i = 1, 2$, are integer, then $(F^{(1)}, F^{(2)})$ and $(G^{(1)}, G^{(2)})$ are $L$-separable by MSSA and the complex-values time series $F^{(1)} + iF^{(2)}$ and $G^{(1)} + iG^{(2)}$ 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 \((F(i), G(i))\) 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

\[
f_k^{(1)} = A \cos(2\pi\omega_k + \varphi_1), \quad f_k^{(2)} = B \cos(2\pi k\omega + \varphi_2).
\]

If \(L\omega\) and \(K\omega\) are integer, then \((F(1), F(2))\) produces two equal eigenvalues in MSSA: \(\lambda_1 = \lambda_2 = (A^2 + B^2)LK/4\), while \(F(1)^2 + iF(2)^2\) produces two nonequal eigenvalues in CSSA:

\[
\lambda_1 = (A^2 + B^2 + 2AB \sin \varphi)LK/4, \\
\lambda_2 = (A^2 + B^2 - 2AB \sin \varphi)LK/4,
\]

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

**Multi-dimensional time series and LRRs**

Consider a system of infinite time series \(X^{(1)}, X^{(2)}, \ldots, X^{(s)}\), choose the window length \(L\) and denote \(X^{(1)}, \ldots, X^{(s)}\) the column trajectory spaces of the series. Let \(X = \text{span}(X^{(1)}, \ldots, X^{(s)})\) be the column trajectory space of the collection of time series \((X^{(1)}, X^{(2)}, \ldots, X^{(s)})\). 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 \(X^{(l)}\) by \(r_l = \dim X^{(l)} \leq L, \ r = l, \ldots, s\). For each time series \(X^{(l)}\) we can write out the minimal LRR governing the time series:

\[
x_{j+r_l}^{(l)} = \sum_{k=1}^{r_l} a_k^{(l)} x_{j+r_l-k}^{(l)}, \quad \text{where} \quad a_{r_l}^{(l)} \neq 0, \ l = 1, \ldots, s.
\]  

(27)

The corresponding characteristic polynomials of the LRR (27) are

\[
F_{r_l}^{(l)}(\mu) = \mu^{r_l} - \sum_{k=1}^{r_l} a_k^{(l)} \mu^{r_l-k}, \quad l = 1, \ldots, s.
\]  

(28)

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

Let

\[
p^{(l)} \quad \text{be number of different roots of the polynomial} \quad F_{r_l}^{(l)}(\lambda),
\]

\[
\mu_m^{(l)} \quad \text{be} \ m\text{-th root of the polynomial} \quad P_{r_l}^{(l)}(\lambda),
\]

\[
k_m^{(l)} \quad \text{be the multiplicity of the root} \quad \mu_m^{(l)}.
\]
Thus, we have

\[ k_1^{(l)} + \ldots + k_p^{(l)} = r_l, \quad l = 1, \ldots, s. \]

The characteristic roots determine the series behavior. For example, if \( k_m^{(l)} = 1 \), then

\[ x_n^{(l)} = \sum_{j=1}^{r_l} C_j^{(l)} (\mu_j^{(l)})^n. \]

Also let

\[ \mu_1, \ldots, \mu_p \quad \text{be the pooled set of roots of polynomials} \quad P_r^{(1)}, \ldots, P_r^{(s)}, \]

\[ k_1, \ldots, k_p \quad \text{be the multiplicities of the roots} \quad \mu_1, \ldots, \mu_p, \]

where multiplicity of a root in the pooled set is equal to the maximum of multiplicities of the corresponding roots in the initial sets.

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

**Proposition 2.** Rank of the infinite multi-dimensional time series \((X^{(1)}, X^{(2)}, \ldots, X^{(s)})\) is equal to

\[ r = \sum_{i=1}^{p} k_i \quad \text{if} \quad L > r. \]

Consider a simple example.

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

\[ f_k^{(1)} = A \cos(2\pi \omega_1 k + \varphi_1), \quad f_k^{(2)} = B \cos(2\pi \omega_2 k + \varphi_2), \quad (29) \]

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

1. For \( \omega_1 = \omega_2 \) the SSA and the MSSA ranks of the sinusoid \((29)\) is equal to 2. The CSSA rank is equal to 1 if \( A = B \) and \( |\varphi_1 - \varphi_2| = \pi/2 \mod \pi \) and is equal to 2 otherwise.

2. For \( \omega_1 \neq \omega_2 \) the SSA, MSSA and CSSA ranks equal 4.

### 7.2. Elements of 2D-SSA theory

**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 \( X = (x_{m,n})_{m,n=0}^{\infty,\infty} \) given in a parametric form:

\[ x_{m,n} = \sum_{k=1}^{r} c_k \lambda_k^m \mu_k^n, \quad (30) \]
where \((\lambda_k, \mu_k) \in \mathbb{C}^2\) 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 \(X\) is equal to \(r\) (or equivalently, the arrays of the form (30) are arrays of finite rank). Note that the exponentials can be also represented in the form\[
\lambda_k = \exp(2\pi i \omega_{x,k} + \phi_{x,k}), \quad \mu_k = \exp(2\pi i \omega_{y,k} + \phi_{y,k}).
\]

We should note the class of arrays of finite rank also contains bivariate polynomials and their products with the functions from (30). An algebraic characterization of the arrays of finite rank can be found in Golyandina and Usevich (2009).

**Real arrays**

Now let us summarize what happens in the case of real arrays \(X\). If (30) is real then for any pair \((\lambda, \mu)\) its complex conjugate \(\overline{\lambda}, \overline{\mu}\) also should be present in (30). We order the roots such that \((\lambda_k, \mu_k)\) are real for \(1 \leq k \leq d\), and other \(2s\) roots are at least partially complex (are complex conjugate pairs), and are arranged as \((\lambda_k, \mu_k) = (\text{conj}(\lambda_{k-s}), \text{conj}(\mu_{k-s}))\) for \(d+1 \leq k \leq d+s\). Then the roots have a representation

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

where \((\rho_{x,k}, \rho_{y,k}, \omega_{x,k}, \omega_{y,k}) \in \mathbb{R}^2 \times [0; 1/2]^2\) are distinct 4-tuples of real numbers, such that \(\omega_{x,k} = \omega_{y,k} = 0\) for \(1 \leq k \leq d\).

Then the representation (30) becomes a sum of \(d + s\) planar modulated sinewaves:

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

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.*

\[
2 \cos(2\pi \omega_x m + \phi_1) \cos(2\pi \omega_y n + \phi_2) = \cos(2\pi (\omega_x m + \omega_y n) + \phi_1 + \phi_2) + \cos(2\pi (\omega_x m - \omega_y n) + \phi_1 - \phi_2)
\]

**8. Acknowledgement**

The paper was partly supported by the project NG13-083 of the Dynasty Foundation in the area of computer science. Konstantin Usevich has received funding from the European Research Council under the European Union’s Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement no. 258581 “Structured low-rank approximation: Theory, algorithms, and applications”.
References

Ade F (1983). “Characterization of textures by ‘Eigenfilters’.” *Signal Processing, 5*(5), 451 – 457.

Alexandrov T (2009). “A method of trend extraction using Singular Spectrum Analysis.” *RevStat, 7*(1), 1–22.

Broomhead D, King G (1986b). “On the qualitative analysis of experimental dynamical systems.” In S Sarkar (ed.), *Nonlinear Phenomena and Chaos*, pp. 113–144. Adam Hilger, Bristol.

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.” *Proc. IEEE, 93*(2), 216–231.

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

Gistat Group (2013). “CaterpillarSSA.” (1996-2013), URL http://gistatgroup.com.

Golub GH, Van Loan CF (1996). *Matrix computations (3rd ed.).* Johns Hopkins University Press, Baltimore, MD, USA. ISBN 0801854148.

Golyandina N (2010). “On the choice of parameters in Singular Spectrum Analysis and related subspace-based methods.” *Stat. Interface, 3*(3), 259–279.

Golyandina N, Florinsky I, Usevich K (2007). “Filtering of digital terrain models by 2D Singular Spectrum Analysis.” *International Journal of Ecology & Development, 8*(F07), 81–94. URL http://eprints.soton.ac.uk/273004/.

Golyandina N, Nekrutkin V, Stepanov D (2003). “Variants of the ‘Caterpillar’-SSA method for analysis of multidimensional time series.” In *Proceedings of II International Conference on System Identification and Control Problems (SICPRO 03)*, pp. 2139–2168. Moscow: V.A.Trapeznikov institute of Control Sciences. (In Russian), URL http://gistatgroup.com/cat/mssa_analysis_ru.pdf.

Golyandina N, Korobeynikov A (2013). “Basic Singular Spectrum Analysis and forecasting with R.” *Computational Statistics & Data Analysis. doi:10.1016/j.csda.2013.04.009.*

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

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

Golyandina N, Shlemov A (2013). “Variations of Singular Spectrum Analysis for separability improvement: non-orthogonal decompositions of time series.” *arxiv.org. URL http://arxiv.org/abs/1308.4022.*
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 Tyrtysnikov (eds.), Matrix Methods: Theory, Algorithms and Applications, pp. 449–473. World Scientific Publishing.

Golyandina N, Zhigljavsky A (2013). Singular Spectrum Analysis for time series. SpringerBriefs in Statistics. Springer.

Golyandina NE, Holloway DM, Lopes FJP, Spirov AV, Spirova EN, Usevich KD (2012). “Measuring Gene Expression Noise in Early Drosophila Embryos: Nucleus-to-nucleus Variability.” In Procedia Computer Science, volume 9, pp. 373–382.

Hannachi A, Jolliffe IT, Stephenson DB (2007). “Empirical orthogonal functions and related techniques in atmospheric science: A review.” International Journal of Climatology, 27(9), 1119–1152.

Hassani H, Heravi S, Zhigljavsky A (2013). “Forecasting UK Industrial Production with Multivariate Singular Spectrum Analysis.” Journal of Forecasting, 32(5), 395–408. ISSN 1099-131X.

Heinig G (1995). “Generalized inverses of Hankel and Toeplitz mosaic matrices.” Linear Algebra Appl., 216(0), 43–59.

Holloway DM, Lopes FJP, da Fontoura Costa L, Travençolo BAN, Golyandina N, Usevich K, Spirov AV (2011). “Gene Expression Noise in Spatial Patterning: hunchback Promoter Structure Affects Noise Amplitude and Distribution in Drosophila Segmentation.” PLoS Computational Biology, 7(2), e1001069.

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

Korobeynikov A (2010). “Computation- and Space-Efficient Implementation of SSA.” Stat. Interface, 3(3), 357–368.

Larsen RM (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.

Monadjemi A (2004). Towards efficient texture classification and abnormality detection. Ph.D. thesis, University of Bristol.
Mourrain B, Pan VY (2000). “Multivariate polynomials, duality, and structured matrices.” *Journal of complexity*, 16(1), 110–180.

Rodríguez-Aragón LJ, Zhigljavsky A (2010). “Singular spectrum analysis for image processing.” *Statistics and Its Interface*, 3(3), 419–426.

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.

Roy R, Kailath T (1989). “ESPRIT: estimation of signal parameters via rotational invariance techniques.” *IEEE Trans. Acoust.*, 37, 984–995.

Sarkar D (2008). *Lattice: Multivariate Data Visualization with R*. Springer, New York. ISBN 978-0-387-75968-5, 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 MJ, Virtue P, Elad M, Pauly JM, Vasanawala SS, Lustig M (2013). “ESPIRiT — an eigenvalue approach to autocalibrating parallel MRI: Where SENSE meets GRAPPA.” *Magnetic Resonance in Medicine*, pp. n/a–n/a. doi:10.1002/mrm.24751.

Van Der Veen AJ, Deprettere EF, Swindlehurst AL (1993). “Subspace-based signal analysis using singular value decomposition.” *Proc. IEEE*, 81(9), 1277–1308. ISSN 0018-9219.

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

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.

Weare BC, Nasstrom JS (1982). “Examples of Extended Empirical Orthogonal Function Analyses.” *Mon Weather Rev*, 110(6), 481–485.

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.
Affiliation:
Nina Golyandina  
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  
Department of Statistical Modelling  
Faculty of Mathematics and Mechanics  
St.Petersburg State University  
Universitetskij pr 28, 198504, St.Petersburg, Russia  
E-mail: anton@korobeynikov.info

Alex Shlemov  
Department of Statistical Modelling  
Faculty of Mathematics and Mechanics  
St.Petersburg State University  
Universitetskij pr 28, 198504, St.Petersburg, Russia  
E-mail: shlemovalex@gmail.com

Konstantin Usevich  
Department ELEC  
Faculty of Engineering  
Vrije Universiteit Brussel  
Pleinlaan 2, 1050 Elsene, Belgium  
E-mail: konstantin.usevich@vub.ac.be