Journal of Statistical Software
July 2021, Volume 98, Issue 15. doi: 10.18637/jss.v098.i15

Dimension Reduction for Time Series in a Blind Source Separation Context Using R

Klaus Nordhausen
Vienna University of Technology

Markus Matilainen
Turku PET Centre

Jari Miettinen
Aalto University

Joni Virta
Aalto University
University of Turku

Sara Taskinen
University of Jyväskylä

Abstract

Multivariate time series observations are increasingly common in multiple fields of science but the complex dependencies of such data often translate into intractable models with large number of parameters. An alternative is given by first reducing the dimension of the series and then modelling the resulting uncorrelated signals univariately, avoiding the need for any covariance parameters. A popular and effective framework for this is blind source separation. In this paper we review the dimension reduction tools for time series available in the R package **tsBSS**. These include methods for estimating the signal dimension of second-order stationary time series, dimension reduction techniques for stochastic volatility models and supervised dimension reduction tools for time series regression. Several examples are provided to illustrate the functionality of the package.

Keywords: blind source separation, supervised dimension reduction, R.

1. Dimension reduction and BSS for time series

In many fields of applied science multivariate time series, $\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,p})^\top$, $t = 1, \ldots T$, are collected. Examples include geophysical time series, telecommunications data, financial time series and biomedical time series, e.g., EEG, MEG and fMRI. Such data are often characterized by two main features, which both pose problems for multivariate time series modelling. First, the p components in multivariate time series data are often correlated and therefore individual time series cannot be modelled using univariate time series models. Models such as multivariate autoregressive moving average (ARMA) and multivariate generalized autoregressive conditional heteroskedasticity (GARCH) include huge numbers of parameters and model fitting becomes computationally impractical unless the models are noticeably simplified. Second, the data can be high-dimensional and might contain a high unknown number of noise components. As an example, consider current biomedical datasets where the number of time series components vary from hundreds to millions, and the main aim of the analysis is to separate the signals of interest from noise. For such high-dimensional data, fitting multivariate time series models might become impossible, and at the very least unreasonable as it is not sensible to assign a huge number of model parameters for noise components. These two features often encountered in multivariate time series data motivate us to consider dimension reduction methods suitable for time series.

In this paper we illustrate how blind source separation (BSS) methods (see for example Comon and Jutten 2010; Nordhausen and Oja 2018) can be used for dimension reduction in a time series context. Recall first that a (linear) BSS model assumes that the observable p-variate time series $\boldsymbol{x} = (\boldsymbol{x}_t)_{t=0,\pm1,\pm2,\ldots}$ satisfy

$$\boldsymbol{x}_t = \boldsymbol{\mu} + \boldsymbol{\Omega}\boldsymbol{z}_t, \quad t = 0, \pm1, \pm2, \ldots, \tag{1}$$

where $\boldsymbol{\mu} \in \mathbb{R}^p$ is a p-variate location vector, $\boldsymbol{\Omega} \in \mathbb{R}^{p\times p}$ is a full-rank mixing matrix and $\boldsymbol{z} = (\boldsymbol{z}_t)_{t=0,\pm1,\pm2,\ldots}$ is a p-variate latent time series satisfying $E(\boldsymbol{z}_t) = \boldsymbol{0}$, $Cov(\boldsymbol{z}_t) = \boldsymbol{I}_p$ and

$$Cov_\tau(\boldsymbol{z}_t) = E(\boldsymbol{z}_t\boldsymbol{z}_{t+\tau}^\top) = \boldsymbol{\Lambda}_\tau \quad \text{is a diagonal matrix for all } \tau = 1, 2, \ldots.$$

The p time series in $\boldsymbol{z}$ are thus weakly stationary and uncorrelated, and the model is often referred to as a second-order source separation model (SOS). If we make the stronger assumption on the independence of the p time series in $\boldsymbol{z}$, then the model is called the independent time series model. The BSS approach for dimension reduction is often preferred since the obtained components can be treated as independent, allowing univariate fitting and prediction methods. For example Van der Weide (2002) and Broda and Paolella (2009) have used this idea for multivariate GARCH modelling. Furthermore, with BSS one can also decide componentwise whether an estimated component is relevant or not. For related research on this, see Matteson and Tsay (2011).

There are two different approaches to define signal and noise in BSS. One commonly used noisy BSS model is the one where an additive external noise vector is included in model (1), that is, we assume that $\boldsymbol{x}_t = \boldsymbol{\mu} + \boldsymbol{\Omega}\boldsymbol{z}_t + \boldsymbol{\epsilon}_t$, $t = 0, \pm1, \pm2, \ldots$, where $\boldsymbol{\epsilon}_t \in \mathbb{R}^p$ is a vector of white noise (Comon and Jutten 2010). The other approach, and the one we prefer, differs from the noisy BSS model in that it assumes that q of the p sources are signal and $p - q$ are internal noise components. In particular, we assume that the source vector $\boldsymbol{z}_t$ can be partitioned into two parts, $\boldsymbol{z}_t = (\boldsymbol{s}_t^\top, \boldsymbol{w}_t^\top)^\top$, so that the first part $\boldsymbol{s}_t \in \mathbb{R}^q$ includes the sources of interest while $\boldsymbol{w}_t \in \mathbb{R}^{p-q}$ represents the noise components. To be more specific, we assume the following BSS model

$$\boldsymbol{x}_t = \boldsymbol{\mu} + \boldsymbol{\Omega}\boldsymbol{z}_t = \boldsymbol{\mu} + \boldsymbol{\Omega}_1\boldsymbol{s}_t + \boldsymbol{\Omega}_2\boldsymbol{w}_t, \quad t = 0, \pm1, \pm2, \ldots, \tag{2}$$

where $\boldsymbol{\mu} \in \mathbb{R}^p$ is a location vector, $\boldsymbol{\Omega} = (\boldsymbol{\Omega}_1, \boldsymbol{\Omega}_2) \in \mathbb{R}^{p\times p}$ is a full-rank mixing matrix, $\boldsymbol{\Omega}_1 \in \mathbb{R}^{p\times q}$ and $\boldsymbol{\Omega}_2 \in \mathbb{R}^{p\times(p-q)}$, and the p-variate latent time series $\boldsymbol{z}_t = (\boldsymbol{s}_t^\top, \boldsymbol{w}_t^\top)^\top$ satisfy

(A1) $E(\boldsymbol{z}_t) = \boldsymbol{0}$ and $Cov(\boldsymbol{z}_t) = \boldsymbol{I}_p$,

(A2) $\boldsymbol{s}_t \in \mathbb{R}^q$ and $\boldsymbol{w}_t \in \mathbb{R}^{p-q}$ are independent.

(A3) $Cov_\tau(\boldsymbol{z}_t) = \begin{pmatrix} \boldsymbol{\Lambda}_\tau & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}$, where $\boldsymbol{\Lambda}_\tau \in \mathbb{R}^{q \times q}$ is a diagonal matrix for all $\tau = 1, 2, \ldots$

Notice that in model (2) no distributional assumptions are made for the noise components $\boldsymbol{w}_t$, we simply require that they do not show any linear autocorrelations. Examples of such noise components include white noise as well as components exhibiting volatility clustering. The latter components have zero linear autocorrelations but non-zero quadratic autocorrelations. Notice also that in the BSS model (2) the components in $\boldsymbol{s}_t$ can be identified at most up to signs and permutations of its components, and $\boldsymbol{\Omega}_1$ at most up to the signs and permutation of its columns. Matrix $\boldsymbol{\Omega}_2$ is identifiable only up to post-multiplication with an orthogonal $(p - q) \times (p - q)$ matrix.

We note that other approaches besides BSS are frequently used for the dimension reduction of multivariate time series data. We next list some of the most well-known of these and discuss the benefits of BSS over them. The simplest options are arguably the classical principal component analysis (PCA) and factor analysis, which however treat the data as independent observations. As the ability to use temporal information is crucial in analyzing time series, a number of extensions of the two methods that allow for this have been suggested in the literature. The most prominent example is given by dynamic factor models, see Stock and Watson (2010); Ensor (2013) and the references therein, where the latent factors are most often assumed to obey a vector autoregressive structure. This is in strict contrast to the BSS model (1) where no structural assumptions on the latent variables are made (beyond temporal uncorrelatedness). As an example of this versatility, consider the methods SOBI and vSOBI discussed in Section 2 and Section 3, respectively, which both comply with the BSS framework but the former assumes the sources to be standard linear processes and the latter that they are series exhibiting stochastic volatility. For a more recent extension of PCA, see the time series PCA (Chang, Guo, and Yao 2018) which is actually based on a slight modification of the model (1) and thus constitutes a form of BSS. Other contenders include, e.g., non-negative matrix factorizations (Cheung, Devarajan, Severini, Turolla, and Bonato 2015; Févotte, Smaragdis, Mohammadiha, and Mysore 2018) and autoencoders (Bianchi, Livi, Mikalsen, Kampffmeyer, and Jenssen 2019). The involved complicated constraints and structures make such models difficult to analyze theoretically, whereas the relatively straightforward form of the BSS model makes it possible to obtain convergence rates and other theoretical guarantees (Miettinen, Illner, Nordhausen, Oja, Taskinen, and Theis 2016). To summarize, the advantages of BSS over its competitors lie in its ability to model temporal data while making minimal assumptions on the latent variables and in the BSS model which is simultaneously complicated enough to be useful and simple enough to study theoretically.

This paper is organized as follows. In Section 2 we recall two classical second-order source separation methods and show how they can be used to estimate the dimension of the interesting source components in model (2). In Section 3 we consider dimension reduction in a case where the time series exhibit stochastic volatility and our main interest is finding out whether some of the components lack stochastic volatility features. In Section 4 several supervised dimension reduction methods for multivariate time series are discussed. Section 5 reviews the existing packages available in R (R Core Team 2021) for multivariate time series and BSS as well as describes the functionality of the R package **tsBSS** (Matilainen, Croux, Miettinen, Nordhausen, Oja, and Taskinen 2021). In Section 6 several examples are given to illustrate the functionality of the **tsBSS** package. The package **tsBSS** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=tsBSS`.

2. Dimension estimation using AMUSE and SOBI

In this section we illustrate how two classical second-order source separation methods, AMUSE (algorithm for multiple unknown signals extraction) by Tong, Soon, Huang, and Liu (1990) and SOBI (second-order blind identification) by Belouchrani, Abed-Meraim, Cardoso, and Moulines (1997), can be used to estimate the dimension of important components in the BSS model. Let us start by recalling the two methods. Hence, assume for a moment that $\boldsymbol{x}_t$ follows a BSS model (2) with $q = p$, that is, there are no noise components included in the model. Assume also, without loss of generality, that $\boldsymbol{\mu} = \mathbf{0}$. The aim of second-order source separation is to find an unmixing matrix $\boldsymbol{W} \in \mathbb{R}^{p \times p}$ such that the component time series in $\boldsymbol{W}\boldsymbol{x}_t$ are standardized and mutually uncorrelated, that is, $\boldsymbol{W}\boldsymbol{x}_t = \boldsymbol{z}_t$ up to location shifts, sign changes and permutations of the components.

Write now $\boldsymbol{x}_t^{st} = Cov(\boldsymbol{x}_t)^{-1/2}\boldsymbol{x}_t$ for a standardized time series. Cardoso and Souloumiac (1993) showed that the standardized series then satisfy $\boldsymbol{x}_t^{st} = \boldsymbol{U}\boldsymbol{z}_t$, where $\boldsymbol{U} \in \mathbb{R}^{p \times p}$ is an unknown orthogonal matrix. The standardization thus solves the BSS problem up to rotation, that is, the unmixing matrix is given by $\boldsymbol{W} = \boldsymbol{U}^\top Cov(\boldsymbol{x}_t)^{-1/2}$. In AMUSE, the rotation matrix $\boldsymbol{U}$ is found simply using the eigendecomposition of the autocovariance matrix with given lag τ, as by assumption, $Cov_\tau(\boldsymbol{x}_t^{st}) = \boldsymbol{U} Cov_\tau(\boldsymbol{z}_t)\boldsymbol{U}^\top = \boldsymbol{U}\boldsymbol{\Lambda}_\tau\boldsymbol{U}^\top$ for any lag τ. For the statistical properties of AMUSE estimators, see Miettinen, Nordhausen, Oja, and Taskinen (2012).

The drawback of the AMUSE procedure is that, in order to separate all p source components, we need to assume that for a chosen lag τ, the eigenvalues in $\boldsymbol{\Lambda}_\tau$ are distinct. As this may not hold in practice, it is often better to use approximate joint diagonalization of several autocovariance matrices as suggested in Belouchrani *et al.* (1997). Their SOBI method is a natural extension of AMUSE as the solution is given by $\boldsymbol{W} = \boldsymbol{U}^\top Cov(\boldsymbol{x}_t)^{-1/2}$ where the orthogonal $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p) \in \mathbb{R}^{p \times p}$ maximizes

$$\sum_{\tau \in \mathcal{T}} \sum_{i=1}^{p} (\boldsymbol{u}_i^\top Cov_\tau(\boldsymbol{x}_t^{st})\boldsymbol{u}_i)^2,$$

that is, we find a rotation that makes the autocovariance matrices defined by a set of lags $\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$ "as diagonal as possible". Different algorithms for approximate joint diagonalization yield naturally different SOBI solutions. The statistical properties of various SOBI estimators are discussed in Miettinen, Nordhausen, Oja, and Taskinen (2014); Illner *et al.* (2015); Miettinen *et al.* (2016) where the performances of the estimators are also compared. In most cases, the SOBI method outperforms the AMUSE method.

For more details about joint diagonalization in general and especially in R, see Miettinen, Nordhausen, and Taskinen (2017) and the references therein. In the following, joint diagonalization will be a reoccuring theme in most discussed methods and we will always use for this purpose an algorithm based on Given's (or Jacobi) rotations as suggested by Clarkson (1988) and implemented in R, for example, in the package **JADE** (Miettinen *et al.* 2017). The joint diagonalization of K symmetric matrices of size $p \times p$ is an operation of complexity $\mathcal{O}(Kp^3)$ (whereas the ordinary eigendecomposition has $K = 1$ and the complexity $\mathcal{O}(p^3)$), and as such some care has to be taken when applying the methods to data sets where the number of variables is measured in hundreds rather than in tens. Moreover, as is common with eigendecomposition-type algorithms, the possibility of numerical issues should be taken into account when working with data sets having p in the high hundreds.

Consider then the BSS model (2) with $q < p$, that is, the model consists of mutually uncorrelated and stationary components as well as of noise components which are not of interest. When using AMUSE, the eigendecomposition of an autocovariance matrix of the standardized vectors $\boldsymbol{x}_t^{st} = Cov(\boldsymbol{x}_t)^{-1/2}\boldsymbol{x}_t$ is now by assumption

$$Cov_\tau(\boldsymbol{x}_t^{st}) = \boldsymbol{U}\tilde{\boldsymbol{\Lambda}}_\tau\boldsymbol{U}^\top = (\boldsymbol{U}_1, \boldsymbol{U}_2)\begin{pmatrix}\boldsymbol{\Lambda}_\tau & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{pmatrix}\begin{pmatrix}\boldsymbol{U}_1^\top\\ \boldsymbol{U}_2^\top\end{pmatrix},$$

where $\boldsymbol{U}_1 \in \mathbb{R}^{p\times q}$, $\boldsymbol{U}_2 \in \mathbb{R}^{p\times(p-q)}$, and $\boldsymbol{\Lambda}_\tau \in \mathbb{R}^{q\times q}$ is a diagonal matrix consisting of the q marginal τ-th autocovariances of the latent series. As noise components do not show any linear autocorrelations, the last $p-q$ diagonal values in $\tilde{\boldsymbol{\Lambda}}_\tau$ are equal to zero. The aim is thus to find a rotation matrix $\boldsymbol{U}_1$ so that the latent source components are given by $\boldsymbol{s}_t = \boldsymbol{U}_1^\top\boldsymbol{x}_t^{st}$ up to sign changes and permutation. In Matilainen, Nordhausen, and Virta (2018), the noise components are identified simply by investigating the diagonal elements of $\tilde{\boldsymbol{\Lambda}}_\tau^2$, where the squaring is used for ordering the sources in decreasing order of interestingness. The noise components are then the last $p-q$ components having zero eigenvalues. For finite samples, the eigenvalues are naturally never equal to zero, and we look for $p-q$ components having "small enough" eigenvalues. In Matilainen *et al.* (2018) a bootstrap-based test for testing the null hypothesis $H_{0k}: \ q = k$ is proposed, and in Virta and Nordhausen (2021) an asymptotic test is given. The signal dimension can then be estimated by testing successively H_{0k} for all possible dimensions $k = 0, \ldots, p-1$ and using the resulting chain of p-values to pin-point the correct dimension. To avoid testing all p hypotheses, one can also use, e.g., a divide-and-conquer approach, see Matilainen *et al.* (2018) for details. Previously similar approaches have been used in the context of independent component analysis (ICA) in Nordhausen, Oja, and Tyler (2017a) and Nordhausen, Oja, Tyler, and Virta (2017b).

The AMUSE method based dimension reduction suffers again from one major drawback. The q components of interest must have non-zero autocovariances with the given lag so that they can be separated from noise components. To overcome this problem, the AMUSE method can be replaced with the SOBI method, which uses K autocovariance matrices in estimation. In order to find all latent source components, it is only required that for each source its autocovariance with one of the lags $\tau \in \mathcal{T}$ must be non-zero. In Matilainen *et al.* (2018) and Virta and Nordhausen (2021), bootstrap and asymptotic tests based on the sum of squared "eigenvalue matrices" $\sum_{\tau\in\mathcal{T}}\tilde{\boldsymbol{\Lambda}}_\tau^2$ are used. Notice that as the SOBI procedure is based on joint diagonalization, matrices $\tilde{\boldsymbol{\Lambda}}_\tau$ are not exact eigenvalue matrices, but obtained as $\tilde{\boldsymbol{\Lambda}} = diag(\boldsymbol{U}^\top Cov_\tau(\boldsymbol{x}_t^{st})\boldsymbol{U})$. Nevertheless, the testing procedure is similar to that used in the case of AMUSE. For a comparison of different approaches for testing the dimension of source subspace, see Matilainen *et al.* (2018) and Virta and Nordhausen (2021).

As already stated, the choice of the lag set $\mathcal{T}$ is of prime importance in the SOBI-based tests as they only recognize signals having autocorrelation on at least one lag in $\mathcal{T}$. As such, $\mathcal{T}$ should be chosen to contain enough lags to identify all the signals, but at the same time be kept small enough, as including too many unnecessary lags can lead into slower computations and noisy estimation in practice. The default set $\{1, \ldots, 12\}$ used in **tsBSS** exhibits this compromise. The previous naturally holds even stronger for AMUSE which allows only for a single lag τ (yielding faster performance in return) and unless expert knowledge on an appropriate lag is available, SOBI should be used instead of AMUSE. The choice of lags for SOBI is for example also discussed in Tang, Liu, and Sutherland (2005); Taskinen, Miettinen, and Nordhausen (2016).

The above idea to use the magnitude of "eigenvalues" to determine which of the components are noise components and which are interesting source components, can also be visually investigated with a scree plot. In such a plot it is however often difficult to decide when an "eigenvalue" is small enough to be related to a noise component. Recently, Luo and Li (2016) extended the scree plot into the ladle plot where, in addition to the information contained in the eigenvalues, also information from the corresponding eigenvectors is used. The main idea in the ladle plot is that the eigenvectors related to noise components span a space and are not well-defined as compared to the eigenvectors related to components with unique signal eigenvalues. Thus measuring this variability using bootstrap sampling and combining it with the information from the eigenvalues gives us a better picture of the number of noise components.

The ladle plot based on AMUSE and SOBI was introduced recently in Nordhausen and Virta (2018). The plots are constructed using stationary time series bootstrapping ideas as follows. Let $\lambda_0^2, \lambda_1^2, \ldots, \lambda_{p-1}^2$ be the (re-indexed) "eigenvalues" in decreasing order and define the normalized function $\phi : \{0, \ldots, p-1\} \to \mathbb{R}$ as

$$\phi_n(k) = \lambda_k^2 \, (1 + \sum_{l=0}^{p-1} \lambda_l^2)^{-1}.$$

Next, let $\hat{\boldsymbol{B}}_k$ contain the first k corresponding columns of the (joint) diagonalizer of AMUSE (SOBI) computed for the original data sample and denote by $\boldsymbol{B}^*_{k,j}$, $j = 1, \ldots, m$, the corresponding quantity for the j-th bootstrap sample. The average variation (around $\hat{\boldsymbol{B}}_k$) of the space spanned by the first k vectors can then be measured as

$$f_n(k) = \frac{1}{m} \sum_{j=1}^{m} \{1 - |\det(\hat{\boldsymbol{B}}_k^\top \boldsymbol{B}^*_{k,j})|\},$$

for $k = 1, \ldots, p-1$ with $f_n(0) := 0$. The ladle function $g_n(k)$ is now obtained simply by summing the functions ϕ_n and f_n,

$$g_n(k) = f_n(k) + \phi_n(k),$$

and visualizing the ladle function then yields the ladle plot. The ladle estimator for the source subspace dimension is the value of k for which $g_n(k)$ is minimal. For further details, the interested reader is referred to Luo and Li (2016) and Nordhausen and Virta (2018).

3. Dimension reduction in the context of volatility clustering

As discussed earlier, AMUSE and SOBI use linear autocovariances in estimation and have proven to be powerful methods when separating uncorrelated linear processes from noise. However, as these two methods treat also components exhibiting volatility clustering as noise, they cannot be used for estimating such time series. If this is our main interest, then extensions of second-order source separation methods are needed.

Matilainen, Nordhausen, and Oja (2015) considered the independent component model for time series (which assumes that the component series in $\boldsymbol{z}_t$ are independent) and proposed two methods which use fourth-order cumulants in estimation: In gFOBI (generalized FOBI), the matrices of fourth cross-moments $\boldsymbol{B}_\tau(\boldsymbol{x}_t) = E[\boldsymbol{x}_{t+\tau}\boldsymbol{x}_t^\top \boldsymbol{x}_t \boldsymbol{x}_{t+\tau}^\top]$ are approximately jointly diagonalized, that is, we find an orthogonal matrix $\boldsymbol{U} \in \mathbb{R}^{p\times p}$ such that it maximizes

$$\sum_{\tau\in\mathcal{T}}\sum_{i=1}^{p}(\boldsymbol{u}_i^\top \boldsymbol{B}_\tau(\boldsymbol{x}_t^{st})\boldsymbol{u}_i)^2.$$

The gFOBI estimator is then given by $\boldsymbol{W} = \boldsymbol{U}^\top Cov(\boldsymbol{x}_t^{st})^{-1/2}$. Notice that by using $\tau = 0$ we obtain the classical FOBI (fourth-order blind identification) method which was proposed already in Cardoso (1989) for solving the independent component analysis problem.

The gJADE (generalized JADE) method, in turn, extends the classical JADE (joint approximate diagonalization of eigenmatrices) method by Cardoso and Souloumiac (1993) to a time series context by using a larger class of fourth-order cumulants in estimation. Write the cross-cumulant matrices as

$$\boldsymbol{C}_\tau^{jk}(\boldsymbol{x}_t) = E[\boldsymbol{x}_{t+\tau}\boldsymbol{x}_t^\top \boldsymbol{E}^{jk}\boldsymbol{x}_t\boldsymbol{x}_{t+\tau}^\top] - Cov_\tau(\boldsymbol{x}_t)(\boldsymbol{E}^{jk}+\boldsymbol{E}^{kj})Cov_\tau(\boldsymbol{x}_t)^\top - trace(\boldsymbol{E}^{jk})\boldsymbol{I}_p,$$

where $\boldsymbol{E}^{jk} = \boldsymbol{e}_j\boldsymbol{e}_k^\top$, $j,k = 1,\ldots,p$ with $\boldsymbol{e}_i$ denoting the vector with a 1 in the i-th coordinate and 0's elsewhere. Then the gJADE solution is given by $\boldsymbol{W} = \boldsymbol{U}^\top Cov(\boldsymbol{x}_t)^{-1/2}$, where orthogonal $\boldsymbol{U} \in \mathbb{R}^{p\times p}$ maximizes

$$\sum_{\tau\in\mathcal{T}}\sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{k=1}^{p}(\boldsymbol{u}_i^\top \boldsymbol{C}_\tau^{jk}(\boldsymbol{x}_t^{st})\boldsymbol{u}_i)^2.$$

Again, the choice $\tau = 0$ reduces to the classical JADE solution.

Another branch of methods designed for separating independent time series exhibiting volatility clustering is based on the use of nonlinearity functions in the estimation. An extension of classical SOBI method was proposed in Matilainen, Miettinen, Nordhausen, Oja, and Taskinen (2017b). Their vSOBI (variant of SOBI) estimate is given by $\boldsymbol{W} = \boldsymbol{U}^\top Cov(\boldsymbol{x}_t^{st})^{-1/2}$, where $\boldsymbol{U} \in \mathbb{R}^{p\times p}$ maximizes

$$\sum_{\tau\in\mathcal{T}}\sum_{i=1}^{p}(E[G(\boldsymbol{u}_i^\top\boldsymbol{x}_t^{st})G(\boldsymbol{u}_i^\top\boldsymbol{x}_{t+\tau}^{st})] - E[G(\boldsymbol{u}_i^\top\boldsymbol{x}_t^{st})]E[G(\boldsymbol{u}_i^\top\boldsymbol{x}_{t+\tau}^{st})])^2,$$

and G can be any twice continuously differentiable function. Commonly used choices for G are $G(y) = y^2$ and $G(y) = \log(\cosh(y))$. The vSOBI method also extends the two FixNA (Fixed-point algorithms for maximizing nonlinear autocorrelation) methods proposed by Hyvärinen (2001) and Shi, Jiang, and Zhou (2009). For the comparisons of gFOBI, gJADE, FixNA and vSOBI, see Matilainen *et al.* (2017b).

The methods recalled above are designed for separating time series which do not show any linear autocorrelations. However, if our interest is in separating uncorrelated stationary time series from components exhibiting volatility clustering, that is, we consider the BSS model (2) with the $p - q$ noise sources actually being components with volatility clustering, the above methods are not applicable. Miettinen, Matilainen, Nordhausen, and Taskinen (2019) combined a second-order source separation method (SOBI) with vSOBI using $G(x) = x^2$. Their gSOBI (generalized SOBI) solution is given as $\boldsymbol{W} = \boldsymbol{U}^\top Cov(\boldsymbol{x}_t^{st})^{-1/2}$, where $\boldsymbol{U} \in \mathbb{R}^{p\times p}$ solves

$$b\sum_{\tau_1\in\mathcal{T}_1}\sum_{i=1}^{p}(\boldsymbol{u}_i Cov_{\tau_1}(\boldsymbol{x}_t^{st})\boldsymbol{u}_i^\top)^2 + (1-b)\sum_{\tau_2\in\mathcal{T}_2}\sum_{i=1}^{p}(E[(\boldsymbol{u}_i^\top\boldsymbol{x}_t^{st})^2(\boldsymbol{u}_i^\top\boldsymbol{x}_{t+\tau_2}^{st})^2]-1)^2.$$

Here $\mathcal{T}_1$ and $\mathcal{T}_2$ are the lag sets for the linear and the quadratic parts, respectively, and $b \in [0, 1]$ assigns the weights for the two parts. The first part of the objective function is designed to find components which are linear processes and the second part components with volatility clustering. In Miettinen *et al.* (2019) the statistical properties of the gSOBI estimator are given, and its performance under different values for b is studied using simulation studies. The value of b should be larger than 0.5 and $b = 0.9$ is suggested for general use.

Miettinen *et al.* (2019) also propose a test for identifying whether there is volatility clustering in the components or not. In order to do this, possible linear autocorrelation of a component needs to be removed first. The presence of linear autocorrelation is measured using a modified version of Ljung-Box test. For the components exhibiting linear autocorrelation, volatility clustering test is performed on ARMA residuals. Finally, the components are ordered according to the value of the test statistic of the volatility clustering test.

The research in Miettinen *et al.* (2019) was motivated by Hu and Tsay (2014) who proposed principal volatility component (PVC) analysis for identifying and ordering components with volatility clustering. With a slight modification, the PVC method can also be seen as a BSS method as it simultaneously diagonalizes two scatter matrices (see Miettinen *et al.* 2019, for more details).

4. Supervised dimension reduction for time series

The methods described above can be considered as unsupervised dimension reduction methods as no specific response, which should be modelled based on $\boldsymbol{x}_t$, was assumed. In supervised dimension reduction it is assumed that we have observed a univariate response time series y_t and that the dimension of $\boldsymbol{x}_t$ should be reduced without losing information about y_t and without knowing the functional relationship between y_t and $\boldsymbol{x}_t$. Supervised dimension reduction methods are well established for iid data, the most popular methods being sliced inverse regression (SIR) (Li 1991) and sliced average variance estimation (SAVE) (Cook 2000). These and other iid supervised dimension reduction methods are implemented, for example, in R in the package **dr** (Weisberg 2002).

Supervised dimension reduction in case of time series data is however much more difficult as the dependence between y_t and $\boldsymbol{x}_t$ may also lag in time. To address this problem Matilainen, Croux, Nordhausen, and Oja (2017a) and Matilainen, Croux, Nordhausen, and Oja (2019) suggested time series versions of SIR and SAVE as well as a weighted linear combination of the two methods. The three approaches are denoted TSIR, TSAVE and TSSH (time series SIR SAVE hybrid), respectively, and recalled in the following. Let now $y = (y_t)_{t=0,\pm1,\pm2,\ldots}$ and $\boldsymbol{x} = (\boldsymbol{x}_t)_{t=0,\pm1,\pm2,\ldots}$ be (weakly and jointly) stationary univariate and p-variate time series, respectively. We then assume that

$$y_t = f(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \ldots; \epsilon_t, \epsilon_{t-1}, \ldots),$$

where function f is unspecified, $\epsilon = (\epsilon_t)_{t=0,\pm1,\pm2,\ldots}$ is an unspecified stationary noise process independent from $\boldsymbol{x}_t$, and the explaining time series $\boldsymbol{x}_t$ follow the BSS model

$$\boldsymbol{x}_t = \boldsymbol{\Omega} \boldsymbol{z}_t + \boldsymbol{\mu},$$

where again $\boldsymbol{\mu} \in \mathbb{R}^p$ is a location vector and $\boldsymbol{\Omega} \in \mathbb{R}^{p \times p}$ is a full rank mixing matrix. The stationary p-variate source time series $\boldsymbol{z}_t$ can be partitioned into $\boldsymbol{z}_t = (\boldsymbol{z}_t^{(1)^\top}, \boldsymbol{z}_t^{(2)^\top})^\top$, where

$\boldsymbol{z}_t^{(1)} \in \mathbb{R}^q$ and $\boldsymbol{z}_t^{(2)} \in \mathbb{R}^{p-q}$. The dimension q denotes the smallest value which fulfills the conditions

(B1) $\mathsf{E}(\boldsymbol{z}_t) = \boldsymbol{0}$ and $\mathrm{COV}(\boldsymbol{z}_t) = \boldsymbol{I}_p$ and

(B2) $(y, \boldsymbol{z}^{(\mathbf{1})^\top})^\top \perp\!\!\!\perp \boldsymbol{z}^{(2)}$.

All the information needed to model y_t is therefore contained in the process $\boldsymbol{z}_t^{(1)}$ and one can write

$$y_t = f(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \ldots; \epsilon_t, \epsilon_{t-1}, \ldots) = f_0(\boldsymbol{z}_t^{(1)}, \boldsymbol{z}_{t-1}^{(1)}, \ldots; \epsilon_t, \epsilon_{t-1}, \ldots) \qquad (3)$$

with another unspecified function f_0, possibly depending on $\boldsymbol{\Omega}$ and $\boldsymbol{\mu}$. As in the unsupervised case, this model is ill-defined in the sense that both $\boldsymbol{z}_t^{(1)}$ and $\boldsymbol{z}_t^{(2)}$ can be multiplied by orthogonal matrices and still fulfill conditions (B1) and (B2). The goal is therefore to find an estimate for an unmixing matrix $\boldsymbol{W}$ such that $\boldsymbol{W}\boldsymbol{x}_t = \boldsymbol{z}_t^{(1)}$ up to an orthogonal transformation. Moreover, one should identify which lagged values $\boldsymbol{z}_t^{(1)}, \boldsymbol{z}_{t-1}^{(1)}, \ldots$ contribute in the model.

TSIR and TSAVE use similar approximate joint diagonalization as the unsupervised methods considered in previous sections. For TSIR Matilainen *et al.* (2017a) suggest to use matrices

$$G_{0,\tau}(\boldsymbol{z}_t, y_t) = Cov(E(\boldsymbol{z}_t | y_{t+\tau})),$$

and for TSAVE the matrices

$$G_{1,\tau}(\boldsymbol{z}_t, y_t) = E((\boldsymbol{I}_p - Cov(\boldsymbol{z}_t | y_{t+\tau}))^2),$$

are suggested in Matilainen *et al.* (2019). The solutions are then given as $\boldsymbol{W} = \boldsymbol{U}^\top Cov(\boldsymbol{x}_t)^{-1/2}$, where the orthogonal matrix $\boldsymbol{U} \in \mathbb{R}^{p \times q}$ maximizes

$$\sum_{\tau \in \mathcal{T}} \sum_{i=1}^{q} (\boldsymbol{u}_i^\top G_{s,\tau}(\boldsymbol{x}_t^{st}, y_t) \boldsymbol{u}_i)^2 \qquad (4)$$

with $s = 0$ for TSIR and $s = 1$ for TSAVE, respectively. To compute the matrices $G_{s,\tau}$ in practice, the response time series y_t is ordered and divided into H slices which are then used to approximate the conditional expectations in $G_{0,\tau}$ and $G_{1,\tau}$.

For a weighted combination of TSIR and TSAVE, TSSH, an orthogonal matrix $\boldsymbol{U} \in \mathbb{R}^{p \times q}$ maximizes

$$\sum_{\tau \in \mathcal{T}} \sum_{i=1}^{q} (\boldsymbol{u}_i^\top ((1-b)\, G_{0,\tau_j}(\boldsymbol{x}_t^{st}, y_t) + b\, G_{1,\tau_j}(\boldsymbol{x}_t^{st}, y_t)) \boldsymbol{u}_i)^2,$$

where $b \in [0, 1]$. Notice that $b = 0$ gives now the TSIR solution and $b = 1$ the TSAVE solution.

In practice the number of sources, that is, the value of q, is unknown and needs to be estimated. Also the important lags regarding the sources need to be found. So far no statistical tests for these are available, however Matilainen *et al.* (2017a, 2019) used the pseudo-eigenvalues

$\lambda_{ij} = \boldsymbol{u}_i^\top G_{s,\tau_j}(\boldsymbol{x}_t^{st}, y_t)\boldsymbol{u}_i$ for determining the value of q as follows. Notice that now $\mathcal{T} = \{\tau_1, \dots, \tau_K\}$. Consider the matrix $\boldsymbol{L} = l_{ij}$ where

$$l_{ij} = \frac{\lambda_{ij}}{\sum_{i=1}^{k}\sum_{j=1}^{K}\lambda_{ij}}, \quad i = 1, \dots, q; \; j = 1, \dots, K,$$

are the scaled pseudo-eigenvalues and the scaling is chosen so that the elements of $\boldsymbol{L}$ add up to 1. Denote the marginal sums of the "eigenvalues" as $\lambda_{i\cdot} = \sum_{j=1}^{K}\lambda_{ij}$ and assume that the unmixing matrix is such that the latent sources are ordered with respect to $\lambda_{1\cdot} \geq \ldots \geq \lambda_{q\cdot}$ and $q = p$. Then Matilainen *et al.* (2017a) suggested four different strategies for finding the appropriate amount of sources and the lags corresponding to the sources by, similarly to the principal component analysis (PCA), trying to explain $100 \times P\%$ of the dependence between the latent sources and the response series. The recommended strategy according to Matilainen *et al.* (2019) is the so called "biggest value" strategy which finds the smallest number r of elements $(i_1, j_1), \dots, (i_r, j_r)$ of $\boldsymbol{L}$ such that $\sum_{k=1}^{r} l_{i_k j_k} \geq P$. For details about the other strategies see Matilainen *et al.* (2017a). For the number of slices Matilainen *et al.* (2019) recommend to use $H = 10$ for TSIR and $H = 5$ for TSAVE.

5. Dimension reduction of multivariate time series in R

5.1. Existing packages

In R, many classes are available for time series data and numerous packages implement modeling multivariate time series, see, for example, the CRAN task view "Time Series Analysis" (`https://CRAN.R-project.org/view=TimeSeries`). Possibly the most comprehensive and general package for multivariate time series modeling is the **MTS** package (Tsay and Wood 2021), offering methods for fitting, among others, multivariate MA, AR, ARMA and multivariate stochastic volatility models, but also dimension reduction methods for time series data. Further R packages that implement unsupervised dimension reduction for time series, mainly through PCA and factor models, include the packages **gdpc** (Peña, Smucler, and Yohai 2021), **PCA4TS** (Chang, Guo, and Yao 2015), **freqdom** (Hormann and Kidzinski 2017) and **tsfa** (Gilbert and Meijer 2005).

R-implementations of numerous blind source separation methods are also available, most of the packages, however, containing mainly BSS methods for iid data. These include, for example, **fICA** (Miettinen, Nordhausen, and Taskinen 2018), **fastICA** (Marchini, Heaton, and Ripley 2019), **ica** (Helwig 2018), **ProDenICA** (Hastie and Tibshirani 2010) and **ICS** (Nordhausen, Oja, and Tyler 2008). The package **JADE** (Miettinen *et al.* 2017) contains, besides iid BSS methods, also AMUSE and SOBI as well as BSS methods for nonstationary time series. The package **BSSasymp** (Miettinen *et al.* 2017) implements the SOBI variant eSOBI (Taskinen *et al.* 2016) and **tensorBSS** (Virta, Koesner, Li, Nordhausen, Oja, and Radojicic 2021) offers BSS approaches for tensor-valued time series. Time series packages which can fit multivariate models in a BSS context by assuming an ICA model are for example **gogarch** (Pfaff 2012) and **rmgarch** (Ghalanos 2019).

5.2. The package tsBSS

The **tsBSS** package (Matilainen *et al.* 2021), which we introduce here, is the most comprehensive package for BSS methods for vector-valued time series. The package **tsBSS** depends on the packages **boot** (Canty and Ripley 2021), **forecast** (Hyndman and Khandakar 2008), **ICtest** (Nordhausen, Oja, Tyler, and Virta 2021), **JADE**, **parallel**, **Rcpp** (Eddelbuettel and François 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014). Most functions in the package assume that the input multivariate time series is either a matrix with p columns and T rows or a corresponding object of class '`mts`', '`xts`' or '`zoo`'.

We next describe the functions implementing the methods discussed in Sections 2, 3 and 4. The AMUSE- and SOBI-based signal dimension testing and estimation methods described in Section 2 are implemented as the functions:

- `AMUSEasymp`, `AMUSEboot`, and `AMUSEladle`
- `SOBIasymp`, `SOBIboot`, and `SOBIladle`

In each function, the user supplies the multivariate time series and the lag(s) to be used. In the hypothesis testing functions (`*asymp` and `*boot`) the signal dimension to be tested is specified using the argument `k`, whereas in the `*ladle` functions the user supplies the maximum number of components to be evaluated as possible signals via the `ncomp` argument. All bootstrapping functions use by default only one core, but offer the possibility of doing parallel computations. Examples on this are shown in Section 6. For the bootstrap hypothesis testing, the user can choose between one parametric and three nonparametric bootstrap versions. For details, see the corresponding help files. For the ladle functions, the user can choose between fixed block bootstrapping and stationary bootstrapping.

The testing functions return objects of class '`ictest`', which is introduced in the package **ICtest** and inherits from class '`htest`', and there are `print`, `components`, `plot`, `ggplot` and `screeplot` methods available for the class. The ladle functions return objects of class '`ladle`', which was again introduced in **ICtest**, and available methods for this class are `print`, `summary`, `components`, `plot`. The actual ladle plot is obtained by calling `ladleplot` or `ggladleplot`.

The functions which implement BSS methods for multivariate time series exhibiting volatility clustering, as described in Section 3, are usually called by the name of the method. Hence, the package has the functions `FixNA`, `gFOBI`, `gJADE`, `PVC` and `vSOBI`. All five functions let the user specify the set of lags to be used. The default is to use the first 12 lags which seems fairly standard in the BSS literature. As there is no natural order of the components, it will usually differ between the methods and the functions usually return the unmixing matrix, the estimated sources, the vector of the lags used and the location of the original series. For financial time series, however, the components of main interest are those which exhibit volatility clustering - therefore all the functions have also the arguments `ordered`, `acfk` and `original`, functioning as follows.

When setting `ordered = TRUE`, the components are ordered according to their degree of volatility clustering as measured by quadratic correlation. For this purpose, `acfk` specifies a vector of lags which will be used to test for serial correlations. Then, if marginal tests of linear correlation detect dependencies at level α, marginal ARMA models will be fitted to each source component. The default is $\alpha = 0.05$ but the user can change it. Then the components whose serial dependence was deemed significant are replaced by their residuals from an ARMA fit using `auto.arima`. For these components, tests of quadratic correlations are performed and the components are ordered according to their test statistics, in order to set the components having the strongest quadratic correlations first. The argument `original`

Type	Name	Output class	Reference
Blind source separation	`gFOBI`	'`bssvol`'	Matilainen *et al.* (2015)
	`gJADE`	'`bssvol`'	Matilainen *et al.* (2015)
	`vSOBI`	'`bssvol`'	Matilainen *et al.* (2017b)
	`FixNA`	'`bssvol`'	FixNA (Shi *et al.* 2009), FixNA2 (Matilainen *et al.* 2017b)
	`gSOBI`	'`bssvol`'	Miettinen *et al.* (2019)
	`PVC`	'`bssvol`'	Hu and Tsay (2014), Miettinen *et al.* (2019)
Dimension estimation	`AMUSEasymp`	'`ictest`' (**ICtest**)	Virta and Nordhausen (2021)
	`SOBIasymp`	'`ictest`' (**ICtest**)	Virta and Nordhausen (2021)
	`AMUSEboot`	'`ictest`' (**ICtest**)	Matilainen *et al.* (2018)
	`SOBIboot`	'`ictest`' (**ICtest**)	Matilainen *et al.* (2018)
	`AMUSEladle`	'`ladle`' (**ICtest**)	Nordhausen and Virta (2018)
	`SOBIladle`	'`ladle`' (**ICtest**)	Nordhausen and Virta (2018)
Autocorrelation testing	`lbtest`	'`lbtest`'	Miettinen *et al.* (2019)
Supervised dimension reduction	`tssdr`	'`tssdr`'	TSIR (Matilainen *et al.* 2017a), TSAVE & TSSH (Matilainen *et al.* 2019)
	`summary.tssdr`	'`summary.tssdr`'	Matilainen *et al.* (2017a)

Table 1: Summary of the functions in **tsBSS** package.

is used to indicate whether the user wants to return the original sources in the desired order or the ordered sources after removing the serial linear correlations. A function for testing the linear and quadratic correlations is also available in the package under the name `lbtest`, see the corresponding help page for details.

In case of `ordered = TRUE`, the output of the stochastic volatility BSS functions will contain many additional features, such as, details of the marginal ARMA fits as well as test statistic values and p values for the marginal linear and quadratic correlation tests. If `original = FALSE`, the source component object `S` naturally does not anymore correspond to the original sources, which are then returned as `Sraw`. The returned object for all the stochastic volatility BSS functions is of the class '`bssvol`', which inherits from the class '`bss`'. Many useful methods such as `components` or `plot` are available for this class.

The previous BSS methods designed for time series with volatility clustering, `FixNA`, `gFOBI`, `gJADE`, `PVC` and `vSOBI`, can deal with linear processes, but the sixth method described in Section 3, `gSOBI`, is clearly a more efficient choice when there are both linear process components and components with volatility clustering. The `gSOBI` function works basically the same way as all other BSS functions mentioned above. In `gSOBI`, however, the user has to specify two sets of lags, one for the "SOBI part" and one for the "vSOBI part". From our experience, it seems that the lag set for the vSOBI part can be much smaller than the lag set for the SOBI part. Furthermore, the user can, via the argument `b`, control how much weight should be given to the SOBI part. The default is `b = 0.9`.

For the supervised dimension reduction methods described in Section 4, the function `tssdr` is available. Via the argument `algorithm`, the user can choose between `"TSIR"` (default), `"TSAVE"` and the hybrid method `"TSSH"`. The other main input arguments are the univariate response series `y` and the multivariate explaining series `X`. The argument `k` controls the lags to

Class	Methods
‘bssvol’	bss.components*, coef*, plot, print
‘lbtest’	print
‘tssdr’	components, plot, print
‘summary.tssdr’	coef, components, plot, print

* inherited from class ‘bss’ (**JADE**)

Table 2: Summary of the methods in classes in **tsBSS** package.

be used and the argument `H` the number of slices. Note that for the TSSH method, different numbers of slices can be used for the TSIR and TSAVE parts by providing a vector of length two. The argument `weight` specifies the weight given to TSAVE. By default, both parts get equal weight. The returned object is of the class ‘`tssdr`’ and the function returns, among other things, the unmixing matrix, the estimated sources and the matrix L as described in Section 4. The most useful methods for an object of this class are `plot` and `summary`. In the `summary` call one can specify, using the argument `type`, one of the four methods to get an indication about which sources and which lags are relevant. The threshold value P needed for this is by default 0.8, but can be changed via the argument `thres`.

The functions and the classes and their methods in **tsBSS** package are summarized in Tables 1 and 2, respectively.

6. Illustrations

6.1. Dimension estimation using AMUSE and SOBI

Following the structure of the paper, we first consider the case where it is assumed that the signals are components with second-order dependencies, and the goal of the analysis is to test hypotheses about the number of signals in order to estimate their number.

For the example, the packages **tsBSS** and **tuneR** (Ligges, Krey, Mersmann, and Schnackenberg 2018) are needed, which we load as follows.

```
R> library("tsBSS")
R> library("tuneR")
```

We create an artificial 20-variate time series by mixing three sound signals available in the **JADE** package with 17 Gaussian white noise processes.

```
R> S1 <- readWave(system.file("datafiles/source5.wav", package = "JADE"))
R> S2 <- readWave(system.file("datafiles/source7.wav", package = "JADE"))
R> S3 <- readWave(system.file("datafiles/source9.wav", package = "JADE"))
R> set.seed(1234)
R> p <- 20
R> NOISE <- matrix(rnorm(50000 * (p - 3)), ncol = p - 3)
R> S <- cbind(S1@left, S2@left, S3@left, NOISE)
R> S <- scale(S, center = FALSE, scale = apply(S, 2, sd))
```

```
R> St <- ts(S, start = 0, frequency = 8000)
R> A <- matrix(runif(p^2, 0, 1), p, p)
R> X <- tcrossprod(St, A)
R> Xt <- ts(X, start = 0, frequency = 8000)
```

Then we use the asymptotic test for SOBI using lags 1 to 10 to test the hypothesis that there are $H_{03}: \; q = 3$ signals. Note that the argument `k` in `SOBIasymp` below is not a tuning parameter but instead specifies which null hypothesis H_{0k} we are interested in testing.

```
R> SOBIasymp(Xt, k = 3, tau = 1:10)

        SOBI test for white noise processes

data:  x
T = 1441.1, df = 1530, p-value = 0.9482
alternative hypothesis: there are fewer than 17 white noise components
```

Thus, the null hypothesis cannot be rejected. Experimenting with different choices of `tau` (not shown here) also gave the same conclusion. For demonstration purposes we test the hypothesis $H_{02}: \; q = 2$ using nonparametric bootstrap with the third bootstrap variant `"np3"` which allows for dependence between the noise components. For further details about the different bootstrap strategies see the help files for `AMUSEboot` and `SOBIboot`.

To speed up the bootstrap computations we use 3 cores and fix for reproducibility the seeds using the `iseed` argument.

```
R> SOBIboot(Xt, k = 2, tau = 1:10, s.boot = "np3", ncores = 3, iseed = 3333)

        ICA subwhite noise bootstrapping test using SOBI and strategy
        np3

data:  x
T = 3886.3, replications = 200, p-value = 0.004975
alternative hypothesis: the last 18 components are not white noise
```

The bootstrap test clearly rejects the null hypothesis which verifies, when combined with the first test, that there are three signals.

To directly estimate the number of signals, the Ladle plot is a natural starting point. We use again SOBI with lags 1 to 10. As `SOBIladle` uses `ts.boot` from the **boot** package (Canty and Ripley 2021) also this can be computed in parallel, by passing the arguments `parallel = "multicore"` and `ncpus` to specify the number of cores. However, as bootstrapping is done separately for each suspected number of dimensions, it is advisable to set the maximum number of components that should be evaluated in order to save computation time. Here we say the maximum is 12 via the `ncomp` argument. The argument `l = 40` specifies that we use stationary bootstrap with expected block lengths of 40.

```
R> SL <- SOBIladle(Xt, tau = 1:10, l = 40, ncomp = 12,
+    parallel = "multicore", ncpus = 3)
```

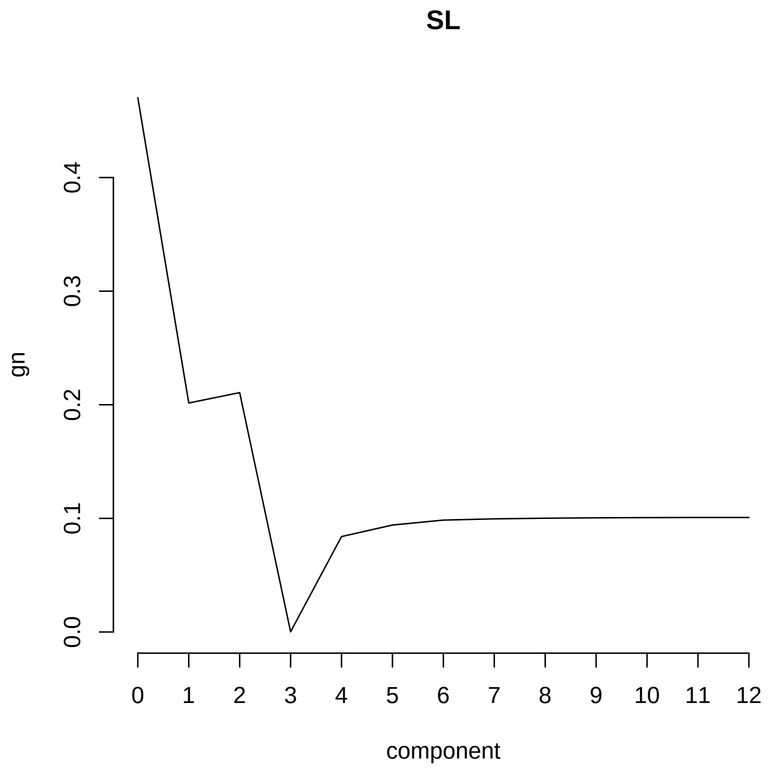

Figure 1: Ladle plot for the sound example data.

The plot is then obtained as

```
R> ladleplot(SL)
```

and shown in Figure 1 where the position of the minimum again verifies that there are three signals in this time series. To see the three signals one can use

```
R> plot(SL, which = "k")
```

where `which = "k"` specifies that one does not want to plot all sources but only those corresponding to the estimated number of signals. The time courses of the three signals are shown in Figure 2.

6.2. Dimension reduction in the context of stochastic volatility

In financial time series the second order correlations are often not of interest and the focus is on the stochastic volatility features of the data. Our preferred method in this context is gSOBI and we will demonstrate its functionality using the exchange rate dataset from **stochvol** package (Kastner 2016), which contains daily bilateral prices of one Euro in 23 currencies. The daily measurements are from January 3, 2000, until April 4, 2012.

In addition to the **tsBSS** package we use for this example also the packages **stochvol** (Kastner 2016) and **zoo** (Zeileis and Grothendieck 2005). For reproducibility purposes for this example also a seed is set.

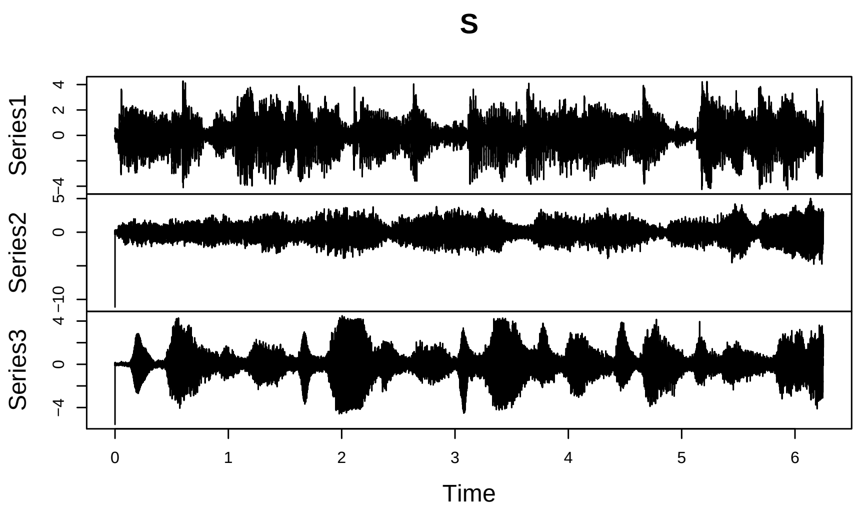

Figure 2: The estimated signals from sound example data based on the ladle estimate using SOBI.

```
R> library("tsBSS")
R> library("zoo")
R> library("stochvol")
R> data("exrates", package = "stochvol")
R> set.seed(1234)
```

We apply the gSOBI method to the logarithmic returns of the original variables using the preferred weight `b = 0.9` for the "SOBI" part. The resulting sources will be ordered according to their volatility clustering level which we evaluate using quadratic autocorrelations after removing componentwise second order autocorrelations by fitting automatically ARMA to those components which exhibit linear autocorrelation. The argument `acfk = 1:5` specifies that the first five lags should be used to test for linear and quadratic autocorrelation, respectively.

```
R> exrlogr <- zoo(apply(exrates[, -(ncol(exrates))], 2, logret))
R> attr(exrlogr, "index") <- as.Date(rownames(exrlogr))
R> gSOBIwrd <- gSOBI(exrlogr, b = 0.9, ordered = TRUE, acfk = 1:5,
+    original = FALSE)
```

The 23 raw sources can be plotted as follows

```
R> plot(gSOBIwrd$Sraw, main = "The raw sources", xlab = "Time")
```

However, due to the dimensionality of the data the figure is not included here. To evaluate which components exhibit serial autocorrelations we look at the `linP` object which returns the marginal p values based on the modified Ljung-Box test. The `armaeff` object is a binary indicator indicating which source components exhibit serial autocorrelation.

```
R> round(gSOBIwrd$linP, 4)
```

```
 [1] 0.4991 0.1480 0.0818 0.7412 0.5197 0.3284 0.3159 0.7355 0.0816
[10] 0.0546 0.3886 0.0740 0.0231 0.1486 0.0618 0.0292 0.2741 0.4634
[19] 0.1543 0.0146 0.1054 0.8372 0.2959
```

```
R> gSOBIwrd$armaeff
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[12] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
[23] FALSE
```

In our example, there are three components that exhibit significant serial linear autocorrelation. The sources of interest, in which the linear autocorrelation has been removed, can then be plotted with the command

```
R> plot(bss.components(gSOBIwrd), main = "The sources", xlab = "Time")
```

However, for space reasons we again refrain from showing the figure here. The next step in the analysis is typically to study volatility at each time point. To estimate these quantites, assuming a stochastic volatility (SV) model, the package **stochvol** offers the function `svsample`. This function uses Markov chain Monte Carlo (MCMC) for the estimation making this quite time consuming. Therefore for demonstration purpose we focus on the three components which are considered to have the most volatility clustering and the three components considered to have the least volatility clustering. For convenience we write a wrapper around `svsample` to extract only the quantity of interest.

```
R> svsample_exph <- function(x) {
+    svsample(x)$summary$latent[, 6]
+  }
R> #estimating the volatilities using MCMC sampling
R> ExpH_gSOBI <- apply(bss.components(gSOBIwrd)[, c(1:3, 21:23)], 2,
+    svsample_exph)
R> ExpH_gSOBI <- zoo(ExpH_gSOBI, order.by = as.Date(rownames(exrlogr)))
R> plot(ExpH_gSOBI, ylim = c(0, 8), main = "Volatility series of components
+    with the most and the least volatility clustering")
```

Figure 3 shows that the last components have clearly much less volatility clustering compared to the first components. To see if the components have significant marginal volatility clustering, the corresponding p values based on a modified Ljung-Box test are available via

```
R> round(gSOBIwrd$volP, 4)
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Based on these results all the components seem to have significant volatility clustering. It is however obvious that the degree of volatility clustering differs considerable between the components and that the first components are much more interesting that the last components.

As a comparison, we perform the same analysis using the (modified) principal volatility component (PVC) analysis. We first look at the p values for the linear and quadratic autocorrelations.

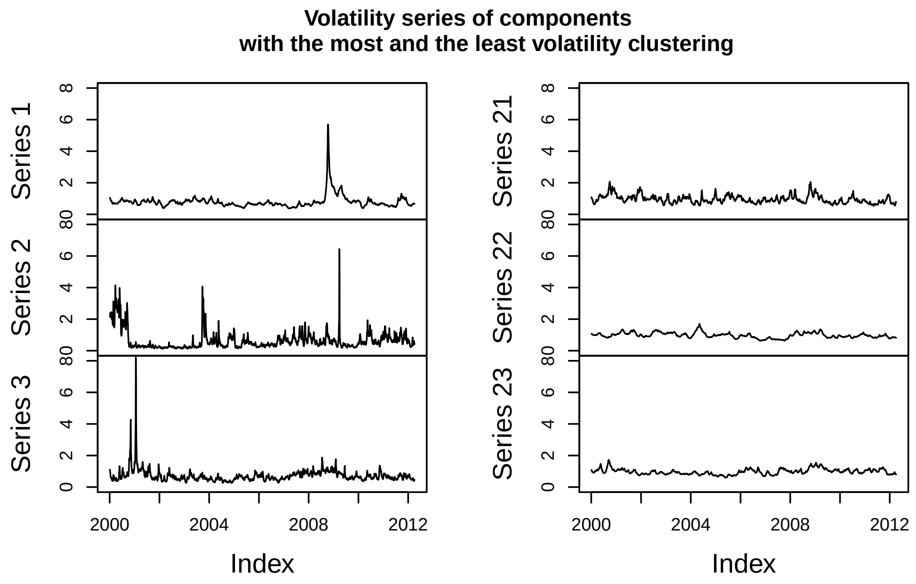

Figure 3: Volatility series of three components with the most volatility clustering and three components with the least volatility clustering extracted using gSOBI.

```
R> PVCwrd <- PVC(exrlogr, ordered = TRUE, acfk = 1:5, original = FALSE)
R> round(PVCwrd$linP, 4)

 [1] 0.5172 0.1416 0.8147 0.0337 0.6906 0.0068 0.7410 0.2194 0.1110
[10] 0.4026 0.2933 0.2200 0.4044 0.1424 0.1574 0.0967 0.2094 0.0000
[19] 0.2863 0.5095 0.0227 0.1612 0.3487

R> PVCwrd$armaeff

 [1] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
[12] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
[23] FALSE

R> round(PVCwrd$volP, 4)

 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Four components seem to have linear autocorrelation and that is removed in order to estimate the volatility of the sources.

Then we compute again the marginal volatilities based on the SV model as in the gSOBI case. The resulting volatility components using PVC are in Figure 4.

```
R> ExpH_PVC <- apply(bss.components(PVCwrd)[, c(1:3, 21:23)], 2, svsample_exph)
R> ExpH_PVC <- zoo(ExpH_PVC, order.by = as.Date(rownames(exrlogr)))
R> plot(ExpH_PVC, ylim = c(0, 8), main = "Volatility series of components
+    with the most and the least volatility clustering")
```

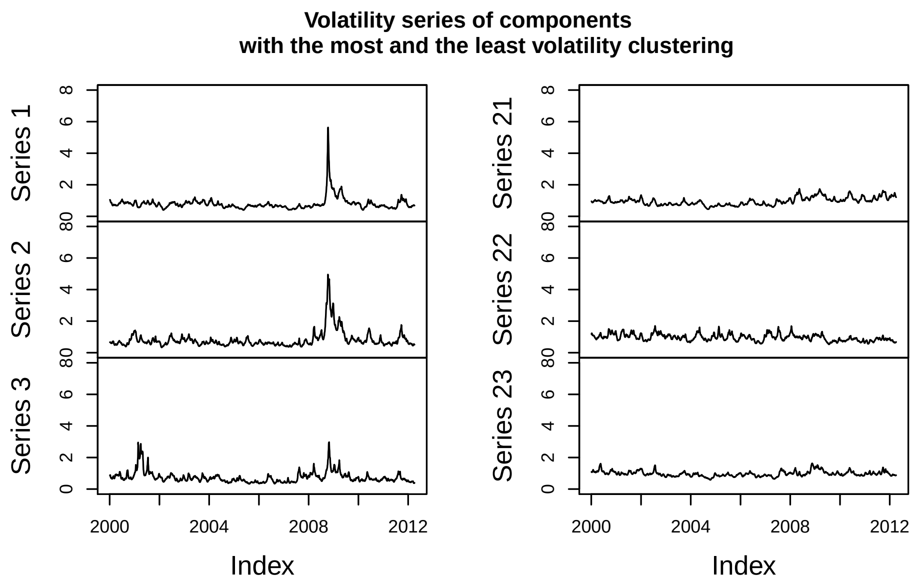

Figure 4: Volatility series of components with the most volatility clustering and three components with the least volatility clustering extracted using PVC.

Finally, we compare the gSOBI and the PVC results to corresponding principal components computed with standard principal component analysis (PCA) which ignores the time structure and is thus not targeting volatility clustering effects. As gSOBI and PVC standardize the components to have variance one, we do so here for the obtained PCs as well.

```
R> PCA <- princomp(exrlogr)
R> PCst <- scale(PCA$scores)
```

After that we remove linear autocorrelations

```
R> linear <- lbtest(PCst, 1:5, "linear")
R> armaeff <- (linear$p_val < 0.05)
R> p <- ncol(PCst)
R> PCst2 <- PCst
R> index <- (1:p)[armaeff]
R> for (j in index) {
+    PCst2[, j] <- forecast::auto.arima(PCst[,j], stationary = TRUE,
+    seasonal = FALSE)$residuals
+ }
R> PCst2 <- zoo(PCst2, order.by = as.Date(rownames(exrlogr)))
```

The components are then ordered according to the volatility test statistic Q. The volatility series of components with the most and the least volatility clustering are plotted in Figure 5.

```
R> vol <- lbtest(PCst2, 1:5, "squared")
R> ordered <- order(vol$TS, decreasing = TRUE)
R> PCst3 <- PCst2[, ordered]
```

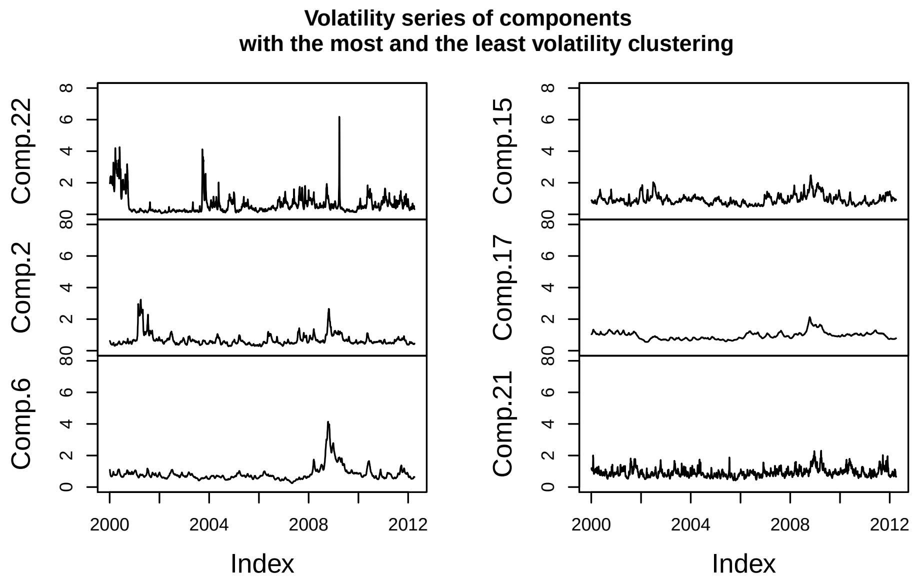

Figure 5: Volatility series of components with the most volatility clustering and three components with the least volatility clustering extracted using PCA.

```
R> ExpH_PCA <- apply(PCst3[, c(1:3, 21:23)], 2, svsample_exph)
R> ExpH_PCA <- zoo(ExpH_PCA, order.by = as.Date(rownames(exrlogr)))
R> plot(ExpH_PCA, ylim = c(0, 8), main = "Volatility series of components
+    with the most and the least volatility clustering")
```

As before, all the components have volatility clustering, but it seems to be higher in the components with the least volatility clustering compared to the volatility series based on gSOBI or PVC. Note that the ordering of the volatility clustering is not related to the order of the PCs. Using for example only a few first PCs might have missed the components exhibiting a lot of volatility clustering. The most volatile component (Comp. 22) is the one with the second lowest variance.

To further compare compare to gSOBI and PVC, note that for PCA the volatility clustering test statistic values are much less dispersed, i.e., even the ones with the lowest test statistic value tend to have more volatility clustering. This is a disadvantage when trying to uncover possible components without volatility clustering.

```
R> round(gSOBIwrd$volTS) #gSOBI

 [1] 2733209 1131552 1056249  695381  166247  157372  156301   86093
 [9]   52503   51997   36702   21050   10577    6927    6095    4237
[17]    2468    2128    1382     764     666      98      86
```

```
R> round(PVCwrd$volTS) #PVC

 [1] 2587720  177967   86407   79737   33781   29551   23391   10914
 [9]   10848    8419    3667    2653    2349    2023    1140    1028
[17]     634     567     524     504     409     391     184
```

```
R> round(vol$TS[ordered]) #PCA

 [1] 803437 234315  67806  45408  27815  23148  22255  21614  13381
[10]  13019  10258   5872   5505   4702   4354   3873   2187   1964
[19]   1702   1657   1416    998    595
```

6.3. Supervised dimension reduction for time series

To conclude the illustration of the **tsBSS** package we demonstrate how to perform TSIR in a supervised dimension reduction context. For that purpose we use the `PRSA` data set (Liang *et al.* 2015) which is available from the UCI Machine Learning Repository (Dheeru and Karra Taniskidou 2017) and can be loaded into R as

```
R> PATH <- paste0("https://archive.ics.uci.edu/ml/machine-learning",
+    "-databases/00381/PRSA_data_2010.1.1-2014.12.31.csv")
R> PRSA <- read.csv(PATH)
```

The dataset includes hourly measurements of fine particulate matter, PM2.5 concentrations, taken at the US Embassy in Beijing, and some meteorological data from Beijing Capital International Airport. The whole dataset is from January 2010 to December 2014. The goal is here to predict the PM2.5 concentration using as predicting time series the hourly measurements of `DEWP` (dew point; ℃), `TEMP` (temperature; ℃), `PRES` (air pressure; hPa), `Iws` (cumulated wind speed; m/s), `Is` (cumulated hours of snow) and `Ir` (cumulated hours of rain). For a complete description of the data set, see Liang *et al.* (2015) and `https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data`.

For simplicity we use the data from March 2013 to October 2014 and impute the few missing values using the package **imputeTS** (Moritz and Bartz-Beielstein 2017) with a simple moving average.

```
R> library("imputeTS")
R> response <- na_ma(PRSA[27721:42360, 6], k = 4, "simple")
R> predictors <- PRSA[27721:42360, -c(1:6, 10)]
```

The response time series in Figure 6 can be plotted as

```
R> plot(ts(response), ylab = "PM 2.5 concentration", xlab = "Time in hours")
```

As a preparation step we make the predicting time series more stationary by taking the first order differences. For the variables `Iws`, `Is` and `Ir` this is done after taking logarithms where, if needed, one was added to make the counts all positive. Also all the time series are then centered and scaled.

```
R> predictors$Iws <- log(predictors$Iws)
R> predictors$Ir <- log(predictors$Ir + 1)
R> predictors$Is <- log(predictors$Is + 1)
R> X <- scale(diff(as.matrix(predictors)))
R> y <- scale(response[-1])
```

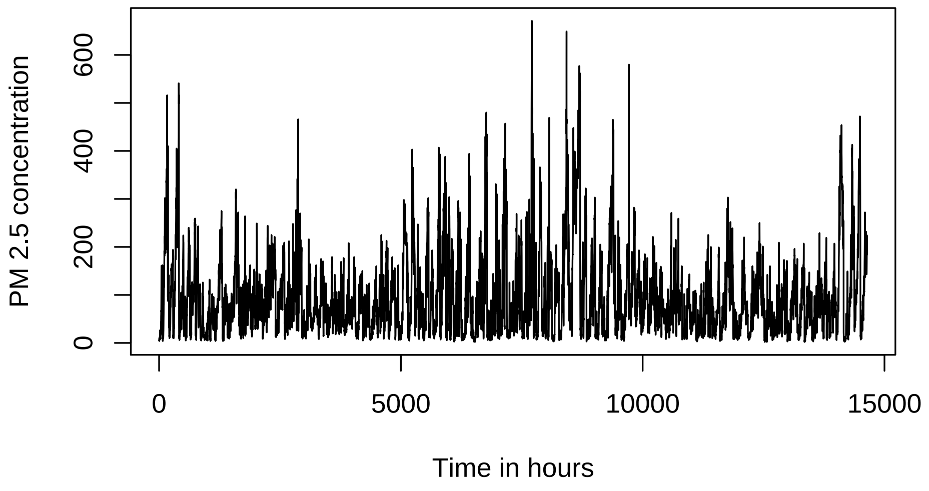

Figure 6: The hourly measured PM 2.5 concentration values.

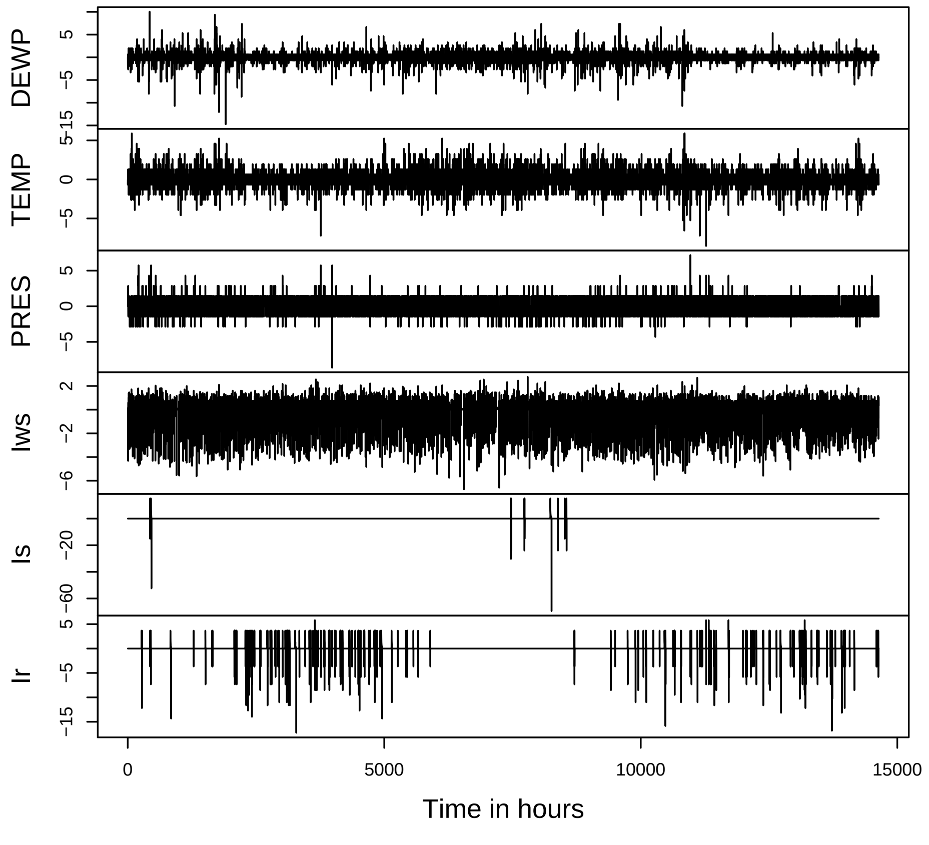

Figure 7: The six times series which should be used to explain the PM 2.5 concentration values.

The six explaining time series are then plotted in Figure 7 with the command

```
R> plot(ts(X), xlab = "Time in hours", main = "", nc = 1)
```

To perform supervised dimension reduction we use the `tssdr` function, which has TSIR as its default. We choose to compute TSIR using the first 12 lags with 10 slices.

```
R> library("tsBSS")
R> TSIR <- tssdr(y, X, H = 10, algorithm = "TSIR", k = 1:12)
```

We could check all the estimated sources using `components(TSIR)`. However, the most convenient way to work with the `TSIR` object, which is of class '`tssdr`', is to use the `summary` function, where one can specify the method and threshold to investigate which directions and lags might be relevant. Among the strategies for choosing the directions and the lags, we choose the biggest values method with a threshold of 0.8,

```
R> TSIRbiggest <- summary(TSIR, thres = 0.8, type = "big")
```

To see the most important results we can just use

```
R> TSIRbiggest

        Summary of TSIR for response y and predictors X

The signal separation matrix W is:

       [,1]  [,2]  [,3]   [,4]    [,5]    [,6]
[1,] -0.652 0.167 0.751 0.0401 -0.0439 -0.0229

The L matrix is:

        Dir.1   Dir.2    Dir.3    Dir.4    Dir.5    Dir.6
Lag 1  0.0251 0.00509 0.008412 0.000396 0.001478 0.000370
Lag 2  0.0412 0.00746 0.006506 0.000663 0.001076 0.000910
Lag 3  0.0578 0.00924 0.003978 0.001520 0.000610 0.000664
Lag 4  0.0725 0.00804 0.002212 0.002298 0.000368 0.000435
Lag 5  0.0881 0.00585 0.000769 0.001099 0.000546 0.001516
Lag 6  0.0949 0.00450 0.000642 0.001324 0.001223 0.000793
Lag 7  0.1021 0.00255 0.002390 0.001082 0.000578 0.000348
Lag 8  0.0990 0.00345 0.001567 0.000614 0.000873 0.000648
Lag 9  0.0844 0.00801 0.001232 0.000270 0.000949 0.000762
Lag 10 0.0678 0.01147 0.001111 0.000915 0.001054 0.000429
Lag 11 0.0558 0.01834 0.001199 0.000650 0.000540 0.000359
Lag 12 0.0453 0.02009 0.002296 0.000811 0.000617 0.000816

Using the biggest values in L to choose the directions and lags:

The relevant combinations of lags and directions are
```

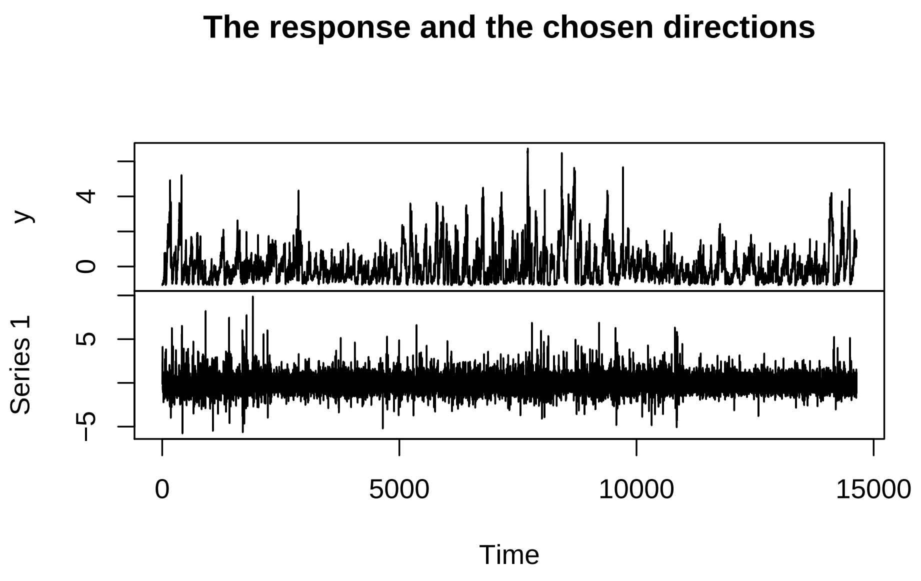

Figure 8: The hourly PM 2.5 concentration values time series together with the component considered relevant by TSIR.

```
      Lag Dir.
 [1,]   7    1
 [2,]   8    1
 [3,]   6    1
 [4,]   5    1
 [5,]   9    1
 [6,]   4    1
 [7,]  10    1
 [8,]   3    1
 [9,]  11    1
[10,]  12    1
[11,]   2    1
```

Following the biggest value strategy, one linear combination of predicting time series seems sufficient to model the response. The weights that the linear combination of interest gives to each variable can be seen from the matrix `W` which can be extracted using `coef(TSIRbiggest)`. The weights corresponding to the differentiated variables of `Iws`, `Is` and `Ir` are very small, whereas the differentiated variables of `DEWP` and `PRES` have the largest impact on the linear combination.

From the last part of the `summary` output it seems also that the most relevant thing is what happened 6–8 hours before as these lags have the largest information content. If the air pressure has gone up and the dew point has gone down during the last 6-8 hours, which might be an indication of drier weather, which would have an effect on how long the pollution particles stay in the air.

The first few values of the chosen source can be printed as

```
R> head(components(TSIRbiggest))
```

```
Time Series:
Start = 1
End = 6
Frequency = 1
[1] -0.1036230  0.8793015  2.2863857  0.6471331  0.6461821  0.5362510
```

The response and the chosen sources can be obtained using `bss.components(TSIRbiggest)`. Using `plot` one obtains a figure with the response time series and the chosen interesting components as shown in Figure 8.

```
R> plot(TSIRbiggest)
```

In practice now a good model for the source components should be found and the relationship between the response and the source component should be explored. This is however beyond the scope of this example.

7. Summary and discussion

We presented the R package **tsBSS** containing implementations of several BSS methods aimed for time series data. Dimension reduction through BSS is an attractive option for the first step in multivariate data analysis, helping avoid the cumbersome modelling of complex dependencies and, in the best case, ridding the data altogether of noise. In its current form the package contains tools for dimension reduction of both second-order stationary models and for time series exhibiting stochastic volatility. It is also the first package providing tools for supervised dimension reduction in a time series context.

Acknowledgments

The work of KN was supported by the CRoNoS COST Action IC1408 and the Austrian Science Fund P31881-N32. The work of ST was supported by the CRoNoS COST Action IC1408. The work of JV was supported by Academy of Finland (grant 321883). We would like to thank the anonymous reviewers for their comments which improved the paper and package considerably.

References

Belouchrani A, Abed-Meraim K, Cardoso JF, Moulines E (1997). "A Blind Source Separation Technique Using Second-Order Statistics." *IEEE Transactions on Signal Processing*, **45**(2), 434–444. doi:10.1109/78.554307.

Bianchi FM, Livi L, Mikalsen KØ, Kampffmeyer M, Jenssen R (2019). "Learning Representations of Multivariate Time Series with Missing Data." *Pattern Recognition*, **96**, 106973. doi:10.1016/j.patcog.2019.106973.

Broda SA, Paolella MS (2009). "CHICAGO: A Fast and Accurate Method for Portfolio Risk Calculation." *Journal of Financial Econometrics*, **7**(4), 412–436. doi:10.1093/jjfinec/nbp011.

Canty A, Ripley BD (2021). ***boot**: Bootstrap Functions (Originally by Angelo Canty for S)*. R package version 1.3-28, URL https://CRAN.R-project.org/package=boot.

Cardoso JF (1989). "Source Separation Using Higher Order Moments." In *International Conference on Acoustics, Speech, and Signal Processing, 1989*, pp. 2109–2112. doi:10.1109/icassp.1989.266878.

Cardoso JF, Souloumiac A (1993). "Blind Beamforming for Non-Gaussian Signals." *IEE Proceedings F (Radar and Signal Processing)*, **140**(6), 362–370. doi:10.1049/ip-f-2.1993.0054.

Chang J, Guo B, Yao Q (2015). ***PCA4TS**: Segmenting Multiple Time Series by Contemporaneous Linear Transformation*. R package version 0.1, URL https://CRAN.R-project.org/package=PCA4TS.

Chang J, Guo B, Yao Q (2018). "Principal Component Analysis for Second-Order Stationary Vector Time Series." *The Annals of Statistics*, **46**, 2094–2124. doi:10.1214/17-aos1613.

Cheung VCK, Devarajan K, Severini G, Turolla A, Bonato P (2015). "Decomposing Time Series Data by a Non-Negative Matrix Factorization Algorithm with Temporally Constrained Coefficients." In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 3496–3499. IEEE. doi:10.1109/embc.2015.7319146.

Clarkson DB (1988). "A Least Squares Version of Algorithm AS 211: The FG Diagonalization Algorithm." *Journal of the Royal Statistical Society C*, **37**(2), 317–321. doi:10.2307/2347359.

Comon P, Jutten C (2010). *Handbook of Blind Source Separation: Independent Component Analysis and Applications*. Academic Press. doi:10.1016/c2009-0-19334-0.

Cook RD (2000). "SAVE: A Method for Dimension Reduction and Graphics in Regression." *Communications in Statistics – Theory and Methods*, **29**, 2109–2121. doi:10.1080/03610920008832598.

Dheeru D, Karra Taniskidou E (2017). "UCI Machine Learning Repository." URL http://archive.ics.uci.edu/ml.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063. doi:10.1016/j.csda.2013.02.005.

Ensor KB (2013). "Time Series Factor Models." *Wiley Interdisciplinary Reviews: Computational Statistics*, **5**(2), 97–104. doi:10.1002/wics.1245.

Févotte C, Smaragdis P, Mohammadiha N, Mysore GJ (2018). "Temporal Extensions of Nonnegative Matrix Factorization." In *Audio Source Separation and Speech Enhancement*, pp. 161–187. John Wiley & Sons. doi:10.1002/9781119279860.ch9.

Ghalanos A (2019). ***rmgarch**: Multivariate GARCH Models.* R package version 1.3-7, URL https://CRAN.R-project.org/package=rmgarch.

Gilbert PD, Meijer E (2005). "Time Series Factor Analaysis with an Application to Measuring Money." *Technical Report 05F10*, University of Groningen, SOM Research School. URL https://research.rug.nl/en/publications/time-series-factor-analysis-with-an-application-to-measuring-mone-2.

Hastie T, Tibshirani R (2010). ***ProDenICA**: Product Density Estimation for ICA Using Tilted Gaussian Density Estimates.* R package version 1.0, URL https://CRAN.R-project.org/package=ProDenICA.

Helwig NE (2018). ***ica**: Independent Component Analysis.* R package version 1.0-2, URL https://CRAN.R-project.org/package=ica.

Hormann S, Kidzinski L (2017). ***freqdom**: Frequency Domain Based Analysis: Dynamic PCA.* R package version 2.0.1, URL https://CRAN.R-project.org/package=freqdom.

Hu YP, Tsay RS (2014). "Principal Volatility Component Analysis." *Journal of Business and Economic Statistics*, **32**, 153–164. doi:10.1080/07350015.2013.818006.

Hyndman RJ, Khandakar Y (2008). "Automatic Time Series Forecasting: The **forecast** Package for R." *Journal of Statistical Software*, **26**(3), 1–22. doi:10.18637/jss.v027.i03.

Hyvärinen A (2001). "Blind Source Separation by Nonstationarity of Variance: A Cumulant-Based Approach." *IEEE Transactions on Neural Networks*, **12**(6), 1471–1474. doi:10.1109/72.963782.

Illner K, Miettinen J, Fuchs C, Taskinen S, Nordhausen K, Oja H, Theis FJ (2015). "Model Selection Using Limiting Distributions of Second-Order Blind Source Separation Algorithms." *Signal Processing*, **113**, 95–103. doi:10.1016/j.sigpro.2015.01.017.

Kastner G (2016). "Dealing with Stochastic Volatility in Time Series Using the R Package **stochvol**." *Journal of Statistical Software*, **69**(5), 1–30. doi:10.18637/jss.v069.i05.

Li KC (1991). "Sliced Inverse Regression for Dimension Reduction." *Journal of the American Statistical Association*, **86**(414), 316–327. doi:10.1080/01621459.1991.10475035.

Liang X, Zou T, Guo B, Li S, Zhang H, Zhang S, Huang H, Chen SX (2015). "Assessing Beijing's PM2.5 Pollution: Severity, Weather Impact, APEC and Winter Heating." *Proceedings of the Royal Society of London A*, **471**(2182). doi:10.1098/rspa.2015.0257.

Ligges U, Krey S, Mersmann O, Schnackenberg S (2018). ***tuneR**: Analysis of Music and Speech.* R package version 1.3.3, URL https://CRAN.R-project.org/package=tuneR.

Luo W, Li B (2016). "Combining Eigenvalues and Variation of Eigenvectors for Order Determination." *Biometrika*, **103**(4), 875–887. doi:10.1093/biomet/asw051.

Marchini JL, Heaton C, Ripley BD (2019). ***fastICA**: FastICA Algorithms to Perform ICA and Projection Pursuit.* R package version 1.2-2, URL `https://CRAN.R-project.org/package=fastICA`.

Matilainen M, Croux C, Miettinen J, Nordhausen K, Oja H, Taskinen S (2021). ***tsBSS**: Blind Source Separation and Supervised Dimension Reduction for Time Series.* R package version 1.0.0, URL `https://CRAN.R-project.org/package=tsBSS`.

Matilainen M, Croux C, Nordhausen K, Oja H (2017a). "Supervised Dimension Reduction for Multivariate Time Series." *Econometrics and Statistics*, **4**, 57–69. `doi:10.1016/j.ecosta.2017.04.002`.

Matilainen M, Croux C, Nordhausen K, Oja H (2019). "Sliced Average Variance Estimation for Multivariate Time Series." *Statistics*, **53**(3), 630–655. `doi:10.1080/02331888.2019.1605515`.

Matilainen M, Miettinen J, Nordhausen K, Oja H, Taskinen S (2017b). "On Independent Component Analysis and Stochastic Volatility Models." *Austrian Journal of Statistics*, **46**, 57–66. `doi:10.17713/ajs.v46i3-4.671`.

Matilainen M, Nordhausen K, Oja H (2015). "New Independent Component Analysis Tools for Time Series." *Statistics & Probability Letters*, **105**, 80–87. `doi:10.1016/j.spl.2015.04.033`.

Matilainen M, Nordhausen K, Virta J (2018). "On the Number of Signals in Multivariate Time Series." In Y Deville, S Gannot, R Mason, MD Plumbley, D Ward (eds.), *International Conference on Latent Variable Analysis and Signal Separation*, pp. 248–258. Springer-Verlag. `doi:10.1007/978-3-319-93764-9_24`.

Matteson DS, Tsay RS (2011). "Dynamic Orthogonal Components for Multivariate Time Series." *Journal of the American Statistical Association*, **106**(496), 1450–1463. `doi:10.1198/jasa.2011.tm10616`.

Miettinen J, Illner K, Nordhausen K, Oja H, Taskinen S, Theis FJ (2016). "Separation of Uncorrelated Stationary Time Series Using Autocovariance Matrices." *Journal of Time Series Analysis*, **37**(3), 337–354. `doi:10.1111/jtsa.12159`.

Miettinen J, Nordhausen K, Oja H, Taskinen S (2012). "Statistical Properties of a Blind Source Separation Estimator for Stationary Time Series." *Statistics & Probability Letters*, **82**(11), 1865–1873. `doi:10.1016/j.spl.2012.06.025`.

Miettinen J, Nordhausen K, Oja H, Taskinen S (2014). "Deflation-Based Separation of Uncorrelated Stationary Time Series." *Journal of Multivariate Analysis*, **123**, 214–227. `doi:10.1016/j.jmva.2013.09.009`.

Miettinen J, Nordhausen K, Taskinen S (2017). "Blind Source Separation Based on Joint Diagonalization in R: The Packages **JADE** and **BSSasymp**." *Journal of Statistical Software*, **76**(2), 1–31. `doi:10.18637/jss.v076.i02`.

Miettinen J, Nordhausen K, Taskinen S (2018). "**fICA**: FastICA Algorithms and Their Improved Variants." *The R Journal*, **10**(2), 148–158. `doi:10.32614/rj-2018-046`.

Miettinen M, Matilainen M, Nordhausen K, Taskinen S (2019). "Extracting Conditionally Heteroskedastic Components Using Independent Component Analysis." *Journal of Time Series Analysis*. doi:10.1111/jtsa.12505.

Moritz S, Bartz-Beielstein T (2017). "**imputeTS**: Time Series Missing Value Imputation in R." *The R Journal*, **9**(1), 207–218. doi:10.32614/rj-2017-009.

Nordhausen K, Oja H (2018). "Independent Component Analysis: A Statistical Perspective." *Wiley Interdisciplinary Reviews: Computational Statistics*, p. e1440. doi:10.1002/wics.1440.

Nordhausen K, Oja H, Tyler DE (2008). "Tools for Exploring Multivariate Data: The Package **ICS**." *Journal of Statistical Software*, **28**(6), 1–31. doi:10.18637/jss.v028.i06.

Nordhausen K, Oja H, Tyler DE (2017a). "Asymptotic and Bootstrap Tests for Subspace Dimension." arXiv:1611.04908v2 [stat.ME], URL https://arxiv.org/abs/1611.04908v2.

Nordhausen K, Oja H, Tyler DE, Virta J (2017b). "Asymptotic and Bootstrap Tests for the Dimension of the Non-Gaussian Subspace." *IEEE Signal Processing Letters*, **24**(6), 887–891. doi:10.1109/lsp.2017.2696880.

Nordhausen K, Oja H, Tyler DE, Virta J (2021). ***ICtest**: Estimating and Testing the Number of Interesting Components in Linear Dimension Reduction.* R package version 0.3-4, URL https://CRAN.R-project.org/package=ICtest.

Nordhausen K, Virta J (2018). "Ladle Estimator for Time Series Signal Dimension." In *2018 IEEE Statistical Signal Processing Workshop (SSP)*, pp. 428–432. doi:10.1109/SSP.2018.8450695.

Peña D, Smucler E, Yohai V (2021). ***gdpc**: Generalized Dynamic Principal Components.* R package version 1.1.2, URL https://CRAN.R-project.org/package=gdpc.

Pfaff B (2012). ***gogarch**: Generalized Orthogonal GARCH (GO-GARCH) Models.* R package version 0.7-2, URL https://CRAN.R-project.org/package=gogarch.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Shi Z, Jiang Z, Zhou F (2009). "Blind Source Separation with Nonlinear Autocorrelation and Non-Gaussianity." *Journal of Computational and Applied Mathematics*, **223**(1), 908–915. doi:10.1016/j.cam.2008.03.009.

Stock J, Watson M (2010). "Dynamic Factor Models." In MP Clements, DF Henry (eds.), *Oxford Handbook of Economic Forecasting*. Oxford University Press.

Tang AC, Liu JY, Sutherland MT (2005). "Recovery of Correlated Neuronal Sources from EEG: The Good and Bad Ways of Using SOBI." *NeuroImage*, **28**(2), 507–519. doi:10.1016/j.neuroimage.2005.06.062.

Taskinen S, Miettinen J, Nordhausen K (2016). "A More Efficient Second Order Blind Identification Method for Separation of Uncorrelated Stationary Time Series." *Statistics & Probability Letters*, **116**, 21–26. doi:10.1016/j.spl.2016.04.007.

Tong L, Soon VC, Huang YF, Liu R (1990). "AMUSE: A New Blind Identification Algorithm." In *IEEE International Symposium on Circuits and Systems*, pp. 1784–1787. doi:10.1109/ISCAS.1990.111981.

Tsay RS, Wood D (2021). ***MTS**: All-Purpose Toolkit for Analyzing Multivariate Time Series (MTS) and Estimating Multivariate Volatility Models.* R package version 1.0.3, URL https://CRAN.R-project.org/package=MTS.

Van der Weide R (2002). "GO-GARCH: A Multivariate Generalized Orthogonal GARCH Model." *Journal of Applied Econometrics*, **17**(5), 549–564. doi:10.1002/jae.688.

Virta J, Koesner CL, Li B, Nordhausen K, Oja H, Radojicic U (2021). ***tensorBSS**: Blind Source Separation Methods for Tensor-Valued Observations.* R package version 0.3.8, URL https://CRAN.R-project.org/package=tensorBSS.

Virta J, Nordhausen K (2021). "Determining the Signal Dimension in Second Order Source Separation." *Statistica Sinica*, **31**(1), 135–156. doi:10.5705/ss.202018.0347.

Weisberg S (2002). "Dimension Reduction Regression in R." *Journal of Statistical Software*, **7**(1), 1–22. doi:10.18637/jss.v007.i01.

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

Affiliation:

Klaus Nordhausen
CSTAT – Computational Statistics
Institute of Statistics & Mathematical Methods in Economics
Vienna University of Technology
Wiedner Hauptstr. 7
1040 Vienna, Austria
E-mail: klaus.nordhausen@tuwien.ac.at
URL: http://klausnordhausen.com/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
July 2021, Volume 98, Issue 15 *Submitted:* 2018-10-11
doi:10.18637/jss.v098.i15 *Accepted:* 2019-12-31