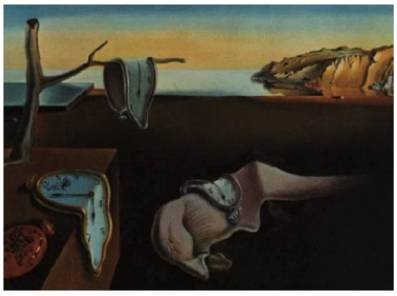
Journal of Statistical Software

Information on Mission | Information for Authors | Style Guide | Volumes | About

Journal Title:	Journal of Statistical Software
Publisher:	University of California Press
P-ISSN:	15487660
Open Access:	YES
Subject:	Statistics and Probability
Citescore:	17
SNIP:	7.237
SJR:	7.636
Quartile:	1



Iodice D'Enza Alfonso, Markos Angelos, Buttarazzi Davide (2018). The idm package: Incremental decomposition methods in R. JOURNAL OF STATISTICAL SOFTWARE, vol. 86, p. 1-24, ISSN: 1548-7660, doi: 10.18637/jss.v086.c04

Journal of Statistical Software

August 2018, Volume 86, Code Snippet 4. *doi: 10.18637/jss.v086.c04*

The idm Package: Incremental Decomposition Methods in R

Alfonso Iodice D'Enza
Università di Cassino

Angelos Markos
Democritus University of Thrace

Davide Buttarazzi
Università di Cassino

Abstract

In modern applications large amounts of data are produced at a high rate and are characterized by relationship structures changing over time. Principal component analysis (PCA) and multiple correspondence analysis (MCA) are well established dimension reduction methods to explore relationships within a set of variables. A critical step of the PCA and MCA algorithms is a singular value decomposition (SVD) or an eigenvalue decomposition (EVD) of a suitably transformed matrix. The high computational and memory requirements of ordinary SVD and EVD make their application impractical on massive or sequential data sets. A series of incremental SVD/EVD approaches are available to address these issues. The **idm** R package is introduced that implements two efficient incremental SVD approaches. The procedures in question share desirable properties that ease their embedding in PCA and MCA. The package also provides functions for producing animated visualizations of the obtained solutions. A comparison of online MCA implementations in terms of accuracy is also included.

Keywords: singular value decomposition, dimensionality reduction, principal component analysis, correspondence analysis.

1. Introduction

Principal component analysis (PCA; Jolliffe 2002) as well as multiple correspondence analysis (MCA; Greenacre 2007) are well established methods for exploratory analysis and visualization of high dimensional data sets. In modern applications, with large amounts of data being produced at a high rate, the applicability of PCA or MCA is unfeasible, or even impossible, because of the limitations that affect eigenvalue decomposition (EVD) and singular value decomposition (SVD), that are at the core of PCA and MCA. In particular, with $\mathbf{X}$ a matrix of n observations and Q attributes, the computational complexity of EVD$\left(\mathbf{X}^\top\mathbf{X}\right)$

and SVD($\mathbf{X}$) is $\mathcal{O}(Q^3)$ and $\mathcal{O}(n^2Q)$, respectively, assuming $n \geq Q$ (Golub and Van Loan 2012). Therefore, the application of EVD and SVD becomes unfeasible for large data sets and impossible for data streams, as the general implementations require all the data to be available when the analysis starts. Several proposals in the literature aim to overcome the limitations of EVD and SVD; the different approaches can be roughly classified into *batch* and *incremental*. Batch methods aim to increase the computational efficiency of EVD and SVD, extending their applicability to very large data sets. For EVD, iterative algorithms such as the LR and the QR (Golub and Van Loan 2012), are widely used for finding the eigenvalues of $\mathbf{X}$. The LR algorithm is based on the successive triangular decomposition of $\mathbf{X}$, whereas the QR algorithm makes use of unitary transformations. Under certain conditions, both algorithms converge to upper triangular form, thereby revealing the eigenvalues of $\mathbf{X}$. On the SVD side, the bilinear diagonalization Lanczos methods considerably reduce the computational complexity of the procedure (Baglama and Reichel 2005).

Incremental methods aim to update an existing decomposition when new data comes in sequentially, i.e., in the case of data streams. Incremental EVD/SVD is widely used in image analysis and face recognition (see, e.g., an incremental PCA implementation in Zhao, Yuen, and Kwok (2006)). Incremental decomposition methods can be roughly classified into the following categories (see Cardot and Degras (2018) for a detailed review): *i)* pertubation mehods (Hegde, Principe, Erdogmus, Ozertem, Rao, and Peddaneni 2006); *ii)* stochastic optimization and related methods (Sanger 1989; Oja 1992; Weng, Zhang, and Hwang 2003; Mitliagkas, Caramanis, and Jain 2013), *iii)* randomized algorithms (Warmuth and Kuzmin 2008); *iv)* secular equations (Gu and Eisenstat 1994) and *v)* heuristic techniques for incremental EVD/SVD (Zha and Simon 1999; Levey and Lindenbaum 2000; Hall, Marshall, and Martin 2002; Ross, Lim, Lin, and Yang 2008; Baker, Gallivan, and Van Dooren 2012). The methods of the last family are particularly interesting, since they are computationally efficient and share desirable properties that ease their embedding in both PCA and MCA (Iodice D'Enza and Markos 2015).

There are several packages in R (R Core Team 2018) that implement PCA, MCA and their variants: package **FactoMineR** (Lê, Josse, and Husson 2008) provides a wide range of factor analysis methods; package **ca** (Nenadić and Greenacre 2007) is specific for simple and multiple correspondence analysis, with several options for interpretation and visualization; package **ade4** (Thioulouse and Dray 2007) provides standard PCA and MCA for the analysis of ecological and environmental data. In all these packages, computations are based on ordinary EVD and SVD eigensolvers.

Efficient batch EVD/SVD implementations are also provided in R: the **svd** package (Korobeynikov and Larsen 2017) is a bridge to the C library **TRLAN** (Wu and Simon 2010) that uses state of the art eigensolvers, including the thick-restart Lanczos method (Wu and Simon 2000); package **irlba** (Baglama, Reichel, and Lewis 2018) implements the implicitly restarted Lanczos method for SVD; package **rARPACK** (Qiu and Mei 2016) is a wrapper of the **ARPACK** library for large scale eigendecompositions (Lehoucq, Sorensen, and Yang 1998). Also, packages **corpcor** (Schaefer, Opgen-Rhein, Zuber, Ahdesmäki, Duarte Silva, and Strimmer 2017) and **gmodels** (Warnes, Bolker, Lumley, and Johnson 2015) provide efficient computations of the SVD.

A series of incremental EVD/SVD methods embedded in the context of PCA, have been recently implemented in the **onlinePCA** package (Degras and Cardot 2016) and were thoroughly compared in terms of computational accuracy (Cardot and Degras 2018).

In this paper, the **idm** R package (Iodice D'Enza, Markos, and Buttarazzi 2018) is introduced that implements two efficient methods for block-wise incremental eigendecomposition, namely *eigenspace arithmetics* (Hall *et al.* 2002) and *block incremental SVD with mean update* (Ross *et al.* 2008), and embeds these methods in dimension reduction techniques for both interval (PCA) and categorical (MCA) data. In particular, for MCA, two variants are implemented (Iodice D'Enza and Markos 2015). The first one refers to the *exact* case and is suitable when all the data is available from the beginning of the analysis, whereas the second one refers to the *online* case i.e., when new data comes in sequentially, as in data streams. Package **idm** is available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=idm.

The **idm** package shares the same purpose with package **onlinePCA**, that is, to provide incremental dimension reduction methods, but it differs in the following aspects. First, in addition to incremental PCA, package **idm** provides incremental MCA implementations. Second, the two incremental decomposition methods provided in package **idm** are not included in package **onlinePCA**. Therefore, in this context, we are also interested in comparing all these methods against each other in terms of computational accuracy, especially when used to provide online MCA solutions. The third distinguishing feature of the **idm** package lies in the animated plots that the user can easily produce to display evolving PCA and MCA solutions. To facilitate the interpretation, static **ggplot2** graphs (Wickham 2009) are complemented by animated two-dimensional plots, created with the **animation** package (Xie 2013). Animated plots are particularly appealing since they allow to monitor the evolution of PCA and MCA solutions as new data blocks arrive.

The paper is structured as follows. In Section 2, PCA and MCA are briefly introduced as matrix decomposition techniques, whereas their incremental versions are described in Section 3. The package description is provided by means of applicative examples in Section 4. A comparison between online MCA algorithms in terms of computational accuracy on simulated and real data is offered in Section 5. A discussion in Section 6 concludes the paper.

2. Dimension reduction methods as a matrix decomposition

This section provides a brief introduction to PCA and MCA from a matrix decomposition viewpoint. Let $\mathbf{X}$ be an $n \times Q$ data matrix, where n is the number of observations and Q is the number of quantitative attributes. We refer in this section to covariance PCA, with the Q attributes on their original scale, whereas in correlation PCA the Q attributes are scaled to have a unit variance (see, e.g., Borgognone, Bussi, and Hough 2001). In this setting, the PCA of $\mathbf{X}$ amounts to the SVD of the following matrix

$$\mathbf{S}_{\text{PCA}} = n^{-1/2} \left(\mathbf{X} - n^{-1} \mathbf{1}\mathbf{1}^\top \mathbf{X} \right) Q^{-1/2}, \tag{1}$$

where $\mathbf{1}$ is an n-dimensional vector of ones. The decomposition is $\mathbf{S}_{\text{PCA}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where $\mathbf{U}$ is an $n \times Q$ orthonormal matrix with left singular vectors on columns, $\mathbf{\Sigma}$ is a diagonal matrix containing the Q singular values and $\mathbf{V}$ is a $Q \times Q$ matrix of right singular vectors. The jth singular value corresponds to the standard deviation of data along the direction of the jth singular vector, $j = 1, \ldots, Q$. Let $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ be the first d columns of $\mathbf{U}$ and $\mathbf{V}$ and let $\tilde{\mathbf{\Sigma}}$ be the matrix of the first d singular values, then $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^\top$ is the rank-d matrix that approximates $\mathbf{S}_{\text{PCA}}$ in the least-squares sense. The principal coordinates for rows and

columns are $\mathbf{F} = n^{1/2}\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}$ and $\mathbf{G} = Q^{1/2}\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}$, respectively.

We define MCA in a very similar way. Let $\mathbf{Z}$ be an $n \times Q$ binary matrix, where n is the number of observations and Q the total number of categories that characterize q categorical variables. The general element is $z_{ij} = 1$ if the ith observation is characterized by the jth category, $z_{ij} = 0$ otherwise. Let $\mathbf{P} = \frac{1}{n \times q}\mathbf{Z}$ be the so-called correspondence matrix, where $n \times q$ is the grand total of $\mathbf{Z}$. The core step of MCA is the matrix decomposition of the standardized residual matrix $\mathbf{S}$, defined as follows

$$\mathbf{S}_{\text{MCA}} = \mathbf{D}_{\mathbf{r}}^{-1/2}\left(\mathbf{P} - \mathbf{r}\mathbf{c}^{\top}\right)\mathbf{D}_{\mathbf{c}}^{-1/2}, \tag{2}$$

where $\mathbf{r}$ and $\mathbf{c}$ are the row and column margins of $\mathbf{P}$, respectively; $\mathbf{D}_{\mathbf{r}}$ and $\mathbf{D}_{\mathbf{c}}$ are diagonal matrices with values in $\mathbf{r}$ and $\mathbf{c}$. As for PCA, the MCA lies in the SVD of $\mathbf{S}_{\text{MCA}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}$, and the principal coordinates in the d-dimensional space are $\mathbf{F} = \mathbf{D}_{\mathbf{r}}^{-1/2}\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}$ for the observations, and $\mathbf{G} = \mathbf{D}_{\mathbf{c}}^{-1/2}\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}$ for the attributes.

3. Incremental dimension reduction

In order to describe an incremental scheme for PCA and MCA we first introduce some necessary definitions. An *eigenspace* is a collection of the quantities needed to define the result of a matrix eigendecomposition, as it involves eigenvalues (singular values), eigenvectors (singular vectors), data mean and size. In particular, with respect to the SVD, for an $n_1 \times Q$ matrix $\mathbf{X}_1$ and an $n_2 \times Q$ matrix $\mathbf{X}_2$, we can specify two eigenspaces as $\Omega_1 = (n_1, \boldsymbol{\mu}_1, \mathbf{U}_1, \mathbf{\Sigma}_1, \mathbf{V}_1)$ and $\Omega_2 = (n_2, \boldsymbol{\mu}_2, \mathbf{U}_2, \mathbf{\Sigma}_2, \mathbf{V}_2)$. The aim of incremental decomposition is to obtain an eigenspace Ω_3 for the row-wise concatenated matrix $\begin{bmatrix}\mathbf{X}_1 \\ \mathbf{X}_2\end{bmatrix}$, using uniquely the information in Ω_1 and the matrix $\mathbf{X}_2$. The total number of observations and the global data mean can be easily updated: $n_3 = n_1 + n_2$ and $\boldsymbol{\mu}_3 = \frac{n_1\boldsymbol{\mu}_1 + n_2\boldsymbol{\mu}_2}{n_3}$.

3.1. Incremental block-wise PCA

The incremental version of PCA is a straightforward adaptation of the *eigenspace arithmetics* approach (Hall *et al.* 2002). As pointed out by Hall *et al.* (2002) adding new data to an existing eigenspace rotates the eigenvectors (singular vectors) and it scales the eigenvalues according to the data spread. Therefore the eigenvectors in $\mathbf{V}_3$ are linear combinations of the already available ones, $\mathbf{V}_1$. In order to deal with a change in dimension, a basis sufficient span $\mathbf{V}_3$ is constructed, where $\mathbf{V}_1$ is augmented by $\mathbf{v}$, the latter given by

$$\mathbf{v} = \text{orth}\left(\psi\left[\mathbf{H}, \mathbf{h}\right]\right). \tag{3}$$

The orth$(\cdot)$ operator stands for a Gram-Schmidt orhogonalization procedure, ψ discards very small column vectors from the matrix, and $\mathbf{v}$ is the set of t eigenvectors that are outside the eigenspace Ω_1; $\mathbf{H}$ is the null space of both $\mathbf{V}_1$ and $\mathbf{V}_2$; $\mathbf{h}$ is the component of the vector joining the means $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ that lies in the null space of both subspaces. More specifically,

$$\mathbf{H} = \mathbf{V}_2 - \mathbf{V}_1\mathbf{V}_1^{\top}\mathbf{V}_2 \quad \text{and} \quad \mathbf{h} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \mathbf{V}_1\mathbf{V}_1^{\top}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2).$$

Finally, the *merged* eigenvectors are given by $\mathbf{V}_3 = [\mathbf{V}_1, \mathbf{v}]\,\mathbf{R}$, where $\mathbf{R}$ is an orthonormal matrix obtained from the SVD of the following block matrix:

$$\begin{bmatrix} \mathbf{\Sigma}_1\mathbf{U}_1^\top & \mathbf{V}_1^\top\mathbf{V}_2\mathbf{\Sigma}_2\mathbf{U}_2^\top \\ 0 & \mathbf{v}^\top\mathbf{V}_2\mathbf{\Sigma}_2\mathbf{U}_2^\top \end{bmatrix} + \begin{bmatrix} \mathbf{V}_1^\top(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_3)\,\mathbf{1}_{n_1} & \mathbf{V}_1^\top(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_3)\,\mathbf{1}_{n_2} \\ \mathbf{v}^\top(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_3)\,\mathbf{1}_{n_1} & \mathbf{v}^\top(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_3)\,\mathbf{1}_{n_2} \end{bmatrix} = \mathbf{R}\mathbf{\Sigma}\mathbf{U}^\top, \tag{4}$$

where $\mathbf{1}_n$ is an n-dimensional vector of ones.

The remaining elements of the SVD-based eigenspace $\mathbf{\Omega}_3$ are given by

$$\mathbf{\Sigma}_3 = \mathbf{\Sigma} \text{ and } \mathbf{U}_3 = \mathbf{U}. \tag{5}$$

The row and column principal coordinates of $\mathbf{\Omega}_3$ are given by $\mathbf{F} = n_3^{1/2}\mathbf{U}_3\mathbf{\Sigma}_3$ and $\mathbf{G} = Q^{1/2}\mathbf{V}_3\mathbf{\Sigma}_3$, respectively.

Note that for each analyzed data block, $\mathbf{X}$, it is assumed that $n > Q$. When the matrix has more attributes than observations, that is when $Q > n$, it is more convenient to perform an EVD on $\mathbf{X}\mathbf{X}^\top$ to obtain $\mathbf{U}$ and then compute $\mathbf{V}$ by means of the transition formula, $\mathbf{V} = \mathbf{X}^\top\mathbf{U}\mathbf{\Sigma}^{-1}$. The case when the whole data set is $Q \gg n$, however, has peculiarities that will be briefly discussed in Section 6.

It is important to outline that the obtained PCA solution using the incremental method described above is *exact*, in the sense that it collapses into the ordinary PCA solution on the covariance matrix. If the PCA solution on the correlation matrix is needed, then the Q attributes need to be scaled in advance. Moreover, the addition of eigenspaces is *commutative* and *associative*, which is convenient for a parallel implementation (see Hall *et al.* 2002).

In terms of computational complexity, the starting and the incoming eigenspaces need a total of $\mathcal{O}(n_1^2 + n_2^2)Q$ operations. In addition, eigenspace merging requires at most $\mathcal{O}(Q + n_2^2 + 1)^2 Q$ operations. When computer memory is a limiting factor, the method is expected to yield much lower space complexity than the ordinary PCA algorithm. A drawback of this procedure, however, is that it requires computing all eigenelements of the covariance matrix. Therefore, it is more suitable when the whole data set is available in advance, as in the case of large data sets, than in online settings.

3.2. Incremental block-wise MCA

Setting the problem in an MCA framework, it is necessary to derive the matrix to be decomposed and the data mean. This is because, in the case of MCA, variables are transformed according to the margins of each data block. In particular, we first express the standardized residual matrix of Equation 2 in covariance matrix form:

$$\mathbf{S} = \underbrace{\frac{\mathbf{Z}}{Q\sqrt{n}}\mathbf{D}_\mathbf{c}^{-1/2}}_{\mathbf{X_1}} - \mathbf{1}_n \underbrace{\frac{1}{\sqrt{n}}\mathbf{1}_\mathbf{c}^\top\mathbf{D}_\mathbf{c}^{1/2}}_{\boldsymbol{\mu}_1^\top}, \tag{6}$$

where $\mathbf{X}_1 = \frac{1}{Q\sqrt{n}}\mathbf{Z}\mathbf{D}_\mathbf{c}^{-1/2}$ is the $n_1 \times Q$ row-wise centered matrix of the first data block and $\boldsymbol{\mu}_1 = \frac{1}{\sqrt{n}}\mathbf{D}_\mathbf{c}^{1/2}\mathbf{1}$ is the data mean. For an incoming data block, we obtain the corresponding $n_2 \times Q$ matrix $\mathbf{X}_2$.

Two different MCA approaches can be defined. If the whole data matrix is known in advance, then the final or global column margins are also known and the eigenspace Ω_3 of the super

matrix $\begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2\end{bmatrix}$ can be obtained using the *eigenspace arithmetics* method of Hall *et al.* (2002), as in the case of incremental PCA. This leads to an *exact* MCA solution. Similar to incremental PCA, the starting and the incoming eigenspaces need a total of $\mathcal{O}(n_1^2 + n_2^2)Q$ operations. In addition, eigenspace merging requires at most $\mathcal{O}(k + n_2^2 + 1)^2 Q$ operations, where $k = Q - q$ is the number of singular values needed for a full MCA solution.

When the whole data set is not known in advance, then the final MCA solution is approximate and Ω_3 can be obtained using *block incremental SVD with mean update*, a method proposed by Ross *et al.* (2008). This method exploits the relationship between the QR decomposition and the SVD to incrementally compute the left and right singular vectors. The complexity of the algorithm is similar to that of the exact case, with a crucial difference: eigenspace merging requires approximately $\mathcal{O}(d+n_2^2+1)^2 Q$ operations, where $d \leqslant k$. This is a decisive advantage of incremental SVD over eigenspace arithmetics, because it does not require computing all k eigenelements, if one is only interested in the d largest eigenvalues. Therefore it is more appropriate for a *live* or *online* MCA approach.

The incremental SVD is based on the following lemma.

Lemma. *Given the SVD of* $\mathbf{X}_1 = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^\top$,

$$\begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2\end{bmatrix} = \begin{bmatrix}\mathbf{U}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{\Sigma}_1 & \mathbf{0}\\ \mathbf{L} & \mathbf{H}\mathbf{Q}^\top\end{bmatrix}\begin{bmatrix}\mathbf{V}_1^\top\\ \mathbf{Q}\end{bmatrix},$$

where $\mathbf{L} = \mathbf{X}_2\mathbf{V}_1$, $\mathbf{Q}$ *is a result from the* $\mathbf{QR}$*-decomposition of* $\mathbf{H} = \mathbf{X}_2 - \mathbf{L}\mathbf{V}_1^\top$ *and* $\mathbf{I}$ *is the identity matrix. In order to take into account the varying mean, the vector* $\sqrt{\frac{nm}{n+m}}\,(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$ *was added to* $\mathbf{X}_2$.

Proof. Let $\mathbf{L} = \mathbf{X}_2\mathbf{V}_1$ be the projection of $\mathbf{X}_2$ onto the orthogonal basis $\mathbf{V}_1$ and $\mathbf{H} = \mathbf{X}_2 - \mathbf{L}\mathbf{V}_1^\top$ the orthogonal component of $\mathbf{L}$. Apply a $\mathbf{QR}$-decomposition to $\mathbf{H}$ to obtain $\mathbf{Q}$. Thus, $\mathbf{H} = \mathbf{X}_2 - \mathbf{L}\mathbf{V}_1^\top \Leftrightarrow \mathbf{X}_2 = \mathbf{H} + \mathbf{L}\mathbf{V}_1^\top \Leftrightarrow \mathbf{X}_2 = \mathbf{H}\mathbf{Q}^\top\mathbf{Q} + \mathbf{L}\mathbf{V}_1^\top$. Therefore,

$$\begin{bmatrix}\mathbf{U}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{\Sigma}_1 & \mathbf{0}\\ \mathbf{L} & \mathbf{H}\mathbf{Q}^\top\end{bmatrix}\begin{bmatrix}\mathbf{V}_1^\top\\ \mathbf{Q}\end{bmatrix} = \begin{bmatrix}\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^\top\\ \mathbf{L}\mathbf{V}_1^\top + \mathbf{H}\mathbf{Q}^\top\mathbf{Q}\end{bmatrix} = \begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2.\end{bmatrix}$$

Apply the SVD to the matrix $\begin{bmatrix}\mathbf{\Sigma}_1 & \mathbf{0}\\ \mathbf{L} & \mathbf{H}\mathbf{Q}^\top\end{bmatrix}$ to obtain $\mathbf{U}_m\mathbf{\Sigma}_m\mathbf{V}_m^\top$.

Finally, $\mathbf{U}_3 = \begin{bmatrix}\mathbf{U}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\mathbf{U}_m$, $\mathbf{\Sigma}_3 = \mathbf{\Sigma}_m$, $\mathbf{V}_3 = \mathbf{V}_m^\top\begin{bmatrix}\mathbf{V}_1^\top\\ \mathbf{Q}\end{bmatrix}$.

In each update, the new row and column margins are given by $\mathbf{D}_\mathbf{r}^{(3)} = n_3^{-1}$ and $\mathbf{D}_\mathbf{c}^{(3)} = \left(n_1\mathbf{D}_\mathbf{c}^{(1)} + n_2\mathbf{D}_\mathbf{c}^{(2)}\right)n_3^{-1}$, respectively. Thus, $\mathbf{D}_\mathbf{c}^{(3)}$ is set to be the average of the "local" margins or the margins of the merged data blocks. Finally, row and column principal coordinates are given by $\mathbf{F}_3 = \left(\mathbf{D}_\mathbf{r}^{(3)}\right)^{-1/2}\mathbf{U}_3\mathbf{\Sigma}_3$ and $\mathbf{G}_3 = \left(\mathbf{D}_\mathbf{c}^{(3)}\right)^{-1/2}\mathbf{V}_3\mathbf{\Sigma}_3$, respectively. □

For a thorough treatment of incremental MCA see Iodice D'Enza and Markos (2015).

Data set	Description
`tweet`	The data set refers to a small corpus of 7,296 messages or tweets mentioning seven major hotel brands (Iodice D'Enza and Markos 2015).
`women`	Data related to 2,107 Spanish working women and the effect of their work on the family (Greenacre 2010).
`enron`	Presence/absence of 80 words in 39,861 e-mail messages. The data is a subset of the Enron e-mail corpus from the UCI Machine Learning Repository (Dua and Karra Taniskidou 2017).

Function	Description
`add_es`	Addition of two eigenspaces using the incremental methods of Hall *et al.* (2002) or Ross *et al.* (2008).
`do_es`	Computation of the eigenspace of a data matrix.
`i_mca`	Computation of the incremental multiple correspondence analysis solution using either *exact* or *live* methods.
`i_pca`	Computation of the incremental principal component analysis solution using the method of Hall *et al.* (2002).
`plot` method for '`i_mca`' objects	Static or animated graphical visualization of the incremental multiple correspondence analysis solution.
`plot` method for '`i_pca`' objects	Static or animated graphical visualization of the incremental principal component analysis solution.
`update` method for '`i_mca`' objects	Update of a given multiple correspondence analysis solution using the method proposed by Ross *et al.* (2008).
`update` method for '`i_pca`' objects	Update of a given principal component analysis solution using the method proposed by Hall *et al.* (2002).

Table 1: Summary of the **idm** package contents.

4. Package description and illustrative examples

In this section, the features of the package are illustrated through real data examples. First, incremental PCA is demonstrated with the `HCS` data set provided by the **caret** package (Kuhn 2008). Then, two incremental MCA approaches, *exact* and *live*, are demonstrated with the `enron` and the `tweet` data sets. The former is taken from the UCI Machine Learning Repository (Dua and Karra Taniskidou 2017) while the latter was reported by Iodice D'Enza and Markos (2015).

The incremental block-wise PCA and MCA techniques described in Section 3 are implemented in the **idm** package, adopting the standard S3 object-oriented system. They can be performed by invoking the main functions `i_pca()` and `i_mca()`, respectively. Accordingly, the `plot()` function allows for both static and animated graphical visualizations of the solution. The complete list of functions and data sets available in package **idm** is summarized in Table 1.

Arguments	Description
`data1`	Matrix or data frame of starting data or full data if `data2 = NULL`.
`data2`	Matrix or data frame of incoming data.
`current_rank`	Rank of approximation or number of components to compute; if empty, the full rank is used.
`nchunk`	Number of incoming data chunks (equal splits of `data2`, `default = 2`) or a vector with the row size of each incoming data chunk.
`disk`	Logical indicating whether the output is saved to hard disk (`default = FALSE`).
Output	**Description**
`rowpcoord`	Row scores on the principal components.
`colpcoord`	Variable loadings.
`eg`	A list describing the eigenspace of a data matrix, with components: `u`: left eigenvectors; `v`: right eigenvectors; `m`: number of cases; `d`: eigenvalues; `orgn`: data mean.
`sv`	Singular values.
`inertia.e`	Percentage of explained variance.
`levelnames`	Attribute labels.
`rowctr`	Row contributions.
`colctr`	Column contributions.
`rowcor`	Row squared correlations.
`colcor`	Column squared correlations.
`nchunk`	A copy of `nchunk` in the return object.
`disk`	A copy of `disk` in the return object.
`allrowcoord`	A list containing the row scores on the principal components produced after each data chunk is analyzed; returned only when `disk = FALSE`.
`allcolcoord`	A list containing the variable loadings on the principal components produced after each data chunk is analyzed; returned only when `disk = FALSE`.
`allrowctr`	A list containing the row contributions after each data chunk is analyzed; returned only when `disk = FALSE`.
`allcolctr`	A list containing the column contributions after each data chunk is analyzed; returned only when `disk = FALSE`.
`allrowcor`	A list containing the row squared correlations produced after each data chunk is analyzed; returned only when `disk = FALSE`.
`allcolcor`	A list containing the column squared correlations produced after each data chunk is analyzed; returned only when `disk = FALSE`.

Table 2: List of `i_pca()` arguments and output with description.

4.1. Incremental PCA: HCS data

An incremental PCA can be performed by invoking the `i_pca()` function that requires two arguments, which can be matrices or data frames, and returns a list of objects of class '`i_pca`'. The description of the available arguments along with the related output is reported in Table 2.

Incremental PCA is demonstrated using data from high-content screening (HCS), which refers

to quantitative information about cells, collected through the analysis of their microscopic images. Several features are usually measured (e.g., size, fluorescence intensity, shape, just to cite a few; see Giuliano *et al.* (1997) for an overview about the HCS approach). The data set used in this example comes from a study conducted by Hill, Lapan, Li, and Haney (2007), aimed to investigate the impact of image segmentation on HCS. The attributes in the HCS data set are measurements of cells. Since multiple attributes refer to a particular characteristic of the cell (e.g., size, shape, morphology) it is reasonable to expect, in a supervised setting, multicollinearity among predictors. Predictive models prefer predictors to be uncorrelated (or at least with low correlation), therefore a common approach is to define a limited number of linear combinations of the attributes via PCA and fit the predictive model using the obtained orthogonal components. This is done both to limit model complexity, as an alternative to model selection and regularization (see, e.g., Gareth, Witten, Hastie, and Tibshirani 2013) and to deal with multicollinearity.

The `HCS` data set is available in the **caret** package (Kuhn 2008) in its pre-processed version as used in Kuhn and Johnson (2013) to apply PCA prior to segmentation. The `HCS` data set is organized in a 2,019 $\times$61 data frame, where the first column is the cell identification number, the second column indicates whether the cell was originally used either in the training or test sample, and the third column is the class variable indicating if the cell was either poorly or well segmented. These first three variables will be excluded from our analysis, while the remaining 58 cell features will be used to perform an incremental PCA.

After installing the **idm** package from the Comprehensive R Archive Network (CRAN), the package and the `HCS` data set are loaded:

```
R> library("idm")
R> data("segmentationData", package = "caret")
```

An incremental PCA is performed by sequentially adding 20 blocks of cells to the PCA solution of the first 150 cells via the following commands:

```
R> HCS <- scale(segmentationData[, -(1:3)])
R> colnames(HCS) <- abbreviate(colnames(HCS), minlength = 5)
R> data1 <- HCS[1:150, ]
R> data2 <- HCS[151:2019, ]
R> res_iPCA <- i_pca(data1, data2, nchunk = 20)
```

First, variable names are abbreviated to the first 5 characters in order to improve the readability of the plots. The `data1` argument specifies the first data block on which PCA is applied and consists of the first 150 rows of the `HCS` data set. Then, `data2` indicates the incoming data that will be used to update the existing PCA solution, consisting of the remaining 1,869 rows. The `nchunk` argument specifies that the incoming observations (rows) will be splitted into 19 equal blocks of 93 cells and one (last) block of 102 cells, and will be sequentially added to the PCA solution. Notice that the number of retained principal components after each incremental update is the maximum possible, i.e., 58, which corresponds to the full rank solution. Fewer dimensions can be specified through the `current_rank` argument. Moreover, in this example, the `HCS` variables are initially standardized to have zero mean and unit standard deviation, so that results are equivalent to the analysis of the correlation matrix. The `i_pca()` function returns the list of objects reported in Table 2. When `disk = FALSE`, the PCA output of each data chunk is returned in lists `allrowcoord`, `allcolcoord`, etc.

Arguments	Description
`x`	Principal component analysis object returned by `i_pca` or the `update` method for '`i_pca`' objects.
`dims`	Numerical vector of length 2 indicating the dimensions to plot on horizontal and vertical axes, respectively; the first dimension is horizontal and the second dimension is vertical.
`what`	Vector of two logicals specifying the contents of the plot(s). The first entry indicates if the scatterplot of observations is displayed and the second entry if the correlation circle of the variable loadings is displayed (`default = c(TRUE, TRUE)` shows both plots).
`dataname`	String prefix used for custom naming of output files; by default, it is the name of the output object.
`labels`	String vector of variable labels.
`animation`	Logical indicating whether animated GIF or PDF files are created and saved to the hard drive or a static plot is created (`default = TRUE`).
`frames`	Number of animation frames shown per iteration (`default = 10`); applicable only when `animation = TRUE`.
`zoom`	Logical indicating whether axes limits change during the animation creating a zooming effect; applicable only when `animation = TRUE`.
`movie_format`	Specifies if the animated plot is saved in the working directory either in `default = "gif"` or `"pdf"` format.
`...`	Further arguments passed to the `plot()` function.

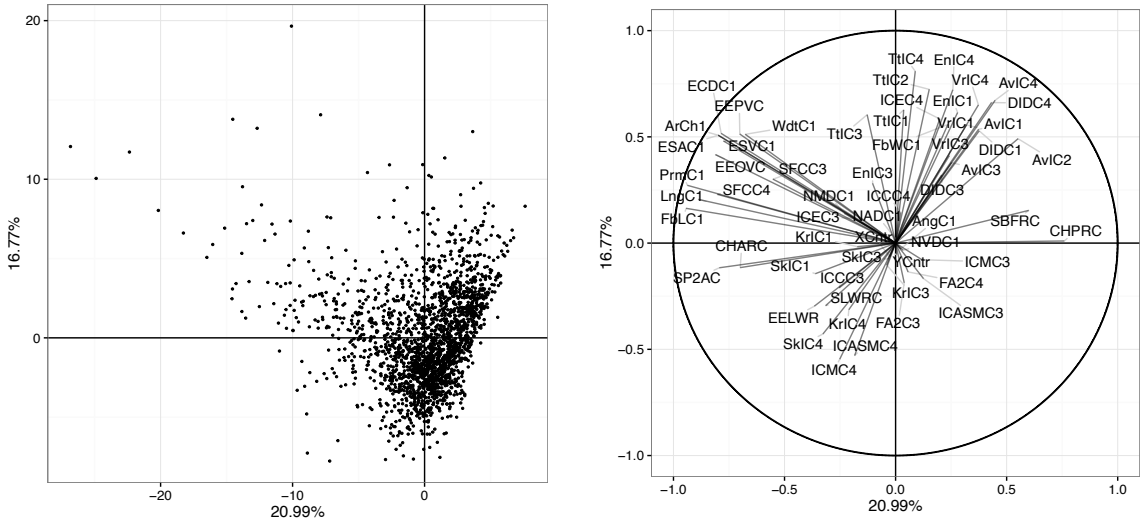
Table 3: List of arguments for the `plot` method for '`i_pca`' objects with description.

If the full data set is not available in advance or the user wants to mimic the online update process, the S3 function `update()` can instead be used to handle one update at a time, as follows:

```
R> data(segmentationData, package = "caret")
R> HCS <- scale(segmentationData[, -(1:3)])
R> colnames(HCS) <- abbreviate(colnames(HCS), minlength = 5)
R> res_PCA <- i_pca(HCS[1:200, ])
R> aa <- seq(from = 201, to = nrow(HCS), by = 200)
R> aa[length(aa)] <- nrow(HCS) + 1
R> for (k in 1:(length(aa) - 1)) {
+    res_PCA <- update(res_PCA, HCS[c((aa[k]):(aa[k + 1] - 1)), ])
+  }
```

The `i_pca()` function is initially called to create a starting PCA solution of the first 200 observations. The ten remaining blocks are subsequently added, one at a time, using `update()`. The first nine blocks of data consist of 200 rows, while the last block consists of 218 rows.

The final solution can be visualized via the `plot()` function by passing as argument an object of class '`i_pca`'. Hence, the `plot` method for '`i_pca`' objects makes a two-dimensional map of the object created by the function `i_pca()` (or the `update` method for '`i_pca`' objects) with respect to the selected principal components. The description of the available arguments for the `plot` method for '`i_pca`' objects is given in Table 3.



(a) (b)

Figure 1: Static PCA representation for the `HCS` data set with respect to principal components 1 (horizontal) and 2 (vertical). (a) Scatterplot of the 2,019 `HCS` observations. (b) Correlation circle of the 58 attributes.

The `animation` argument specifies whether a static or an animated representation of the PCA solution will be obtained. The animations are produced using the **animation** package (Xie 2013) and the animated PDF file should be viewed using Adobe Acrobat Reader. In order to create the animated displays, ImageMagick (or GraphicsMagick) needs to be installed on the user's system. When `movie_format = "pdf"` the following files are produced: a stand-alone `"pdf"` file containing the animation, a TEX file with the LATEX code that can be embedded in a manuscript or Beamer presentation, and another PDF file containing all the single frames of the animation. When `movie_format = "gif"`, a GIF file is produced that contains all the frames. In practice, the animated plots turn out to be more helpful for the interpretation, and allow the user to monitor the evolution of incremental PCA results.

The static plot of the final solution for the `HCS` data invokes the following command:

```
R> plot(res_iPCA, animation = FALSE)
```

The two-dimensional scatterplot of the observations is reproduced in Figure 1(a). The horizontal axis represents the first principal component that explains about 21% of the total variance, while the vertical axis represents the second principal component that accounts for about 17% of the variance. The `dims` argument allows the user to specify which dimensions are to be plotted, with default `dims = c(1, 2)`. The `res_iPCA$inertia.e` object can be invoked to explore the variance retained by each principal component. For instance, the first five components explain about 62% of the total variation, while the first ten components explain about 78% of total variation. The variance explained by the first twenty components is above 90%. Hence, the 58 dimensions of the `HCS` data set can be substantially reduced by using PCA. This is not entirely surprising, as this technique is widely applied in high-content screening.

The correlation circle, depicted in Figure 1(b), represents the variable loadings with respect

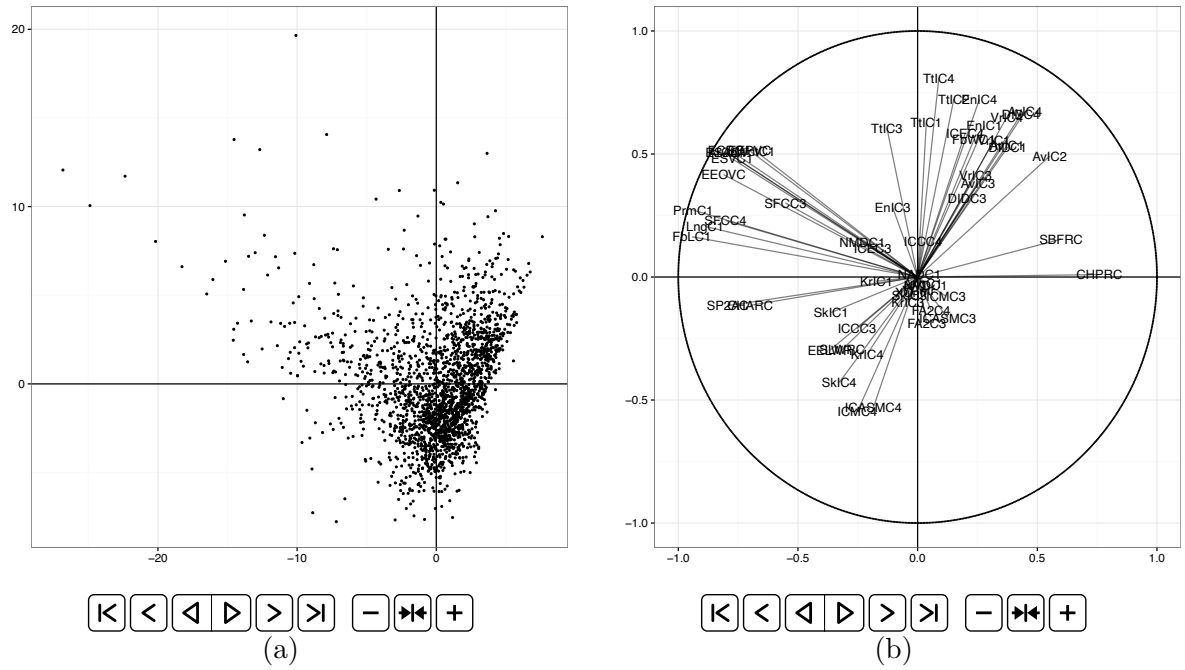


Figure 2: Animated incremental PCA representation for the `HCS` data set with respect to principal components 1 (horizontal) and 2 (vertical). (a) Scatterplot of the 2,019 cells. (b) Correlation circle of the 58 attributes (the control panel below the plots enables to activate the animation).

to the first two principal components. Two well separated groups of variables can be distinguished: one contributing more to the first principal component (i.e., SP2AC, WdtC1, FbLC1, among many others), and one contributing more to the second principal component (i.e., TtlC4, DIDC4, VrlC4 and others). In addition, it can be clearly observed that many variables, those showing very short segments in Figure 1(b), provide very small contributions to the first two principal components. The interpretation of the graphical output can also be facilitated by the use of the so-called row and column contributions (`res_iPCA$rowctr, res_iPCA$colctr`) and correlations (`res_iPCA$rowcor, res_iPCA$colcor`) allowing to detect among the observations and the attributes which ones are well projected and which ones contribute more to the construction of the axes.

In order to visualize the various steps leading to the final PCA solution, the animated plot can be obtained via the following command:

```
R> plot(res_iPCA, animation = TRUE, frames = 25, movie_format = "pdf")
```

The `frames` argument specifies the number of animation frames shown per iteration. The `animation` argument is set, by default, equal to `TRUE`.

The animated plots for the `HCS` data set are reproduced in Figure 2. In particular, the animated scatterplot of the observations on the two-dimensional subspace is reproduced in Figure 2(a). The animation takes place in twenty steps. First, a remarkable anticlockwise rotation takes place when the first incoming data block (93 cells) is added to the initial data block (150 cells). While the PCA solution continues to update, the plot shows how

the sequence of the remaining 19 blocks of observations distributes with respect to the first two principal components. Then, when the last block of data arrives (102 observations), the animation converges to the final solution. It is important to note that the PCA solution referring to the starting data block (specified with the `data1` argument), is already on the plot when the second data block arrives.

Additional information can be deduced from the animated plot focusing on the correlation patterns among variables, reproduced in Figure 2(b). For instance, some correlations become stronger as more information becomes available. The animated plot can also reveal if variables initially contributing more to one principal component turn out to contribute more to another principal component once the PCA has arrived at its final solution. Nonetheless, the animated plot clearly reveals which blocks of incoming observations (cells) substantially alter the PCA solution.

4.2. Incremental "exact" MCA: Enron data

The two versions of incremental MCA can be applied through the `i_mca()` function. The function requires two arguments, which can be matrices or data frames, and returns a list of objects of class '`i_mca`'. The description of available arguments, along with the related output is reported in Table 4. In particular, through the `method` argument, it is possible to specify which approach is to be implemented. When `method = "exact"`, dimensionality reduction is based on the method proposed by Hall *et al.* (2002), that is suitable when all the data is available from the beginning of the analysis. On the other hand, when `method = "live"`, dimensionality reduction is based on the method proposed by Ross *et al.* (2008), that refers to the case when new data comes in, as in data streams. As mentioned in Section 3.2, the main difference between `"exact"` and `"live"` lies in the calculation of the column margins of the input matrix; see Iodice D'Enza and Markos (2015) for further details.

An incremental MCA using the *exact* method is applied to a subset of the Enron e-mail corpus from the UCI Machine Learning Repository (Dua and Karra Taniskidou 2017). The Enron corpus is a collection of 39,861 e-mail messages with roughly 6 million tokens and a 28,102 terms vocabulary. From the original corpus, we selected the 80 most frequent words and created a binary 39,681 $\times$80 presence/absence data set.

After loading the data, an exact incremental MCA solution can be obtained by:

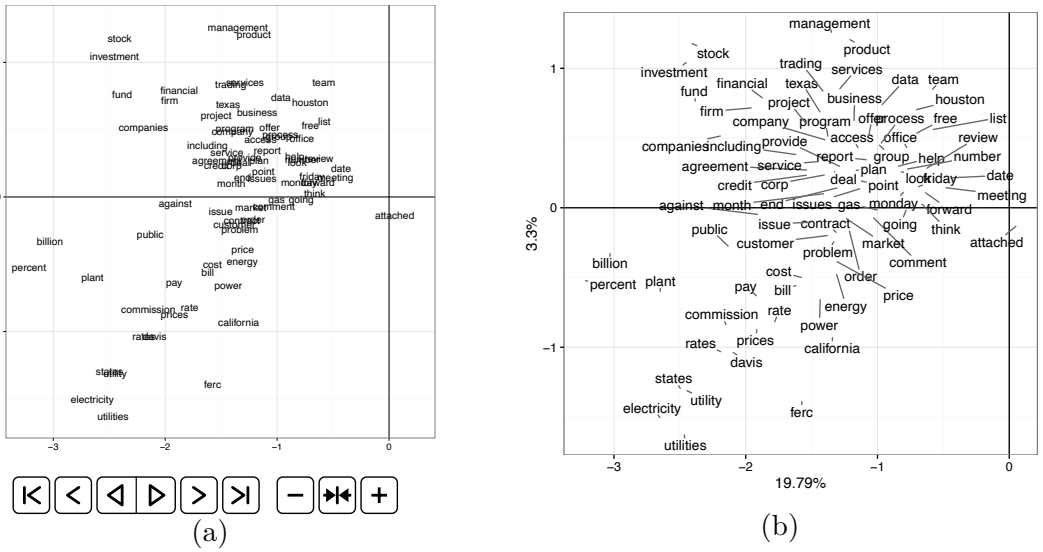
```
R> data("enron", package = "idm")
R> data1 <- enron[1:500, ]
R> data2 <- enron[501:39861, ]
R> res_iMCAh <- i_mca(data1, data2, method = "exact", nchunk = 10)
```

where `data1` represents the starting data over which MCA is applied, while `data2` contains the incoming data used to update the starting MCA solution. In addition, the `nchunk` argument specifies that the incoming data is split into ten equally-sized data blocks.

The results can be visualized via the `plot()` function. The arguments available for the `plot` method for '`i_mca`' object are equivalent to those reported in Table 3 for the `plot` method for '`i_pca`' objects. A notable difference is that the plot (either static or animated) can be customized by the `contrib` argument so that:

Arguments	Description
`data1`	Matrix or data frame of starting data or full data if `data2 = NULL`.
`data2`	Matrix or data frame of incoming data.
`method`	String specifying the type of implementation: `"exact"` or `"live"`.
`current_rank`	Rank of approximation or number of components to compute; if empty, the full rank is used.
`nchunk`	Number of incoming data chunks (equal splits of `data2`, `default = 2`) or a vector with the row size of each incoming data chunk.
`ff`	Number between 0 and 1 indicating the "forgetting factor" used to down-weight the contribution of earlier data blocks to the current solution. When `ff = 0` (default) no forgetting occurs; applicable only when `method = "live"`.
`disk`	Logical indicating whether the output is saved to hard disk (`default = FALSE`).
Output	
`rowpcoord`	Row principal coordinates.
`colpcoord`	Column principal coordinates.
`rowcoord`	Row standard coordinates.
`colcoord`	Column standard coordinates.
`sv`	Singular values.
`inertia.e`	Percentages of explained inertia.
`levelnames`	Column labels.
`rowctr`	Row contributions.
`colctr`	Column contributions.
`rowcor`	Row squared correlations.
`colcor`	Column squared correlations.
`rowmass`	Row masses.
`colmass`	Column masses.
`nchunk`	A copy of `nchunk` in the return object.
`disk`	A copy of `disk` in the return object.
`ff`	A copy of `ff` in the return object.
`allrowcoord`	A list containing the row principal coordinates produced after each data chunk is analyzed; returned only when `disk = FALSE`.
`allcolcoord`	A list containing the column principal coordinates produced after each data chunk is analyzed; returned only when `disk = FALSE`.
`allrowctr`	A list containing the row contributions after each data chunk is analyzed; returned only when `disk = FALSE`.
`allcolctr`	A list containing the column contributions after each data chunk is analyzed; applicable only when `disk = FALSE`.
`allrowcor`	A list containing the row squared correlations produced after each data chunk is analyzed; returned only when `disk = FALSE`.
`allcolcor`	A list containing the column squared correlations produced after each data chunk is analyzed; returned only when `disk = FALSE`.

Table 4: List of `i_mca()` arguments and output with description.



(a) (b)

Figure 3: (a) Animated incremental MCA representation for the `enron` data set using the *exact* approach with respect to dimensions 1 (horizontal) and 2 (vertical). (b) Static batch MCA representation for the `enron` data set given by `plot(i_mca(enron), binary = TRUE, animation = FALSE)`, where `binary = TRUE` is used to display word presence only (the control panel below plot (a) enables to activate the animation).

- if `contrib = "none"`, contributions are not indicated in the plot (default);
- if `contrib = "cor"`, the larger the size of an attribute label, the higher its relative contribution;
- if `contrib = "ctr"`, the larger the size of an attribute label, the higher its absolute contribution.

When the data is binary or indicate presence/absence, another argument specific to the `plot` method for '`i_mca`' objects is `binary`, a logical indicating whether only the categories associated with presence will be displayed on the plot.

The animated plot of exact incremental MCA on the `enron` data set is obtained invoking the command:

```
R> plot(res_iMCAh, animation = TRUE, what = c(FALSE, TRUE), binary = TRUE,
+    frames = 25, movie_format = "pdf")
```

The animated plot is reproduced in Figure 3(a) and can be compared with the results provided by a batch MCA on the same data (Figure 3(b)). The final solution obtained by exact incremental MCA is equivalent to that provided by batch MCA. The inertia explained by the first two dimensions approximates 23%. Words close to each other are related in that they occur together in the same e-mail messages.

4.3. Incremental "live" MCA: Twitter data

The "live" approach turns out to be especially useful when the entire data set is not available from the beginning of the analysis. In particular, following Ross *et al.* (2008), the MCA

solution is updated when new data blocks arrive, as in the case of data streams. For instance, consider the `tweet` data set available in the **idm** package. The data consists of 7,296 tweets mentioning seven major hotel brands, extracted via the **twitteR** package (Gentry 2015) within a time period of six days. The variables considered are:

- `Brand`: the hotel brand mentioned in the tweets (Hilton, Intercontinental, Marriott, Bestwestern, Starwood, Hyatt, Choice), coded as 1 to 7.
- `Sentiment`: a sentiment polarity variable on a four-point scale (negative "−", mixed "+/−", positive "+", very positive "++"), coded as 1 to 4.
- `UserVis`: the user popularity/visibility given by number of followers in Twitter on a three-point scale (low, medium, high), coded as 1 to 3.

An online MCA can be performed via the `i_mca()` function, setting the `method` argument equal to `"live"`. An additional parameter, the forgetting factor `ff`, can be used to give exponentially less weight to older data blocks. A value of `ff = 1` (default value) implies that no past data is forgotten at every update of the solution. After loading the data set, the results according to the live approach are obtained invoking the commands:

```
R> data("tweet", package = "idm")
R> data1 <- tweet[1:100, ]
R> data2 <- tweet[101:7296, ]
R> res_iMCA1 <- i_mca(data1, data2, method = "live", current_rank = 2,
+    nchunk = 5)
```

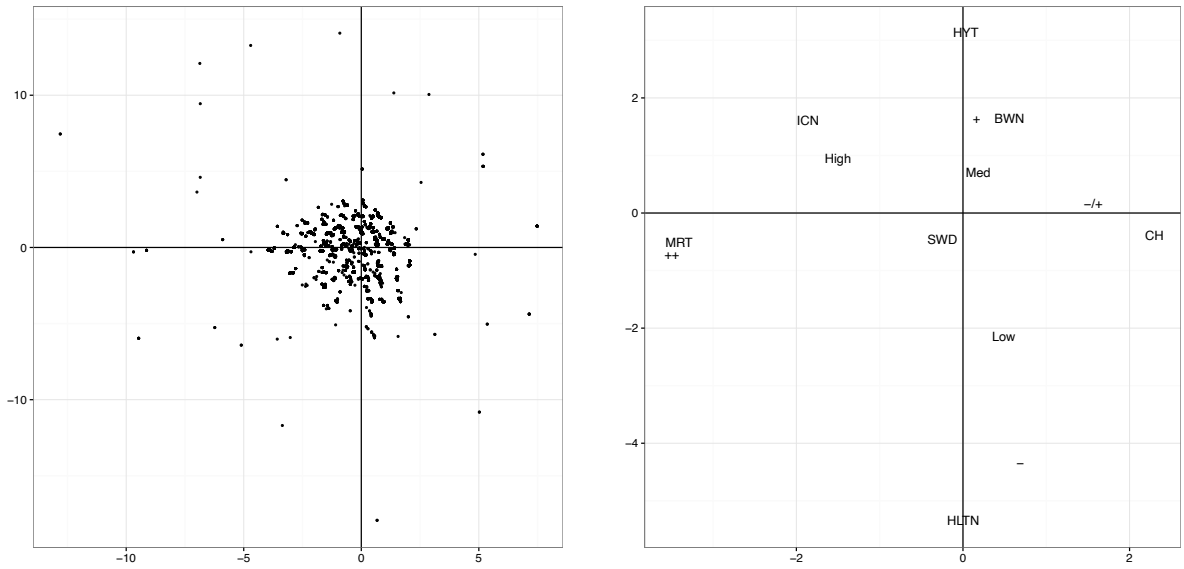
First, the starting MCA solution is computed over the first 100 tweets. Then, five equally sized data blocks are sequentially analyzed. A reduced-rank solution is obtained by keeping only the first `current_rank = 2` dimensions. It is important to outline that the `i_mca()` function immitates in a way the data stream scenario, as the whole data set is available in advance. The `update` method for '`i_mca`' objects can instead be used to handle one update at a time.

The solution, either static or animated, can be visualized by using the `plot` method for '`i_mca`' objects. To facilitate interpretation, custom attribute labels are provided. Hence, the animated maps of live MCA are given by:

```
R> labels <- c("HLTN", "ICN", "MRT", "BWN", "SWD", "HYT", "CH",
+    "-", "-/+", "+", "++", "Low", "Med","High")
R> plot(res_iMCA1, labels = labels, animation = TRUE, movie_format = "pdf")
```

The maps of Figure 4 show how tweet and attribute associations change as new data becomes available. Focusing on the final configurations, the first (horizontal) axis opposes Marriott hotels, receiving very positive comments (left of axis) to the others (right of axis). The second (vertical) axis opposes Hilton hotels, associated with negative comments mostly made by users of low visibility (bottom of axis) to all the others (top of axis). The corresponding static plots can also be obtained with the `plot` method for '`i_mca`' objects.

Consequently, while the *exact* approach can deal with large data sets, the greatest value of the *live* approach is that it well suits the nature of data streams.



(a) (b)

Figure 4: Animated incremental MCA representation for the `tweet` data set using the *live* approach with respect to dimensions 1 (horizontal) and 2 (vertical). (a) Tweets and (b) attributes (the control panel below the plots enables to activate the animation).

5. A comparison of online MCA approaches

A recent study by Cardot and Degras (2018) presented a series of incremental approaches that lead to online PCA and compared their statistical accuracy, computation time, and memory requirements. Seven incremental algorithms were considered: block stochastic power method (BSP; Mitliagkas *et al.* 2013), incremental PCA (IPCA; Arora, Cotter, Livescu, and Srebro 2012), block incremental PCA (BIPCA; Levey and Lindenbaum 2000), generalized Hebbian algorithm (GHA; Sanger 1989), candid covariance-free incremental PCA (CCI; Weng *et al.* 2003), stochastic gradient ascent (SGA; Oja 1992) and subspace network learning (SNL; Oja 1992). All these methods are implemented in the **onlinePCA** R package (Degras and Cardot 2016).

Since MCA can be described as a weighted PCA (see Section 3.2), all the above algorithms can be interchangeably used to provide online MCA solutions. Thus, it would be interesting to investigate how these different approaches compare to each other, as well as to block incremental SVD with mean update (BISVD), the method implemented in the **idm** package. Hence, in this section we present experimental results to demonstrate the efficiency of eight online MCA implementations on both simulated and real data sets. The focus of our attention is on a comparison in terms of accuracy, which is defined as the similarity between the ordinary MCA and the incremental MCA configurations in d dimensions.

5.1. Simulated data

The function `poLCA.simdata()` in the **poLCA** package (Linzer and Lewis 2011) was used to simulate categorical variables that match the data-generating process assumed by the basic latent variable model. The number of latent classes could randomly vary between 2 and 8 and the number of categories between 2 and 5, with randomly generated probabilities of occurence of the different categories. Following Cardot and Degras (2018), the number of observations, n, was generated with $n \in \{10^3, 10^4, 10^5, 10^6\}$ and the number of variables $Q \in \{10, 100\}$. The eight algorithms under comparison were initialized by the MCA of the first 25% of the observations and then run on the remaining 75%. The number of estimated dimensions, d, was fixed to 5. The tuning parameters of the stochastic methods, SGA, SNL and GHA, were selected by trial-and-error, as described in Cardot and Degras (2018). Note that BSP, BIPCA and BISVD allow for block-wise updates, whereas the other methods allow for vector updates only (one row at a time). For these methods, the recommended optimal block size was used (see Levey and Lindenbaum 2000; Mitliagkas *et al.* 2013; Iodice D'Enza and Markos 2015).

The similarity between the ordinary MCA and online MCA configurations in principal coordinates was measured using the R index, that equals $1 - m^2$, where m^2 is the symmetric orthogonal Procrustes statistic. The index ranges from 0 to 1 and can be interpreted as a correlation coefficient; it was calculated using the function `protest()` of the **vegan** package (Oksanen *et al.* 2018). Ordinary MCA was applied using the **ca** package (Nenadić and Greenacre 2007).

Table 5 shows the similarity values of the R index (averaged over 1,000 replications) between ordinary MCA and each one of the eight online MCA variants under comparison, using the first $d = 5$ dimensions and different values of n and Q. A first remark is that online MCA configurations are getting more similar to those of ordinary MCA as the number of observations increases from 10^3 to 10^6, for all methods. Also, as expected, when the number of variables increases, the performance generally deteriorates, given that the number of dimensions is fixed. Stochastic approximation methods perform quite similar to each other with the exception of BSP, which generally provides more accurate solutions. Moreover, it is easy to observe that the three block-based algorithms, BSP, BIPCA and BISVD, are more accurate than vector-based methods. BISVD, the method implemented in the **idm** package, clearly outperforms every other method with the exception of BSP; BISVD and BSP have similar performance as n gets larger. It is important to outline, however, that block-based methods are expected to be less efficient than vector-based approaches in terms of computation time.

5.2. Real data

The accuracy of the eight online MCA approaches was also evaluated using two real-world data sets, the `enron` and `tweet` data sets, described in Section 4. The number of estimated dimensions, d, was set to 2 and 6, and the starting data block to 300 and $1,000$, respectively. All other settings were as specified in the experiments on simulated data.

Table 6 shows the R index between ordinary MCA and each one of the eight online MCA approaches for the two data sets. In line with the results obtained on the simulated data, BISVD and BSP clearly outperform all other methods. Notice that for BISVD, two dimensions suffice to obtain a highly accurate MCA solution on the `enron` data set ($R = 0.99$).

Data size	Value	BSP	IPCA	BIPCA	GHA	CCI	SGA	SNL	BISVD
$10^3 \times 10$	Mean	0.88	0.76	0.76	0.76	0.76	0.76	0.76	0.89
	SD	0.05	0.06	0.06	0.06	0.06	0.06	0.06	0.05
$10^3 \times 100$	Mean	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.87
	SD	0.04	0.07	0.07	0.07	0.07	0.07	0.07	0.08
$10^4 \times 10$	Mean	0.90	0.86	0.86	0.86	0.86	0.86	0.86	0.90
	SD	0.06	0.07	0.07	0.07	0.07	0.07	0.07	0.07
$10^4 \times 100$	Mean	0.94	0.92	0.92	0.92	0.92	0.92	0.92	0.95
	SD	0.03	0.07	0.07	0.07	0.07	0.07	0.07	0.04
$10^5 \times 10$	Mean	0.93	0.92	0.92	0.92	0.92	0.92	0.92	0.93
	SD	0.07	0.08	0.09	0.08	0.08	0.08	0.08	0.08
$10^5 \times 100$	Mean	0.95	0.94	0.94	0.94	0.94	0.94	0.94	0.95
	SD	0.03	0.05	0.05	0.05	0.05	0.05	0.05	0.04
$10^6 \times 10$	Mean	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
	SD	0.06	0.05	0.06	0.06	0.06	0.06	0.06	0.06
$10^6 \times 100$	Mean	0.98	0.97	0.97	0.97	0.97	0.97	0.97	0.98
	SD	0.04	0.05	0.05	0.05	0.05	0.05	0.05	0.05

Table 5: Accuracy of eight online MCA approaches for the first $d = 5$ dimensions

.

Data set	Size	BSP	IPCA	BIPCA	GHA	CCI	SGA	SNL	BISVD
`enron`	39,861 ×80	0.94	0.59	0.74	0.79	0.74	0.79	0.74	0.99
`tweet`	7,296 ×3	0.98	0.95	0.95	0.95	0.95	0.95	0.95	0.98

Table 6: Accuracy of eight online MCA approaches for the `enron` and `tweet` data sets.

6. Concluding remarks

In this paper we have presented the **idm** package for the R environment. It implements two efficient SVD-based procedures to provide incremental block-wise PCA and MCA variants. Both procedures allow to keep track of the data mean, which is a desirable property in the context of PCA and MCA, so as to simultaneously update the center of the low-dimensional space of the solution. A distinct feature of *eigenspace arithmetics*, the approach by Hall *et al.* (2002), is that the addition of eigenspaces is commutative and associative, which is convenient for a parallel implementation. On the other hand, BISVD, the method of Ross *et al.* (2008), has a computational advantage in that it can produce an approximate, reduced-rank solution. This property makes the method more appealing for online MCA scenarios. Experiments on simulated and real data sets indicated that BISVD compares well with alternative online methods in terms of accuracy. Another important feature of the package is that it includes the option for animated visualization of the PCA/MCA solutions. Animated plots can be used to facilitate interpretation but could also serve didactic purposes (e.g., the illustration of theoretical concepts of PCA/MCA).

In this work, we considered the incremental case when new observations, n, come in, whereas Q, the number of attributes, is fixed. In this framework, even if $Q > n$ for the first few

data blocks, n is expected to increase as new data blocks are analyzed and eventually become greater than Q. We did not consider, however, the case when the whole data set refers to further attributes (e.g., genes) describing a fixed set of observations, that is, scenarios with fixed n as Q increases. The application of PCA when Q is at least as large as n can lead to solutions being inconsistent and/or hard to interpret, as pointed out by Johnstone and Lu (2009). The authors show the consistency of PCA by introducing a simple model involving a single component

$$\mathbf{x}_i = \nu_i \boldsymbol{\rho} + \sigma \mathbf{z}_i, \tag{7}$$

where $\mathbf{x}_i$ is a Q-dimensional data vector, $\boldsymbol{\rho} \in \mathbb{R}^Q$ is the single component being estimated, $\nu_i \sim N(0, 1)$ are *iid* random effects, $\mathbf{z}_i$ are independent noise vectors (also Gaussian) and σ is the noise standard deviation. The estimator $\hat{\boldsymbol{\rho}}$ is the dominant eigenvector of the sample covariance matrix given by $n^{-1} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top$. Johnstone and Lu (2009) show that, assuming a positive signal-to-noise ratio, $\hat{\boldsymbol{\rho}}$ is a consistent estimate of $\boldsymbol{\rho}$ if and only if $\frac{Q}{n} \to 0$; we refer the reader to the original paper for details. In a more recent paper, however, Lee, Zou, and Wright (2010) pointed out that inconsistency of the sample eigenvectors does not necessarily imply poor performance of PCA. For example, PCA has been successfully applied in genome-wide association studies for accurate estimation of ethnicity (Ma, Kosorok, and Fine 2006), and in PC regression for microarrays (Price, Patterson, Plenge, Weinblatt, and Shadickand D. Reich 2006). Still, the authors warn the reader about the possibility of poor performance of PCA when $Q > n$.

In conclusion, incremental settings where $Q \gg n$ are interesting problems that are worth further investigation and are left for future work. Furthermore, extensions to other incremental versions of well-known multivariate analysis methods based on incremental EVD/SVD can be considered in future versions of the package.

References

Arora R, Cotter A, Livescu K, Srebro N (2012). "Stochastic Optimization for PCA and PLS." In *50th Annual Allerton Conference on Communication, Control, and Computing, Allerton Park & Retreat Center, Monticello, IL, USA, October 1–5, 2012*, pp. 861–868.

Baglama J, Reichel L (2005). "Augmented Implicitly Restarted Lanczos Bidiagonalization Methods." *SIAM Journal on Scientific Computing*, **27**(1), 19–42. doi:10.1137/04060593x.

Baglama J, Reichel L, Lewis BW (2018). ***irlba**: Fast Truncated SVD, PCA and Symmetric Eigendecomposition for Large Dense and Sparse Matrices*. R package version 2.3.2, URL https://CRAN.R-project.org/package=irlba.

Baker CG, Gallivan KA, Van Dooren P (2012). "Low-Rank Incremental Methods for Computing Dominant Singular Subspaces." *Linear Algebra and Its Applications*, **436**(8), 2866–2888. doi:10.1016/j.laa.2011.07.018.

Borgognone MG, Bussi J, Hough G (2001). "Principal Component Analysis in Sensory Analysis: Covariance or Correlation Matrix?" *Food Quality and Preference*, **12**(5–7), 323–326. doi:10.1016/s0950-3293(01)00017-9.

Cardot H, Degras D (2018). "Online Principal Component Analysis in High Dimension: Which Algorithm to Choose?" *International Statistical Review*, **86**(1), 29–50. doi:10.1111/insr.12220.

Degras D, Cardot H (2016). ***onlinePCA**: Online Principal Component Analysis*. R package version 1.3.1, URL https://CRAN.R-project.org/package=onlinePCA.

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

Gareth J, Witten D, Hastie T, Tibshirani R (2013). *An Introduction to Statistical Learning: With Applications in R*. Springer-Verlag, New York.

Gentry J (2015). ***twitteR**: R Based Twitter Client*. R package version 1.1.9, URL https://CRAN.R-project.org/package=twitteR.

Giuliano KA, DeBiasio RL, Dunlay RT, Gough A, Volosky JM, Zock J, Pavlakis GN, Taylor DL (1997). "High-Content Screening: A New Approach to Easing Key Bottlenecks in the Drug Discovery Process." *Journal of Biomolecular Screening*, **2**(4), 249–259. doi:10.1177/108705719700200410.

Golub GH, Van Loan CF (2012). *Matrix Computations*, volume 3. JHU Press.

Greenacre M (2007). *Correspondence Analysis in Practice*. 2nd edition. Chapman & Hall/CRC, Boca Raton. doi:10.1201/9781420011234.

Greenacre MJ (2010). *Biplots in Practice*. Fundacion BBVA.

Gu M, Eisenstat SC (1994). "A Stable and Efficient Algorithm for the Rank-One Modification of the Symmetric Eigenproblem." *SIAM Journal on Matrix Analysis and Applications*, **15**(4), 1266–1276. doi:10.1137/s089547989223924x.

Hall P, Marshall D, Martin R (2002). "Adding and Subtracting Eigenspaces with Eigenvalue Decomposition and Singular Value Decomposition." *Image and Vision Computing*, **20**(13–14), 1009–1016. doi:10.1016/s0262-8856(02)00114-2.

Hegde A, Principe JC, Erdogmus D, Ozertem U, Rao YN, Peddaneni H (2006). "Perturbation-Based Eigenvector Updates for On-Line Principal Components Analysis and Canonical Correlation Analysis." *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, **45**(1–2), 85–95. doi:10.1007/s11265-006-9773-6.

Hill A, Lapan P, Li Y, Haney S (2007). "Impact of Image Segmentation on High-Content Screening Data Quality for SK-BR-3 Cells." *BMC Bioinformatics*, **8**(340). doi:10.1186/1471-2105-8-340.

Iodice D'Enza A, Markos A (2015). "Low-Dimensional Tracking of Association Structures in Categorical Data." *Statistics and Computing*, **25**(5), 1009–1022. doi:10.1007/s11222-014-9470-4.

Iodice D'Enza A, Markos A, Buttarazzi D (2018). ***idm**: Incremental Decomposition Methods*. R package version 1.8.2, URL https://CRAN.R-project.org/package=idm.

Johnstone IM, Lu AY (2009). "On Consistency and Sparsity for Principal Components Analysis in High Dimensions." *Journal of the American Statistical Association*, **104**(486), 682–693. doi:10.1198/jasa.2009.0121.

Jolliffe I (2002). *Principal Component Analysis*. John Wiley & Sons.

Korobeynikov A, Larsen RM (2017). ***svd**: Interfaces to Various State-of-Art SVD and Eigensolvers*. R package version 0.4.1, URL https://CRAN.R-project.org/package=svd.

Kuhn M (2008). "Building Predictive Models in R Using the **caret** Package." *Journal of Statistical Software*, **28**(5), 1–26. doi:10.18637/jss.v028.i05.

Kuhn M, Johnson K (2013). *Applied Predictive Modeling*. Springer-Verlag, New York. doi:10.1007/978-1-4614-6849-3.

Lê S, Josse J, Husson F (2008). "**FactoMineR**: An R Package for Multivariate Analysis." *Journal of Statistical Software*, **25**(1), 1–18. doi:10.18637/jss.v025.i01.

Lee S, Zou F, Wright FA (2010). "Convergence and Prediction of Principal Component Scores in High-Dimensional Settings." *The Annals of Statistics*, **38**(6), 3605–3629. doi:10.1214/10-aos821.

Lehoucq R, Sorensen D, Yang C (1998). ***ARPACK** User's Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. URL http://www.caam.rice.edu/software/ARPACK/.

Levey A, Lindenbaum M (2000). "Sequential Karhunen-Loeve Basis Extraction and Its Application to Images." *IEEE Transactions on Image Processing*, **9**(8), 1371–1374. doi:10.1109/83.855432.

Linzer DA, Lewis JB (2011). "**poLCA**: An R Package for Polytomous Variable Latent Class Analysis." *Journal of Statistical Software*, **42**(10), 1–29. doi:10.18637/jss.v042.i10.

Ma S, Kosorok MR, Fine JP (2006). "Additive Risk Models for Survival Data with High-Dimensional Covariates." *Biometrics*, **62**(1), 202–210. doi:10.1111/j.1541-0420.2005.00405.x.

Mitliagkas I, Caramanis C, Jain P (2013). "Memory Limited, Streaming PCA." In *Advances in Neural Information Processing Systems*, pp. 2886–2894.

Nenadić O, Greenacre M (2007). "Correspondence Analysis in R, with Two- And Three-Dimensional Graphics: The **ca** Package." *Journal of Statistical Software*, **20**(3), 1–13. doi:10.18637/jss.v020.i03.

Oja E (1992). "Principal Components, Minor Components, and Linear Neural Networks." *Neural Networks*, **5**(6), 927–935. doi:10.1016/s0893-6080(05)80089-9.

Oksanen J, Blanchet FG, Friendly M, Kindt R, Legendre P, McGlinn D, Minchin PR, O'Hara RB, Simpson GL, Solymos P, Stevens MHH, Szoecs E, Wagner H (2018). ***vegan**: Community Ecology Package*. R package version 2.4-6, URL https://CRAN.R-project.org/package=vegan.

Price A, Patterson N, Plenge R, Weinblatt M, Shadickand D Reich N (2006). "Principal Components Analysis Corrects for Stratification in Genome-Wide Association Studies." *Nature Genetics*, **38**, 904–909. doi:10.1038/ng1847.

Qiu Y, Mei J (2016). ***rARPACK**: Solvers for Large Scale Eigenvalue and SVD Problems*. R package version 0.11-0, URL https://CRAN.R-project.org/package=rARPACK.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Ross DA, Lim J, Lin RS, Yang MH (2008). "Incremental Learning for Robust Visual Tracking." *International Journal of Computer Vision*, **77**(1–3), 125–141. doi:10.1007/s11263-007-0075-7.

Sanger TD (1989). "Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network." *Neural Networks*, **2**(6), 459–473. doi:10.1016/0893-6080(89)90044-0.

Schaefer J, Opgen-Rhein R, Zuber V, Ahdesmäki M, Duarte Silva AP, Strimmer K (2017). ***corpcor**: Efficient Estimation of Covariance and (Partial) Correlation*. R package version 1.6.9, URL https://CRAN.R-project.org/package=corpcor.

Thioulouse J, Dray S (2007). "Interactive Multivariate Data Analysis in R with the **ade4** and **ade4TkGUI** Packages." *Journal of Statistical Software*, **22**(5), 1–14. doi:10.18637/jss.v022.i05.

Warmuth MK, Kuzmin D (2008). "Randomized Online PCA Algorithms with Regret Bounds That Are Logarithmic in the Dimension." *Journal of Machine Learning Research*, **9**(10), 2287–2320.

Warnes GR, Bolker B, Lumley T, Johnson RC (2015). ***gmodels**: Various R Programming Tools for Model Fitting*. R package version 2.16.2, URL https://CRAN.R-project.org/package=gmodels.

Weng J, Zhang Y, Hwang WS (2003). "Candid Covariance-Free Incremental Principal Component Analysis." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **25**(8), 1034–1040. doi:10.1109/tpami.2003.1217609.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag, New York.

Wu K, Simon H (2000). "Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems." *SIAM Journal on Matrix Analysis and Applications*, **22**(2), 602–616. doi:10.1137/s0895479898334605.

Wu K, Simon H (2010). "**TRLAN** User Guide." *Technical report LBNL-42953*, Lawrence Berkeley National Laboratory. URL http://crd-legacy.lbl.gov/~kewu/trlan.html.

Xie Y (2013). "**animation**: An R Package for Creating Animations and Demonstrating Statistical Methods." *Journal of Statistical Software*, **53**(1), 1–27. doi:10.18637/jss.v053.i01.

Zha H, Simon HD (1999). "On Updating Problems in Latent Semantic Indexing." *SIAM Journal on Scientific Computing*, **21**(2), 782–791. doi:10.1137/s1064827597329266.

Zhao H, Yuen PC, Kwok JT (2006). "A Novel Incremental Principal Component Analysis and Its Application for Face Recognition." *IEEE Transactions on Systems, Man, and Cybernetics B*, **36**(4), 873–886. doi:10.1109/tsmcb.2006.870645.

Affiliation:

Alfonso Iodice D'Enza, Davide Buttarazzi
Department of Economics and Law
Università di Cassino e del Lazio Meridionale
03043 Cassino, Italy
E-mail: iodicede@unicas.it, d.buttarazzi@unicas.it

Angelos Markos
Department of Primary Education
Democritus University of Thrace
68100 Alexandroupoli, Greece
E-mail: amarkos@eled.duth.gr

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
August 2018, Volume 86, Code Snippet 4 *Submitted:* 2016-04-05
doi:10.18637/jss.v086.c04 *Accepted:* 2017-04-29

Dichiarazione Sostitutiva di Atto Notorio
(ART.47 E ART.19 D.P.R.28.12.2000 N.445)

Il sottoscritto IODICE D'ENZA ALFONSO nato a Napoli il 18/07/1977 , residente a Napoli, Via Pontano, 3, Codice Fiscale: DCDLNS77L18F839F, con riferimento alla domanda di partecipazione per il conseguimento dell'abilitazione scientifica nazionale alle funzioni di professore universitario di Prima Fascia nel settore concorsuale 13/D1- Statistica (Decreto Direttoriale n. 553 del 26 febbraio 2021), consapevole che le dichiarazioni mendaci sono punite ai sensi degli artt. 483, 495, 496, del codice penale e delle leggi speciali in materia,

DICHIARA SOTTO LA PROPRIA RESPONSABILITÀ CIVILE E PENALE DI ESSERE COAUTORE DEL SEGUENTE LAVORO FRUTTO DELL'ATTIVITÀ DI RICERCA CONGIUNTA:

IODICE D'ENZA Alfonso, Markos Angelos, Buttarazzi Davide (2018). The idm package: Incremental decomposition methods in R. JOURNAL OF STATISTICAL SOFTWARE, vol. 86, p. 1-24, ISSN: 1548-7660, doi: 10.18637/jss.v086.c04

- che la redazione del lavoro è frutto di una stretta e paritetica collaborazione fra gli autori che ne condividono l'impostazione, le ipotesi di ricerca ed i risultati conseguiti;

- ai fini dell'individuazione dell'apporto individuale nei lavori in collaborazione, che l'apporto individuale di Alfonso Iodice D'Enza si è maggiormente esplicitato nelle sezioni 2, 3 e 6, in coerenza con le ricerche e le tematiche affrontate e desumibili dal curriculum e dal percorso scientifico del candidato.

Letto, confermato e sottoscritto.

Napoli, 29 maggio 2023

Alfonso Iodice D'Enza