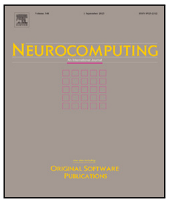
Neurocomputing 606 (2024) 128396

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Original Software Publication

Clusterwise Independent Component Analysis (C-ICA): An R package for clustering subjects based on ICA patterns underlying three-way (brain) data

Jeffrey Durieux [a,b,c,*], Serge Rombouts [a,b,d], Marisa Koini [e], Juan Claramunt Gonzalez [a], Tom Wilderjans [a,b,f,g]

[a] Methodology and Statistics Unit, Institute of Psychology, Leiden University, Wassenaarseweg 52, Leiden, 2333 AK, The Netherlands
[b] Leiden Institute for Brain and Cognition, Leiden University, The Netherlands
[c] Econometrics Institute, Erasmus School of Economics, Erasmus University, Burgermeester Oudlaan 50, Rotterdam, 3062 PA, The Netherlands
[d] Department of Radiology, Leiden University Medical Center, Albinusdreef 2, Leiden, 2300 RC, The Netherlands
[e] Department of Neurology, Medical University of Graz, Neue Stifingtalstrasse 6, Graz, 8010, Austria
[f] Research Group of Quantitative Psychology and Individual Differences, Faculty of Psychology and Educational Sciences, KU Leuven, Tiensestraat 102, Leuven, 3000, Belgium
[g] Department of Clinical Psychology, Vrije Universiteit Amsterdam, van der Boechorstraat 7, Amsterdam, 1081 BT, The Netherlands

ARTICLE INFO

Communicated by S. Wang

Keywords:
ICA
Cluster analysis
fMRI
Precision medicine
Three-way data
R package

ABSTRACT

In many areas of science, like neuroscience, genomics and text mining, several important and challenging research questions imply the study of (subject) heterogeneity present in three-way data. In clinical neuroscience, for example, disclosing differences or heterogeneity between subjects in resting state networks (RSNs) underlying multi-subject fMRI data (i.e., time by voxel by subject three-way data) may advance the subtyping of psychiatric and mental diseases. Recently, the Clusterwise Independent Component Analysis (C-ICA) method was proposed that enables the disclosure of heterogeneity between subjects in RSNs that is present in multi-subject rs-fMRI data [1]. Up to now, however, no publicly available software exists that allows to fit C-ICA to empirical data at hand. The goal of this paper, therefore, is to present the `CICA` R package, which contains the necessary functions to estimate the C-ICA parameters and to interpret and visualize the analysis output. Further, the package also includes functions to select suitable initial values for the C-ICA model parameters and to determine the optimal number of clusters and components for a given empirical data set (i.e., model selection). The use of the main functions of the package is discussed and demonstrated with simulated data. Herewith, the necessary analytical choices that have to be made by the user (e.g., starting values) are explained and showed step by step. The rich functionality of the package is further illustrated by applying C-ICA to empirical rs-fMRI data from a group of Alzheimer patients and elderly control subjects and to multi-country stock market data. Finally, extensions regarding the C-ICA algorithm and procedures for model selection that could be implemented in future releases of the package are discussed.

1. Introduction

During the last decade, in several fields of science, complex research questions urged researchers to collect three-way data. A first example is clinical neuroscience in which deeper insights into mental diseases and their subtypes were obtained by studying multi-subject resting state fMRI data. Such data measure the intensity of the blood oxygen level dependent (BOLD) imaging response [2] across time for several voxels administered from different subjects (i.e., voxel by time by subject data; see, for example, [3]). As a second example, in bioinformatics, disease related (regulatory) genetic pathways were disclosed through analyzing microarray gene expression data that were measured across several tissues or subjects over time (i.e., genes by tissue or subject by time; see [4–6]). Three-way data are also commonly collected to tackle emerging questions in other fields of science, like chemometrics, text mining, financial data, marketing and psychology (also see Section 2.1).

The most challenging research questions often pertain to the systematic patterns underlying such three-way data. For example, it is known that brain functioning involves voxels that act together in resting state functional connectivity networks (RSNs; [3,7]) and that genes exert their combined influence through regulatory pathways consisting of co-regulated/co-expressed genes [8,9]. An often used analysis method that enables the identification of these RSN patterns (from fMRI data)

* Corresponding author.
E-mail address: j.durieux@fsw.leidenuniv.nl (J. Durieux).

https://doi.org/10.1016/j.neucom.2024.128396
Received 17 April 2024; Received in revised form 25 July 2024; Accepted 10 August 2024
Available online 22 August 2024

and regulatory pathways (from microarray gene expression data) is Independent Component Analysis (ICA; [10]).[1] This method decomposes a multivariate signal, such as fMRI data, into a set of spatial statistically independent components (i.e., the RSNs) and a linear mixing matrix (i.e., the time courses associated to the RSNs).

An important goal of clinical neuroscience is to get a deeper understanding of mental diseases and its (sub)types. As research showed that disruptions in the integrity of RSNs are associated to various (subtypes of) neurodegenerative and psychiatric diseases such as depression disorder [12], schizophrenia [13], Alzheimer's disease [14], frontotemporal dementia [15] and Parkinson's disease [16], a principal challenge of the field is to reveal differences and similarities in RSNs between (groups of) subjects. To this end, researchers often collect multi-subject rs-fMRI data and extract RSNs by means of ICA.

In general, two ICA strategies can be applied to rs-fMRI data of multiple subjects to uncover similarities and differences in RSNs between (groups of) subjects. The first strategy is to apply ICA on the rs-fMRI data of each subject separately. For each subject, a -different- set of RSNs and associated time courses are estimated and one can manually identify relevant (similarities and differences in) RSNs patterns. This, however, could become a rather tedious job as many comparisons between RSNs across subjects need to be made. The second strategy, often referred to as Group-ICA, consists of first temporally concatenating the rs-fMRI data of all subjects together into one large dataset and then apply ICA to this large dataset. As such, a (common) set of RSNs that are related to all subjects from the sample is identified, along with the associated time courses for each subject.[2] To uncover similarities and differences in RSNs between (known) subject groups, where the grouping could, for example, be based on clinical diagnoses, researchers often apply group-ICA on the data of each group and compare the common RSNs across groups. Both mentioned strategies can be performed by using MELODIC [19], which is available in the FMRIB Software Library (FSL; [20]) or by adopting the Group ICA fMRI Toolbox (GIFT; [21]). As such, group-ICA reveals differences in RSNs between known subject groups.

Recently Zhang et al. [22] noted that a significant conceptual barrier that prevents the progressing of research regarding disease related functional connectivity is the heterogeneity in diseases and the fact that most proposed analysis methods do not have the ability to capture this variability in RSNs between (groups of) subjects. Indeed, patients with the same clinical diagnosis can exhibit rather different symptomatology, which suggests heterogeneity in RSNs among patients with the same clinical diagnosis. This large variability in RSNs within clinical patient groups calls for new methodologies that enable the automatic subtyping of subjects such that the large variability in RSNs is accounted for. Current methods, such as Group-ICA, are not suitable for an automatic subtyping since these methods require that the groups are defined a priori (e.g., based on a clinical diagnosis) instead of that they can be learned or derived from the data.

A promising way to uncover disease related heterogeneity in RSNs between patients is to combine exploratory clustering (unsupervised learning) techniques with ICA. Such a method allows for identifying unknown groups of patients -which could be identified as neurofunctional subtypes [1] or biotypes [23]- in an automatic fashion and detecting the differences in RSNs among these patient groups. A novel method in this regard is Clusterwise Independent Component Analysis (C-ICA), which was presented by [1]. This method clusters patients into homogeneous groups such that patients from the same group have common RSNs patterns and patients that are allocated to different groups are characterized by substantively different RSNs. The subgroups and their associated (differences in) RSNs that are derived from the data by C-ICA may provide valuable information about (existing) psychiatric and neuropsychological disorders and their subtypes.

Up to now, however, there is no comprehensive and easy to use software available to apply C-ICA to empirical multi-subject rs-fMRI data. The goal of this paper, therefore, is to introduce the R [24] package **CICA** which contains a set of functions that can be used to perform C-ICA [25] and interpret and visualize its output. In Section 2, an overview of the C-ICA model and associated loss function is presented along with an Alternating Least Squares (ALS) type of algorithm to estimate the C-ICA model parameters and strategies for model selection. Next, in Section 3, the main functions of **CICA** are explained and illustrated with a simulated dataset. Moreover, also functions that facilitate the visual inspection and interpretation of a C-ICA solution are presented. Section 4 gives an empirical illustration of C-ICA using multi-subject rs-fMRI data concerning AD patients and healthy control subjects. Another empirical example concerning multi-country stock market data is also presented. Using these empirical data sets, the working of the **CICA** package is further demonstrated and some extra functionality of the package is shown. Moreover, in Section 4, the results of a small simulation study are presented, investigating the computation speed of two types of component estimation procedures provided in the **CICA** package. Finally, some concluding remarks, including improvements for future releases of the **CICA** package, are presented in Section 5.

2. Data, model, loss function, algorithm and model selection

2.1. Data and pre-processing

2.1.1. Three-way multi-subject rs-fMRI data

A multi-subject rs-fMRI data set $\underline{\mathbf{X}}$ consists of a set of rs-fMRI scans $\mathbf{X}_i$ (V voxels $\times$ T time points) which contain data about the BOLD response for several subjects i ($i = 1, \ldots, N$). Each scan $\mathbf{X}_i$ measures how the intensity of the BOLD response varies over time for a large set of voxels. As can be seen in Fig. 1 (upper left array), such a data set can be represented as a three-way data array $\underline{\mathbf{X}}$ in which voxels constitute the first way (i.e, rows), time points the second way (i.e., columns) and subjects the third way (i.e., array slices).

2.1.2. Pre-processing of rs-fMRI data

It is well known that the BOLD signal measured by (rs-)fMRI contains a lot of noise and is usually mixed with non-neural sources of variability which are often not of interest to a researcher [27]. As a consequence, the raw BOLD signal data are not ready for an analysis with (C-)ICA (or with any other kind of analysis). It is therefore, in order to produce valid results, necessary to pre-process the data first before performing any kind of analysis. Specific pre-processing steps for (rs-)fMRI involve, among others, brain extraction, motion correction, temporal and spatial filtering, non-linear registration to a standard MNI152 space and spatial smoothing (for the pre-processing steps used for the example data discussed in Section 4 see, [28]). These pre-processing steps can be performed by using specialized software tools such as FSL [20]. Discussing these tools, however, goes beyond the scope of the current paper (for an example see, [29]).

Besides the standard pre-processing steps for rs-fMRI data discussed above, two further pre-processing steps should be considered when

[1] To keep the text short and focused, for the rest of this section and Section 2, we will only discuss the example of rs-fMRI data. However, a similar argumentation can be given for other types of three-way data (see also Section 2.1 and Section 4.2 for an example using three-way data concerning stock market data). Note, for example, that ICA has been used in many diverse fields, like biomedical engineering, medical imaging, speech enhancement, remote sensing, communication systems, exploration seismology, geophysics, econometrics and data mining [11].

[2] Often, in order to interpret the RSNs, the common set of RSNs are projected back to each individual's data space using a back projection procedure from the GIFT toolbox (known as back-reconstruction see; [17]) or a dual-regression step that is available in FSL [18]. As such, different RSNs per subject are obtained, which reside in the data space of each subject and can be linked in a one-to-one fashion to the common RSNs.

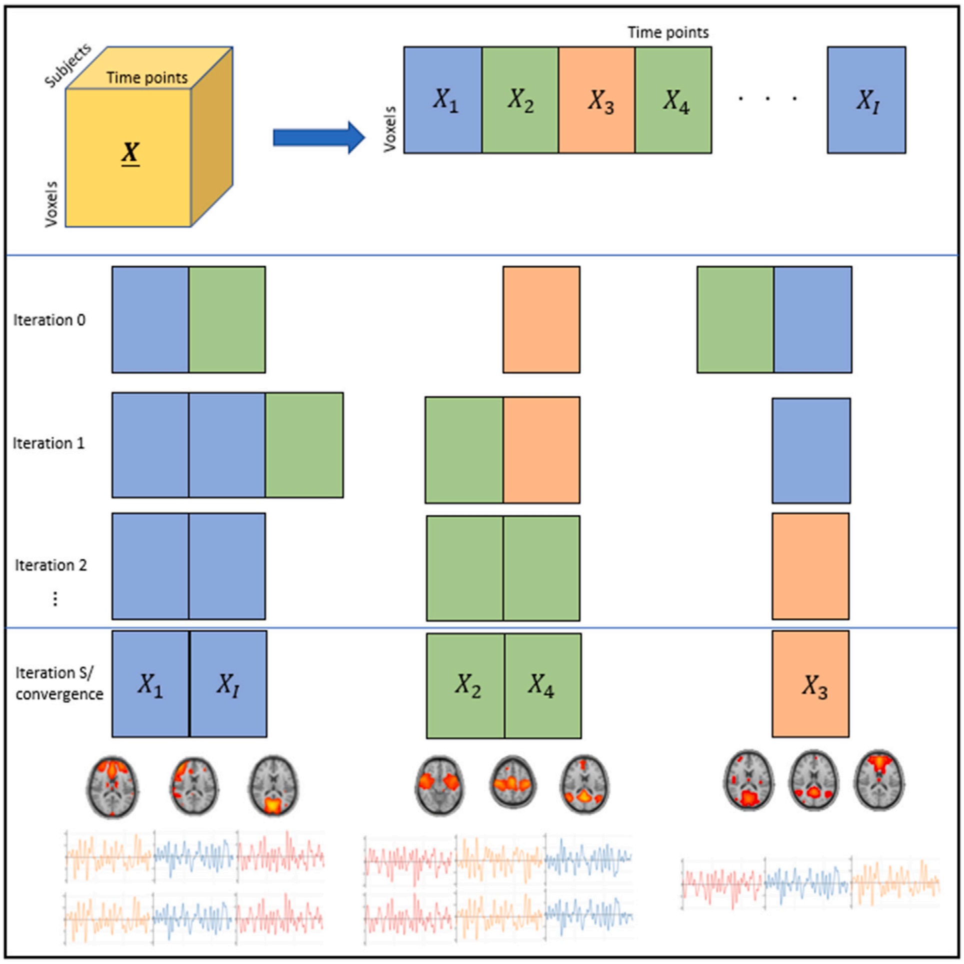


Fig. 1. Graphical illustration of Clusterwise Independent Component Analysis (C-ICA). In the upper panel, a three-way data array (in this example, resting-state fMRI data: voxel by time by subject) is displayed. The slices of the array denote the rs-fMRI data of different subjects and are color coded to indicate the true subject clustering (i.e., subjects with the same color belong to the same cluster). This array is stored as a `list` object in R, with each element of the list being a different data slice (i.e., subject). In the middle panel, the clustering procedure is visualized. Starting from an initial subject partition with $R = 3$ clusters, which could be randomly or (pseudo-)rationally [26] obtained or user defined (iteration 0), the clustering is updated in several iterations (iteration 1, 2, …, S). In each iteration of the algorithm, Independent Component Analysis (ICA) is performed on the (concatenated) data of each cluster to estimate cluster-specific parameters, and subjects/matrices are reallocated to their best fitting cluster. These steps are repeated until convergence (iteration S). In the bottom panel, once the algorithm has converged resulting in an optimal subject partition (with again $R = 3$ clusters), for each cluster, the RSNs (i.e., independent components displayed in the brain figures) and associated time courses (i.e., mixing matrices) are estimated by concatenating along the temporal dimension the voxel × time matrices of subjects belonging to the cluster in question and applying ICA to this concatenated matrix. Note that the number of RSNs Q per cluster is kept constant across clusters (here at $Q = 3$). Note further that the time courses associated to the RSNs are subject-specific (i.e., differ between subjects), whereas the RSNs are only cluster-specific. Therefore, for the first and second cluster, two sets of $Q = 3$ time courses are displayed and only a single set of $Q = 3$ time courses is displayed for the third cluster (i.e., per cluster, the number of time courses equals the number of subjects).

applying C-ICA. First, a row-wise centering of each data matrix $\mathbf{X}_i$ such that each voxel in each matrix has a mean value of 0 across time. Second, each data matrix $\mathbf{X}_i$ can be normalized, resulting in all data matrices having an equal amount of variability and all matrices having the same weight in the C-ICA analysis. This pre-processing step, which is known as block scaling (and is implemented in the presented software), consists of scaling the values in a data matrix such that the sum of squared entries of that matrix equals a given value (see Section 3.1). This pre-processing step is advised in order to account for the large, but arbitrary, scale differences that may exist between the observed data matrices, which impact the estimates of the C-ICA parameters and obscure the underlying cluster structure. Note that similar pre-processing steps are also warranted when applying C-ICA to other types of data than rs-fMRI.

2.1.3. Other types of three-way data

Although we will mainly use multi-subject rs-fMRI data to illustrate the **CICA** package in this paper (see Section 4.1), three-way data also often occur in various other fields of science. Therefore, as a second example, we also illustrate the **CICA** package using multi-country stock market data (see Section 4.2). Also there, C-ICA and the `CICA` package can be used to model heterogeneity in ICA patterns underlying the data. Further examples of three-way data, which are presented in Table 1, that come from different areas of science and to which C-ICA can be applied are listed below:

- Task based -instead of resting state- multi-subject fMRI data (i.e., subjects by voxels by time/trials/tasks) that can be used to extract functional connectivity patterns that are related to task characteristics [30].
- In bioinformatics, as also briefly mentioned in the introduction, C-ICA can be used to discover different genetic regulatory pathways accounting for synergistic effects of multiple genes that are underlying the data of different patient groups [31].
- In chemometrics, three-way data are encountered frequently [32]. In particular, dynamic or time-resolved (or longitudinal) metabolomics or proteomics data are analyzed to uncover the underlying mechanisms in the metabolism -in terms of metabolites or proteins- that, for example, are related to the onset of diseases.

Table 1
Overview of several types of three-way data that stem from different fields of science. For each data type, the three ways involved are listed, along with the substantive interpretation of the ICA components underlying the data. Regarding clustering, often heterogeneity in ICA components is expected between the entities of the first way.

Field	Data	Way 1	Way 2	Way 3	ICA component
Neuropsychology	resting-state or task-based fMRI	subjects	voxels	time/trials/tasks	(rs-) networks
Bioinformatics	microarray gene expression data	genes	subjects	time	regulatory pathways
Chemometrics	metabolomic/proteomic data	subjects	metabolites/proteins	time	biological mechanisms
Text mining	text streams	text	vocabulary (terms)	time (period)	topics
Finance	dynamic transaction-level data	traders	trade behaviors	time	trading trends
Finance	Stock market data	countries	trading days	companies	stock trends
Marketing	product purchasing data	subjects	products	variables	purchasing styles
Psychology	longitudinal behavioral data	subjects	behaviors/symptoms	time/conditions	syndromes/profiles

C-ICA allows for disentangling patient groups that are governed by different underlying biological mechanisms.

- C-ICA can be adopted in text mining in which the goal is to infer topics common to text streams that are coded based on a vocabulary (i.e., text stream by term by time period data; [33,34]). C-ICA can be used to groups text streams in terms of the topics underlying these streams.
- Financial data can give rise to three-way data. For example, transaction-level data contain information about trading behaviors from several traders followed over time (i.e., traders by trading variables by time). ICA can be used here, for example, for portfolio selection [35]. The goal here is to infer correlated trading patterns or trends and C-ICA can cluster traders based on the trading patterns they use.
- Stock market data can also give rise to three-way data (also see Section 4.2). For example, daily closing prices from different stocks across different countries (i.e., country by trading days by companies) can be collected. ICA can be used to extract general patterns from these stocks that are related to macro economical trends. As such C-ICA can be used to cluster countries based on these stock market trends.
- In marketing, researchers want to uncover purchasing styles based on data in which several subjects rate products on a set of (purchasing) variables (i.e., subject by product by purchasing info variables). C-ICA can function here as a way to uncover subject heterogeneity in purchasing styles [36].
- In psychology, based on data about symptoms or behaviors of subjects over time (i.e., subject by symptom/behavior by time), researchers try to infer the syndromes patients suffer from that are causing these behaviors and symptoms. C-ICA can be used to group subjects based on the syndromes they suffer from -and herewith account for syndrome comorbidity- in order to, for example, subtype psychiatric disorders.

Of course, also for these example data sets, appropriate pre-processing steps, which may be specific for the field of application and/or the type of data (e.g., specific steps for microarray vs chemometric data), are needed to get valid results when analyzing these data with C-ICA.

2.2. C-ICA model

In order to cluster subjects based on underlying RSN patterns and the similarities and differences therein (between subjects), in C-ICA the subjects are allocated to R mutually exclusive non-empty clusters (i.e., a subject partition). To this end, each $\mathbf{X}_i$, which is the voxel by time ($V \times T$) data matrix of subject i, is decomposed as follows:

$$\mathbf{X}_i = \sum_{r=1}^{R} p_{ir}\mathbf{S}^r\mathbf{A}_i^T + \mathbf{E}_i \tag{1}$$

where the binary elements p_{ir}, which can be collected in the binary partition matrix $\mathbf{P}$ ($N \times R$), equal 1 when subject i belongs to cluster r and equals 0 when subject i is not allocated to cluster r. Further, the independent components or RSN patterns for cluster r ($r = 1, \dots, R$) are presented in the columns of the $\mathbf{S}^r$ ($V \times Q$) matrices. The $\mathbf{A}_i$ ($T \times Q$) matrices contain the time courses (i.e., mixing matrix) for subject i that are associated to the common RSNs for the cluster r to which subject i belongs. The number of RSNs is indicated by Q and in C-ICA is kept equal across clusters (and subjects). In C-ICA, the RSN patterns in $\mathbf{S}^r$, which are specific for each cluster, are considered as the common group RSN patterns that best represent the data $\mathbf{X}_i$ of all subjects i being allocated to the cluster in question. For these common RSNs it is assumed that they are (as) non-normally distributed (as possible) and (as much as possible) mutually statistical independent. Finally, the elements in $\mathbf{E}_i$ are the error terms for each data block i.

2.3. CICA loss function

The parameters of the C-ICA model are the partitioning matrix $\mathbf{P}$, the subject specific mixing matrices $\mathbf{A}_i$ and the cluster specific independent component matrices or RSNs matrices $\mathbf{S}^r$. The aim of an analysis with C-ICA -given values for the number of clusters R and components Q- is to find the values of these parameters that minimize the following least squares loss function:

$$L(\mathbf{P}, \mathbf{S}^r, \mathbf{A}_i) = \sum_{i=1}^{N} \|\mathbf{X}_i - \sum_{r=1}^{R} p_{ir}\mathbf{S}^r\mathbf{A}_i^T\|^2 \tag{2}$$

2.4. C-ICA algorithm

In order to estimate the parameters $\mathbf{P}$, $\mathbf{S}^r$ and $\mathbf{A}_i$ of the C-ICA model (see Eq. (1)), an Alternating Least Squares (ALS) type of algorithm is used. The C-ICA algorithm alternates between two (conditional) updating steps until no improvement in the loss function Eq. (2) occurs: (1) re-estimating the ICA parameters $\mathbf{S}^r$ and $\mathbf{A}_i$ while keeping $\mathbf{P}$ fixed, and (2) updating the partition matrix $\mathbf{P}$ while fixing the $\mathbf{S}^r$ and $\mathbf{A}_i$ matrices. More specifically, the algorithm takes the following four steps:

1. Find starting values for the partition matrix $\mathbf{P}$. To this end, each subject is assigned to one of the R clusters randomly, implying an initial subject partition with (non-empty) clusters of (almost) equal size. The algorithm, however, can also be seeded with a user-defined start or a (pseudo-)rationally obtained start, like the partition obtained from the two step clustering procedure of [26].
2. Update the cluster specific C-ICA parameters $\mathbf{A}_i$ and $\mathbf{S}^r$ conditionally on the current subject partition in $\mathbf{P}$. This update is performed -per cluster- by, first, temporally concatenating the data $\mathbf{X}_i$ of all subjects i that were assigned (in the previous step) to cluster r, which results in a large matrix $\mathbf{X}^r$. Next, for each cluster r, the ICA parameters are obtained by applying ICA to each $\mathbf{X}^r$, resulting in an update of $\mathbf{S}^r$ and the $\mathbf{A}_i$'s for each subject belonging to that cluster. After computing the cluster specific ICA parameters, the C-ICA loss function (see Eq. (2)) is re-evaluated (see Step 4).

3. Re-estimate the subject partition matrix $\mathbf{P}$ while keeping the cluster specific ICA parameters $\mathbf{A}_i$ and $\mathbf{S}^r$ fixed. In this step, for each subject separately, the optimal cluster membership is determined by evaluating for each cluster the fit of the subject under consideration to that cluster. The (mis)fit of a subject to a cluster is measured by the partition criterion $L_{ir} = \|\mathbf{X}_i - \hat{\mathbf{X}}_i^{(r)}\|^2$ and each subject is (re-)allocated to the cluster r for which L_{ir} reaches its minimal value. In the L_{ir} formula, $\hat{\mathbf{X}}_i^{(r)}$, which denote the predicted values when subject i is assumed to belong to cluster r, can be calculated as $\hat{\mathbf{X}}_i^{(r)} = \mathbf{S}^r \hat{\mathbf{A}}_i^{(r)^T}$. Here, $\mathbf{S}^r$ is computed in step 2 and $\hat{\mathbf{A}}_i^{(r)}$ is calculated through the formula $\hat{\mathbf{A}}_i^{(r)} = \mathbf{X}_i \mathbf{S}^{r^T} (\mathbf{S}^r \mathbf{S}^{r^T})^{-1}$, in which $(\ldots)^{-1}$ denotes a matrix inverse and $\mathbf{S}^{r^T}$ the transpose of matrix $\mathbf{S}^r$.
4. Check the convergence criterion. Steps 2 and 3 are repeated until the difference in loss function values between two consecutive evaluations is smaller than some user specified tolerance value (e.g., .000001).

The above presented ALS algorithm, which is a special case of a block relaxation algorithm [37], results in a series of loss function values (Eq. (2)) that are nonincreasing (i.e., the optimization criterion improves -or stays the same- in each step). As the loss function is bounded from below by zero (i.e., squared loss cannot become negative), the ALS algorithm -under very mild conditions- is guaranteed to converge to at least a local optimal solution [37].

Indeed, the ALS algorithm is sensitive to local minima, a problem that is also encountered with the most commonly used clustering procedures in the literature (i.e., k-means clustering and model-based clustering; see, for example, [38,39]). It is therefore advised to rerun the algorithm several times using random initializations of the partition matrix $\mathbf{P}$ and selecting the best solution encountered across all these runs (i.e., a multi-start procedure). In particular, randomly determining an initial subject partition often leads to a starting partition with (almost) equally sized clusters. There is, however, no guarantee that the optimal solution has equally sized clusters. As a consequence, in general, random start partitions are very far off from the optimal partition and often -after performing ALS iterations- result in a local optimal solution that is suboptimal. A way to improve the probability of the algorithm retaining the global optimal solution is to find starting partitions that are closer to the -unknown- global optimal partition. Therefore, it is also recommended to additionally seed the ALS algorithm with a set of rational and pseudo-rational starts. As an example of a rational start, one can use the estimated subject partition(s) from the clustering procedure of [26]. In this procedure, single subject ICA is performed on the data $\mathbf{X}_i$ of each subject separately with the given number of RSNs (Q).[3] Next, for each pair of subjects, the modified-RV coefficient [40] is calculated between the subjects' RSNs ($\mathbf{S}_i$). After storing these values in a (dis)similarity matrix, several rational partitions can be obtained by applying hierarchical clustering -using different linkage methods- to this (dis)similarity matrix. The pseudo-rational partitions can be obtained by slightly perturbating the rationally obtained partitions (e.g., switch cluster labels for, for example, 10% of the subjects). Note that the idea of initializing a clustering procedure using a partition retrieved from another clustering procedure is quite common and has met increasing support in the literature throughout the years (for an overview, see [41]). The original idea was proposed by [42], where it was shown that initializing k-means using partitions obtained by an agglomerative hierarchical clustering method (e.g., Ward's method) improves cluster recovery. In a similar vein, the partition(s) obtained by the clustering procedure from [26] may improve the cluster recovery of C-ICA.

[3] Performing ICA on each subject separately is equivalent to performing C-ICA selecting for $R = N$ clusters.

Note that the computational load of C-ICA is quite significant. During the ALS iterations, indeed, multiple fastICA runs are performed for each cluster R on relatively large concatenated datasets. In order to overcome this limitation, the **CICA** package has an additional functionality where the ICA based component estimation is replaced with an eigenvalue decomposition (EVD). Although EVD cannot extract the correct direction of the components (i.e., components will not be independent), it is able to derive the correct component space (and hence using EVD does not affect the estimation of P). After finding P (with EVD's) fastICA can be used to retain the correct independent components per cluster. This results in a faster computation of the clustering $\mathbf{P}$, also see Section 4.3. Moreover, it is strongly recommended to perform the multiple starts of the ALS procedure in parallel. These multiple starts can be performed in parallel, either on a computer cluster or locally. For example, scripts can be sourced in parallel on a local machine,[4] with different starts supplied using the `userGrid` argument.

2.5. C-ICA model selection: determining the optimal Q and R

When applying C-ICA, the number of clusters R and components Q present in the data needs to be determined during the analysis as in practice the true number of clusters and components underlying a multi-subject rs-fMRI data set is (almost) never known beforehand. A commonly adopted strategy to address this problem is to run C-ICA with several (combinations of) values for R and Q and using a procedure for model selection to determine the C-ICA model with the optimal R and Q. For example, given a set of C-ICA solutions with varying R and Q, one could use a sequential scree test procedure (for more information, see [1,43]) to identify the optimal R and Q. In short, during this two-step model selection procedure, in the first step, the optimal number of clusters R is determined (across all values for Q) and then, in step 2, conditional on this optimal R, the optimal number of components Q is identified. In both steps, scree ratios are used to determine the optimal number of clusters/components. A scree ratio is computed by dividing the difference in C-ICA loss function values between a less complex model and the model under consideration by the difference in loss function values between the model under consideration and a more complex model. Large scree ratios are desired since this implies that the less complex model fits the data worse compared to the model under consideration (i.e., large numerator), whereas a model that is more complex does not fit the data considerable better than the model under consideration (i.e., a small denominator; see [1] and Section 3.1.3 for a thorough explanation and example of this procedure).

3. The C-ICA package

CICA is the R package the performs the C-ICA procedure by [1]. The package is available at CRAN[5] and can be directly installed using the `install.packages("CICA")` function. **CICA** contains several R functions that can be called by the user. An overview of these functions is given in Table 2 and each of them will be explained in detail in the following sections. In Section 3.1, the different steps of a typical analysis with C-ICA are discussed, along with examples of the functions from the **CICA** package that can be used to perform these steps. Note that all steps will be illustrated using a small data set with simulated data that can be generated using a function that is included in the package. In the following three sections, additional functionality of the **CICA** package is presented that gives the user more options regarding data input (Section 3.2), selecting (pseudo-)rational and user-defined starts to seed the ALS algorithm (Section 3.3) and output handling (Section 3.4).

[4] In Rstudio this can be done by sourcing the script as a background job.
[5] https://CRAN.R-project.org/package=CICA.

Table 2
Overview of the R functions that are available in the CICA package.

Function	Type	Description
C-ICA	main function	Performs C-ICA and returns the results in an object of class `CICA` or `MultipleCICA`*
FindRationalStarts	auxiliary function	Determines (pseudo-)rational starts and returns an object of class `rstarts`
SequentialScree	auxiliary function	Performs the sequential model selection procedure on an object of class `MultipleCICA`
computeRVmat	auxiliary function	Computes RV coefficients between all pairwise matrices of a list (is used by `FindRationalStarts()`)
loadNIfTIs	input data function	Loads NIfTI files from a directory and puts them in a list for use in `CICA()`
Sr_to_nifti	output data function	Writes RSNs estimate by `CICA()` to NIfTI files
matcher	S3 method	Matches estimated RSNs to given template RSNs
plot.CICA	S3 method	Plot method for `CICA` class type
plot.rstarts	S3 method	Plot method for `rstarts` class type
plot.ModSel	S3 method	Plot method for `ModSel` class type
summary.CICA	S3 method	Summary method for `CICA` class type
Sim_CICA	simulation function	Simulate data based on C-ICA model

Note. Most of the auxiliary functions are used within the main function `CICA()` but are also made available (separately) for users such that they also can be employed outside the `CICA()` function. Hence, users can, for example, perform the two step procedure from [26] separately using the function `FindRationalStarts()`. *When C-ICA analyses with different numbers of clusters R and/or numbers of RSNs/components Q are requested, the output is of class `MultipleCICA`.

3.1. Different steps for performing C-ICA

The goal of an analysis with C-ICA is to cluster the subjects based on differences in RSNs underlying their rs-fMRI data. To this end, a C-ICA procedure consisting of several steps is performed. In Fig. 2, a flowchart of the different steps is shown. To illustrate the different steps of a typical analysis with C-ICA, we will use a simulated fMRI data set that contains scans (i.e., voxel activations measured over time) for 60 subjects coming from four true clusters, each with 15 subjects. The data of each subject consists of activity values for 100 voxels measured across 10 time points (i.e., fMRI volumes). We use a relatively low number of voxels and time points in order to reduce computation time for this illustration; in practice often a larger number of voxels and/or time points is used (and can be simulated by the user using the `Sim_CICA()` function). Note that as these are simulated data, we know the true clustering of the subjects as well as the true RSNs per cluster ($\mathbf{S}^r$'s in Fig. 1) and the associated true time courses per subject (i.e., $\mathbf{A}_i$'s in Fig. 1). For generating the true cluster specific RSNs, we simulated five underlying independent non-normally distributed components (i.e., $\mathbf{S}^r$) per cluster. For the true time courses, we simulated subject specific time courses (5 for each subject) from a uniform distribution. In total, the data consists of 60 matrices (i.e., 4 clusters with 15 subjects) of size 100 (voxels) by 10 (time points). The data are not pre-processed. A user, however, is advised to center the matrices such that each row/voxel has a mean equal to zero and to block scale the matrices to have an equal sum of squares across data matrices. These pre-processing steps are performed using the function included in the **CICA** package (see Section 2.1 and analysis steps further below).

After simulating (or loading data as is done in Section 4) the data into R (step 0), the different steps of C-ICA are: (1) performing C-ICA with random and (pseudo-)rational (and user-defined) starts, (2) selecting the optimal number of clusters R and components Q (i.e., model selection) and (3) interpreting the C-ICA output. Below, these steps will be explained in detail and illustrated using the simulated data set presented above.

3.1.1. Step 0. Simulate data

The data can be simulated[6] by:

```
CICA_data <- Sim_CICA(Nr = 15, Q = 5,
    R = 4, voxels = 100, timepoints =
    10, E = 0.4, overlap = .25,
    externalscore = TRUE)
```

When running this code, a data object named `CICA_data` is added to the R global environment. This data object is a `list` object of length 6 where the first element (named `P`) is a vector of length 60 containing the true subject partition (i.e., cluster memberships). Note that the data here consist of 4 clusters (`R = 4`) and 15 subjects per cluster (`Nr = 15`). The second element (named `X`) is a `list` object of length 60 (i.e., the number of subjects in the data). This object contains the input data for the `CICA` function, that is, the 60 voxel × timepoints matrices, one matrix for each of the 60 subjects; here each data matrix contains the BOLD response for 100 (`voxels = 100`) voxels across 10 (`timepoints = 10`) time points.[7] The third element (named `Sr`) in the `CICA_data` list object is a `list` object of length 4 (i.e., the number of clusters), where each element of that list is a voxel × RSN matrix. Here, each matrix contains the five (`Q = 5`) true RSNs per cluster. The fourth element (named `Air`) is a nested list existing of four (`R = 4`) elements, one for each cluster; each element in this list is a new list with as many elements as there are subjects in the corresponding cluster. Each element of the nested list contains the time course matrix $\mathbf{A}_i$ for a particular subject. The fifth element (named `RVs`) is an $R \times R$ symmetric matrix containing all pairwise modified-RV coefficients for the simulated cluster specific component matrices $\mathbf{S}^r$. Note that this object is only included in the output when the argument `overlap` is not `NULL` but a value equal to or larger than zero. A relatively small value for the argument `overlap` (e.g., 0.01) generates a cluster specific component structure that contains a large amount of overlap and vice versa. When `overlap` equals `NULL` (the default option), no overlap is simulated between the clusters specific components. The procedure for

[6] In order to replicate these data the function call `set.seed(1)` should be used before running the simulation function.

[7] The C-ICA algorithm is programmed such that independence between the components corresponding to the rows, which here as the rows pertain to voxels can be interpreted as RSNs, is assumed (i.e., spatial C-ICA). When the user wants to apply C-ICA with independence on the column components of the data (i.e., temporal C-ICA), the data should be transposed first. A list of matrices can easily be transposed using the command `lapply(CICA_data$X t)`. Note that when the current data are transposed and fed to C-ICA, cluster-specific time courses are estimated, which are assumed to be the same for all subjects belonging to a cluster, along with subject-specific RSNs. The goal of our analysis, however, is to find differences in RSNs between subject clusters and thus the rows need to pertain to voxels.

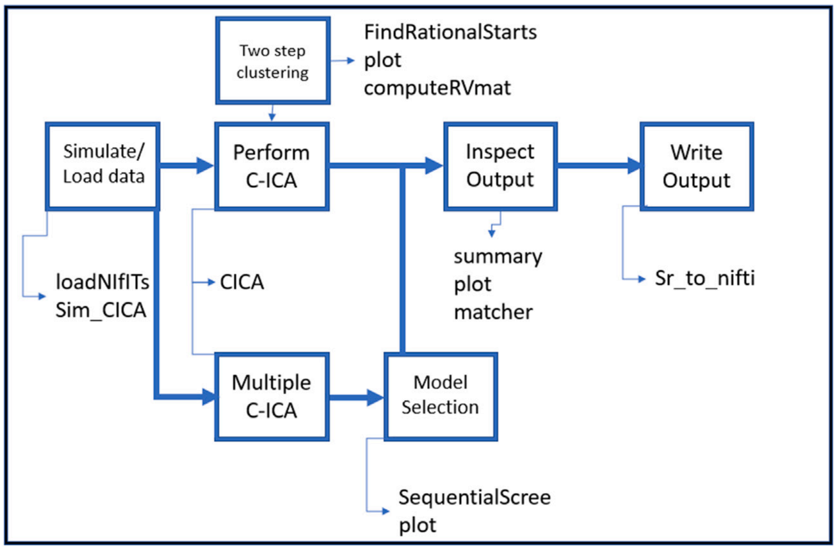


Fig. 2. Flowchart of the different steps of an analysis with C-ICA, with the R functions that can be adopted to perform each step also being listed. For example, in order to load the data for a C-ICA analysis, the `loadNIfTIs()` function can be used.

generating overlap in underlying clusters is more thoroughly described in [1]. Note that the data also contains 40% (`E = 0.4`) of noise. Finally, since the function argument `externalscore` equals `TRUE`, a sixth element (named `externalscore`), which contains a vector with 60 simulated scores from a normal distribution, is present in the output. When the argument equals `FALSE`, no external variable with simulated scores is provided. This vector represents the score on an external variable that can be used in an analysis of variance (ANOVA) test in order to validate the obtained subject clustering (see further and also in Section 4).

3.1.2. Step 1. Performing C-ICA

1a. Main function

The main function from the **CICA** package is `CICA()`, which is used to perform C-ICA with given values for R and Q (and with different types of starts, see step 1b). In order to apply C-ICA to a multi-subject dataset stored in `CICA_data$X`, the following code can be used:

```
multiple_output <- CICA(DataList = CICA_
    data$X, nComp = 2:6, nClus = 1:5,
    method = "fastICA", RanStarts = 30,
    RatStarts = "all", pseudo = c(0.1,
    0.2), pseudoFac = 2, userDef = NULL,
    userGrid = NULL, scalevalue = 1000,
    center = TRUE, maxiter = 100,
    verbose = TRUE, ctol = .000001,
    checks = TRUE)
```

Here, `CICA_data$X` is a `list` object with a length equal to the number of subjects (see above). Each element of this list is a voxel × time matrix (i.e., $\mathbf{X}_i$ in Fig. 1), one matrix for each subject (here, 60 subjects). The arguments `nComp` and `nClus` refer to the number of components and clusters to be estimated, respectively. The `method` argument refers to the type of component estimation to be used, either the default estimation fastICA (`"fastICA"`) or a (faster) eigenvalue decomposition based procedure (`"EVD"`). Note that either an integer or a vector (e.g., `nClus = c(1,3,4,6)`) can be inputted to these two arguments. In the former case, C-ICA will be fitted with the specified number of components and clusters. In the latter case, a C-ICA analysis will be performed for each combination of a specified number of components and clusters. Alternatively, to give the user more control over the requested combinations of the number of components and clusters, a `data.frame` object where the first column represents the number components and the second column the number of clusters can be supplied to the `userGrid` argument.[8] In that case, the input parameters `nComp` and `nClus` are ignored by the `CICA()` function and a C-ICA analysis will be fitted for each row of the data frame with the specified number of components and clusters in that row. Note that we will analyze the simulated data selecting for $R = 1$ up to $R = 5$ clusters and $Q = 2$ up to $Q = 6$ components, which results in 25 C-ICA models being fitted (i.e., the 25 possible combinations of R and Q). In order to select the optimal number of components Q and clusters R, next, a model selection procedure will be employed on the output (see Step 2).

The argument `RanStarts` sets the number of random starts (here 30).[9] The arguments `RatStart`, `pseudo`, `pseudoFac` and `userDef` refer to the use of (pseudo-)rational and user-defined starts and are explained in more detail below (step 1b) and in Section 3.3 (explaining the use of `FindRationalStarts()`). For our analysis, rational starts (8 in total, see further) are requested via the argument `RatStarts = "all"`. The rational starts are also perturbated such that pseudo-rational starts are obtained. Here, 10% and 20% of the cluster labels from the rational starts are switched to another label (i.e., `pseudo = c(0.1, 0.2)`) and this is done twice as specified by the `pseudoFac = 2` argument. User defined starting values can also be inputted through the `userDef` argument. The user defined starting values should be stored in a `data.frame` object where each start is stored in a column.

The arguments `scalevalue` and `center` refer to the pre-processing options of block scaling and mean-centering of voxels, respectively (see Section 2.1). Note that the default setting is that the data matrices (per subject) are row-wise centered (i.e., mean of zero across time points per voxel; `center = TRUE`) and matrix-wise scaled (called block scaling, which implies that each data matrix is normalized to have an equal sum of squared entries of 1000; `scalevalue = 1000`). However, by changing the value of `scalevalue`, data blocks

[8] This can be accomplished, for example, by the following code: `RQrange = as.data.frame(rbind(c(2,4),c(3,5),c(6,2)))` and specifying `userGrid=RQrange` in the `CICA()` call.

[9] In order to avoid the possibility that a user requests too many random starts (which causes a long computation time), the maximum number of random starts is set to 10% of a Stirling number of the 2nd kind. This number represents how many ways N objects can be allocated to R clusters or subsets with the restriction that each cluster contains at least one object.

can be normalized to any desired sum of squares value and by setting `scalevalue = NULL` no block scaling of the matrices is performed (i.e., scale differences between data matrices are kept in the data). Setting `center = FALSE` implies that no row-wise centering is applied. Of course, the user can pre-process the data `X` in any user-defined way and input that pre-processed `X` as input to `CICA()`. In that case, centering (`center = FALSE`) and block scaling (`scalevalue=NULL`) should be switched off.

The argument `maxiter` (default is 100) sets the maximum number of iterations that the ALS algorithm performs and influences the convergence (i.e., when setting `maxiter` too low the algorithm may not converge and yield a very suboptimal solution) and the computation time of the algorithm (i.e., increasing `maxiter` will increase computation time). Also the tolerance value `ctol` (with .000001 as default) influences the convergence of the ALS algorithm as it determines the amount of decrease in loss function value (Eq. (2)) that is minimally needed to let the algorithm iterate further. Note that setting this value too large results in suboptimal solutions (i.e., too fast convergence) and using a too low value unnecessarily lengthens computation time. Finally, when the `verbose` argument equals `TRUE`, which is the default, the progress of the analysis is printed to the console. Here, information about the type of start and the evolution of the loss function value over iterations is shown. Finally, the argument `checks` refers to whether all input arguments should be checked (i.e., `checks = TRUE`). Here, it is checked whether the input arguments are correct. For example, when rational starts are requested, it is checked whether the correct linkage methods are specified.

The output of the `CICA()` function is an object of either class `CICA` (when a single C-ICA model is fitted with a given value for R and Q) or class `MultipleCICA` (when, as in our example, multiple C-ICA models with various R-Q combinations are fitted). The latter is simply a list of multiple `CICA` class objects and can be used as input for a model selection procedure (see step 2). An object of class `CICA` contains 8 different objects. Here, the first object (`P`) contains a vector that denotes the estimated cluster membership for each subject. The second object (`Sr`) is a `list` object with length equal to the specified number of clusters R. Each element contains the estimated RSNs of a particular cluster (i.e., $\mathbf{S}^r$), with each column referring to a different RSN. The object `Ais` is a `list` object containing the estimated subject-specific time course matrices (i.e., $\mathbf{A}_i$), with each element of the list pertaining to a different subject and the subjects being in the same order as in the input data `X`. In each matrix, the time courses are stored in the columns. The fourth object (`Loss`) contains the loss function value (see Eq. (2)) of the optimal solution retained by C-ICA. The fifth object (`IndLoss`) is a vector with length equal to the number of objects and contains the loss function values of each subject separately (i.e., the L_{ir} values of each subject associated to its optimal cluster). Note that the sum of these individual loss function equals the overall loss function. The sixth object (`LossStarts`) is a vector with the loss function values of the converged solutions (i.e., after ALS iterations) for all starts -either random, (pseudo-)rational or used-defined- used in the ALS algorithm. The seventh object (`iterations`) denotes the number of iterations that were performed to obtain the optimal solution that yielded the lowest overall loss function value. Finally, the eighth object `starts` is a `data.frame` that contains all starting partitions (stored as columns) used to seed the ALS algorithm. The column names of this data frame indicate whether the start is a user-defined, random, rational or pseudo-rational start. Moreover, for the latter two types of starts, the column name indicates on what type of hierarchical clustering procedure it is based and specifically for the pseudo-rational starts, the name also includes the percentage of perturbation used to obtain that particular pseudo-rational start (see Section 3.3).

1b. Determining (pseudo-)rational (and user-defined) starting partitions

To increase the probability of the C-ICA algorithm to retain the global optimal subject partition, a multi-start procedure consisting of a mix of random, rational, and pseudo-rational (and user-defined) starts is advised. A random start is generated in the `CICA()` function by randomly dividing the subjects across the requested number of clusters R (specified by the `nClus` argument), hereby guaranteeing that each cluster contains at least a single subject (i.e., no empty clusters are allowed). A rational start partition can be obtained by running a simpler clustering procedure than C-ICA to the data. In the `CICA()` function, a rational start is obtained by running the two step clustering procedure as presented in [26] (see Sections 2.4 and 3.3 for more information). A pseudo-rational initial subject partition can be generated by perturbing to a small extent a rationally obtained subject partition. This is implemented in `CICA()` by randomly reallocating a given proportion (specified via the `pseudo` and `pseudoFac` arguments) of the subjects to another cluster (with equal probability) in such a way that no empty cluster(s) is created. By setting the arguments `RanStarts`, `RatStarts`, `pseudo` and `pseudoFac` in the `CICA()` call, the user can determine the number of random (`RanStarts`), rational (`RatStarts`; maximum is 8, see Section 3.3) and pseudo-rational starts. Note that the argument for `RatStart` should be an unambiguous abbreviation of one of the linkage methods for the `hclust()` function. That is, the argument should be a vector with one (or more) of the following elements: `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"median"` or `"centroid"`. If the argument is equal to `"all"`, the eight aforementioned methods will be used. For each generated rational start, one or more pseudo-rational starts are created by switching for a certain proportion of subjects the cluster label from the corresponding rational start. The value(s) in (the vector) `pseudo` indicates the proportion of subjects for which cluster labels should be switched and `pseudoFac` denotes how often this switching procedure should be repeated. Note that this results in multiple pseudo-rational starts being generated from a given rational start with a given proportion of labels to be switched. For example, when three rational starts were determined, `pseudo=c(.10,.20)` and `pseudoFac=10`, for each rational start, 10 pseudo-rational starts are generated by switching 10% of the labels and 10 pseudo-rational starts are created by changing 20% of the labels. As such, for each of the eight rational starts, 20 (i.e., `length(pseudo) * pseudoFac`) pseudo-rational starts are generated, resulting in total in 60 pseudo-rational starts being fed to the C-ICA algorithm, along with 3 rational starts and `RanStarts` random starts.

Besides random and (pseudo-)rational starts, the C-ICA algorithm can also be supplied with user-defined initial subject partitions. These partitions could result from previous research or could represent expectations or hypotheses of the researcher regarding the optimal subject clustering.[10] These user-defined partitions could also be (pseudo-)rational starts that the user generated by applying some chosen or custom made clustering procedure(s) to the data. After storing the user-defined partition vectors (i.e., cluster membership vectors) as columns in a `data.frame` object, these starts then can be provided to C-ICA by the `userDef` argument (for more information, see also Section 3.3).

3.1.3. Step 2. Model selection

When calling the `CICA()` function, the user needs to specify the number of clusters R and components Q. As these numbers in practice are almost always unknown, an often adopted solution is to apply C-ICA

[10] By adding user-defined starts, researchers can add a confirmatory aspect to the cluster analysis, which is typically a fully exploratory analysis. Indeed, when, for example, existing diagnostic labels for the subjects are entered as a user-defined start, it can be evaluated how close these labels are (in terms of grouping and loss function value) to the optimal solution retained by C-ICA.

with several values for R and Q[11] and to select the optimal R and Q relying on, for example, a sequential model selection procedure (see Section 2.5). To this purpose, the `CICA()` function from the **CICA** package contains an option for running C-ICA models with several values of R and Q and a function `SequentialScree()` that performs the sequential model selection procedure. In step 1a of Section 3.1, it was shown how to fit multiple C-ICA models, selecting for multiple combinations of values of R and Q. In our example, C-ICA models were fitted with (all possible combinations of) 2 to 6 components and 1 to 5 clusters. Thus, in total, 25 different C-ICA models were fitted and the output is stored in a named list of class `MultipleCICA`. This output object, in order to perform the model selection, can be passed to the `SequentialScree()` function as follow[12]:

```
ModSelOutput <- SequentialScree(multiple
    _output)
```

By calling this function, the sequential scree test procedure (for more information, see [1,43]) is performed using the loss function values of the multiple fitted C-ICA models with varying Q and R (see Section 2.5). The output of this function is an object of class `ModSel` and an associated summary S3 method provides information about the selected model, the optimal R and Q value and the scree test values computed in the intermediate steps.[13] For the current simulated data example, the optimal number of clusters and RSNs is equal to the true number of clusters ($R = 4$) and RSNs ($Q = 5$) that were used to generate the data. In R, this can be requested as follow: `ModSelOutput$optimalQ` and `ModSelOutput$optimalR`.

Additionally, an S3 plot method is available for objects of class `ModSel`, which plots a multiple lines scree plot. This plot can be requested as follows:

```
plot(ModSelOutput)
```

As a result, as displayed in Fig. 3, the loss functions values are plotted against the number of RSNs Q for different number of clusters R (i.e., one line in a different color for each number of clusters). The selected model is clearly indicated with a dark green star shaped point. This plot shows that, for each number of clusters, the loss values decrease when adding more RSNs but this only up to $Q = 5$. Further, loss values also decrease when the number of clusters is increased and this until $R = 4$.

Of course, the user can also apply other model selection methods, like, for example the CHull method [44,45] for which an R package called **multichull** exists [46].

[11] Note that the first and last value for Q and R cannot be selected by definition and therefore a suitable range for R and Q should be specified.

[12] Note that a user can also pass a `data.frame` object to this function, where the rows pertain to the different CICA models fitted with varying R and Q. The first column should denote the number of components Q, the second column the number of clusters R and the last column the corresponding loss function value.

[13] The output of this function is a list with six elements, the first two denote the optimal number of selected Q and R, respectively The third elements is a `data.frame` which is used in the model selection procedure. It contains information about the fitted models (combinations of Q and R) and their associated loss function value. The fourth element contains a matrix with the computed scree test values from the first step, the fifth element is a vector with mean scree test values (averaged over the different Q). The maximum value here refers to the optimal number of clusters R. The sixth element is a vector of scree test values conditional on the optimal number of clusters computed in the first step. Similar as before, the maximum value refers to the optimal number of RSNs Q.

3.1.4. *Step 3. Interpreting C-ICA output*

Step 3a. Subject clustering

In order to facilitate the readability of the output of the `CICA()` function, a S3 summary method for output objects of class `CICA` is included in the **CICA** package. For example, the output for the C-ICA solution with $R = 4$ clusters and $Q = 5$ components can be printed in the console by calling:

```
summary(multiple_output$Q_5_R_4)
```

The `summary()` function prints three pieces of information about the solution (with the requested R and Q) retained by the C-ICA algorithm (see below). Note that when multiple C-ICA models were fitted, a user should specify a specific model (via `multiple_output$Q_5_R_4`, for example). First, the estimated subject partition is presented in a table (i.e., matrix **P**). The subjects of the estimated partition are simply numbered from 1 to N by default, but can also have names depending on whether the input data list supplied to `CICA()` also contains names. Secondly, a frequency table indicating the number of subjects per cluster for the retained solution is printed. For our example data set, the solution retained by C-ICA has four clusters with 15 subjects each. Finally, the loss function value of the retained solution is displayed to the console (here, 22649.44).

```
Partitioning matrix P:

        Cluster 1 Cluster 2 Cluster 3 Cluster 4
Subj 1          1         0         0         0
Subj 2          1         0         0         0
Subj 3          1         0         0         0
Subj 4          1         0         0         0
Subj 5          1         0         0         0
Subj 6          1         0         0         0
Subj 7          1         0         0         0
Subj 8          1         0         0         0
Subj 9          1         0         0         0
Subj 10         1         0         0         0
Subj 11         1         0         0         0
Subj 12         1         0         0         0
Subj 13         1         0         0         0
Subj 14         1         0         0         0
Subj 15         1         0         0         0
Subj 16         0         1         0         0
Subj 17         0         1         0         0
Subj 18         0         1         0         0
Subj 19         0         1         0         0
Subj 20         0         1         0         0
Subj 21         0         1         0         0
Subj 22         0         1         0         0
Subj 23         0         1         0         0
Subj 24         0         1         0         0
Subj 25         0         1         0         0
Subj 26         0         1         0         0
Subj 27         0         1         0         0
Subj 28         0         1         0         0
Subj 29         0         1         0         0
Subj 30         0         1         0         0
Subj 31         0         0         1         0
Subj 32         0         0         1         0
Subj 33         0         0         1         0
Subj 34         0         0         1         0
Subj 35         0         0         1         0
Subj 36         0         0         1         0
Subj 37         0         0         1         0
Subj 38         0         0         1         0
Subj 39         0         0         1         0
```

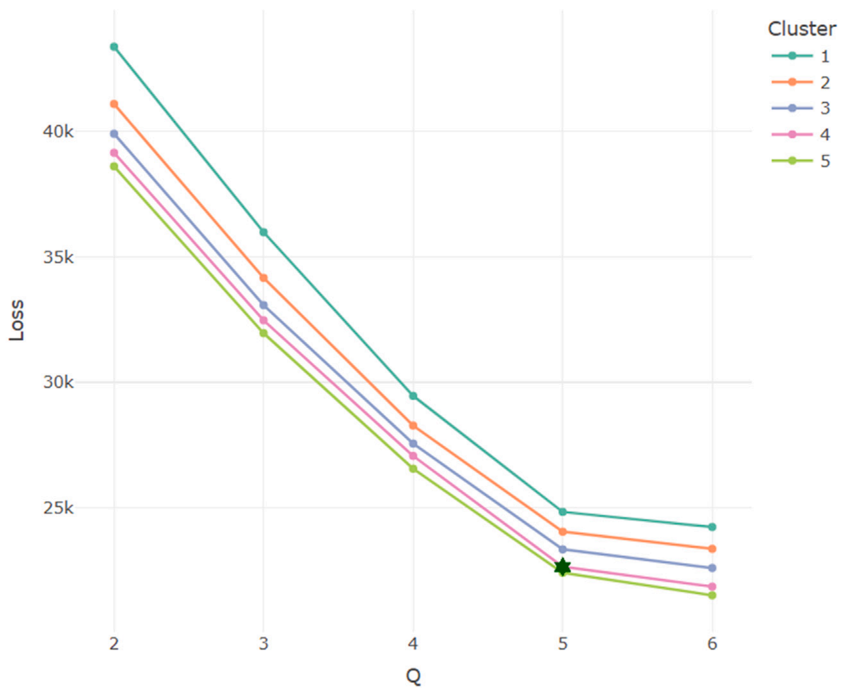


Fig. 3. Multiple lines scree plot where the C-ICA loss function values are presented against the number of components Q for different number of clusters R (indicated by separate lines in a different color). The optimal model in this simulated data example is a model with $Q = 5$ RSNs and $R = 4$ clusters (clearly indicated), which also is the true model underlying the data.

```
Subj 40         0         0         1         0
Subj 41         0         0         1         0
Subj 42         0         0         1         0
Subj 43         0         0         1         0
Subj 44         0         0         1         0
Subj 45         0         0         1         0
Subj 46         0         0         0         1
Subj 47         0         0         0         1
Subj 48         0         0         0         1
Subj 49         0         0         0         1
Subj 50         0         0         0         1
Subj 51         0         0         0         1
Subj 52         0         0         0         1
Subj 53         0         0         0         1
Subj 54         0         0         0         1
Subj 55         0         0         0         1
Subj 56         0         0         0         1
Subj 57         0         0         0         1
Subj 58         0         0         0         1
Subj 59         0         0         0         1
Subj 60         0         0         0         1

Tabulation of clustering:

Cluster 1 Cluster 2 Cluster 3 Cluster 4
15        15        15        15

Loss function value of optimal solution is:
22649.44
```

When the user has some additional information about the subjects (e.g., demographic or symptom information), the obtained subject partition can be validated by, for example, testing in an ANOVA or a pairwise t-test (with corrections for multiple testing) whether (and which) subject clusters differ on some external variable(s). For example, various symptoms may be more prevalent in some clusters than in other clusters. To this end, the subject clustering (stored in `multiple_output$Q_5_R_4$P`) can be inputted as a dependent variable in an ANOVA[14] or pairwise t-test. By running the code below, we can test whether the obtained subject clusters differ in terms of a clinical score (e.g., Mini Mental State Examination -MMSE- which is an often used measure for evaluating cognitive functioning):

```
pairwise.t.test(CICA_data$externalscore,
     multiple_output$Q_5_R_4$P, p.adjust
    .method = "bonferroni")
```

Which gives the following output:

```
Pairwise comparisons using t tests with pooled SD

data: CICA_data$externalscore and multiple_output$
Q_5_R_4$P

1      2      3
2 <2e-16      -      -
3 0.021  <2e-16      -
4 <2e-16 0.081  <2e-16

P value adjustment method: bonferroni
```

In our example data, it can be seen that subjects in each cluster significantly (at an alpha level of .05) differ from each other except for subjects from cluster 2 and 4 ($p = 0.081$). In order to compute the marginal means for the MMSE score, the following code can be used:

```
aggregate(CICA_data$externalscore, list(
    multiple_output$Q_5_R_4$P), FUN =
    mean)
```

Giving the following result:

[14] One can perform an ANOVA with the following code: `aov(CICA_data$externalscore multiple_output$Q_5_R_4$P)`.

```
  Group.1        x
1       1 13.05894
2       2 26.78497
3       3 11.94004
4       4 25.84787
```

Both cluster 1 and 3 seem to be more cognitively impaired (as indicated by a lower mean MMSE score) than subjects from cluster 2 and 4. The use of this type of tests might help a user to validate and/or interpret the obtained subject clustering, which is based on neurofunctional information, by relating it to an external (clinical) score.

As the data used here was simulated and thus the true clustering is known, we can evaluate how good C-ICA uncovered this true clustering. To this end, the Adjusted Rand Index (ARI; [47]) from the **mclust** package [48] can be used.[15] The ARI is a measure that compares two partitions and equals zero when both partitions do not agree above chance level and equals one in the case of a perfect agreement between the two partitions. This measure is ideal since it accounts for arbitrary cluster permutations (i.e., the label switching problem which is also present for many other clustering procedures; also see [1]). The following code can be used to compute the ARI between the true partition (stored in `CICA_data$P`) and the retained partition by C-ICA (stored in `multiple_output$Q_5_R_4$P`):

```
adjustedRandIndex(CICA_data$P, multiple_
    output$Q_5_R_4$P)
```

It appears, as can be seen in Table 3 (top row), that C-ICA uncovered the true partition perfectly as it has an ARI of one (fourth column). This can also be seen by making a confusion table with the true (in the rows) and obtained (columns) clustering:

```
table(CICA_data$P, multiple_output$Q_5_R
    _4$P )
```

The result, as can be seen in the code output below, shows a one-to-one (perfect) match between the true and the obtained clusters (hence, ARI equals 1 as indicated above).

```
   1  2  3  4
1  15 0  0  0
2  0 15  0  0
3  0  0 15  0
4  0  0  0 15
```

In a similar way, we can assess the quality of the rational and pseudo-rational starting partitions used in the C-ICA algorithm (in this case based on eight different types of linkage methods). These partitions were obtained from the two step clustering procedure presented in [26], which can be computed with the `FindRationalStarts()` function (see Section 3.3). Since rational starts were requested in the `CICA` function call, the generated rational initial subject partitions are included -as columns- in the matrix `multiple_output$Q_5_R_4$starts` and can be evaluated as follow:

```
apply(multiple_output$Q_5_R_4$starts,
    MARGIN = 2, FUN = adjustedRandIndex,
     y = CICA_data$P)
```

[15] The Balanced Accuracy, which is more often used in supervised learning applications, can also be used. This measure represents the arithmetic mean of sensitivity and specificity. After accounting for possible cluster permutations, one could compute the balanced accuracy (and other classification evaluation statistics) using the **caret** package [49].

With this `apply` function call, the estimated rational and pseudo-rational starting partitions (but also the used random starting partitions) are compared to the true subject partition (i.e., `CICA_data$P` using the function `adjustedRandIndex()`).[16] The resulting ARI values are shown in Table 3 (only for the rational starts), with the values in the second column pertaining to the initial partitions (before ALS iterations) and the values in the fourth column to the partition after running the ALS iterations (i.e., converged solution). It seems, as can be seen from this table (fourth column) that, apart from the rational starts based on the two versions of Ward's linkage method and McQuitty, none of the rational starts based on a (other) hierarchical clustering linkage method resulted in the true partitioning (i.e., ARIs not being equal to one). Moreover, by comparing the second column (before ALS iterations) to the fourth column (converged solution), almost all of these rational starts directly converged, after one iteration, into a local minimum (i.e., their loss values are equal to the loss value of the solution retained by C-ICA). The rational starts based on both types of Ward's linkage methods and McQuitty performed the best as they resulted in a perfect solution (i.e., ARI = 1) with the same loss function value as for the C-ICA retained solution. Note that the initial subject partitions based on these linkage methods already closely resembled the true clustering (ARI = .955 and .790), implying that it was relatively easy for the ALS iterations to move towards the global minimum as the initial start was already close to this minimum. Note that this is a finding similar to the results from [42], where it was observed that rational starting partitions are more likely to yield good results when the starting partition is already close to the (optimal) final solution. Remarkably, as can be seen in the top row of Table 2, the initial (random) partition from the ALS run that yielded the optimal C-ICA solution did not resemble the true partition at all (i.e., ARI of .029). These results indicate again that it is important to seed the C-ICA algorithm with enough starting partitions that are a good mix of random and (pseudo-)rational starts. Of course, adding more starts also increases the computation time (unless infrastructure that allows for parallel computing is used). As such, the procedure of [26] can be used as a quick procedure to get rational starts that give a reasonable -but not optimal- subject partition. C-ICA, however, takes a bit longer but yields a better subject partition.

In practice, of course, the true partition is not known and the ARI between the C-ICA retained solution and the true solution thus cannot be computed. However, in that case, the ARI can be used to compare the obtained clustering with an existing labeling of the subjects (e.g., diagnostic labels). This could assist the user in interpreting and validating the obtained clusters (for an example, see Section 4).

Step 3b. RSNs per cluster and subject-specific time courses

To inspect the RSNs per cluster, an S3 plot method is included in the **CICA** package for objects of class `CICA`. In order to call the plot function, the following code can be used:

```
plot(multiple_output$Q_5_R_4, cluster =
    1)
```

The function automatically infers whether the output is based on a C-ICA analysis of (rs-)fMRI input data or whether another type of data was used as input. When fMRI data was inputted, which is inferred by the number of voxels (for an illustration, see step 3b of Section 4), a neuroimage viewer is started where the cluster-specific RSNs (i.e., columns in $\mathbf{S}^r$, see Fig. 1) can be viewed interactively. This viewer is based on the R package **papayar** [50], which is a wrapper to R for the JavaScript based medical research image viewer[17]. When the C-ICA output contains RSNs where the number of voxels is either

[16] `MARGIN = 2` indicates that the function specified in the `apply()` function is performed columnwise.

[17] https://mangoviewer.com/papaya.html

Table 3
Adjusted Rand Indices between the true subject partition and estimated partitions before and after the iterations of C-ICA (only reporting rational starts).

Method	ARI of start partition	Loss of start partition	ARI of solution	Loss of solution
C-ICA*	.029	24 044.20	1	22 649.44
Ward.D	.955	22 683.09	1	22 649.44
Ward.D2	.955	22 683.09	1	22 649.44
Complete	.453	23 202.88	.675	23 045.14
Single	0	24 021.55	0	24 021.55
Average	−.001	24 029.74	−.001	24 029.74
McQuitty	.790	22 866.60	1	22 649.44
Median	−.001	24 031.79	−.001	24 031.79
Centroid	−.001	24 029.74	−.001	24 029.74

Note. Adjusted Rand Index (ARI) between the true subject partition and the solution retained by C-ICA and the estimated rational starts (based on hierarchical clustering using eight different linkage methods). The second and fourth column present the ARI before (i.e., initial partition) and after the ALS iterations (i.e., converged solution), respectively. In the third and fifth column, the loss of the corresponding C-ICA solution before (column 3) and after (column 5) ALS iterations is presented. C-ICA (top row) was run with 30 random starts and 23 out of these 30 random starts converged to the same (optimal) solution.

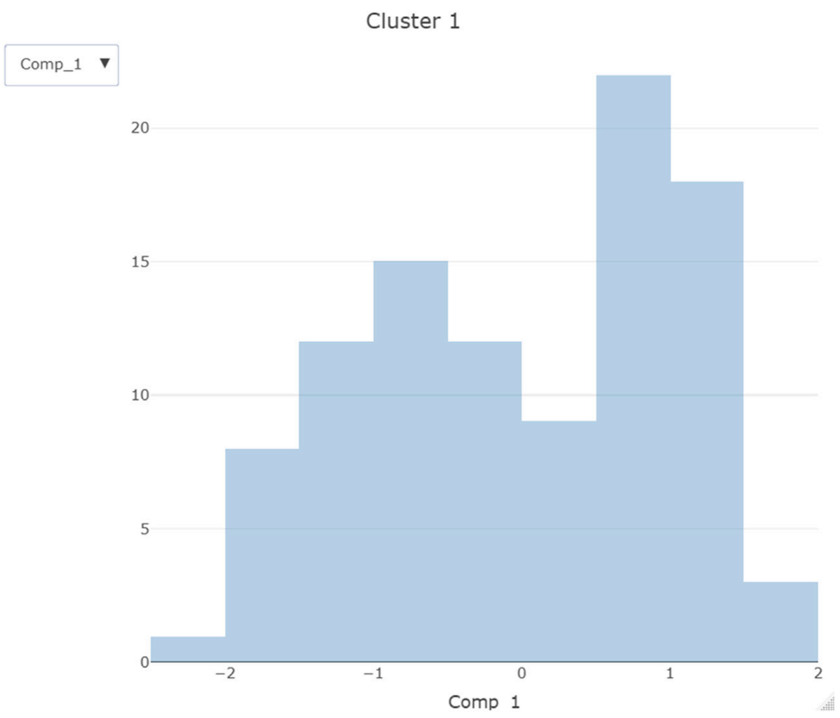


Fig. 4. Plot resulting from applying the S3 plot method on a `CICA` class object, selecting for `cluster = 1`. Since other data than fMRI data are used as input data for this example, histograms of the independent RSNs from the cluster specified by `cluster` (here, cluster number 1) are displayed for further inspection. Note that in R, the independent RSNs can be viewed one by one interactively by selecting the corresponding RSN index via the drop-down menu in the upper left corner of the plot.

14812, 116380 or 902629, a structural brain NIfTI file with a spatial resolution of 8 mm, 4 mm and 2 mm, respectively is used to plot the estimated RSNs on this brain structure (see Section 4 for an example). When no fMRI data were inputted or the number of voxels differs from these three above mentioned numbers, as in the current example, an interactive histogram is plotted using the **plotly** package [51]. This histogram shows the distributions of the RSNs from the cluster (number) that is specified by the input argument `cluster` (see Fig. 4).

It can also be insightful to plot the estimated subject specific time courses ($\mathbf{A}_i$'s). This can simply be achieved by the following code: `plot(multiple_output$Q_5_R_4$Ais[[subject]][,q])`. Here, `subject` refers to the subject number and `q` to the RSN number (also see Fig. 8).

Step 3c. RSN/component matching

A well-known difficulty when interpreting the output of ICA (and therefore also of C-ICA) is that the RSNs -within a cluster- show up in a random order (as opposed to, for example, PCA where components are ordered in terms of decreasing amounts of variance explained). That is, for different but equivalent (C-)ICA solutions (with the same optimal partition and loss value), the RSNs per cluster can be presented in the output in a different order. In practice this means that a particular RSN (e.g., the default mode network; DMN) can be in a different position when comparing ICA solutions. Moreover, for C-ICA, similar RSNs across clusters are hard to detect as in the output these similar RSNs can have a different position in each cluster (for more information on this cluster and component permutation ambiguity, see [1]). In order to avoid a tedious manual RSN matching procedure, a final S3 method associated with an object of class `CICA` is provided. This function, `matcher()`, automatically matches RSNs from different clusters to each other, and thereby facilitates the interpretation of the estimated RSNs in each cluster. The matching procedure is based on the absolute Tucker congruence [52] value, with a value of 0 meaning similarity between RSNs at chance level and a value of 1 indicating perfect match (after accounting for some trivial scaling and reflection of the RSN). Note that high Tucker values for a RSN across clusters can occur which means that a certain RSN is present in multiple clusters. The function can be used in the following way:

```
matcher(multiple_output$Q_5_R_4,
    reference = 1, RV = TRUE)
```

By default, the RSNs from the first cluster (`reference = 1`) are taken as reference templates against which the RSNs of other clusters are matched/compared to. Here, all RSNs from cluster 1 are compared with the RSNs from other clusters. The user can alter the reference cluster by changing the `reference` input parameter (i.e., `reference = 2` for taking the RSNs of the second cluster as reference templates). As a result, in the output, as shown below, multiple tables are presented. The first table (`matchIndexMatrix`) displays the indices of the matched components, whereas the second table (`matchTuckerMatrix`) shows the amount of agreement of each matched RSN with its template in terms of an absolute Tucker value.[18] Note that the output also contains the Tucker values for all comparisons of the RSNs of all clusters with the template RSNs. In the output below, one can see that, for example, the second component from the reference cluster -which is cluster 1- matches best with component 3 from cluster 2 (with an agreement of .906) and component 1 from cluster 3 (agreement of .885) and component 2 from cluster 4 (agreement of .808).

```
$matchIndexMatrix
Template    Cluster  1 Cluster  2 Cluster  3 Cluster  4
Component 1          1          1          2          1
Component 2          2          3          1          2
Component 3          3          5          4          3
Component 4          4          4          3          5
Component 5          5          2          5          3

$matchTuckerMatrix
Template    Cluster  1 Cluster  2 Cluster  3 Cluster  4
Component 1          1      0.834      0.884      0.741
Component 2          1      0.906      0.885      0.808
Component 3          1      0.664      0.692      0.632
Component 4          1      0.843      0.891      0.834
Component 5          1      0.601      0.684      0.558
```

When the reference argument is a string variable with a path to a folder containing a NIfTI file with templates of known RSNs (e.g., default mode network), the estimated RSNs for each cluster are matched to these known RSN templates (for an example of this use of the `matcher()` function, see Section 4). By using this function, one can easily see, for each cluster, on which position in the output an RSN known from the literature resides and how large the agreement is between this RSN template and the RSN in each cluster that resembles the template the most.

When the argument `RV` of the `matcher()` function equals `TRUE` (note that the default value is `FALSE`), the modified-RV coefficient [40] is computed between all pairs of cluster-specific RSNs and these values are presented as a third table in the output. When `RV=FALSE`, no modified-RV coefficients are computed, which saves computation time. The modified-RV coefficient, which is a matrix correlation, gives an indication of the amount of overall similarity between two sets of RSNs (here, the RSNs of two clusters), with a value of one implying perfect similarity and a value of zero similarity at chance level. These values, which are presented in the code output below, can be used to assess whether clusters are quite similar to or very different from each other in terms of their corresponding RSNs. From these results, it appears that there is quite some overlap between the estimated RSNs between clusters, which is not an unexpected result since the data simulation procedure specifically generated clusters with a large overlap in RSNs. For example, the estimated RSNs from cluster 1 yields a modified-RV value of .827 with the RSNs from cluster 2, whereas the RSNs from cluster 3 and 4 have an modified-RV value of .816.

```
$RVs
      [,1]  [,2]  [,3]  [,4]
[1,] 1.000 0.827 0.833 0.834
[2,] 0.827 1.000 0.840 0.832
[3,] 0.833 0.840 1.000 0.816
[4,] 0.834 0.832 0.816 1.000
```

3.2. Input data functions

Rs-fMRI data are usually stored in a specific open file format, known as a Neuroimaging Informatics Technology Initiative (NIfTI) file.[19] The use of this specific file format allows for compatibility between different neuroimaging software tools, like FSL [20] and GIFT [21]. The **CICA** package can also make use of this specific file format.

A NIfTI file contains fMRI data of a single subject and can be seen as a four-way array where the first three dimensions (x, y, z) refer to voxels (spatial locations in three dimensions) and the fourth dimension (t) is a time dimension. The coordinates on the x dimension refer to coordinates of an axis that divides the left and right part of the brain (the sagittal plane), the y dimension refers to an axis that divides the anterior part and the posterior part of the brain (the coronal plane) and the z dimension refers to an axis that divides the upper part from the lower part of the brain (the axial plane). When fMRI data are analyzed, specific pre-processing steps should be performed first, like non-linear registration to a standard MNI152 space, brain extraction, motion correction and spatial smoothing (see Section 2.1 and the analysis of the empirical data in Section 4). Additionally, it is important to note that for the **CICA** package, the voxels should be vectorized and stored row-wise in a matrix where each row corresponds with a voxel and each column refers to a time point (i.e., volume). The data of each subject should therefore be converted to a voxel × time point matrix $\mathbf{X}_i$.[20]

As C-ICA uses fMRI data of multiple subjects, all $\mathbf{X}_i$ matrices need to be stored in a `list` object. In order to facilitate the use of the **CICA** package, we provide an easy to use function called `loadNIfTIs()` that automatically converts multi-subject (rs-) fMRI data in NIfTI format to a `list` object where each element refers to a voxel × time point matrix of a single subject which can be supplied to the main function `CICA()`. As shown below, this function has two arguments `dir` and `toMatrix`:

```
X <- loadNIfTIs(dir = "<path>", toMatrix
    = TRUE)
```

The first argument should be a character string with the path name of the folder where all the NIfTI files, one for each subject, are stored. The second argument, `toMatrix` is a boolean value where the default argument (TRUE) results in the function storing the NIfTI files as voxel

[18] An absolute value is displayed in the output for easy of reading. The raw Tucker values are also given in the output from the `matcher()`, but omitted from the output for the current example.

[19] https://nifti.nimh.nih.gov/.

[20] Note that a dataset should be stored as a voxel × time point matrix and not as a time point × voxel matrix, which is the convention in the neuroimaging community. The reason for this is that the underlying ICA function `icafast()` from the **ica** package [53] that is used in `CICA()` is programmed in such a way that statistical independence is applied to the components pertaining to the row elements of the matrix. For voxel × time point data this implies that spatial ICA is performed which estimates independent spatial maps (and not independent time courses).

× time point matrices, which is needed when one wants to analyze the data with the `CICA()` function. The matrices of all subjects are collected in a `list` object `X`, which is the required input format for the `CICA()` function (see Step 1a of Section 3.1). When the argument of `toMatrix` equals `FALSE`, the NIfTI files are loaded into the `R` global environment, preserving the original four dimensions of a NIfTI file, which results in `X` being a list of 4D-arrays of class `niftiImage`. Note that when the data are loaded in this format, they cannot be analyzed with C-ICA. This format, however, allows that typical analyses for neuroimaging data (e.g., smoothing data, applying a mask) can be performed on each loaded NIfTI file using the **fslr** package [20,54].[21] Of course, after performing pre-processing options using the **fslr** package, NIfTI files can be first saved and subsequently loaded into `R` using the `loadNIfTIs()` function such that a C-ICA analysis can be performed on the data. Moreover, this alternative format further makes it possible to inspect the NIfTI files one by one by running the command `papaya(list(X[ index ]))`, where `index` refers to the location (i.e., element number) of the required NIfTI file in the output list (here `X`) from the `loadNIfTIs()` function.

3.3. Generating (pseudo-)rational and user-defined starts and associated S3 plotting methods

The ALS algorithm that is used to estimate the parameters of the C-ICA model is sensitive to local minima (see Section 2.3). It is therefore strongly advised, in order to help the algorithm to retain a global minimum, to run the algorithm using multiple starts that are a combination of randomly and (pseudo-)rationally generated initial subject partitions, possibly augmented with a set of user-defined start partitions (based on previous research or a priori expectations of the researcher). Such a multi-start procedure can be implemented in the C-ICA algorithm, as explained in Section 3.1, by setting in the `CICA()` function the arguments `RanStarts`, `RatStarts`, `pseudo` and `pseudoFac` and user-defined starts can be added through the argument `rational`. The **CICA** package also allows the user to generate rational and pseudo-rational starts outside the `CICA()` function and to inspect these user-generated starts through the function `FindRationalStarts()`.

To determine rational starts, the `FindRationalStarts()` (and `CICA()`) function relies on the two-step procedure from [26]. In this procedure, single subject ICA's are performed (with the `icafast()` function from the ica package; [53]) for a given number of components Q (argument `nComp`), yielding Q RSNs per subject. Next, the modified RV-coefficient [40] between all possible pairs of subject-specific RSNs is computed, which quantifies the amount of similarity of two sets of RSNs. This last step can be performed in the **CICA** package through the `computeRVmat()` function, which has as input a list of length N with each element being a matrix with Q columns (one matrix for each of the N subjects and the RSNs stored in the columns). The output of this function is an $N \times N$ dissimilarity matrix of class `dist`,[22] which can be given as input to a function for hierarchical clustering, like `hclust()`. In `hclust()`, the user can choose between several linkage methods (e.g., complete and single linkage, Ward's method) by setting the `method` argument. An initial subject partition with a given number of clusters R (argument `nClust`) can be obtained by cutting the obtained dendrograms at a particular height (which is determined by `cutree`). A pseudo-rational start can be generated by randomly selecting a given proportion of the subjects (argument `pseudo`) and reallocating these subjects with equal probability to a different cluster and this in such a way that no cluster becomes empty. The user can generate rational and pseudo-rational starts through the following call to `FindRationalStarts()`:

```
Out_starts <- FindRationalStarts(
    DataList = CICA_data$X, RatStarts =
    c("single", "complete", "ward.D2"),
    pseudo = c(.1,.2), pseudoFac = 5,
    nComp = 5, nClus = 4, scalevalue =
    1000, center = TRUE, verbose = TRUE)
```

Here, `DataList`, `nComp`, `nClus`, `scalevalue`, `center` and `verbose` work the same as for the `CICA()` function (see Section 3.1). The `RatStart` argument can be used to determine which and how many rational starts need to be computed. In total, eight different options for rational starts are offered, which will be computed when the default `RatStart="all"` is specified. These options correspond with the eight linkage methods implemented in the `hclust()` function (see the explanation of the `method` argument for `hclust()`). When a single string is given for `RatStart`, only a rational start based on that linkage method (e.g., complete, Ward) is computed.[23] Multiple rational starts can be generated by inputting a vector of strings to `RatStart`. In our example here, three rational starts are obtained by using single and complete linkage and Ward's method (`RatStart=c("single", "complete", "ward.D2")`). It is checked whether the obtained rational starts are unique. When this is not the case, the rational start(s) is replaced by a (unique) random start.[24]

To compute pseudo-rational starts, which are always based on the generated rational starts, a single value or a vector of values (in between zero and one) should be inputted at the `pseudo` argument. The inputted value(s) indicates the proportion of labels from each unique rationally obtained partition that are switched (i.e., perturbated) in order to generate pseudo-rational starts. The argument `pseudoFac` specifies how many pseudo-rational starts need to be generated from a given rational start with a given proportion of label switches. In total, `length(RatStart) * length(pseudo) * pseudoFac` pseudo-rational starts will be (maximally) computed, next to `length(RatStart)` rational starts, as for each (unique) rationally obtained partition `length(pseudo) * pseudoFac` pseudo-rational starts will be generated. In our example, for each rational start, 10 pseudo-rational starts will be generated: 5 by switching 10% and 5 by switching 20% (pseudoFac = 5) of the cluster labels from the corresponding unique rational start partition. As such, in total (maximally) three rational and 30 pseudo-rational starts are obtained.

The output of this function (here `Out_starts`) is of class `rstarts` and is a list that contains two elements; `rationalstarts` and `RVdist`. The first element is a `data.frame` where each column is a rationally or pseudo-rationally obtained subject partition vector, with the rational starts being in the most left columns, and the rows corresponding to the subjects (in the same order as in the inputted data in `DataList`). Note that user-defined starts can be added (as new columns) to this `data.frame` (see below). The second object in the output of the function, `RVdist`, which is stored as a `dist` object, is a dissimilarity matrix based on RV coefficients computed between the RSNs of each subject. This dissimilarity matrix was supplied to the hierarchical clustering procedure in order to compute the rational starts (see above). It is of course possible to supply this object to other clustering procedures chosen by the user, which allows the user

[21] Note that FSL should be installed on your machine in order to use the functions from this package.

[22] Note that the modified-RV values are similarities (i.e., a value of 1 implies perfect congruence). However, in this function, the similarities are transformed into dissimilarities by taking the square root of 1 minus the similarity value.

[23] For this argument, the following agglomeration methods are available: "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" or "centroid". Note that these are the eight different methods used by the `hclust()` function from the **stats** package.

[24] Random starts are generated until a random start is found which is not present yet in the set of starting partitions, herewith also preventing empty clusters.

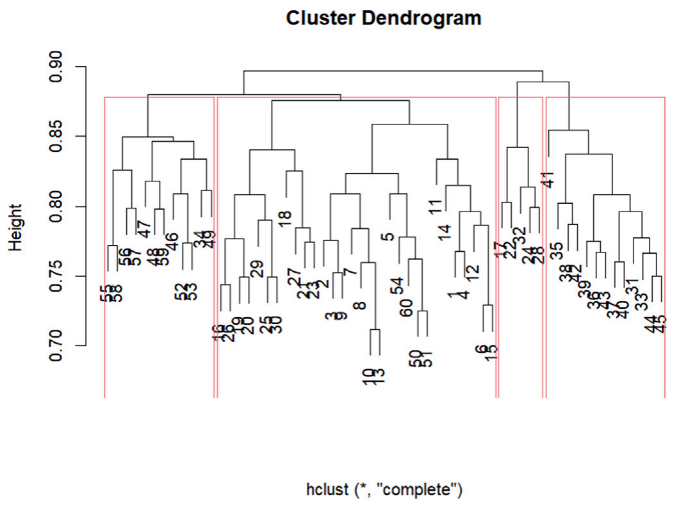


Fig. 5. Visualization of the dendrogram resulting from applying the S3 plot method to an `rstarts` class object and selecting for `type` = 1. Here, complete linkage (`method` = "complete") is used to obtain the dendrogram.

to compute other rational (and pseudo-rational) starts. These start partitions generated by the user can be given as input to the C-ICA algorithm by inputting these starts as user-defined starts (see below).

The object of class `rstarts` has an associated S3 plot method, which can be used when the `RVdist` element is not `NULL`. When the argument `type` equals 1, a dendrogram of a hierarchical clustering is shown. By default, the complete linkage method is used, but a user can specify another linkage method (e.g., average linkage) by changing the argument `method` in the call of the plot function. The result of applying the two-step procedure of [26] with complete linkage to our example data set can be viewed by:

```
plot(Out_starts, type = 1, nClus = 4,
    method = "complete")
```

By specifying a specific linkage method that can be used by the `hclust()` function, a dendrogram with the obtained clustering is plotted (see Fig. 5).

When `type` equals 2, a low dimensional representation of the subjects is plotted, which is obtained by applying classical (metric) MultiDimensional Scaling (MDS; [55]) to the (dis) similarity matrix in the `RVdist` element. By default, a two dimensional representation is shown. An interactive three dimensional representation can also be requested by setting the argument `mdsdim` equal to 3. The subjects presented in the low dimensional representation can be color coded based on the estimated clustering with `nClus` clusters from a hierarchical clustering procedure with a linkage method that is specified by `method`. A three dimensional representation of the subjects from our example data set, color coded based on the four-cluster solution obtained by running Ward's hierarchical clustering, can be viewed by using the following command:

```
plot(Out_starts, type = 2, mdsdim = 3,
    nClus = 4, method = "ward.D2")
```

Running this code results in an interactive three dimensional representation, which is presented in Fig. 6, of the subjects. In such a lowdimensional (MDS) representation, subjects that are close to each other -in Euclidean distance- are more similar to each other -in terms of RV- than subjects further apart from each other. This representation of the subjects can be adopted by the user to come up with a user-defined subject clustering that can be inputted to the ALS algorithm (see below).

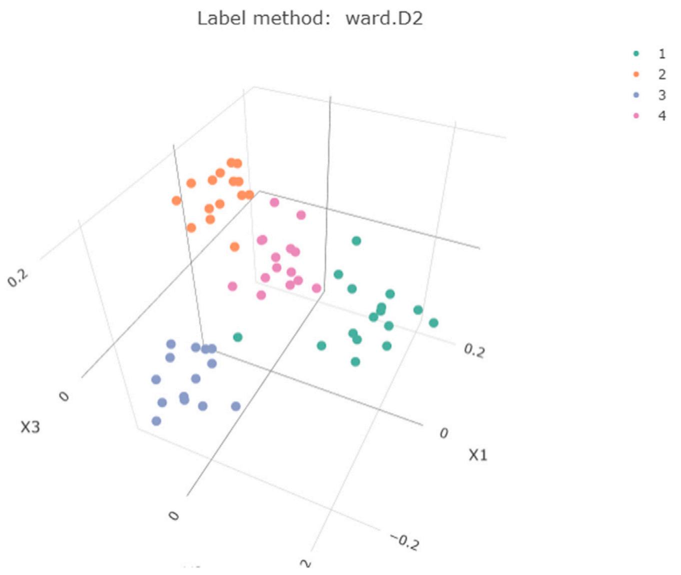


Fig. 6. Three dimensional (`mdsdim` = 3) MDS plot resulting from applying the S3 plot method on an `rstarts` class object and selecting for `type` = 2. The objects are color coded based on the estimated clustering from a hierarchical cluster analysis using Ward's clustering method (`method` = "ward.D2") and cutting the dendrogram such that four clusters are selected. Note that within R, the figure can be rotated interactively.

A user can also add user-defined starts to the ALS algorithm. To this end, these user-defined starts should be collected in a `data.frame`. Here the number of rows should be equal to the number of subjects N and the number of columns to the number of user-defined starts. Thus, each column represents a different user-defined start partition vector. The values in this vector, which indicate cluster membership, should go from 1 up to the number of clusters R (`nClus`) and no empty clusters are allowed. To seed the C-ICA algorithm with these user-defined starts, the `data.frame` object should be given to the `rational` input argument of the `CICA()` function (see step 1a of Section 3.1). Note that this `data.frame` is allowed to contain a combination of (pseudo-)rational (for example generated by `FindRationalStarts()`) and user-defined start partitions.

3.4. Output functions

Sometimes researchers may want to inspect the cluster-specific RSNs with their software tool of choice, such as FSLeyes [56], or do further analyses on these RSNs in other software tools. In order to facilitate the interoperability between different software tools, the **CICA** package has a function called `Sr_to_nifti()` that enables the conversion of the estimated RSNs per cluster (i.e., $\mathbf{S}^r$) to a NIfTI file format. The function can be used as follow:

```
Sr_to_nifti(x = multiple_output$Q_5_R_4,
    write = TRUE, ...)
```

Here, the function argument `x` refers to an object of class `CICA()`. When `write` equals `TRUE`, the RSNs are converted to NIfTI files saved to the current working directory as individual NIfTI files. Note that the RSNs are stored in the 4-th dimension of a NIfTI file. In order to save storage space, the option is given to not save the RSNs images (`write` = `FALSE`). In this case, nothing is saved to the current working directory and only a list of NIfTI files is returned within the global environment of R. This is given as an option since subsequent analyses can then be done on the NIfTI files and/or the NIfTI images within the global environment can be inspected further using the `plot()` function. As a finale note, the function internally uses the function `writeNifti()` from the **RNifti** package [57]. Arguments to this function can be passed to the `Sr_to_nifti()` call by using the ellipsis functionality provided (i.e., the ... argument).

4. Applications

4.1. Example 1. Multi-subject rs-fMRI data

In this section, the **CICA** package will be demonstrated using an empirical multi-subject rs-fMRI dataset. Note that these data are not included in the **CICA** package due to the large size of the data.

4.1.1. Step 0. Load data

The dataset is a sub selection of a (larger) dataset that contains samples of two subpopulations, namely patients diagnosed with Alzheimer's disease (AD, 77 patients) and elderly control subjects (EC, 173 subjects). For this example application, we selected 20 AD patients and 20 EC subjects that show the largest differences -between groups- in underlying RSNs. By selecting two sub-groups with clear differences in RSNs, we can evaluate the performance of C-ICA more thoroughly as now the data contain two clear subject clusters that differ in underlying RSNs. A more detailed description of the full data including the performed pre-processing steps (e.g., brain extraction, motion correction, non-linear registration to a standard MNI152 space and spatial smoothing) is presented in [28] and more information about the sub selection procedure can be found in [1].

In order to load the data, which are stored in 40 NIfTI files, one for each subject, the following code can be used:

```
data_fmri <- loadNIfTIs(dir = "<path>",
    toMatrix = TRUE)
```

Here, a character string with the path to the directory containing the 40 NIfTI files should be provided to the `dir` argument. The 40 voxel $\times$ time point matrices, containing the fMRI data for each subject separately, are loaded and stored in the list object `data_fmri`. Note that the order of the subjects is determined by the ordering of the NIfTI files in the folder where they are stored. The names of the original NIfTI files are used for naming the matrices in the `data_fmri` object and can be viewed by typing:

```
names(data_fmri)
```

4.1.2. Step 1. Performing C-ICA with several starting partitions

Step 1a. Performing C-ICA

A C-ICA analysis is performed using the following code:

```
output_CICA <- CICA(DataList = data_fmri
    , nComp = 2:30, nClus = 1:4, method
    = "fastICA", RanStarts = 30,
    RatStarts = "all", pseudo = c(0.1,
    0.2), pseudoFac = 2, scalevalue =
    1000, center = TRUE, maxiter = 100,
    verbose = TRUE, ctol = .000001)
```

Here, C-ICA is applied on the input data object `data_fmri` selecting for $R = 1, \dots, 4$ clusters and $Q = 2, \dots, 30$ RSNs per cluster. Note that when $R = 1$, no clustering is performed and a Group-ICA [17] type of analysis is performed.

The 8 rational (i.e., `RatStarts = "all"`) and 16 pseudo-rational starts, for each RSNs Q and cluster R combination is computed within the `CICA()` function. Further, the C-ICA algorithm is also run with a total of 30 random starts (`RanStarts = 30`). The other arguments are specified as before (see Step 1 of Section 3.1). The solution that yields -after ALS iterations- the lowest loss function value (see Eq. (2)) is saved in the final output.[25]

Step 1b. Inspecting rational and pseudo-rational starts

Note that in the last previous section, rational and pseudo-rational starts are computed within the main `CICA()` function using the two step clustering procedure form [26]. However, as mentioned in Section 3.3, the two step procedure could also be performed separately and the results can be inspected visually using the associated plot methods (i.e., either a dendrogram or MDS configuration of the rational starting partition). Note that only a single RSNs and cluster combination (e.g., `nComp = 5` and `nClus = 2`) can be provided to this function:

```
Out_starts_EmpirApplic <-
    FindRationalStarts(DataList = data_
    fmri, RatStarts = "all", nComp = 5,
    nClus = 2, pseudo = c(0.1, 0.2),
    pseudoFac = 2)
```

The two step procedure (see Section 3.3) is performed to the data selecting for two clusters (`nClus = 2`) and 5 RSNs per cluster (`nComp = 5`). Regarding rational starts, as the default setting `RatStart="all"` is used, 8 rational starts are generated, one for each linkage method that is available in the `hclust()` function. Additionally, since a vector of proportions is supplied to the `pseudo` argument, the rationally obtained partitions are perturbated in order to generate pseudo-rational starts. More specifically, for each unique rational start, `length(pseudo)` unique pseudo-rational starts are generated by reassigning 10% and 20% (i.e., the values in `pseudo`) of the subjects from the corresponding rational start to a different cluster (see Section 3.3). This is repeated 2 times (i.e., `pseudoFac = 2`). As such, in total, 32 pseudo-rational start partitions are obtained along with the 8 rational starts discussed above. These rational and pseudo-rational starts, are stored in the columns of the `starts` field of the `Out_starts_EmpirApplic` object. If necessary, these starts can also be provided to the `CICA()` function via the `userDef` argument.

4.1.3. Step 2. Model selection

In total, 116 different C-ICA models are fitted (i.e., all possible combinations of 2 to 30 components and 1 to 4 clusters). In order to select the optimal model, the `SequentialScree()` function can be used as follow:

```
ModSel <- SequentialScree(output_CICA)
```

Here, the sequential scree test is performed (for more information, [1,43]). For this analysis, the optimal model appears to be a model with $R = 3$ clusters and $Q = 13$ cluster specific RSNs.

4.1.4. Step 3. Interpreting the C-ICA output

3a. Subject clustering

The results of the estimated subject partition can be viewed by applying the `summary()` function to the `CICA` output object `output_CICA` and specifying a particular value for R and Q (`output_CICA$Q_13_R_3`):

```
summary(output_CICA$Q_13_R_3)
```

[25] Note that the current command performs each start of the C-ICA algorithm sequentially, implying that each start is a completely independent procedure (i.e., an embarrassingly parallel computation problem). The computations for all starts can be performed in parallel on a computer cluster; in that case, `RanStarts = 1` should be used and, when (pseudo-)rational and/or user-defined starts are inputted these partition vectors should be provided separately to the `userDef` argument.

```
Partitioning matrix P:

       Cluster 1 Cluster 2 Cluster 3
Subj1          1         0         0
Subj2          1         0         0
Subj3          1         0         0
Subj4          1         0         0
Subj5          1         0         0
Subj6          1         0         0
Subj7          1         0         0
Subj8          1         0         0
Subj9          1         0         0
Subj10         1         0         0
Subj11         1         0         0
Subj12         1         0         0
Subj13         1         0         0
Subj14         1         0         0
Subj15         1         0         0
Subj16         1         0         0
Subj17         1         0         0
Subj18         1         0         0
Subj19         1         0         0
Subj20         1         0         0
Subj21         1         0         0
Subj22         0         1         0
Subj23         0         1         0
Subj24         0         1         0
Subj25         0         0         1
Subj26         0         0         1
Subj27         0         0         1
Subj28         0         0         1
Subj29         0         0         1
Subj30         0         0         1
Subj31         0         0         1
Subj32         0         0         1
Subj33         0         0         1
Subj34         0         0         1
Subj35         0         0         1
Subj36         0         0         1
Subj37         0         1         0
Subj38         0         0         1
Subj39         0         1         0
Subj40         0         1         0

Tabulation of clustering:

Cluster 1 Cluster 2 Cluster 3
       21          6        13

Loss function value of optimal solution is:
25969.82
```

As shown in the output above, 21 subjects are allocated to cluster 1, 6 subjects to cluster 2 and 13 subjects to cluster 3. The loss value associated with this optimal subject partition equals 25969.82 (see Eq. (2)). Since diagnostic labels for the subjects (i.e., AD vs. EC) are available for the current application, one can evaluate the amount of agreement between these labels (in the current example, `labels` is a vector with the diagnostic labels for each subject: a factor with two levels) and the obtained clustering by making a crosstabulation between them:

```
table(labels, output_CICA$Q_13_R_3$P)
```

```
labels  1  2  3
AD      1  6 13
EC     20  0  0
```

The results show that all EC subjects, together with one AD patient, are allocated to the first estimated cluster and that the other AD patients are divided into two (sub-)clusters; the majority of AD patients (13) are allocated to the third estimated cluster, whereas a minority of the AD patients (6) form a relatively small cluster (the second estimated cluster). All in all, the obtained clustering that was selected by the model selection procedure agrees to a good extent with the diagnostic labels. That is, a single cluster with all EC subjects (albeit including also one AD patient) appears, along with two (sub-)clusters containing only AD patients.

Note that for each subject a Mini Mental State Exam (MMSE) score is present. As such, in order to further validate the subject clustering, pairwise t-tests are performed:

```
pairwise.t.test(MMSE, output_CICA$Q_13_R
    _3$P, p.adjust.method = "bonferroni"
    )
```

Running this code gives the following output:

```
Pairwise comparisons using t tests with pooled SD

data:  dfQ3$y and dfQ3$P

  1       2
2 0.00077 -
3 2.6e-06 1.00000

P value adjustment method: bonferroni
```

As expected, the subjects from cluster 1 (mainly EC patients) significantly differ in terms of MMSE score from both the subjects in cluster 2 and 3 (AD patients), whereas no significant difference in MMSE score is observed for the two AD patients clusters. However, due to the relative small sample size, assumptions might not be fully met. The mean MMSE score per cluster can be calculated by the following command:

```
aggregate(MMSE, list(RES$Q_13_R_3$P),
    FUN=mean)
```

```
  Group.1        x
1       1 27.28571
2       2 19.83333
3       3 19.00000
```

The mean MMSE score equals 27.29, 19.83 and 19 for cluster 1, 2 and 3, respectively, showing that subjects from the mainly ED cluster have less cognitive decline than the subjects from the two AD clusters which have very similar levels of cognitive decline. As a final suggestion, in order to facilitate the interpretation of a C-ICA solution, when more external grouping variables (e.g., gender) are supplied to the second (list) argument in the `aggregate()` function, the mean MMSE score for each combination of the factor levels of these external variables is computed, which further may enhance the understanding of the obtained solution. Note that it does not make sense here to compute ARI and BA between diagnostic labels (2 groups) and the obtained subject partition (3 clusters) as both differ in terms of the number of groups/clusters.

3b RSNs per cluster and subject-specific time courses

In order to view the estimated cluster-specific RSN's $\mathbf{S}^r$, the `plot()` function can be applied to the `CICA` output object:

```
plot(output_CICA$Q_13_R_3)
```

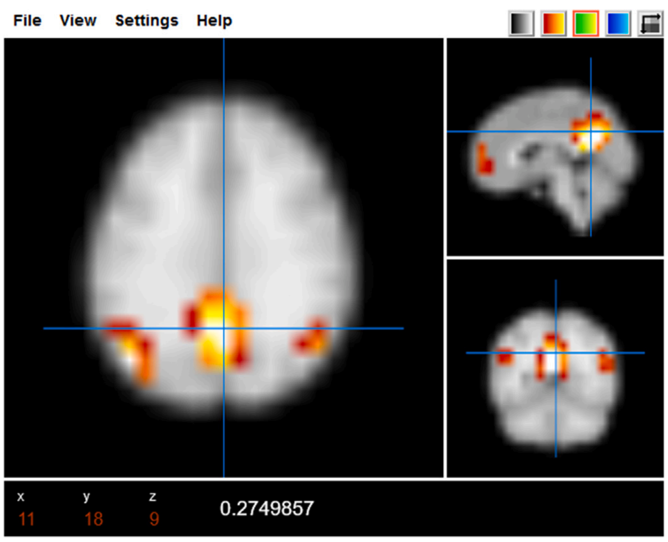


Fig. 7. Result of applying the S3 plot method to a CICA class object. Here, the estimated DMN for cluster 1 is shown.

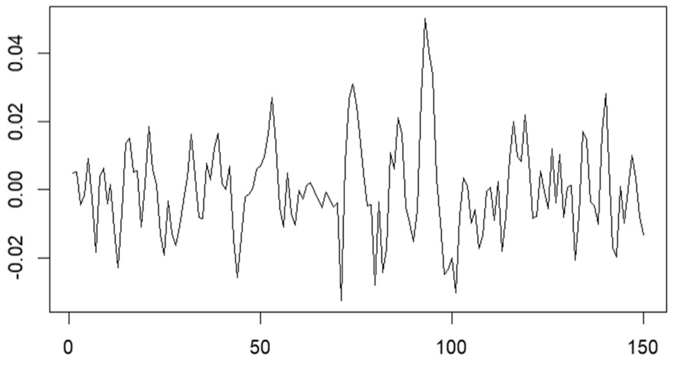


Fig. 8. Plot of the estimated time course for subject 1 (belonging to cluster 1) that is associated with the DMN.

As the inputted data are based on NIfTI files and loaded into R by using the `loadNIfTIs()` function (see Step 0), the `plot()` call will start an interactive neuroimage viewer in the plotting pane of RStudio [58] or in a web browser (when base R is used or specifically requested in the plotting pane in RStudio). It is advised to view the RSNs in a web browser since it enables a more intuitive functionality of scrolling through different RSNs for each cluster separately. An example of the neuroimage viewer can be seen in Fig. 7 in which the estimated RSN from cluster 1 is shown that most closely matches the DMN (for info on the matching procedure, see Section 3.4).

Of course, one can also further explore the time courses. In order to plot a time course associated to a RSN for a particular subject, the following code can be used:

```
plot(output_CICA$Q_13_R_3$Ais[[1]][ ,1],
    type ="l")
```

This results, as shown in Fig. 8, in a line plot (specified using the argument `type = "l"`) of the first ([, 1]) time course of the first ([[1]]) subject, which for the current application is the time course associated to the DMN (i.e., the first RSN of the cluster to which subject 1 belongs).

3c. RSN/component matching

As is true for ICA, the C-ICA algorithm outputs the RSNs (per cluster) in a random order. In practice this means that certain biologically relevant RSNs, such as the DMN for example, can appear in a different position in the output each time (C-)ICA is performed. Moreover, two very similar RSNs may be present at different positions across the estimated clusters. In order to facilitate the interpretation of the cluster-specific RSNs of a C-ICA solution, one can identify in each cluster the RSN that best matches a given template RSN (such as the DMN). To this end, the `matcher()` function can be used:

```
matcher(output_CICA$Q_13_R_3, reference
    = "TemplateFolder/KnownNetworks.nii.
    gz")
```

The `matcher()` function takes an output object of class `CICA` as its first argument and as a second argument (`reference`) a character string with the location of (a folder containing) a NIfTI object that includes a set of template RSNs of interest to the researcher.[26] The `matcher()` function then identifies in each cluster which RSN matches each template RSN the closest (for more information on this procedure, see Section 3.4). The output of the matching procedure is shown below. The output consists of two parts; first, for each template RSN (presented in the rows), the position of the best matching RSN in each cluster (one column per cluster) is presented. For example, template component number 5 -which is the DMN, a relevant RSN for AD- best matches with the first RSN from cluster 1, the third RSN from cluster 2 and the second RSN from cluster 3. When the user wants to view these matched RSNs, the above-mentioned neuroimaging viewer can be adopted.

```
$matchIndexMatrix
            NIfTI   Cluster  1   Cluster  2  Cluster  3
Component  1      1           2           5           8
Component  2      2          13           2           4
Component  3      3           8           6           9
Component  4      4          10           8           7
Component  5      5           1           3           2
Component  6      6           9           1           1
Component  7      7           3           5           9
Component  8      8           4           5          10
```

The second part of the output, using the same format as the previous output, displays the amount of agreement or matching (in terms of absolute Tucker's congruence coefficient) between the template and estimate RSN per cluster. Here, it can be seen that the estimated DMN from the first clusters -with mainly EC subjects- has a stronger similarity (0.709) with the template RSN than the estimated DMN from cluster 2 with 6 AD patients (0.279) or cluster 3 with 13 AD patients (0.545).

```
$matchTuckerMatrix
            NIfTI  Cluster  1  Cluster  2  Cluster 3
Component  1     1      0.807      0.413      0.582
Component  2     2      0.440      0.255      0.377
Component  3     3      0.602      0.249      0.308
Component  4     4      0.653      0.489      0.379
Component  5     5      0.709      0.279      0.545
Component  6     6      0.584      0.411      0.536
Component  7     7      0.480      0.166      0.317
Component  8     8      0.507      0.160      0.288
```

It should be noted, however, that the output of the `matcher()` function is only meant as a tool to aid the user in identifying certain RSNs of interest in the C-ICA output. One should not blindly or automatically apply the matcher function on the C-ICA output. We strongly

[26] A template with well-known RSNs (file `rsn8.nii.gz`) can be downloaded from https://www.fmrib.ox.ac.uk/datasets/royalsoc8/.

advise to inspect the matched RSNs with the interactive viewer and to also examine the other -non matched- RSNs per cluster. Besides matching cluster-specific RSNs to a given set of template RSNs, one can also match the RNSs per cluster to each other (for an example of this, see Section 4.2).

4.2. *Example 2. Multi-country stock market data*

In this section, as a second example, the **CICA** package will be demonstrated using an empirical multi-country stock market dataset. Note that the stock market data are publicly available on the Open Science Framework.[27] The data consist of daily closing prices of different stocks from multiple countries. A selection of 20 countries is used for this example, and for each of these countries, the daily closing prices of the 15 largest companies are included. The daily closing prices from June 27, 2022, to June 25, 2024, are included in these data. Note that each of the 20 countries may have different market closing days. To avoid missing values, we selected only those trading days when all markets were open. The final dataset consists of a three-way data structure where the first mode represents the 20 countries, the second mode represents the trading days (303), and the third mode represents the different companies (15).

A general goal of performing C-ICA on this type of data is to cluster countries into homogeneous groups based on similarities and differences in underlying (macro-)economic patterns (i.e., the cluster-specific components). This allows researchers to investigate and explore different stocks from different companies across multiple countries and extract the main economic patterns from them in a concise manner.

4.2.1. *Step 0. Load data*

Since these types of data are not stored in a NIfTI format (see Section 4.1), the data can simply be loaded using the R function `load()`:

```
data_stocks <- load(file = "stock_data.
    Rdata")
```

where `file` specifies the location of the `stock_data.Rdata` file that contains the data for the current example.

In order to view the names of the different countries, the following code can be used:

```
names(data_stocks)
```

4.2.2. *Step 1 and 2 performing C-ICA and a model selection*

Similar to the previous example, a C-ICA analysis is performed using the following code:

```
output_CICA <- CICA(DataList = data_
    stocks, nComp = 1:10, nClus = 1:5,
    method = "fastICA",RanStarts = 30,
    RatStarts = "all", pseudo = 0.1,
    pseudoFac = 2, scalevalue = 1000,
    center = TRUE, maxiter = 100,
    verbose = TRUE, ctol = .000001)
```

The explanation of each of the input arguments can be viewed in Section 4.1. In short, a C-ICA analysis is performed on centered and block-scaled data, selecting from all possible combinations of 1 to 10 components and 1 to 5 clusters. Both random (30) and (pseudo-) rational starts are used. Note that the (pseudo-) rational starts are performed directly within the function call. If necessary, this can also be performed in a separate function call (i.e., `FindRationalStarts()`) as explained before. Next, in order to select an optimal model, the sequential scree test (for more information, see [1,43]) on these fitted models can be performed:

```
ModSel <- SequentialScree(output_CICA)
```

Results indicate that the optimal model is a parsimonious model with $R = 2$ clusters and $Q = 3$ components. As mentioned before, a `plot` method can be used to visualize the results, if necessary (i.e., `plot(ModSel)`).

4.2.3. *Step 3. Interpreting the C-ICA output*

Similar to the previous example, **CICA** provides several S3 methods for interpreting a fitted CICA model. To inspect the subject clustering, the `summary()` method can be used:

```
summary(output_CICA$Q_3_R_2)
```

This yields the following output for the current example:

```
Partitioning matrix P:

            Cluster 1 Cluster 2
USA                 0         1
Japan               0         1
Germany             1         0
China               0         1
India               0         1
UK                  1         0
France              1         0
Brazil              0         1
Italy               1         0
Canada              1         0
Mexico              0         1
Australia           1         0
Southkorea          0         1
Spain               1         0
Indonesia           1         0
Netherlands         1         0
Turkey              1         0
Switzerland         0         1
Poland              0         1
Belgium             0         1

Tabulation of clustering:

Cluster 1 Cluster 2
       10        10

Loss function value of optimal solution is:
5930.034
```

As shown in the output above, the optimal model yields an equally sized clustering solution. Countries from different geographical regions are included in both clusters. Cluster 1 consists of relatively large European economies, such as Germany, the UK, France, Italy, Spain, and the Netherlands. Cluster 2 primarily includes South Asian (India) and East Asian (Japan, China, and South Korea) countries, along with other populous nations such as the USA, Brazil, and Mexico.

To further interpret the (clustering) results, the aforementioned `matcher()` function can be used. This function aims to match the estimated cluster-specific components to each other and/or to study their similarity. The function can be called using the following code:

```
matcher(output_CICA$Q_3_R_2, RV = TRUE)
```

Which provides the following output:

[27] https://osf.io/uhp8g/

```
$matchIndexMatrix
                Cluster  1 Cluster  2
Component  1           1            1
Component  2           2            2
Component  3           3            3

$matchTuckerMatrix
                Cluster  1 Cluster  2
Component  1           1       0.960
Component  2           1       0.877
Component  3           1       0.476

$RVs
       [,1]  [,2]
[1,] 1.000 0.741
[2,] 0.741 1.000
```

Note that for the current example, no templates were provided to match the estimated components to (an example of this option is given in Section 4.1). Therefore, by default, the components are matched with the estimated components from the first cluster.

The results of the `matcher()` function indicate that the first two components across clusters are strongly related to each other (the absolute Tucker value is .960 and .877 for the first and second pair, respectively). Note that the function uses absolute Tucker values for computing an index matching, but the raw Tucker values are also given as output when calling the `matcher()` function (not shown here). Inspecting these values indicates that the Tucker values for these components were negative, suggesting that they show a strong relationship but in an opposite direction. The third component for both clusters yielded a positive Tucker value of .476, indicating some differences between the clusters in terms of this estimated component.

Fig. 9 displays the cluster-specific components (in black) from both clusters. The negative relationship between the first components of Cluster 1 and Cluster 2 can clearly be seen. The first component from Cluster 1 (top-left figure) shows an upward trend and therefore may represent a macroeconomic effect that reflects increasing stock prices. The first component from Cluster 2, which has a negative Tucker value of −0.96 compared to the first component from Cluster 1, shows a completely opposite downward trend, reflecting decreasing stock prices.

The second components from both clusters (see middle row in Fig. 9) also show a related but opposite trend. Here, the component from the first cluster shows a U-shaped trend, whereas the component from the second cluster shows an inverted U-shape. Note that these types of patterns are relevant for technical analysis of stock market data. The second component from the first cluster (U-shaped trend) is also known as a rounding bottom or a saucer [59,60], and is used for determining whether a certain market is undergoing a reversal from a downward trending market (i.e., bearish market) towards an upward trending market (i.e., bullish market). The detection of this inflection point highlights a potential opportunity to buy stocks. A similar but opposite rationale can be used for the second component from the second cluster. This rounding top trend indicates the movement of an upward trending market towards a downward trending market, potentially indicating that a sale of stocks might be warranted.

The third component from both clusters shows a somewhat mixed pattern, which is also reflected in the relatively low Tucker congruence value of 0.476. Here, the component of the first cluster shows an initial upward trend, followed by a rapid decline or market correction. From May and June 2023, the component shows a relatively stable pattern towards the end of June 2024. The third component from the second cluster shows a similar pattern, but instead of a rapid decline and stabilization, a relatively slow decline occurs followed by a relatively steep recovery to its former peak.

As mentioned in Section 2.2, the C-ICA model also estimates mixing matrices $\mathbf{A}_i$ ($T \times Q$), which, in contrast to the rs-fMRI example, are now 20 country-specific mixing matrices of size 15 (different companies per country) × 3 components. One could inspect these estimated mixing matrices by using the following code:

```
output_CICA$Q_3_R_2$Ais[[1]]
```

Here, the index 1 corresponds to the first country in our current example (which is the USA, allocated to the second cluster) and provides the following output:

```
             [,1]         [,2]         [,3]
AMZN  -0.16739090 -0.172593418 -0.044494746
AAPL  -0.18441262  0.173836497 -0.133638743
MA    -0.20923386  0.058545095  0.109927589
WMT   -0.19403230  0.035174657  0.204771377
UNH    0.40889720 -0.105341244 -0.309377847
DIS    0.40504063 -0.229714652  0.111533821
JNJ    0.73578031  0.170220849 -0.193567470
V     -0.21487524  0.046984683  0.107034226
HD    -0.03425618 -0.021749254  0.096333820
GOOGL -0.19433559 -0.088921725  0.002680569
JPM   -0.18928952  0.001946022  0.211567514
PG    -0.07773008  0.056707787  0.184283085
NVDA  -0.18816909 -0.062735662  0.167699732
MSFT  -0.24448122  0.002249541  0.022074554
TSLA   0.34848847  0.135390823 -0.536827480
```

The values from the mixing matrix can be interpreted as weights, for example, one could see that the weight of JNJ (Johnson & Johnson company) is relatively high for the first component (0.736). This indicates that the stock of this company closely resembles the first cluster-specific component from cluster 2. To visualize this, in Fig. 9, we superimposed the company's stock (in red) onto the estimated component (see top right panel). Indeed, the price of the JNJ stock shows a declining trend over the past two years, which is now summarized by the estimated component.

Another advantage of **CICA** is that a user can inspect the largest mixing weights per country (allocated to a particular cluster) on a particular component. For example, Table 4 shows the largest weights for the second component from each country allocated to the first cluster. From this table, it is evident that most companies are from the financial sector. This suggests that the typical U-shaped (rounding bottom) trend observed for the second component from the first cluster is largely influenced by companies related to the financial sector.

Although the causes of changes in stock prices for a particular company are often multifactorial, one could propose a possible explanation for the change in trend of the second component from the first cluster. During 2023, central banks in various economic regions increased interest rates to combat rising inflation rates [61]. The shift from a downward trend to an upward trend could potentially reflect the increase in profitability margins on loans and deposits.

4.3. Improving the computational speed of `CICA`

As noted in Section 2.4, the procedure can be time consuming due to the multiple ICA's that are performed within the steps of the ALS algorithm. To increase the speed of computations, an additional option has been added to the `CICA` function, which can be called using the `method` argument. The increase in computational speed is due to a modification in the computation of the cluster-specific components $\mathbf{S}^r$. In the original **CICA** version, which is used throughout this paper, fastICA (i.e., `method = "fastICA"`) was used to compute the cluster-specific components. This involves a whitening procedure where the input data is decorrelated and scaled to unit variance, followed by the computation of an orthogonal rotation matrix that

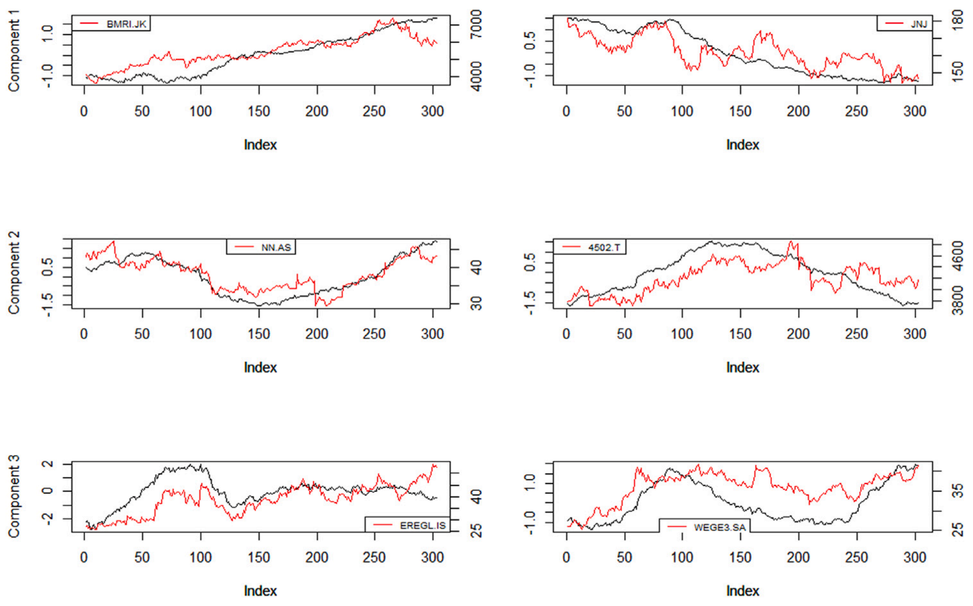


Fig. 9. Plot of the estimated independent components (in black) for a two cluster-three component solution on stock market data. For each component, the individual stock that yields the highest loading is also plotted (in red).

Table 4
Largest estimated mixing weights per country for the second estimated component from cluster 2.

Country	Company	Mixing Weight
Germany	Deutsche Bank	.239
UK	Barclays	.242
France	Vivendi	.391
Italy	Banca Monte dei Paschi	.323
Canada	Canadian Imperial Bank of Commerce	.267
Australia	National Australia Bank Limited	.400
Spain	Merlin Properties	.377
Indonesia	Adaro Energy Indonesia	.372
Netherlands	NN Group	.412
Turkey	Akbank	.290

Table 5
The average computational speed (in seconds) of C-ICA applied to data with varying number of voxels for two different component estimation procedures, standard deviations in parentheses. The last column indicates the relative speed of `method = "EVD"` compared to `method = "fastICA"`.

Voxels	`method = "ICAfast"`	`method = "EVD"`	Relative speed increase (ratio)
2000	541.42 (37.15)	37.14 (2.84)	14.58
4000	862.72 (76.11)	55.58 (6.45)	15.52
6000	1169.48 (102.73)	77.35 (9.29)	15.12
8000	1428.93 (117.35)	92.21 (14.09)	15.50

maximizes the negentropy of the independent components [10]. The new option (i.e., `method = "EVD"`) omits the computation of the orthogonal rotation matrix, potentially reducing computational time significantly. During this procedure, the **Rfast** package [62] is used to quickly compute a covariance matrix of the input data, followed by a fast EVD on the covariances related to the cluster-specific input data (determined by the first step of the ALS algorithm).

It is hypothesized that this change in computation does not affect the reallocation steps (see Algorithm Step 3), as both fastICA and the eigenvalue decomposition compute the same component subspace, which does not influence the least squares estimates within the ALS algorithm steps. The primary difference is that fastICA employs a rotation of the components in the reduced subspace to maximize their independence, whereas the EVD based method does not. This effectively means that for the EVD based procedure, the final cluster-specific components are estimated in a suboptimal way. Whereas the clustering **P**, however, is optimally estimated. To estimate the final cluster-specific components correctly, one could perform separate fastICA's on each of the concatenated datasets that are determined by the (final) clustering results from a CICA analysis first performed using the proposed (faster) EVD-based procedure.

In a small simulation study, the increase of computational speed is demonstrated. For the simulation, using the `Sim_CICA()` function, a total of 20 subjects consisting of two non-overlapping clusters R and five independent components Q for each cluster were generated. The number of time points equals 100 and the amount of additive Gaussian noise equals 10%. This data generation procedure was replicated a total of ten times and the simulated data were subjected to both the `CICA` function using `method = "fastICA"` and `method = "EVD"`. The results, shown in Table 5, indicate that the new option is indeed faster than the rigorously tested (see [1]) fastICA based `CICA` version. Note that the increase in speed is significant with larger numbers of voxels (mean time equals around 24 min versus 1.5 min for the ICA and EVD procedure respectively). This is a significant advantage given the large datasets commonly encountered in neuroscience and other fields of study. Note, however, that the additional EVD based procedure is not rigorously tested using either simulation or empirical data. Future research should therefore investigate the new procedure more thoroughly.

5. Summary and discussion

In this article, we have introduced the R software package **CICA** which allows to fit the C-ICA model [1] to three-way data in order to discover (subject) heterogeneity in this type of data. Applied to multi-subject rs-fMRI data, which are three-way data (i.e., voxel by time by subject), this software enables the simultaneous estimation of a subject partition, the associated cluster-specific RSNs and the subject-specific time courses underlying the data (for a comprehensive overview see, [1]). The package also contains functions to choose good starting values for the C-ICA model parameters and to determine the optimal number of clusters and components for an empirical data set at hand (i.e., model selection). Further, the software package also includes several S3 methods that facilitate the interpretation of C-ICA results, such as an interactive neuroimaging viewer for inspecting estimated

RSNs and a RSN template matcher to determine how much an extracted RSN by C-ICA corresponds with a known RSN (e.g., the default mode network).

C-ICA was originally developed specifically for multi-subject resting-state fMRI data in order to disclose subject heterogeneity in underlying RSNs. The **CICA** package, however, can also be used to detect heterogeneity in other types of three-way data as long as the goal of the analysis is detecting heterogeneity -in one of the three ways of the data- and statistical independence can be assumed between the ICA components/patterns underlying the data. To give an example, ICA is also often used to analyze multi-subject electroencephalogram (EEG) data (see for example [63]). In EEG, changes in electrical activity over time are recorded for several brain regions -using electrodes- and these data can be arranged as a time $\times$ electrode matrix for each subject (hence, three-way data). By applying C-ICA to these data, a researcher may disclose relevant clusters of patients, where each patient cluster is associated with substantively different underlying brain processes (i.e., ICA components of EEG activity). Further, besides resting-state fMRI, C-ICA can also be adopted to explore subject heterogeneity in ICA components underlying task-based fMRI data since for these type of data ICA can also be used to extract functional connectivity patterns that are linked to performing certain tasks [30]. Other possible areas of application for C-ICA are, amongst others, bioinformatics, chemometrics, text mining, financial data, marketing and psychology. As such, C-ICA allows to detect heterogeneity in regulatory genetic pathways (bioinformatics), mechanisms underlying the metabolism related to a disease (metabolomics/proteomics), text streams topics (text mining), trading behaviors (financial data), purchasing styles (marketing) and psychiatric syndromes (psychology).

To estimate the parameters of the CICA model, an ALS type of algorithm is used. A well-known drawback of this type of algorithms is that they are sensitive to local minima, which implies that sometimes a suboptimal clustering solution is retained. To alleviate this problem, which is commonly encountered in many clustering procedures, a multi-start procedure using random start partitions is implemented in the **CICA** package. Moreover, the package also allows the inclusion of rational, pseudo-rational and user-defined starting partitions. Regarding the first, the package adopts the two-step clustering procedure of [26]. By seeding the CICA algorithm with a diverse set of randomly and (pseudo-)rationally obtained start partitions, local minima can be avoided maximally.

In order to save computation time, it is advised to run these multiple starts in parallel, either on a computer cluster or locally. For example, scripts can be sourced in parallel on a local machine, with the different starts supplied using the `userGrid` argument for example. Related to the computation time, the current version of **CICA** includes an experimental version of an EVD based component estimation (instead of a fastICA estimation), which allows for a significant decrease in computation time. This computation does not necessarily extract the right true underlying components, but may indeed compute the correct clustering (under mild circumstances at least). After computing the correct clustering, a final fastICA estimation (one for each cluster R), can be computed in order to identify the correct independent components.

Future versions of the **CICA** package may include other (types of) algorithms than ALS to estimate the parameters of the C-ICA model. One example of such an (type of) algorithm is Iterated Local Search (ILS). This (type of) algorithm could be a good competitor for the current ALS algorithm for C-ICA as research showed that ILS outperformed an ALS-like algorithm for K-means clustering [64], which to some extent is a cluster analysis procedure that is similar to C-ICA (i.e., a clustering procedure based on optimizing a sum of squares objective function). The general idea behind ILS is that the algorithm can escape from local minima, which is not possible for ALS, by performing a local search around a suboptimal converged solution (i.e., a local minimum). During this local search, some of the object labels are perturbated and the ALS algorithm is restarted from this perturbated solution. By perturbating the object labels of a locally optimal solution often a worse solution is obtained. However, performing ALS steps on this worse solution may result in the algorithm discarding (and escaping from) the locally optimal solution and retaining a better solution that has a larger chance to be the globally optimal solution.

Future releases of the **CICA** package may also extend the functionality for model selection, which for C-ICA -and for many other clustering and dimension reduction procedures- always will stay a challenging problem. Indeed, selecting the optimal number of clusters and ICA patterns for a data set at hand is a non-trivial task. To tackle this problem, future research should search for new and better procedures than the sequential model selection procedure presented in this paper (and implemented in the package); these procedures then may be added to the package. Promising results in this regard, although in another (clustering) context than C-ICA, were obtained with the CHull method [44,45] and information theory based procedures (e.g., AIC) for model selection [65]. Future research should evaluate whether these procedures also can be used to solve the complex model selection problem in C-ICA and thus warrant inclusion in the **CICA** package.

CRediT authorship contribution statement

Jeffrey Durieux: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Serge Rombouts:** Writing – review & editing, Supervision, Project administration, Funding acquisition. **Marisa Koini:** Writing – review & editing, Data curation. **Juan Claramunt Gonzalez:** Writing – review & editing, Software, Methodology. **Tom Wilderjans:** Writing – review & editing, Supervision, Software, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The research leading to the results reported in this paper was sponsored by a Talent Grant (Grant Number 406.16.563) from the Netherlands Organization for Scientific Research (NWO).

References

[1] J. Durieux, S.A.R.B. Rombouts, F. de Vos, M. Koini, T.F. Wilderjans, Clusterwise Independent Component Analysis (C-ICA): Using fMRI resting state networks to cluster subjects and find neurofunctional subtypes, J. Neurosci. Methods 382 (2022) 109718, http://dx.doi.org/10.1016/j.jneumeth.2022.109718.

[2] S. Ogawa, T.-M. Lee, A.R. Kay, D.W. Tank, Brain magnetic resonance imaging with contrast dependent on blood oxygenation, Proc. Natl. Acad. Sci. 87 (24) (1990) 9868–9872.

[3] E. Canario, D. Chen, B. Biswal, A review of resting-state fMRI and its use to examine psychiatric disorders, Psychoradiology 1 (1) (2021) 42–53, http://dx.doi.org/10.1093/psyrad/kkab003, arXiv:https://academic.oup.com/psyrad/article-pdf/1/1/42/37906708/kkab003.pdf.

[4] M.J. Hawrylycz, E.S. Lein, A.L. Guillozet-Bongaarts, E.H. Shen, L. Ng, J.A. Miller, L.N. Van De Lagemaat, K.A. Smith, A. Ebbert, Z.L. Riley, et al., An anatomically comprehensive atlas of the adult human brain transcriptome, Nature 489 (7416) (2012) 391–399.

[5] J. Lonsdale, J. Thomas, M. Salvatore, R. Phillips, E. Lo, S. Shad, R. Hasz, G. Walters, F. Garcia, N. Young, et al., The genotype-tissue expression (GTEx) project, Nat. Genet. 45 (6) (2013) 580–585.

[6] M. Melé, P.G. Ferreira, F. Reverter, D.S. DeLuca, J. Monlong, M. Sammeth, T.R. Young, J.M. Goldmann, D.D. Pervouchine, T.J. Sullivan, et al., The human transcriptome across tissues and individuals, Science 348 (6235) (2015) 660–665.

[7] B. Biswal, F. Zerrin Yetkin, V.M. Haughton, J.S. Hyde, Functional connectivity in the motor cortex of resting human brain using echo-planar MRI, Magn. Reson. Med. 34 (4) (1995) 537–541.

[8] W. Liebermeister, Linear modes of gene expression determined by independent component analysis, Bioinformatics 18 (1) (2002) 51–60.

[9] S.-I. Lee, S. Batzoglou, Application of independent component analysis to microarrays, Genome Biol. 4 (2003) 1–21.

[10] A. Hyvärinen, Fast and robust fixed-point algorithm for independent component analysis, IEEE Trans. Neur. Net. 10 (3) (1999) 626–634, http://dx.doi.org/10.1109/72.761722.

[11] G.R. Naik, D.K. Kumar, An overview of independent component analysis and its applications, Informatica (Ljubl.) 35 (1) (2011).

[12] I.M. Veer, C. Beckmann, M.-J. Van Tol, L. Ferrarini, J. Milles, D. Veltman, A. Aleman, M.A. Van Buchem, N.J. Van Der Wee, S.A.R.B. Rombouts, Whole brain resting-state analysis reveals decreased functional connectivity in major depression, Front. Syst. Neurosci. 4 (2010) 41.

[13] A.G. Garrity, G.D. Pearlson, K. McKiernan, D. Lloyd, K.A. Kiehl, V.D. Calhoun, Aberrant "default mode" functional connectivity in schizophrenia, Amer. J. Psychiatry 164 (3) (2007) 450–457.

[14] O. Dipasquale, L. Griffanti, M. Clerici, R. Nemni, G. Baselli, F. Baglio, High-dimensional ICA analysis detects within-network functional connectivity damage of default-mode and sensory-motor networks in Alzheimer's disease, Front. Hum. Neurosci. 9 (2015) 43.

[15] E.G. Dopper, S.A.R.B. Rombouts, L.C. Jiskoot, T. den Heijer, J.R.A. de Graaf, I. de Koning, A.R. Hammerschlag, H. Seelaar, W.W. Seeley, I.M. Veer, et al., Structural and functional brain connectivity in presymptomatic familial frontotemporal dementia, Neurology 83 (2) (2014) e19–e26.

[16] K.T. Olde Dubbelink, D. Stoffers, J.B. Deijen, J.W. Twisk, C.J. Stam, A. Hillebrand, H.W. Berendse, Resting-state functional connectivity as a marker of disease progression in Parkinson's disease: A longitudinal MEG study, NeuroImage: Clin. 2 (Suppl. C) (2013) 612–619.

[17] V.D. Calhoun, T. Adalı, G.D. Pearlson, J.J. Pekar, A method for making group inferences from functional MRI data using independent component analysis, Hum. Brain Mapp. 14 (3) (2001) 140–151.

[18] C.F. Beckmann, C.E. Mackay, N. Filippini, S.M. Smith, Group comparison of resting-state FMRI data using multi-subject ICA and dual regression, NeuroImage 47 (Suppl. 1) (2009) S148.

[19] C.F. Beckmann, S.M. Smith, Probabilistic independent component analysis for functional magnetic resonance imaging, IEEE Trans. Med. Imaging 23 (2) (2004) 137–152.

[20] M. Jenkinson, C. Beckmann, T. Behrens, M. Woolrich, S. Smith, FSL, NeuroImage 62 (2012) 782–790.

[21] V. Calhoun, T. Adali, Group ICA of fMRI toolbox (GIFT), 2004, Online at http://icatb.sourceforge.net.

[22] J. Zhang, A. Kucyi, J. Raya, A.N. Nielsen, J.S. Nomi, J.S. Damoiseaux, D.J. Greene, S.G. Horovitz, L.Q. Uddin, S. Whitfield-Gabrieli, What have we really learned from functional connectivity in clinical populations? NeuroImage 242 (2021) 118466.

[23] A.T. Drysdale, L. Grosenick, J. Downar, K. Dunlop, F. Mansouri, Y. Meng, R.N. Fetcho, B. Zebley, D.J. Oathes, A. Etkin, et al., Resting-state connectivity biomarkers define neurophysiological subtypes of depression, Nat. Med. 23 (1) (2017) 28–38.

[24] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2017, URL https://www.R-project.org/.

[25] J. Durieux, S.A.R.B. Rombouts, T.F. Wilderjans, CICA: clusterwise independent component analysis, 2021, R package version 0.0.0.9000.

[26] J. Durieux, T.F. Wilderjans, Partitioning subjects based on high-dimensional fMRI data: comparison of several clustering methods and studying the influence of ICA data reduction in big data, Behaviormetrika (2019) 1–41.

[27] M.A. Lindquist, The statistical analysis of fMRI data, Stat. Sci. 23 (4) (2008) 439–464.

[28] F. de Vos, M. Koini, T.M. Schouten, S. Seiler, J. van der Grond, A. Lechner, R. Schmidt, M. de Rooij, S.A. Rombouts, A comprehensive analysis of resting state fMRI measures to classify individual patients with Alzheimer's disease, NeuroImage 167 (2018) 62–72.

[29] O. Esteban, R. Ciric, K. Finc, R.W. Blair, C.J. Markiewicz, C.A. Moodie, J.D. Kent, M. Goncalves, E. DuPre, D.E. Gomez, et al., Analysis of task-based functional MRI data preprocessed with fMRIPrep, Nat. Protoc. 15 (7) (2020) 2186–2202.

[30] J. Xu, S. Zhang, V.D. Calhoun, J. Monterosso, C.-S.R. Li, P.D. Worhunsky, M. Stevens, G.D. Pearlson, M.N. Potenza, Task-related concurrent but opposite modulations of overlapping functional networks as revealed by spatial ICA, Neuroimage 79 (2013) 62–71.

[31] J.M. Engreitz, B.J. Daigle Jr., J.J. Marshall, R.B. Altman, Independent component analysis: mining microarray data for fundamental human gene expression modules, J. Biomed. Inform. 43 (6) (2010) 932–944.

[32] A. Smilde, P. Geladi, R. Bro, Multi-way Analysis: Applications in the Chemical Sciences, John Wiley & Sons, 2005.

[33] R. Rapp, Mining text for word senses using independent component analysis, in: Proceedings of the 2004 SIAM International Conference on Data Mining, SIAM, 2004, pp. 422–426.

[34] H.A. Shabat, N.A. Abbas, Independent component analysis based on natural gradient algorithm for text mining, in: 2020 1st. Information Technology to Enhance E-Learning and Other Application, IT-ELA, IEEE, 2020, pp. 72–76.

[35] A. Hitaj, L. Mercuri, E. Rroji, Portfolio selection with independent component analysis, Finance Res. Lett. 15 (2015) 146–159.

[36] H. Ahn, E. Choi, I. Han, Extracting underlying meaningful features and canceling noise using independent component analysis for direct marketing, Expert Syst. Appl. 33 (1) (2007) 181–191.

[37] J. De Leeuw, Block-relaxation algorithms in statistics, in: Information Systems and Data Analysis, Springer, 1994, pp. 308–324.

[38] D. Steinley, Local optima in K-means clustering: what you don't know may hurt you, Psychol. Methods 8 (3) (2003) 294.

[39] D. Steinley, Profiling local optima in K-means clustering: developing a diagnostic technique, Psychol. Methods 11 (2) (2006) 178.

[40] A.K. Smilde, H.A.L. Kiers, S. Bijlsma, C.M. Rubingh, M.J. van Erk, Matrix correlations for high-dimensional data: the modified RV-coefficient, Bioinformatics 25 (3) (2009) 401–405.

[41] D. Steinley, K-means clustering: a half-century synthesis, Br. J. Math. Stat. Psychol. 59 (1) (2006) 1–34.

[42] G.W. Milligan, An examination of the effect of six types of error perturbation on fifteen clustering algorithms, psychometrika 45 (3) (1980) 325–342.

[43] K. De Roover, E. Ceulemans, M.E. Timmerman, K. Vansteelandt, J. Stouten, P. Onghena, Clusterwise simultaneous component analysis for analyzing structural differences in multivariate multiblock data, Psychol. Methods 17 (1) (2012) 100.

[44] E. Ceulemans, H.A. Kiers, Selecting among three-mode principal component models of different types and complexities: A numerical convex hull based method, Br. J. Math. Stat. Psychol. 59 (1) (2006) 133–150.

[45] T.F. Wilderjans, E. Ceulemans, K. Meers, CHull: A generic convex-hull-based model selection method, Behav. Res. Methods 45 (2013) 1–15.

[46] M. Vervloet, T. Wilderjans, J. Durieux, E. Ceulemans, Multichull: A generic convex-hull-based model selection method, 2017, R package version 1.0.0, URL https://CRAN.R-project.org/package=multichull.

[47] L. Hubert, P. Arabie, Comparing partitions, J. Classification 2 (1985) 193–218.

[48] L. Scrucca, M. Fop, T.B. Murphy, A.E. Raftery, mclust 5: clustering, classification and density estimation using Gaussian finite mixture models, R J. 8 (1) (2016) 289–317, http://dx.doi.org/10.32614/RJ-2016-021.

[49] M. Kuhn, caret: classification and regression training, 2022, R package version 6.0-93, URL https://CRAN.R-project.org/package=caret.

[50] J. Muschelli, papayar: view medical research images using the papaya JavaScript library, 2016, R package version 1.0, URL https://CRAN.R-project.org/package=papayar.

[51] C. Sievert, Interactive Web-Based Data Visualization with R, Plotly, and Shiny, Chapman and Hall/CRC, 2020, URL https://plotly-r.com.

[52] L.R. Tucker, A Method for Synthesis of Factor Analysis Studies, Tech. Rep., Educational Testing Service Princeton Nj, 1951.

[53] N.E. Helwig, ica: independent component analysis, 2018, R package version 1.0-2, URL https://CRAN.R-project.org/package=ica.

[54] J. Muschelli, E. Sweeney, M. Lindquist, C. Crainiceanu, fslr: connecting the FSL software with R, R J. 7 (1) (2015) 163.

[55] I. Borg, P.J. Groenen, Modern Multidimensional Scaling: Theory and Applications, Springer Science & Business Media, 2005.

[56] P. McCarthy, FSLeyes, 2021, http://dx.doi.org/10.5281/zenodo.5576035.

[57] J. Clayden, B. Cox, M. Jenkinson, RNifti: fast R and C++ access to NIfTI images, 2021, R package version 1.3.1, URL https://CRAN.R-project.org/package=RNifti.

[58] RStudio Team, RStudio: Integrated Development Environment for R, RStudio, PBC, Boston, MA, 2021, URL http://www.rstudio.com/.

[59] J.-L. Wang, S.-H. Chan, Trading rule discovery in the US stock market: An empirical study, Expert Syst. Appl. 36 (3) (2009) 5450–5455.

[60] A. Zapranis, P.E. Tsinaslanidis, A novel, rule-based technical pattern identification mechanism: Identifying and evaluating saucers and resistant levels in the US stock market, Expert Syst. Appl. 39 (7) (2012) 6301–6308.

[61] Financial and climate policies for a high-interest rate era, 2023, https://www.imf.org/en/Publications/GFSR/Issues/2023/10/10/global-financial-stability-report-october-2023. (Accessed 02 July 2024).

[62] M. Papadakis, M. Tsagris, S. Fafalios, Rfast: a collection of efficient and extremely fast R functions, 2023, R package version 2.1.0, URL https://CRAN.R-project.org/package=Rfast.

[63] R. Huster, S.M. Plis, V.D. Calhoun, Group-level component analyses of EEG: validation and evaluation, Front. Neurosci. 9 (2015) 254.

[64] P. Merz, An iterated local search approach for minimum sum-of-squares clustering, in: M. R. Berthold, H.-J. Lenz, E. Bradley, R. Kruse, C. Borgelt (Eds.), Advances in Intelligent Data Analysis V, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 286–296.

[65] J. Rossbroich, J. Durieux, T.F. Wilderjans, Model selection strategies for determining the optimal number of overlapping clusters in additive overlapping partitional clustering, J. Classification (2022) 1–38.

Jeffrey Durieux is a researcher in the field of methodology and statistics. His interests are developing new data driven machine learning techniques. Another related research interest is to program easy to use software such that other researchers can apply and benefit from state of the art (computationally demanding) statistical methods.

Serge A.R.B. Rombouts is professor of methods of cognitive neuroimaging at the Department of Radiology at the Leiden University Medical Center and the Institute of Psychology at the Faculty of Social and Behavioral Sciences at Leiden University. He studies innovative brain scan technologies for applications in early diagnosis of various neurological and psychiatric disorders, and the visualization of medication effects on brain activity.

Marisa Koini is an associate professor and clinical neuropsychologist at the Medical University of Graz. Her research focuses on dementia, healthy aging, multiple sclerosis, and neurofibromatosis type 1. She also has expertise in both functional and structural imaging.

Juan Claramunt Gonzalez is a computer scientist and information specialist at the Methodology and Statistics Unit of the Institute Psychology at Leiden University. He obtained his Master's degree in Methodology and Statistics at Utrecht University along with the European Master of Official Statistics (EMOS) at Statistics Netherlands.

Tom F. Wilderjans is an associate professor at the Methodology and Statistics Research Unit (Psychology) within the Faculty of Social and Behavioral Sciences at Leiden University. His research is centered on developing and evaluating novel data analytical strategies for clustering and/or dimension reduction. As such, he tries to identify subject clusters that qualitative and/or quantitative differ in underlying mechanisms (like brain patterns). To this end, he performs extensive simulation studies (utilizing high-performance computing infrastructure), applies the techniques to empirical (brain) data (like fMRI) and develops statistical software. His additional interests include model selection problems regarding the number of clusters and underlying dimensions/components.