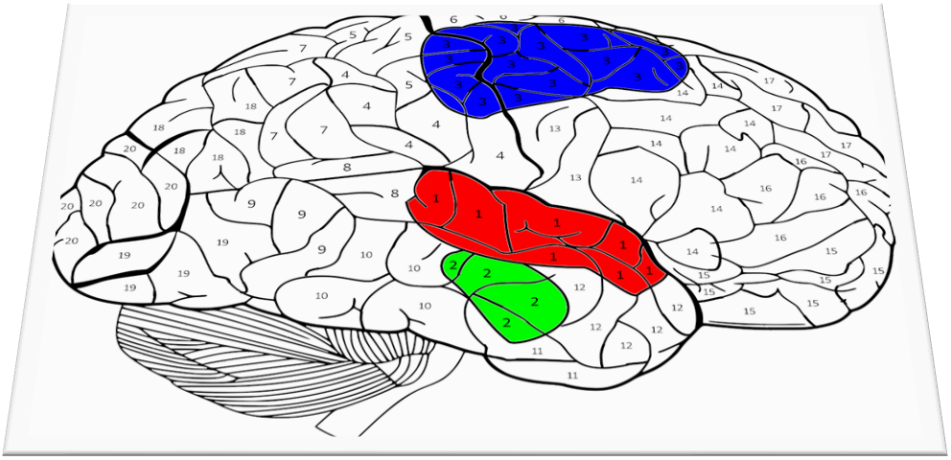
Niels Reuter

PAINTING THE BRAIN BY NUMBERS

Introducing an open-source approach to
automated regional connectivity-based parcellation



Painting the Brain by Numbers

Introducing an open-source approach to automated regional connectivity-based parcellation

Inaugural thesis pressented to the Faculty of Mathematics and Natural Sciences of Heinrich-Heine University Düsseldorf
for the degree of
Doctor of Philosophy in Natural Sciences
by

Niels Reuter

from Heerlen

Jülich/July 2, 2021

From the Institute of Neuroscience and Medicine 7, Brain and Behavior
of Forschungszentrum Jülich

Printed by permission of the Faculty of Mathematics and Natural Sciences of
Heinrich Heine University Düsseldorf

Examiners:

1. Prof. Dr. Simon B. Eickhoff
2. Prof. Dr. Tobias Kalenscher

Date of the oral defence: March 3, 2022

This thesis is primarily based on a published work (Reuter et al 2020). Chapters 2 and 3 are extensions of this published work in adapted form. The author of this thesis contributed to the published work by planning the underlying research, coding the software central to the study, performing the data processing and analysis, as well as writing and revising the paper.

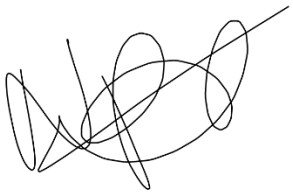
The work herein was conducted at the Institute of Neuroscience and Medicine 7, Brain and Behavior, at Forschungszentrum Jülich and the Department of Mathematics and Natural Sciences at Heinrich Heine Universität and was funded by the Deutsche Forschungsgemeinschaft (DFG*, EI 816/11-1), the National Institute of Mental Health (R01-MH074457), the Helmholtz Portfolio Theme "Supercomputing and Modelling for the Human Brain" and the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement 785907 (HBP SGA2)*

"Such is the vastness of his genius that he can outwit even himself."

– Steven Erikson, Deadhouse Gates

Acknowledgments

I am grateful to my supervisors, Simon B. Eickhoff and Kaustubh R. Patil, for their ongoing support and valuable advice. Thanks go to the collaborators of my work, Sarah Genon, Shahrzad Kharabian, Felix Hoffstaedter, Xiaojin Liu, and Tobias Kalenscher, as well as Kyesam Jung and Jianxiao Wu for testing and advising on CBPtools and Benjamin Poldrack and Alex Waite for helping to make the example data available. Further thanks go to Mario Senden and Matthieu Gilson, whom I am proud to call both colleagues and friends. I would furthermore like to thank Arie van der Lugt who recognized my passion for science and provided an early platform for its expression. Tom de Graaf, Michelle Moerel, Job van den Hurk, and Bettina Sorger whose help was invaluable while studying at the Faculty of Psychology and Neuroscience at Maastricht University. Further thanks go to all my colleagues at the INM-7, particularly Jianxiao Wu, without whom the years I have spent as a Ph.D. student would not have been as enjoyable. Lastly, I thank my parents for everything they have done to support my education and the skills they have taught me to help me get to this point in my life.



Niels Reuter

Julich, March 8, 2022

Summary

Regional connectivity-based parcellation (rCBP) is a widely used procedure for investigating the structural and functional differentiation within a region-of-interest (ROI) based on its long-range connectivity. No standardized software or guidelines currently exist for applying rCBP, making the method only accessible to those who develop their own tools. A historical background to rCBP has been provided in **chapter 1**, which continues with the aim of this work: introducing CBPtools, an open-source software package implementing rCBP. The chapter concludes by detailing various methods and concepts associated with the rCBP procedure.

CBPtools is a Python (version 3.5+) package that allows users to run an extensively evaluated rCBP analysis workflow on a given ROI. It currently supports two modalities: resting-state functional connectivity and structural connectivity based on diffusion-weighted imaging, along with support for custom connectivity matrices. Analysis parameters are customizable, and the workflow can be scaled to many subjects using a parallel processing environment. Parcellation results with corresponding validity metrics are provided as textual and graphical output. Thus, CBPtools provides a simple plug-and-play yet customizable way to conduct rCBP analyses. **Chapter two** discusses architectural choices, scope, and software dependencies, followed by a thorough description of all implemented features as well as a step-by-step guide through the processing pipeline.

In **chapter three** we demonstrate the utility of CBPtools using a voluminous data set on an average compute-cluster infrastructure by performing rCBP on three ROIs prominently featured in parcellation literature. A side-project on the investigation of potential issues regarding outliers in the data set is added as **chapter four**.

In closing we discuss our findings, provide recommendations, and suggest future extensions to the CBPtools software in **chapters five** and **six**. CBPtools is capable of reproducing parcellations found in existing literature. It offers flexibility in terms of customization while remaining easy to use. By providing an open-source software we aim to promote reproducible and comparable rCBP analyses and, importantly, make the rCBP procedure readily available.

Contents

Acknowledgments 6

Summary 7

Contents 8

Figure Index 11

Table Index 12

External Resources 13

Part One Introduction 15

1 Background 16

1.1 Connectivity-Based Parcellation 16

1.2 Motivation and Aim 20

1.3 Measures of Brain Connectivity 21

1.3.1 Resting-state functional MRI 21

1.3.2 Diffusion-weighted MRI 23

1.3.3 Other measures 25

1.4 Clustering Techniques 25

1.4.1 k-means clustering 26

1.4.2 Spectral clustering 27

1.4.3 Agglomerative clustering 29

1.4.4 Alternative Procedures 30

1.5 Cluster Evaluation 30

1.5.1 Silhouette Coefficient 31

1.5.2 Davies-Bouldin Index 32

1.5.3 Calinski-Harabasz Index 32

1.6 Open Science 32

Part Two Methodology 37

2 Implementation 38

2.1 Architecture 39

2.1.1 Validation 39

2.1.2 Workflow Generation 41

2.1.3 Tasks module 42

2.2 Data Structure 43

2.3 Dependencies 44

2.4 Workflow 47

2.5 Getting Started 47

2.5.1 Installation 47

2.5.2 Setup 48

2.5.3 Validation 49

2.5.4 Execution 50

2.6 Processing 52
2.6.1 Masking 52
2.6.2 rsfMRI Connectivity 54
2.6.3 dMRI Connectivity 56
2.6.4 Clustering 56
2.6.5 Grouping 57
2.6.6 Validity 58
2.6.7 Similarity 59
2.6.8 Report 59
2.7 Methods Explanations 60
2.7.1 Median Filtering 60
2.7.2 Nuisance Signal Regression 61
2.7.3 Relabelling Strategy 62
2.8 Alternative Approaches 64
2.8.1 Cluster Methods 64
2.8.2 Multi-session Data 65
2.8.3 Single-subject Parcellation 66
2.8.4 Automatically deriving the seed mask from an atlas 66
2.8.5 Using reference images 67
2.8.6 Running multiple CBPtools projects in parallel 67
2.9 Benchmarks 68
2.10 Extending CBPtools 70
2.10.1 Modifying the Repository 70
2.10.2 Extending a Task 71
2.10.3 Modifying the Workflow 72
2.10.4 Extending the Configuration 73

Part Three Empirical Work 77
3 Example Data 78
3.1 Data and pre-processing 78
3.2 Analysis Procedure 80
3.2.1 Parcellation Results 82
3.2.2 Amygdala long-range dMRI connectivity 82
3.3 SMA Parcellation 83
3.3.1 Results 83
3.3.2 Discussion 86
3.4 Insula Parcellation 86
3.4.1 Results 87
3.4.2 Discussion 89
3.5 Amygdala Parcellation 90
3.5.1 Results 90
3.5.2 Discussion 96
4 Outlier Detection 98
4.1 Approach 98
4.2 Results 99

4.3 Additional Exploratory Analyses 100
4.4 Discussion 102

Part Four Discussion 105
5 General Discussion 106
5.1 CBPtools 106
5.1.1 Future Extensions and Refactoring 108
5.2 Example Data 112
6 Conclusion 115

Bibliography 117

Abbreviations 129

Appendices 131
Appendix 1: Built-in rules for input validation 132
Appendix 2: Participants file example 135
Appendix 3: Confounds file example 136
Appendix 4: List of configuration parameters 137
Appendix 5: Task input, output, and parameters 147
Appendix 6: Benchmark results 151
Appendix 7: Software Mentions 155
Appendix 8: Python Package Mentions 157
Appendix 9: Usage Example 157

Figure Index

Figure 1 k-means clustering example 27

Figure 2 Spectral clustering example 28

Figure 3 Agglomerative clustering example 30

Figure 4 Popularity of programming language (PYPL) ratings 34

Figure 5 CBPtools workflow diagram 45

Figure 6 Example of extracting seed voxels from a target mask 53

Figure 7 Subsampling example 53

Figure 8 Median filtering example 61

Figure 9 Relabelling strategy to obtain group cluster labels 63

Figure 10 Reference similarity heatmap example 67

Figure 11 Forking and cloning the CBPtools repository 71

Figure 12 Visualization of the R preSMA-SMA, R insula, and R amygdala 79

Figure 13 R preSMA–SMA results from the rCBP procedure 84

Figure 14 R preSMA-SMA validity metrics and parcels 85

Figure 15 R Insula results from the rCBP procedure 88

Figure 16 Validity metrics for the R insula 89

Figure 17 R amygdala results from the rCBP procedure 92

Figure 18 R amygdala metrics and parcels for the 2, 3, 4, and 5-cluster solutions 93

Figure 19 R amygdala parcels for k = [2, 3, 4, 5] using different target features 94

Figure 20 Similarity between R amygdala clustering results with different target features 95

Figure 21 Mapping of target voxels that contributed most to the R amygdala 2-cluster solution 95

Figure 22 Methods for detecting deviant connectivity profiles (outliers) 99

Figure 23 Group-level clustering of R insula with and without outlier-removal 100

Figure 24 Correlation between the deviant index and component number 101

Figure 25 Spearman correlation components to acquisition quarter 101

Table Index

Table 1. Duration per task for the rsfMRI preSMA-SMA parcellation 69

Table 2. Duration per task for the dMRI preSMA-SMA parcellation 69

Table 3. Participants file example 135

Table 4. Confounds file example 136

Table 5. Masking task parameters, input, and output 147

Table 6. rsfMRI Connectivity task parameters, input, and output 147

Table 7. dMRI Connectivity task parameters, input, and output 148

Table 8. Clustering (k-means) task parameters, input, and output 148

Table 9. Clustering (spectral) task parameters, input, and output 149

Table 10. Clustering (agglomerative) task parameters, input, and output 149

Table 11. Grouping task parameters, input, and output 149

Table 12. Validity task parameters, input, and output 149

Table 13. Similarity task parameters, input, and output 150

Table 14. Report task parameters, input, and output 150

Table 15. Maximum RSS in MB (rsfMRI) 151

Table 16. Maximum VMS in MB (rsfMRI) 151

Table 17. Maximum USS in MB (rsfMRI) 151

Table 18. Maximum PSS in MB (rsfMRI) 152

Table 19. Maximum CPU load per second in MB (rsfMRI) 152

Table 20. Maximum RSS in MB (dMRI) 153

Table 21. Maximum VMS in MB (dMRI) 153

Table 22. Maximum USS in MB (dMRI) 153

Table 23. Maximum PSS in MB (dMRI) 154

Table 24. Maximum CPU load per second in MB (dMRI) 154

External Resources

CBPtools GitHub Repository – https://github.com/inm7/cbptools

CBPtools Documentation – https://cbptools.readthedocs.io/en/latest/

CBPtools PyPi Package – https://pypi.org/project/cbptools/

Part One

Introduction

1 Background 16
1.1 Connectivity-Based Parcellation 16
1.2 Motivation and Aim 20
1.3 Measures of Brain Connectivity 21
1.4 Clustering Techniques 25
1.5 Cluster Evaluation 30
1.6 Open Science 32

1 Background

Mapping the human brain in an effort to understand its organizational principles is a monumental task, dating back to the early 1900's with Korbinian Brodmann's famous publication on 'Vergleichende Lokalisationslehre der Groβhirnrinde' [Localization in the Cerebral Cortex; Brodmann (1909)]. The advent of modern non-invasive in-vivo neuroimaging technologies, such as magnetic resonance imaging (MRI), has been a driving growth in this research field. Accompanying progress in neuroimaging data analysis techniques allows a range of connectivity measurements from various MRI modalities. Brain organization can then be probed by analysing the patterns in these measurements.

1.1 Connectivity-Based Parcellation

Early brain mapping studies used local histological (i.e., cytoarchitectonic and myeloarchitectonic) properties to map cortical areas. Changes in local properties (e.g., thickness of cortical layers, cellular composition, etc.) were used to mark borders of otherwise homogenous areas. The Brodmann (1909) areas are a pioneering exemplary work of this method, followed later by a more detailed cortical atlas by von Economo and Koskinas (1925), who correctly assumed that cytoarchitectonic differentiation relates to functional differentiation (Triarhou 2006) and could hence be used to localize brain function. Modern studies applying histology-based parcellation of the brain use computational methods such as border mapping (as opposed to manual delineation/drawing) to differentiate areas within the brain. One such example is the JuBrain cytoarchitectonic atlas

[available through the JuBrain Anatomy Toolbox; Eickhoff et al. (2005)] which consists of cytoarchitectonic probabilistic maps of many brain regions. Using ten human post-mortem brains, cytoarchitectonic areas were analysed using an observer-independent border-mapping approach. This approach follows the assumption that laminar patterns in the cytoarchitecture are similar within an area but change abruptly at the border between areas.

While modern computation improves reproducibility, efficiency, and accuracy, histology-based parcellation remains very labour intensive and lacks generalizability due to small sample sizes, as it requires the acquisition and processing of ex vivo materials (i.e., post-mortem tissue). In contrast, modern neuroimaging techniques such as magnetic resonance imaging (MRI) are less laborious and non-invasive as they allow for the in vivo acquisition of whole-brain images. Connectivity measurements taken from various MRI modalities then allow for the investigation of structural and functional differentiation in the brain. Brain organization can subsequently be probed by analysing the patterns in these measurements. One such technique is connectivity-based parcellation (CBP), an umbrella term for a widely used and diverse set of procedures to delineate whole- and regional brain organization, originally conceived by Behrens et al. (2003) in their seminal work on the thalamus. Thalamic voxels were grouped based on similarity in their connection strengths to the rest of the brain (using manual delineation from similarity matrices), i.e., their connectivity profile, obtained through probabilistic tractography on diffusion MRI (dMRI) data. Voxels grouped together form homogeneous units, i.e., parcels, with regards to the measured connectivity marker that best describes the input data at hand. These parcels were validated by their correspondence to histologically delineated thalamic nuclei. Cohen et al. (2008) applied the same method to resting-state functional MRI (rsfMRI) connectivity markers (using a border-mapping approach in line with traditional histological work), finding local transitions in patterns of functional connectivity. Subsequent application of CBP using connectivity markers derived from rsfMRI and dMRI data has been widespread. In addition to being a less labour-intensive approach that can make use of large samples [e.g., the Human Connectome Project (HCP) data (Van Essen et al 2013), the 1000BRAINS study (Caspers et al 2014)], CBP methods can be used to map functional areas. Taken together, this contributed to the rapid adoption and increasing popularity of mapping the human brain using CBP.

CBP is appealing for its various applications, such as atlas mapping, hypothesis generation, and location mapping [cf. Eickhoff et al. (2015)] among

others. From a data processing point of view, it is a useful data compression technique for dealing with large data sets as well as complicated and time-consuming processing and analysis approaches. Questions regarding the brain's architecture such as whether the brain consists of highly specialized units (i.e., brain regions) each responsible for a particular set of functions (as opposed to a general-purpose device), and the degree of such specialization have been debated and investigated for over a century. Such highly specialized regions have been found not only for low-level function (e.g., sensory and motor processes), but also for high-level function (e.g., language, facial and place recognition, and more) [cf. Kanwisher (2010)]. Due to its ability to find local transitions in patterns of functional connectivity, the CBP procedure is effective in finding answers to questions regarding functional specialization and differentiation at the granularity of the whole-brain or a single delineated region.

CBP is an automated procedure that divides a region of interest (ROI) into functionally distinguishable and often spatially consistent parcels. The parcellation of grey matter into functionally specialized parcels is a useful approach to create a compression model of an ROI. Voxels within a parcel are homogeneous in connectivity profile to one another, relative to voxels outside of the parcel. Hence, the procedure functions as a data reduction method, as analyses can now be performed at the more manageable level of parcels, rather than at the level of voxels. While connectivity profiles across a parcel may vary (van den Heuvel and Hulshoff Pol 2010; de Reus and van den Heuvel 2013), these variations are expected to be smaller than those between parcels. Therefore, despite being a simplified data representation, parcels and by extension their borders are meaningful and reproducible. The approach can be applied to an ROI covering the whole brain resulting in a 3D brain atlas, where the connectome is now a parcel by parcel matrix, rather than a much larger voxels by voxels matrix (Smith et al 2013). Alternatively, an ROI covering only a predefined segment of grey matter can be used to investigate whether the ROI contains functionally distinct subunits (i.e., parcels), how they are shaped, and where they are located. Predominantly, connectivity profiles of ROI voxels cover all grey matter voxels in the brain. Approached differently, target regions can be predefined based on a-priori hypotheses (e.g., using only regions that have been associated with the ROI in existing literature, or regions that have a specific behavioural relation), creating an organizational model of the ROI restricted to a particular behavioural or structural domain. Note that regardless of the chosen approach, there is no one true parcellation as many different criteria can be used to segment an ROI.

Instead, the CBP approach defines the most optimal subdivision of an ROI given the characteristics of the underlying data.

The CBP procedure furthermore provides a map of the ROI, as opposed to mapping cognitive and behavioural aspects or the effects of (dys)function onto regions in the brain. The former explains the neurobiological profile of the ROI, i.e., its segmentation based on the measured functional or structural properties, whereas the latter only localizes a singular effect constrained by experimental design and ignorant of the underlying neurobiological composition (e.g., cell structure, thickness of cortical layers, etc.). As such, the former can provide a map of the functional segregation of parcels, whereas the latter cannot be used for constructing a map of the brain. In tandem, however, the former may inform the latter to improve the precision of the localization (Devlin and Poldrack 2007). Furthermore, CBP can be used for generating novel hypotheses (Eickhoff et al 2015) by investigating the existence of CBP-derived substructures within an ROI.

In this work sole focus will be placed on regional CBP (rCBP), an approach that aims to investigate the structural and functional differentiation within a delimited area of grey matter (as opposed to the whole brain) based on its long-range connectivity. Unlike whole-brain CBP where the area to be parcellated consists of all the grey matter (Schaefer et al 2018), rCBP focuses on a particular ROI, hence allowing an in-depth analysis by uncovering the internal differentiation of a region. A common approach to mapping the human brain through rCBP is to cluster voxels/vertices into parcels. A clustering algorithm is used to group voxels/vertices within a given ROI based on similarity in their connection strengths to a set of target voxels/vertices, i.e., their connectivity profile. Voxels clustered together form homogenous units, i.e., parcels, with regards to the measured connectivity marker that best describes the input data at hand. The parcels are often spatially consistent as neighbouring voxels usually exhibit more similar connectivity patterns than those further away. Thus, the rCBP procedure can map functional or structural subdivisions/clusters within a particular ROI. rCBP derived parcels are known to match with histological parcellation (Bzdok et al 2013), but they may also provide subdivisions pertaining to different sources of information not revealed by cytoarchitectonic mapping alone (Clos et al 2013). As each MRI modality yields a different aspect of brain connectivity, rCBP on each modality can yield differing parcellations with different interpretations. Commonly used imaging modalities include, but are not limited to, resting-state blood-oxygen-level dependent (BOLD) time-series used to measure task-independent functional connectivity, diffusion-weighted imaging

(DWI) based probabilistic diffusion tractography to estimate anatomical fibre-connectivity, as well as meta-analytic connectivity modelling (MACM) as a measure of task-dependent functional connectivity and co-activation patterns. Due to the different interpretations that may result from each modality, a multimodal approach [e.g. Genon et al. (2018) and Plachti et al. (2019)] may be used to compare unimodal parcellations.

1.2 Motivation and Aim

Despite its popularity, no standardized software or guidelines currently exists for applying rCBP, making the method only accessible to those who develop their own tools. It is therefore a challenging and time-consuming procedure to employ. As neuroscience makes a transition towards big-data, with prominent examples such as the HCP (Van Essen et al 2013), and the 1000BRAINS study (Caspers et al 2014) having well over a thousand subjects, it becomes an increasing necessity to add support for high-throughput computation and parallel processing. Furthermore, the numerous options available at each step of the rCBP procedure paired with the absence of uniform guidelines make it difficult to have comparable results. For example, choice of clustering algorithm may influence the clustering results, with options such as k-means clustering, spectral clustering, or hierarchical clustering (Von Luxburg 2007; Hastie et al 2013).

To resolve these issues, CBPtools, an open-source distributed workflow for rCBP enclosed in a Python package, is introduced. By unifying the methodological choices behind the procedure into a customizable workflow, it offers a fast, stable, and reproducible means to parcellate brain regions. Furthermore, computational demands highlighted by complex algorithms and large data sets are mitigated by efficient parallel execution of the procedure. CBPtools offers a common working ground to conduct reproducible and data-driven parcellation analyses effortlessly and efficiently.

CBPtools parcellates an ROI and provides the output as NIfTI images along with commonly used cluster-validity metrics. The tool's approach and methods are derived from a substantial body of parcellation works (Wang et al 2015; Bzdok et al 2015; Chase et al 2015; Barron et al 2015; Hardwick et al 2015; Eickhoff et al 2016; Muhle-Karbe et al 2016; Genon et al 2017; Genon et al 2018) and consists of a customizable rCBP workflow allowing users to specify the input data and a range of parameters through a configuration file. CBPtools can calculate connectivity matrices from resting-state or DWI data, but they may instead be

provided directly as input. It then computes parcellations based on the connectivity matrices (projected onto NIfTI images of the ROI and as NumPy array files, as well as 3D voxel plots) and outputs validity metrics for their evaluation. Note that the procedure outlined here utilizes hard clustering. Therefore, when connectivity markers are assumed to change through soft transition (i.e., showing a gradient), parcels generated through this procedure should not be interpreted as neurobiological units but as a simplified data representation (or compression model).

1.3 Measures of Brain Connectivity

Rather than using data obtained through histology, CBPtools relies on measures of connectivity obtained through neuroimaging techniques such as MRI. CBPtools is (currently) capable of interpreting the two most frequently used MRI data modalities: rsfMRI data for functional connectivity, and dMRI data for anatomical connectivity. Various other connectivity modalities popular in rCBP, such as MACM and structural covariance (SC) are viable contenders to extend the breadth of CBPtools, made possible by CBPtools' modular implementation (see Ch. 2 [p. 38]) and open source distribution. Despite different neurobiological properties showing similar patterns of organization (von Economo and Koskinas 1925; Zilles et al 2002) and convergence between prior brain maps (e.g., cytoarchitectonic maps) and rCBP-derived maps being used as an external validity metric, by no means is there a gold-standard measure of connectivity [or parcellation method, for that matter (Eickhoff et al 2018)]. Importantly, the selected connectivity measure critically influences the interpretation of the ensuing parcellation.

1.3.1 Resting-state functional MRI

Functional MRI measures brain activity as it changes in (near) real time by detecting the changes in the level of oxygen in the blood. It exploits the correspondence of blood-oxygen-level to neuronal activity using the BOLD contrast (Ogawa et al 1990), which is a measure of the oxygenated to deoxygenated haemoglobin ratio. Haemoglobin transports oxygen throughout the body, including the brain, and has different magnetic properties when it is oxygenated as to when it is not carrying oxygen. Current theory proposes that active brain areas require more oxygen than inactive areas, hence during activity the blood vessels expand allowing for a larger influx of oxygenated haemoglobin.

This influx causes the BOLD signal to rise considerably and quickly, known as the BOLD response. The (near) real time nature of this neuroimaging method allows for the detection of changes in brain activity when performing tasks or at rest.

For rsfMRI the aforementioned approach is used to measure intrinsic brain activity in a resting (i.e., task-negative) state. That is, even when tasks are not actively performed, cortical activity still reflects consistent patterns of activation in the form of low frequency (~0.1 Hz) fluctuations structured as repeatedly (re-)emerging anti-correlated functional networks (Biswal et al 1995; Raichle et al 2001; Fox et al 2009; Deco et al 2011). Spontaneous activity is not random, as it reveals the organization of coherent functional networks. rsfMRI can therefore be used to explore the functional organization of the brain and map functional networks.

CBPtools assumes that the rsfMRI data has been treated with necessary fMRI pre-processing (e.g., head-motion estimation, slice time correction, susceptibility distortion correction, confound estimation) including realignment and normalization to a template space. Denoising based on independent component analysis like Automatic Removal of Motion Artifacts (ICA-AROMA) (Pruim et al 2015) or FMRIB's ICA-based X-noiseifier (FIX) (Salimi-Khorshidi et al 2014) is encouraged if suitable. FIX in combination with mean white matter and cerebrospinal fluid signal regression has been shown to work well in the context of rCBP [i.e., improved cluster stability and consistency of clusters between neuroimaging modalities (Plachti et al 2019)]. CBPtools only includes pre-processing facilities to perform band-pass filtering, spatial smoothing, and nuisance signal regression (see Sect. 2.6.2 [p. 54]). One possible all-in-one pre-processing pipeline for fMRI data is fMRIPrep (Esteban et al 2019) which, like CBPtools, is a Python-based free and open source software package. Alongside using several popular Python packages for handling neuroimaging data (e.g., NiBabel (Brett et al 2019) for NIfTI image handling, nitime (Rokem et al 2009) for time-series analysis, and nipype (Gorgolewski et al 2011) for facilitating interaction between various neuroimaging software) it makes use of various neuroimaging tools that are commonly used for pre-processing MRI data, such as FSL (Jenkinson et al 2012), ANTs (Avants et al 2011), AFNI (Cox 1996), FreeSurfer (Dale et al 1999; Fischl et al 1999), and more. Note that fMRIPrep was not used to pre-process any of the data included in this work.

To obtain rsfMRI connectivity markers for a given ROI, the subject-wise time-series of the ROI voxels and a set of target voxels (e.g., whole-brain grey matter voxels) are correlated. The correlations between one seed voxel and all target voxels form the connectivity profile of that seed voxel. Connectivity is calculated using linear correlations between the ROI voxels of the timeseries of a subject (x), and the target voxels of the timeseries of the same subject (y). The target can be any part of the brain, although in the example data used here (see Ch. 3 [p. 78]) a subsampled whole-brain grey matter mask is applied (for an explanation of subsampling, see Sect. 2.6.1 [p. 52]). Both x and y are first standardized and then correlated by transposing y and taking its dot product of x, dividing this by the number of voxels in x and transposing the result again (for a code example, see Sect. 2.6.2 [p. 54]). In the event there are voxels without sufficient variance (i.e., variance not exceeding a threshold of the smallest representable number in Python's NumPy package, `np.finfo(np.float32).eps`) within either the target or ROI masked time-series, the standardization will have failed (i.e., a division by zero will have occurred on account of the standard deviation being zero). Hence, all correlations resulting from a computation with a 'not a number' (NaN) element are set to zero. If a Fisher's Z transform is to be performed on the connectivity matrices, then values at precisely 1 or -1 will result in an infinite value. To prevent this, they are set slightly below 1 and above -1 (resp.).

1.3.2 Diffusion-weighted MRI

dMRI exploits the diffusion of water molecules to obtain a magnetic resonance contrast. Patterns in water molecule diffusion can reveal details about tissue architecture, as the Brownian motion of the water molecules will not be spherical (i.e., anisotropic) when blocked unevenly (i.e., from a particular direction). In white matter it is assumed that the water molecule diffusion is blocked primarily by axonal myelin sheaths, hence it can be used for fibre tractography. When blocked the diffusion will be elliptical, aligned along the orientation of the fibres. This technique is called diffusion tensor imaging (DTI). By using this technique fibres can be identified which can be further processed to provide a measure of connectivity between regions in the brain.

For CBPtools, the raw anatomical and diffusion data must first be pre-processed to the point where the data can be used as input to FSL's probtrackx2 tool. Common pre-processing steps for dMRI data include brain segmentation and

skull stripping, bias correction, eddy current correction, and estimation of the diffusion data. FSL's bedpostx must be used for the last step, as only this tool provides the necessary input for probtrackx2. The dMRIPrep software (Richie-Halford et al 2020), an fMRIPrep inspired processing pipeline intended for diffusion data, can be used for pre-processing.

First, the T1-weighted (T1w) anatomical image must be skull-stripped. There are various approaches to skull stripping, each with their own advantages and disadvantages (Kalavathi and Prasath 2016). A popular approach is to segment the T1w anatomical image into different tissue types and subtract the skull from the original image. The resulting image can be binarized to serve as a whole-brain T1w mask. Bias correction is used to remove intensity bias from regions where the intensity should be equivalent. That is, visual observation of the raw diffusion image may reveal variations in the intensity contrast of cortical regions. Since there is no diffusion in cortical grey matter regions the intensity should not differ. Eddy current correction corrects for distortions and alignment issues that may have occurred due to head movement during the acquisitions of the diffusion data. Every few acquisitions there will be a maximum intensity acquisition (b0). The b0 acquisitions can be realigned to one another as they do not differ in intensity. The realignment parameters can then be used to correct distortions in acquisitions between the b0's. Following this, the whole-brain T1w mask is linearly co-registered to an average of the diffusion data to bring the mask to diffusion space. Linear image co-registration can be used when both the mask and the diffusion data are from the same subject (i.e., native). Warp fields must then be created between the diffusion space and a common reference space, e.g., Montreal Neurological Institute (MNI) 152 space, to allow further outputs to share the same space and thus be comparable. The diffusion parameters can then be estimated using bedpostx. Output from this tool can be used as input for CBPtools.

Connectivity markers for dMRI are obtained using probtrackx. This tool produces sample streamlines to identify the number of streamlines that pass through or connect voxels, known as the connectivity distribution. It iteratively uses the voxel-wise bedpostx distributions on diffusion parameters to draw an orientation, move in that direction, and assess whether there are termination criteria. In doing so, it takes crossing fibres into account. CBPtools uses the ROI as a seed region from which the streamlines originate, so that probtrackx can find where streamlines connect to in a target region (commonly the whole brain). This results in an ROI by target connectivity matrix.

1.3.3 Other measures

Other measures of connectivity that are commonly used in the context of rCBP are MACM and SC. These measures differ from the aforementioned in that they are based on covariance measurements across a pool (i.e., a pool of studies for MACM, and a pool of subjects for SC). This means that the connectivity markers are derived at the group level, rather than at the subject level as is the case for rsfMRI and dMRI-based connectivity markers.

The MACM procedure uses coordinate-based peak activations from many published functional neuroimaging experiments, constrained by relevant search criteria in the form of taxonomic categories (e.g., neuroimaging modality, functional/behavioural association, gender, etc.). The studies these peaks are derived from are aggregated in databases such as BrainMap (Fox and Lancaster 2002), Neurosynth (Yarkoni et al 2011), and NeuroVault (Gorgolewski et al 2015). Co-activation likelihood estimation (Eickhoff et al 2012) is then used to find convergence across studies, under the assumption that functional connectivity between brain regions reliably co-activates. Co-activation between each ROI voxel and all voxels in a target area (e.g., the whole brain) then functions as that voxels' connectivity profile. Taken together, this forms the ROI voxel by target voxel connectivity matrix taken over a group of studies. SC instead describes the correlation between anatomical metrics such as cortical thickness or volume between pairs of voxels (Wright et al 1999), e.g., each ROI voxel to a set of target voxels, over a pool of subjects. What results here is again one group connectivity matrix containing the correlations between the anatomical metrics over a pool of subjects.

Aside from the modalities mentioned here, CBPtools can also interpret connectivity matrices given directly as input. Therefore, any measure of connectivity can be used to obtain parcels, despite the measure not being executed by CBPtools.

1.4 Clustering Techniques

Clustering, within the context of rCBP, is a procedure to group (i.e., cluster) a set of voxels such that within-group similarity of their connectivity profiles is high whereas between-group similarity is low. There are various algorithms that can be used for clustering a set of voxels that each differ in their own respects. Each clustering algorithm has a different approach to forming and finding clusters;

hence the understanding of the resulting clusters is likewise different. Furthermore, each clustering algorithm can be fine-tuned with a range of parameters, the values of which depend on the data set and the purpose behind the clustering. The fine-tuning, as well as parameter selection, is therefore an iterative process using prior knowledge to obtain the most biologically meaningful results. For all clustering algorithms listed here, the input is an ROI voxel by target voxel connectivity matrix (see Sect. 1.3 [p. 21]). That is, the ROI voxels are the samples and the connectivity to all target voxels (i.e., the connectivity profile) are the features.

1.4.1 k-means clustering

k-means clustering is a centroid-based clustering algorithm in which a cluster is formed around a centroid. First, k cluster centroids are randomly chosen from all samples (i.e., seed voxels; dotted circles in **Fig. 1**). Each sample is assigned to its nearest centroid by computing the squared Euclidean distance of the sample's features (i.e., the connectivity profile of the seed voxel) to each of the centroids and assigning the sample to the centroid it is closest to (see samples **A** through **G** in **Fig. 1**). Once all samples have been assigned, each centroid is reassigned to the mean of all samples assigned to it. Samples are then re-assigned to their nearest centroids again. These steps iterate until either n iterations (commonly 10.000) have passed, or until the difference between a centroid's original assignment and its reassignment is smaller than a predefined tolerance value. To offset the random nature of the initial centroid selection, the entire procedure is repeated i times [commonly 256 for rCBP, as suggested by Nanetti et al. (2009)], resulting in a cluster solution that has the best ratio of low within-cluster distance to high between-cluster distance.

The k-means algorithm is relatively simple to implement and efficient for a large variety of data types, hence it enjoys a great deal of popularity in the scientific community. It is fast and efficient in its computational cost and compared to agglomerative and spectral clustering, k-means is the faster algorithm (its time complexity is often classified as $O(n)$, i.e., linear in the number of data points). However, this depends on the number of iterations required to reach convergence which is hard to classify as it depends on the data and initial seed. While the algorithm is effective for a large variety of data types, it assumes the clusters in the data should have a notion of a centre, are of uniform size, and are spherical (i.e., the mean of each cluster converges towards its centre and the nearest cluster centre is the correct assignment). Despite this, k-means does not

consider the size of the clusters. Results of the algorithm can vary depending on the initial choice of seeds (i.e., the selection method of the cluster centroid), which may result in convergence to a local minimum (as opposed to the more desired global minimum). This can be mitigated (but not fully overcome) by both having a large enough i and using an initialization algorithm such as k-means++ which spreads out the initial cluster centroid assignments. Furthermore, k-means does not optimize the number of k clusters, but instead takes it as an input parameter. That is, k-means will always return the defined k number of clusters, yielding poor results when set inappropriately. The choice of k is often ambiguous and depends on the scale and distribution of the data (i.e., the ROI and its connectivity profiles), hence requiring validation (see Sect. 1.5 [p. 30]). To accommodate validation, it is advised to explore a range of k. Lastly, the k-means algorithm can be sensitive to outliers. An outlier as a starting seed may result in a cluster of its own, whereas the assignment of an outlier to one cluster from another may impact the means of both clusters changing them considerably.

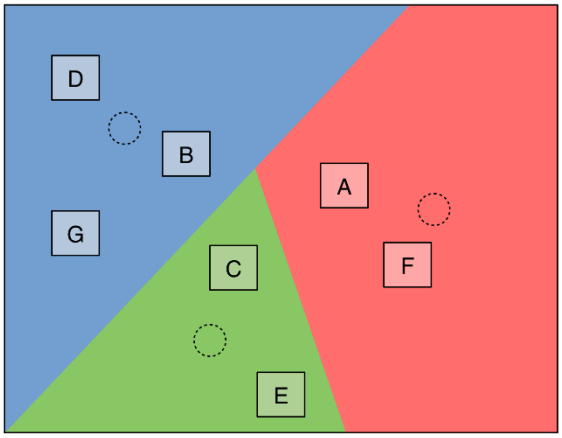


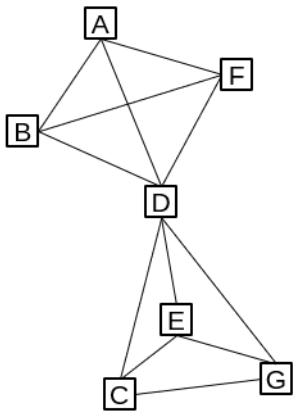
Figure 1 k-means clustering example. This example illustrates the assignment of samples (i.e., seed voxels) A through G to their nearest cluster centroid (dotted circles) for a 3-cluster solution (the blue, green, and red areas representing the clusters)

1.4.2 Spectral clustering

Spectral clustering is an affinity-based clustering algorithm which, in simple terms, performs dimensionality reduction prior to applying a standard clustering algorithm (e.g., k-means). First, a similarity graph, i.e., a graph representing the relationship between the samples, is created (see the graph on the left of **Fig. 2**). There are various ways to approach this, such as computing a graph of nearest neighbours or using a radial basis function kernel. Regardless of the chosen algorithm, it must produce similarity scores, i.e., non-negative values that increase

with similarity. The affinity matrix of this graph contains values expressing how similar the samples are to one another (see the matrix on the right of **Fig. 2** for an example of an *unweighted* affinity matrix). The degree matrix of the similarity graph is a diagonal matrix containing the degree (i.e., the sum of the similarity values for each sample) on the diagonal. The graph Laplacian of the similarity graph is given by subtracting the affinity matrix from the degree matrix. The samples are then embedded in a low-dimensional space (i.e., spectral embedding) by computing the first k eigenvectors of the graph Laplacian as a feature vector for each sample, where k is the desired number of clusters. This results in a *samples by features* matrix serving as input for a clustering algorithm.

A key strength of the spectral clustering algorithm over the k-means algorithm is that it does not make strong assumptions on the statistics of the clusters (i.e., the form/shape of the clusters). Furthermore, unlike k-means, it is not at risk getting stuck in local minima and therefore does not require multiple initializations. It can also be implemented efficiently for large data sets as the adjacency matrix is sparse. That said, very noisy datasets can cause problems, as the most informative eigenvectors are not necessarily the top ones. Computing the eigenvectors is also a computational bottleneck. In terms of computational efficiency, spectral clustering is the slowest out of the three listed algorithms (with a time complexity of $O(n^3)$, where n is the number of data points). The choice of a good similarity graph is furthermore not trivial (Von Luxburg 2007), and the algorithm is quite unstable under different choices of parameters for the similarity graph.



$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure 2 Spectral clustering example. A similarity graph of nodes A through G (left) representing seed voxels, and an unweighted affinity matrix (right) showing the connection between seed voxels

1.4.3 Agglomerative clustering

Agglomerative clustering is a 'bottom-up' hierarchical clustering technique in which each sample is initially considered to be a cluster of its own. Samples/clusters are iteratively merged until the desired number of clusters has been reached (see **Fig. 3**). Comparatively, divisive clustering, a 'top-down' hierarchical clustering algorithm, instead initializes with all samples being part of one big cluster which is gradually subdivided. First, a proximity matrix is calculated which contains a measure of similarity or distance (e.g., Euclidean or Hamming distance) between the samples/clusters. The two most similar (or closest) clusters are merged, and the proximity matrix is updated until the desired cut-off point (i.e., k number of clusters) has been reached. Since clusters (at iterations higher than the initial iteration) consist of more than one sample (i.e., seed voxels), there are various ways to calculate how close or similar clusters are. For example, the single linkage algorithm uses the minimum of the distances between all samples of two clusters, whereas the complete linkage algorithm instead uses the maximum of the distances, and the average linkage algorithm takes the average of the distances between all samples of two clusters. Therefore, the optimal choice of linkage algorithm depends on the form of the data. The single linkage algorithm is best for clusters of different sizes and shapes, but it is also very sensitive to noise. The complete and average linkage algorithms are not as affected by noise but have a larger bias towards global patterns in the data.

Agglomerative hierarchical clustering differentiates itself from k-means clustering in that it does not initialize with a random seed (i.e., k-means initializes with a random set of centroids that varies between initializations). Therefore, when using the same parameters and data the hierarchical clustering algorithm will always provide the same results. In terms of computational efficiency, agglomerative clustering lies in between the k-means clustering and spectral clustering algorithms (with a quadratic time complexity, i.e., $O(n^2)$). Since the merges of data points (i.e., voxels) are final (as opposed to k-means, where reassignment can take place during each iteration), problems may occur with noisy, high dimensional data.

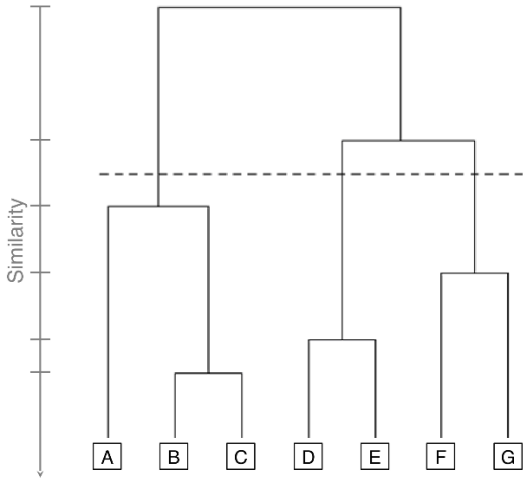


Figure 3 Agglomerative clustering example. Seed voxels A through G are iteratively merged into clusters (bottom to top) until the desired number of 3 clusters has been reached (dotted line)

1.4.4 Alternative Procedures

Instead of employing an unsupervised machine learning algorithm, other (non-clustering) procedures may be used. For example, probabilistic, graded (Bajada et al 2017), or boundary mapping (Cohen et al 2008) approaches are also employed. Probabilistic or 'soft' approaches, such as fuzzy c-means clustering (Bezdek et al 1984) and independent component analysis (ICA) provide continuous and probabilistic clusters that may overlap with one another, as opposed to the discrete and binary 'hard' clustering approaches described in the previous sections. When suspecting functional gradients in the ROI, e.g., retinotopic or tonotopic maps in visual and auditory regions, respectively, a 'hard' clustering approach that clearly separates clusters may not be appropriate. In such cases, graded mapping approaches such as spectral reordering may reveal the underlying neurobiological structure of the ROI more accurately. Boundary mapping, on the other hand, identifies abrupt local changes between the connectivity profiles of voxels to identify the borders between adjacent clusters rather than identifying clusters as a grouping of voxels. Hence, borders rather than clusters are established to divide an ROI.

1.5 Cluster Evaluation

Finding the appropriate number of clusters is a challenging and unresolved problem. There may not even be a *true* number of clusters as the brain has a multilevel organization (Eickhoff et al 2018). It is common to probe a range of k

values starting at two to a number determined through prior knowledge of the ROI and the source data (i.e., based on neuroimaging modality, granularity of the data, or the selected target regions) structure. Therefore, the clustering solutions at different k need to be evaluated. One possibility is using external validation which contrasts the clustering solutions against a predetermined structure, which is independent on the source data (i.e., using a predefined cytoarchitectonic parcellation as an external reference for clustering results of the same region). In the absence of information for external validation (as is frequently the case), internal cluster validation can help to select an optimal clustering in a data-driven way. Several such metrics can be used to rank the clustering results. However, different metrics often produce divergent results. Baarsch et al. (2012) evaluated the Dunn index, the Davies-Bouldin index, the Calinski-Harabasz index, the Silhouette index, the Point Biserial measure, the Pakhira-Bandyopadhyay-Maulik score, and Sum-of-Squares, concluding Sum-of-Squares to be most effective, followed closely by the Silhouette index. Popular alternatives like the Davies-Bouldin index and the Calinski-Harabasz index were only moderate contenders, while the Dunn index performed poorly. Even the best measure, however, was only correct in 60% of the test-cases. Nevertheless, validity metrics are often tested on simulated or simple data sets which might not generalize to the complexity inherent in the connectivity data. Furthermore, many more validity metrics exist [such as the I index, which was tested to perform well in a review by Maulik et al. (2002)]. In general, it is difficult to deem any single validity metric to be good for clustering as data properties may vary significantly between data sets. Therefore, *CBPtools* provides several validity metrics. Sufficient care must be given when deciding which measure to rely upon.

1.5.1 Silhouette Coefficient

The name of the Silhouette coefficient is derived from a technique that provides a graphical representation of clusters based on their dispersion and separation, showing which samples lie within and which samples lie between clusters (Rousseeuw 1987). It provides a measure that indicates how similar a sample is to the cluster it has been assigned to (i.e., how well the sample has been clustered). The measure ranges between -1 and +1, where a positive value indicates that the sample is more similar to its own cluster than to the nearest neighbouring cluster. A value near 0 indicates overlapping clusters, whereas negative values indicate samples have been assigned to the wrong cluster. In the case of low or negative values, a different number of clusters may provide a better

fit for the data. It is calculated using the mean intra-cluster distance and the mean nearest-cluster distance for each sample, where any distance metric can be used (e.g. Euclidean distance or Hamming distance). The average silhouette score over all clustered samples is then used as the coefficient representing how well the data has been clustered.

1.5.2 Davies-Bouldin Index

Introduced by Davies and Bouldin (1979), the Davies-Bouldin index evaluates the similarity within a cluster and differences between clusters. In the ideal case, the similarity within a cluster is high (i.e., less dispersion), whereas the similarity between clusters is low (i.e., clusters are farther apart – more separation). As such, it can be defined as the ratio of inter-cluster distances to intra-cluster distances. The measure has a minimum value of 0 without an upper bound, where a lower value indicates a better clustering.

1.5.3 Calinski-Harabasz Index

The Calinski-Harabasz index, also known as the variance ratio criterion, was proposed by Calinski and Harabasz (1974) as a method of cluster analysis. Like the Silhouette coefficient and the Davies-Bouldin index, it makes use of the ratio of inter-cluster dispersion and intra-cluster separation. The inter-cluster dispersion is the sum of squares of the distances between the centre of each cluster and every point in the cluster. The intra-cluster separation is the sum of squares of the distances between the centre of each cluster and the centroid of the data set weighted by the size of the cluster. Unlike the Davies-Bouldin index, which also uses cluster centres for calculating separation, the Calinski-Harabasz score uses the centroid of the entire data set instead of the other clusters.

1.6 Open Science

A fundamental aspect of science is the distribution of peer-reviewed scientific results, fostering an environment for collaboration, critiquing, replication, and recycling both data and code. Several obstructions exist that hinder the growth of such an environment, such as paywalls of for-profit publishers, usage restrictions on published data, and the lacking availability of source code. The open science movement aims to break these barriers by making scientific research, i.e., publications, data, and software, openly available to anyone. Open science is

considered an umbrella term for five proposed schools of thought on the creation and dissemination of scientific results. Each school of thought is concerned with a different aspect of open science [i.e., technological architecture, accessibility of knowledge creation, impact measurement, collaboration, and access to knowledge (Fecher and Friesike 2014)]. Overall, the aims of each school of thought benefit transparency, accessibility, and collaboration causing the quality and impact of new scientific developments to grow. As a result of the movement, many journals now require the publication of code and data along with results, and universities and research centres across the globe increasingly adopt open access policies for affiliated publications.

The open science concept is closely tied to (and encompasses) the open-source model for software. Open source practices, such as source code publication on software forges (e.g., GitHub, GitLab, SourceForge, bitbucket, etc.), encourage collaboration and transparency, which in turn complements traditional software development (i.e., top-down) with distributed labour and intelligence (i.e., bottom-up) (Riehle et al 2009). The transparent nature of open-source projects furthermore helps with the discovery of software flaws/bugs which is particularly important when scientific results depend on (or are derived with) the underlying software. Collaborative efforts have resulted in tools that are widely used in the scientific community, enhancing replicability and reproducibility of results, and further stimulating the creation of new tools (i.e., by building upon existing open-source software). For example, CBPtools is written in Python which is developed under an OSI-approved open-source license. Several open-source Python packages (e.g., NumPy, SciPy, scikit-learn, NiBabel, and snakemake) without which the development of CBPtools would not be feasible are made use of. Over the years the Python programming language has become a valuable asset to the data science community by virtue of its ease-of-use, open-source license, wide applicability, and vast library of specialized packages. The PYPL index[1], which indexes the popularity of programming languages based on how often users search for a tutorial of the language on Google, shows a steady growth in popularity of Python, whereas MATLAB, a non-free closed-source language, is in decline (see **Fig. 4**). The rise in popularity of Python is furthermore captured with other metrics, such as GitHub unique contributions[2] and the TIOBE index[3].

[1] http://pypl.github.io/PYPL.html (accessed 2/5/2020)
[2] https://octoverse.github.com/ (accessed 2/5/2020)
[3] https://www.tiobe.com/tiobe-index/ (accessed 2/5/2020)

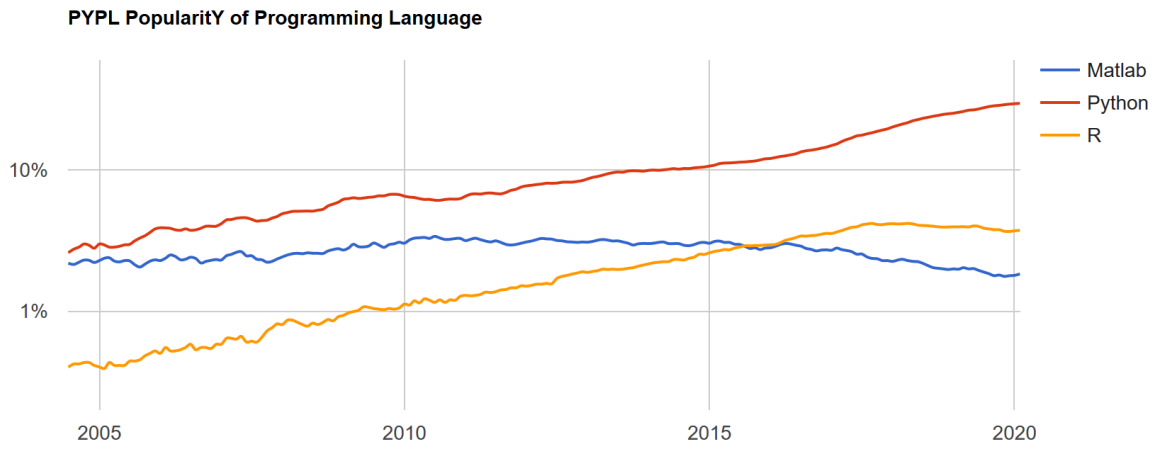


Figure 4 Popularity of programming language (PYPL) ratings. The popularity of the three most common programming languages in Neuroscience, MATLAB, Python, and R, are displayed in blue, red, and yellow, respectively. Time is on the x-axis, whereas the share of search results in on the y-axis (i.e., as of this writing Python sits at a 29.88% share).

CBPtools is distributed under an open-source license and the source code is made available on GitHub[1]. Furthermore, the software is available on Python's official third-party software repository, the Python Package Index (PyPi), which is used as a default source for the pip package manager (which is Python's default package manager and is distributed along with it). By unifying the methodological choices behind the procedure into a customizable workflow, it offers a fast, stable, and reproducible means to parcellate brain regions. Furthermore, the open-source nature allows others to contribute to the code to improve and extend the procedure.

[1] https://github.com/inm7/cbptools

Part Two

Methodology

2 Implementation 38
2.1 Architecture 39
2.2 Data Structure 43
2.3 Dependencies 44
2.4 Workflow 47
2.5 Getting Started 47
2.6 Processing 52
2.7 Methods Explanations 60
2.8 Alternative Approaches 64
2.9 Benchmarks 68
2.10 Extending CBPtools 70

2 Implementation

CBPtools is written in Python (version 3.5+) to exploit Python's prolific presence in the data science community and can be installed with pip (`pip install cbptools`). Capitalizing on pre-existing and widely used packages, such as NumPy, SciPy, NiBabel, and scikit-learn, a range of methods needed for the rCBP procedure was made available. Hence, the software is very accessible (on account of Python and its libraries being free and open source), as well as compatible with many operating systems. CBPtools makes use of snakemake (Köster and Rahmann 2012), an easy-to-use and well-documented workflow management system with parallel processing capabilities that allows the workflow execution to be scaled to various processing environments (i.e., server, cluster, grid, or cloud environments). Through snakemake, CBPtools is compatible with job schedulers that support shell script (such as SLURM or HTCondor). Furthermore, CBPtools can be resumed with partially processed data (e.g., due to hardware failure) making it stable and efficient for use on real world data. The combination of snakemake's command line execution and an easily modifiable configuration file make it possible to set up and run the software without any programming knowledge.

2.1 Architecture

One of the aims in developing CBPtools was to make it easy to use while maintaining modularity and expandability of the code. It was opted to use a command line interface instead of a graphical user interface, as the former is significantly easier to maintain and update. Importantly, users do not require any programming knowledge to use the tool. Furthermore, using CBPtools requires no training, although it is recommended to read the online documentation to gain an understanding of the workflow procedure.

The CBPtools software can be roughly divided into three distinct parts. The *validation suite* ensures the configuration file is interpreted correctly and evaluates the input data. The *workflow generator* uses the input data and user configuration to dynamically generate a Snakefile (i.e., a snakemake compatible workflow file). Finally, the *tasks module* incorporates various internal and external functions for producing the rCBP (interim) output. Each of these three parts can easily be modified and expanded.

Possible future expansions should always remain within the scope of the project. That is, support for more measures of connectivity, algorithms for cluster separation, validity measures, and output formats. Notably, future additions should restrict themselves to their respective parts. For example, a new connectivity measure should not result in data that cannot be used seamlessly in subsequent parts. Any such addition would require dependencies throughout the processing pipeline which subsequent modifications would have to consider. As a result, the software is at risk of becoming bloated and unmaintainable. Features outside of the project's scope, such as data pre-processing or outlier detection, are better suited as stand-alone Python packages that support the same data formats as CBPtools does.

2.1.1 Validation

The first step to perform rCBP with CBPtools is to create a configuration file containing all relevant parameters with the users' desired values. When the user triggers the creation of a new project this file is loaded as a Python nested dictionary using the `pyyaml` package. It is then passed as an argument to the `Validator` class. This class uses a 'YAML ain't mark-up language' (YAML) schema that contains the rules/requirements for each parameter (i.e., key/value

pair), such as whether the parameter is required, under what conditions it is used, its default value, and so on. The below code snippet shows an example schema of the binarization parameter used for mask pre-processing.

```
1.  binarization:
2.    type: float
3.    min: 0.0
4.    default: 0.0
5.    dependency: [modality: [rsfMRI, dmri]]
6.    required: false
7.    desc: "Threshold above which voxels in the ROI mask image are ..."
```

This is a non-required parameter of type `float` (i.e., a real decimal number) with default value of 0.0. The default value will also be used when generating an example configuration file. When defined, it cannot be smaller than 0.0 and it is only used when the data set modality is either rsfMRI or dMRI (i.e., it is not used when the input consists only of connectivity matrices). Lastly, the parameter has a description which is used both as an inline code comment to make collaboration more efficient, and for automatically generating the online documentation section of the configuration parameters[1]. There are more built-in rules/requirements (see Appx. 1) and a means to create custom rules (i.e., rules that apply only to one parameter). Adding a new parameter is as simple as adding a new entry to the schema.

The `Validator` class is instantiated when the `cbptools create` entry point is used. It first iterates over all entries in the schema and creates a meta-data object for each parameter. This is done to ensure that each parameter has the same structure, and that only one class must be modified when parameter attributes change. Each parameter rule/requirement is defined as a class method of the `Validator` class and must be prefixed by `_rule_` (i.e., `_rule_type()` for validating the object type of the parameter's value). Custom rules are instead prefixed with `_rule_custom_`. When a rule fails to be validated (e.g., the type should be `float` but the value is a `string`) a `RuleError` is raised, which is caught and logged. Note that the validation procedure only ends once the entire configuration file has been assessed, so that the resulting log can be as descriptive as possible. However, even one `RuleError` is enough to fail validation. Rule methods are always passed the parameter's meta-data object. For custom rules it is furthermore possible to reference the current class object (i.e., `self`) to access other parameter values.

[1] https://cbptools.readthedocs.io/en/latest/configuration/parameters.html

If needed default parameter values will be assigned and unused, empty, or non-existing parameters are deleted. This will always be logged on the warning level. The validated and cleaned configuration is then passed on as a nested dictionary to the setup procedure. It is also stored as a YAML file in the project directory, although it is only used as a user reference as the configuration values are placed directly in the Snakefile.

The `Validator` class is fast and efficient, intended only to capture mistakes in formatting and parameter definitions. It is not intended to assess whether files exist and have the proper type and meta-data. A more thorough evaluation of the data set files is done instead by the `Setup` class which creates a `DataSet` object of the relevant modality and evaluates the header information of all given files. Only file headers were chosen to be evaluated as they provide sufficient information for the marginal validation CBPtools performs, while significantly reducing computation time for setting up the project. Despite this precaution, the setup procedure is not fast. However, it is a crucial step to prevent faulty input data from causing errors late in the workflow. Such errors may cause a significant waste of compute time and resources, as interim data may need to be discarded.

The `DataSet` class has methods for evaluating modality-specific and modality-agnostic files, the former of which are grouped by modality (i.e., all rsfMRI files are evaluated in one class method). Once this validation is complete, the Setup class then generates disk space and random-access memory (RAM) usage estimates which Snakemake needs for some tasks when they are executed in a cluster environment. Lastly, the `Setup` class executes the generation of the workflow file and stores all relevant files in the user-defined project directory.

2.1.2 Workflow Generation

Each task in the workflow consists of a class which inherits its methods from the `BaseRule` superclass (note that tasks are called *rules* in Snakemake, hence the terminology). Each task object is instantiated with a copy of the configuration file and contains several default properties: the `is_active` property returns *True* if the task is being used in the current configuration of the workflow; and the `input`, `output`, `log`, `benchmark`, `threads`, `resources`, `params`, `run`, and `shell` properties are each rule-contained parameter lists in the Snakefile syntax. These parameter lists are expected to return either a dictionary (if used) or a `None` object (if unused, i.e., an empty value).

Each task class must start with the `Rule` prefix as this is used to collect all available rules when building the workflow. When inheriting a method from the superclass, its contents can be overwritten to allow each task to handle its parameters differently when generating the output dictionary. Various helper functions exist to assist in this, such as helpers for extracting values from the configuration file and wrappers for formatting output such that it can be interpreted by Snakemake. The `RuleAll` class is special, as it is always included as the top-most task in the workflow. Snakemake uses this task as a starting point for generating a directed acyclic graph, thus it must contain all the final output the workflow will generate. An understanding of this structure is necessary to create, modify, or expand workflow tasks (see Sect. 2.10 [p. 70]).

The `build_workflow` utility function evaluates the `is_active` property of each task and parses the properties of active tasks into the Snakefile. This file is saved to disk as 'Snakefile' (no extension), as this is the default file name for a Snakefile. Other file names are possible, although they then must be referenced by name when executing Snakemake.

New workflow tasks can be added as a class inheriting the `BaseRule` superclass. The `RuleAll` class should only be modified if the task generates new output that no other tasks rely upon, as Snakemake will not execute tasks for which there is no expected output. No other tasks nor any workflow building utilities have to be modified. The processing code for a task can be added as a function to the `tasks` module (see next section). It is referenced in the `run` or `shell` parameter list. Tasks are structured such that extending the workflow is simple and does not require changing existing functionality.

2.1.3 Tasks module

Each task is associated either with a shell command or a Python function that receives input and generates output (defined as a parameter list, as described in Sect. 2.1.2 [p. 41]). The Python functions are part of the *tasks* module, which contains a function for each relevant task. The arguments to the function are passed from the task's `input`, `output`, `params`, and `log` methods as dictionaries. No function in the *tasks* module has a return value. Instead, they each create or copy one or more files to a location derived from the `output` dictionary. The functions are called by Snakemake when the relevant rule is executed.

2.2 Data Structure

The input data can be divided into modality-agnostic and modality-specific categories. Modality-agnostic input data includes (1) a binary 3-dimensional NIfTI ROI file in the 3-dimensional NIfTI image data format, (2) an optional 3-dimensional target mask in the same data format, used to define the connections that are considered for each ROI voxel. If not provided by the user, the FSL distributed average MNI152 T1 whole-brain grey matter group template (2mm isotropic) will be used as the target in which case the required input data should match the same MNI152 template as well, and (3) a participants file as a tab-separated text file with a column called `participant_id` containing all unique identifiers of the subjects to be included in the study.

Modality-specific data depends on the selected input modality, i.e., *rsfMRI*, *dMRI*, or *connectivity*. For rsfMRI data, a 4-dimensional time-series NIfTI image per subject must be provided, optionally accompanied by a tab-separated text file containing confounds for each time point as columns (see Appx. 3 for an example confounds table). The dMRI modality requires input necessary to perform FSL's probtrackx2, consisting of: (1) outputs from bedpostx, (2) a brain extraction (BET) binary mask file, (3) a transform file taking seed space to DTI space (either a FLIR matrix or FNIR warpfield; optional), and (4) a file describing the transformation from DTI space to seed space (optional unless input file 3 is defined). Each of these files is subject-specific and can be obtained from FSL's bedpostx output. Connectivity matrices may be provided as source input in lieu of rsfMRI or dMRI data. They must be provided in an ROI-voxel by target-voxel shape, along with a binary 3-dimensional mask of the ROI in NIfTI image data format, and a NumPy array of voxel coordinates in the order that the ROI voxels are represented in the connectivity matrix.

To define input data for the rCBP procedure, the full file paths must be added to the configuration file. CBPtools offers example configuration files using the `cbptools example –get data-modality` command, where datatype is replaced by either *connectivity*, *rsfmri*, or *dmri*, reflecting the different input data modalities. The absolute file path for subject-wise files should be specified as a template, i.e., containing the string `{participant_id}` which will be replaced by the ids of the subjects included in the rCBP project (through the inclusion of the aforementioned participants file). All input data should be quality controlled prior to using CBPtools, as only marginal validation is performed on the input data. Faulty data may halt processing until the issues are resolved, but in the worst

case such data may provide output without explicit warnings that this output should not be trusted. Further specified during the setup are parameters to transform the connectivity matrices (e.g., cubic or Fisher's Z transform, or feature reduction through principal component analysis), the clustering parameters (e.g., the range of k clusters requested) and validity measures, as well as the desired output file formats. Each of these parameters are likewise specified in the configuration file.

2.3 Dependencies

Python (version 3.5+) is required to install CBPtools. Aside from FSL's probtrackx, all dependencies are installed through the CBPtools `setup.py` file. The following Python packages are required for CBPtools to work: matplotlib (3.0.3), nibabel (>=2.5.0), numpy (>=1.17.0), pandas (>=0.25.0), pyyaml (>=5.1.1), scikit-learn (>=0.21.3), scipy (>=1.3.0), seaborn (>=0.9.0), and snakemake (>=5.5.4). Both matplotlib and seaborn are used in tandem to generate various plots as output of the workflow. Nibabel is used for loading and handling NIfTI images, including transformations. Numpy is used for performing various multi-dimensional array manipulations (e.g., computing the connectivity matrices and modifying NIfTI image data). Pandas is used for loading tab-separated files and handling tabular data, and it conveniently interfaces with seaborn when generating plots. Pyyaml is used for loading YAML files and converting them to Python (nested) dictionaries, primarily used for interpreting the configuration file and validation schema. Scikit-learn implements various clustering and cluster validity algorithms, complemented by the large library of scientific functions available from scipy. Lastly, snakemake is the workflow manager that makes parallel processing and management of resources possible when executing the CBPtools workflow.

FSL must be installed manually, as it is not available from the default repositories and requires Python 2 to be installed. This dependency is only required if the dMRI modality will be used, in which case it is recommended to use a UNIX-based operating system (e.g., Linux or OS X). For such systems FSL can be installed from the Neurodebian (Halchenko and Hanke 2012) repository. On systems running Windows, the installation requires further steps as it either uses the Windows Subsystem for Linux (Windows 10 only) or a virtual machine

to emulate Linux. The FSL documentation has detailed instructions on its installation[1].

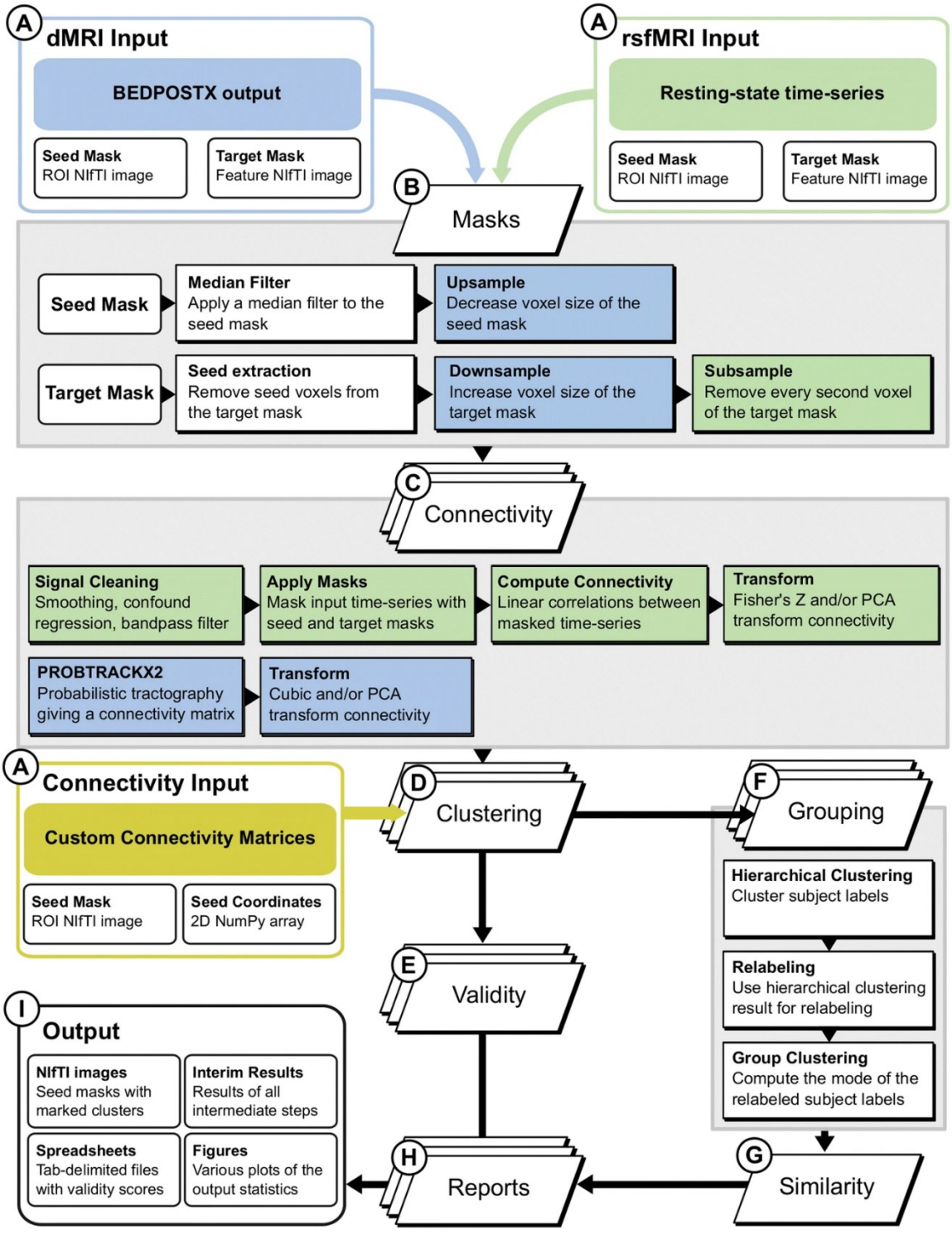


Figure 5 CBPtools workflow diagram. The rCBP procedure can be applied to dMRI (blue) or rsfMRI (green) data separately. Steps marked with multiple boxes are executed in parallel, and

[1] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FslInstallation

colours indicate procedures only applied to input data of the same colour. White boxes are applied regardless of the type of input data.

2.4 Workflow

Fig. 5 outlines the CBPtools workflow for applying the rCBP procedure to rsfMRI, dMRI, or connectivity matrix data. After customizing the parameters of the procedure, input data (**A**) is processed through each step (**B** through **H**) of the workflow, culminating in the output (**I**). Note that the different types of input are not processed in parallel but must instead be set up and executed as different CBPtools projects. Each has the same section key (**A**) to highlight the different types of input data CBPtools supports, and at what stage of the processing the input data is used.

Different types of input may require different processing steps. These steps are marked in different colours (i.e., green for rsfMRI, blue for dMRI, and yellow for connectivity-matrix input data). Steps and procedures with a white background apply to any input data. A quick-start guide containing example data is provided in Appx. 9 as well as on the GitHub project page and online documentation[1].

2.5 Getting Started

The following sections outline how to install, configure, and execute the CBPtools pipeline. A quick start guide is available in the online documentation, which explains how to perform rCBP using CBPtools on a data set in four simple steps.

2.5.1 Installation

CBPtools requires a Python3 (>=3.5) installation. All its dependencies will be installed except for the probtrackx2 tool from the FSL library. This tool is necessary to perform probabilistic tractography on diffusion-weighted imaging data. Hence, if no dMRI data is used then probtrackx2 is not necessary. To see whether probtrackx2 is installed and accessible within the installation environment, the terminal command below can be used. If it is not available, the FSL installation manual[2] can be consulted to install it.

```
$  probtrackx2 --help
```

[1] https://cbptools.readthedocs.io/en/latest/overview/quickstart.html
[2] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FslInstallation

We recommend installing CBPtools within a virtual environment[1]. Any packages installed within a virtual environment will not influence or be influenced by packages outside of it. Hence, versions can be preserved within various analysis environments. It can easily be set up using the venv tool which is packaged with Python3. The command below will create a virtual environment called `cbptools` in the `.venv` directory in the home folder. The second line activates the environment. Although recommended, this is entirely optional.

```
$  python3 -m venv ~/.venv/cbptools
$  source ~/.venv/cbptools/bin/activate
```

CBPtools can be installed from the *Python Package Index* (PyPi) using the pip package installer which, like venv, is packaged with Python3.

```
$  pip install cbptools
```

Alternatively, the development version can be installed directly from *GitHub*.

```
$  pip install git+https://github.com/inm7/cbptools
```

CBPtools and its dependencies will now be installed.

2.5.2 Setup

The first step to creating a CBPtools project is writing a configuration file. This file contains the links to the data set that is to be used, as well as the parameters for the tasks that will process the data. An example configuration file can be obtained which will have a set of parameter and data keys containing default and placeholder values in the YAML format. Note that before use the placeholder data values must be changed to correctly point to the input data. Certain required parameters do not have default values (i.e., the range of clusters to obtain from the parcellation procedure) and must therefore be entered as well. An example configuration file with the requested input modality can be obtained by issuing the command below.

```
$  cbptools example --get modality
```

By changing the 'modality' argument to either 'rsfmri', 'dmri', or 'connectivity', an example file with parameters matching that input data type is

[1] https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/

created. Not all parameter and data keys are represented in the example configuration file. For a complete list of permitted keys, see Appx. 4.

There are three top-level keys in the file: *modality*, *data*, and *parameters*.

- Modality: CBPtools will expected this type of input data and handle validation and setup accordingly. It currently accepts the supported data modalities: 'rsfMRI', 'dMRI', and 'connectivity' (**Fig. 5A**).
- Data: Within this key all external inputs to the workflow are defined. Which files are expected depends crucially on the modality key. Input data is described in more detail in Sect. 2.2 [p. 43].
- Parameters: A key that contains all the parameters for rCBP processing as outlined in the workflow. All available parameters are described in more detail in Appx. 4.

When both the input data and the configuration file are properly defined (and quality controlled for the former), the setup procedure can be started to create a CBPtools project folder.

```
$ cbptools create --config /path/to/config.yaml --workdir
/path/to/workdir
```

The `--config` (alternative `-c`) parameter is used to define the path to the configuration file. The `--workdir` (alternative `-w`) parameter is used to assign the directory in which the project files will be placed. This command will immediately run the validation of the configuration file and input data. Upon success, a project folder will be created at the `--workdir` location (i.e., `/path/to/workdir` in this case). Note that if the `--workdir` already exists and contains files, the `--force` argument must be used to force the setup procedure into using that folder (and potentially overwriting existing files). This is a safety measure to ensure the user is aware, and agrees, that files may be overwritten.

2.5.3 Validation

The `create` command will validate both the configuration file and the data set. The configuration file is parsed as a YAML file and then validated against a schema containing the requirements each key and value pair must meet. If the parsing fails or necessary requirements are not met, the *create* procedure will halt without proceeding to validate the input data. Next, the input data is validated. This validation is only marginal and focuses primarily on common mistakes that

would hinder CBPtools from performing its processing. It is not recommended to rely on it for ensuring the input data contains no flaws. File paths in the configuration are automatically converted from relative to absolute paths. It is recommended to use absolute paths to allow the inclusion of the configuration file regardless of its location. This part of the validation takes longer, as each external file is validated individually. When a file belonging to a subject fails to validate, the subject will automatically be excluded from the project and the user will be warned.

Once validation finishes (be it successfully or not) a log file is provided either at the current location (in case of failure) or inside the project directory (in case of success). The log file contains any error and warning given during the validation, as well as information on the interpretation of the input data (e.g., which nuisance signal regressors were found, if and where probtrackx2 was found, etc.). Reading the log file is strongly recommended.

2.5.4 Execution

Successful completion of the `create` command will generate a project directory at the specified `--workdir` location. From within this directory the workflow can be executed using the snakemake workflow management system, installed as a dependency for CBPtools. Snakemake uses the project's Snakefile, a workflow generated from the configuration file in a snakemake compatible format. Issuing the relevant snakemake command will start processing the data. Below several common use-cases are listed. For a more in-depth description the snakemake documentation[1] can be consulted.

```
$  snakemake -j 8 --resources mem_mb=20000
```

The command above issues a run using 8 threads and at most 20 GB of RAM. Snakemake will automatically manage the allocation of jobs based on the available resources. Commonly, the CBPtools workflow will be executed on a cluster environment. The project will have a `cluster.json` file containing all relevant cluster parameters. Generally, only the `"__default__"` field requires customization for the cluster the workflow is to be executed on. These default settings are inherited for each task and only overwritten if defined for the task within the `cluster.json` file. Note that any parameter used is provided in the `--cluster`

[1] https://snakemake.readthedocs.io/en/stable/executable.html

(alternatively `-c`) argument. The code snippet below contains an example of the `"__default__"` field.

```
{
    "__default__" :
    {
        "account" : "my account",
        "time" : "01:00:00",
        "n" : 1,
        "N" : 1,
        "c" : 1,
        "partition" : "core",
        "out" : "log/{rule}-%j.out",
        "name" : "unknown",
        "mem" : "1000M"
    }
}
```

Note that specific fields, such as `"mem"` (RAM usage in MB) and `"time"` (maximum time a job can run) are overwritten by task-specific fields. For example, the connectivity computation may require a higher `"mem"` allocation depending on the size of the input data. The example command given below uses the `cluster.json` contents for executing the workflow on a cluster using the SLURM scheduler. The `cluster.json` contents are used to fill the wildcards given for the cluster argument (i.e., `{cluster.partition}` becomes `core` using the above default example).

```
$ snakemake -j 999 -w 240 -u cluster.json --resources mem_mb=20000 -c
  "sbatch -p {cluster.partition} -n {cluster.n} -N {cluster.N} -t
  {cluster.time} -c {cluster.c} --mem-per-cpu={cluster.mem} --
  out={cluster.out} --job-name={cluster.name}"
```

The `-c` argument contains the command Snakemake uses to start jobs through SLURM (i.e., the `sbatch` command in the above example). The `-w 240` tells snakemake to wait at least 240 seconds for files to appear on the file system. File system latency may delay the file from being visible to snakemake. If output files are not generated, snakemake will assume the job was not executed properly and halt further processing. The number of jobs `-j` is set to 999, which has snakemake issue all available jobs instantly. After all, SLURM now manages the allocation of jobs instead of snakemake and holds jobs that cannot be executed immediately in queue.

Aside from the `mem_mb` resource, CBPtools also makes use of a custom `io` resource. When several jobs that are executed in parallel attempt to access large files, they may slow one another down. Since only the connectivity task is at risk of having large files as input, each connectivity job is assigned one I/O token. That is, when `--resources io=10` is set, only 10 such jobs may run at the same

time. Whichever I/O value works best depends crucially on the file system and cluster setup.

After running one of these commands the processing will have started. The coming sections detail the steps of the processing pipeline.

2.6 Processing

Here all processing tasks are outlined in a step-by-step fashion. The order in which processing options are mentioned is the order in which they are processed by CBPtools. For a detailed overview of all the in- and output for each task, consult Appx. 5.

2.6.1 Masking

The masking task will generate and/or pre-process the seed and target masks. If a `region_id` (`data: masks: region_id`) is specified, the given seed mask will be treated as an atlas. The `region_id` can be a singular integer or a list of integers. Each voxel within the atlas that has a value occurring in the list of region-ids is used to construct a (composite) binary seed mask. If no target mask is given, the default MNI 152 T1 whole-brain grey matter group template (2 mm isotropic) is used as a target mask. Next, both seed and target masks are binarized (if necessary). If no binarization threshold is set, a default threshold of 0 will be used.

Optionally, for each selected voxel within the binary seed mask, median filtering reassesses its selection based on its neighbourhood. The spatial nearest neighbours are taken for each voxel (resulting in a 3x3x3 matrix of the selected voxel and its neighbours) and the median selection value (i.e., median of all values in the matrix) is assigned as the new selection value for this voxel. An in-depth explanation of median filtering is given in Sect. 2.7.1 [p. 60].

Next, seed voxels are optionally removed from the target mask (i.e., have their values set to 0 indicating they are no longer part of the mask; see **Fig. 6**). This procedure can also optionally remove a border around the seed region to reduce the influence of smoothing. Application of this method ensures that ROI to ROI (i.e., within-ROI) connectivity is ignored. Within-ROI connectivity (i.e., the connectivity between every pair of voxels within the seed mask) tends to be high due to their relative proximity to one another and may therefore dominate

the clustering. Whether doing so leads to better or more biologically relevant parcellation results, however, is unclear.

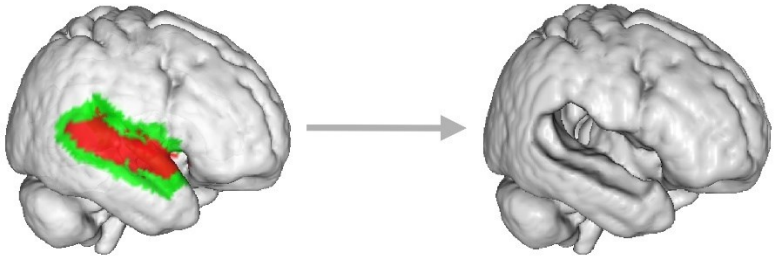


Figure 6 Example of extracting seed voxels from a target mask. The ROI (red area) and a 5 mm border surrounding the ROI (green area) are extracted from a whole-brain grey matter mask

The target mask is now optionally subsampled (see **Fig. 7**). This option is only available to the rsfMRI modality and is recommended when smoothed BOLD time-series are used. It processes the target mask in such a manner that only ever second voxel in each dimension is kept under the spatial-smoothness assumption that neighbouring voxels provide a relatively similar signal. This can significantly reduce computation time while preserving most of the information due to spatial smoothness.

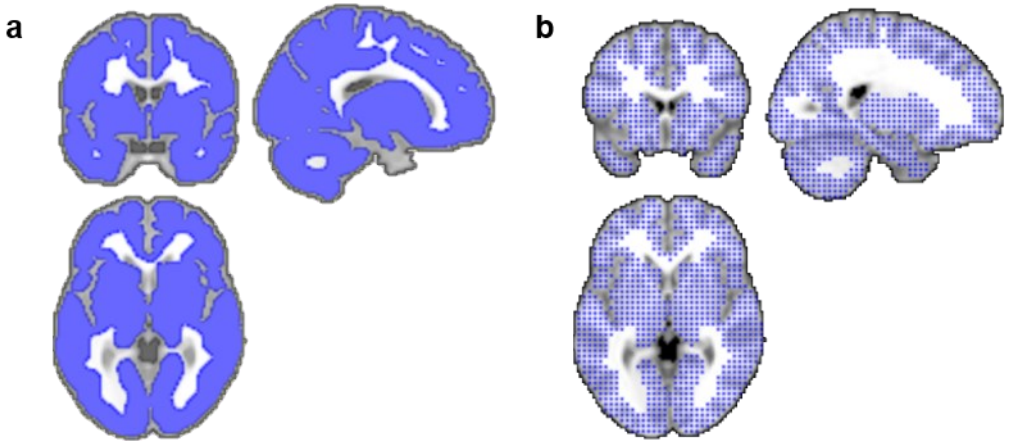


Figure 7 Subsampling example. **a** whole-brain grey matter mask without subsampling applied. **b** whole-brain grey matter mask with subsampling applied.

When the dMRI modality is used, the seed and target masks can optionally be up- and downsampled, respectively. This upsampling option spreads the seed voxels to cover a larger area (reflecting a higher resolution for use with probtrackx2), while maintaining the same number of voxels (which is necessary so that ROI voxels can be mapped back upon the original ROI mask). Thus,

voxels within the upsampled seed mask will be spread out equidistantly over a larger area with no direct neighbouring voxels as a result of not increasing their amount. The target-mask can be downsampled from a higher to a lower resolution, resulting in fewer voxels covering the same space (i.e., larger voxels) which can reduce computation time for probtrackx2.

Lastly, the x, y, and z indices (i.e., coordinates in voxel space) are taken from all seed voxels in C-contiguous order. This results in a 2-dimensional NumPy array (stored as `.npy`) of shape n by 3, where n is the number of voxels in the seed mask. The indexing order is the same order of the seed voxels in the connectivity matrices generated by CBPtools and is used for mapping the cluster labels onto the seed mask.

2.6.2 rsfMRI Connectivity

The time-series are optionally smoothed using the nibabel package (`nibabel.processing.smooth_image`) and a FWHM value in mm over which to smooth. CBPtools uses the default smoothing mode, `nearest`, as it is the recommended choice for smoothing[1]. The seed and target masks are then applied separately to the time-series, resulting in a seed-masked time-series and target-masked time-series matrix (i.e., seed voxels by timepoints, and target voxels by timepoints, respectively). Next, the variance for each masked voxel is calculated and voxels with a variance below tolerance (defined as `numpy.finfo(np.float32).eps`, the smallest representable positive number) are marked as low-variance voxels. Since the calculation to obtain the connectivity matrix includes a division by the standard deviation, low-variance voxels will return `inf` or `NaN` values. These values will be set to 0. The low-variance error thresholds defined in the configuration file are used to check whether there are too many low-variance voxels in the seed- or target-masked time-series. If this is the case, the connectivity computation is aborted, and an empty connectivity matrix is returned. This act is logged, and at a later stage (the stage in which individual results are grouped, see **Fig. 5F**) in the workflow processing will halt and provide a more detailed error log.

Next, if a confounds file is provided the selected columns (all columns if none are selected) will be linearly regressed out of the time-series signal for both the

[1] https://nipy.org/nibabel/reference/nibabel.processing.html#nibabel.processing.smooth_image

seed-masked and target-masked time-series. The code snippet below shows how this is applied using the NumPy package.

```
import numpy as np
time_series = time_series - np.dot(confounds, np.linalg.lstsq(confounds, data, rcond=-1)[0])
```

A fast-Fourier transform is then optionally applied on the seed- and target-masked time-series separately, using the defined filtering band and repetition time.

The seed-based correlation is computed using the seed- and target-masked time-series and a delta degrees of freedom (ddof) of 0, resulting in a connectivity matrix. The code snippet below shows how the connectivity matrix is computed using the NumPy package.

```
import numpy as np

# Standardization
x, y = map(lambda z: (z - np.mean(z, axis=0)) / np.std(z, axis=0, ddof=0), (x, y)
)

# Correlation
r = (y.T.dot(x) / x.shape[0]).T.astype(np.float32)
```

All resulting values that are `inf` or `NaN` are set to 0. All values at or above 1 are set slightly lower than 1, and all values at or below -1 are set slightly higher than -1. This accommodates the subsequent (optional) Fisher's Z (Arctanh) transform, which would otherwise return `inf` for values at 1 or -1.

If multi-session input data is used, then for each subject a connectivity matrix per session will be computed. These matrices are averaged, resulting in one connectivity matrix per subject. Lastly, an optional principal component analysis (PCA) transformation can be applied to the connectivity matrix using the scikit-learn package (`sklearn.decomposition.PCA`). First, the SciPy package is used for detrending (`scipy.signal.detrend`, using the `constant` detrending type). PCA then reduces the number of target features in the connectivity matrix with a user-defined numerical value (an integer larger than 1 returns that many components, whereas a float between 0 and 1 returns components explaining that percentage of variance).

2.6.3 dMRI Connectivity

Using probtrackx2, a tractography analysis is run producing sample streamlines[1]. This produces a folder with various output files, of which the seed points to target points connectivity matrix, `fdt_matrix2.dot`, is used for further processing. The file is produced as a sparse matrix in F-contiguous order which CBPtools densifies using the SciPy package (`scipy.sparse.coo_matrix` and the `.todense()` method). This matrix is now referred to as a connectivity matrix.

Tasks in the CBPtools workflow expect their inputs to be in C-contiguous order. Reordering is required since the connectivity matrix as provided by probtrackx2 is F-contiguous. This procedure is performed using the `cbptools.image.get_f2c_order` method, which provides reordering indices using the seed mask such that an F extraction order is turned into a C extraction order. This new order is applied to the x-axis (seed voxels) of the connectivity matrix, but not the y-axis (as the target value ordering is not used for the remaining procedures in the CBPtools workflow).

As was done for rsfMRI connectivity, if multi-session input data is used, now all connectivity matrices for a subject will be averaged. A cubic transform is now optionally applied to the connectivity matrix using the NumPy package (`numpy.power`) with a power of 1/3. Finally, an optional PCA transform can be applied to the connectivity matrix using the same procedures as described for rsfMRI connectivity.

2.6.4 Clustering

Clustering can be separated into three different tasks, one for each of the three currently available clustering algorithms: k-means, spectral clustering, and hierarchical clustering. Each algorithm has its own set of cluster options that are defined in the configuration file and all use the connectivity matrix obtained from the previous task as input. If the modality is set to 'connectivity', then instead the user-defined connectivity matrices are used, and the workflow starts with this task. The task will run n times on each connectivity matrix, where n is the length of the range of cluster numbers requested (i.e., a configuration of `n_clusters = [2, 3, 4]` issues 3 clustering tasks per connectivity matrix).

[1] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FDT/UserGuide#ProbtrackXOutput

The k-means, spectral clustering, and agglomerative clustering algorithms are each implemented using the scikit-learn (`sklearn`) package. That is, `cluster.KMeans`, `cluster.SpectralClustering`, and `cluster.AgglomerativeClustering`, respectively, along with the cluster options as defined in the configuration file. For spectral clustering, the algorithm may fail due to a `numpy.linalg.LinAlgError` (commonly due to a too liberal stopping criterion value for eigendecomposition of the Laplacian matrix) or because the requested number of clusters was not returned. If that is the case, CBPtools will store an empty output file and create a warning in the log file. Once all clustering tasks have finished executing, further processing will halt and provide a more detailed error log. This allows the user to decide how to proceed (e.g., change the cluster options, exclude problematic subjects, or reconsider other (pre-)processing steps) on a case-by-case basis. The cluster labels obtained from the selected clustering algorithm will be given as output for this task.

Once the clusterings (and therefore also connectivity matrices) are computed for all subjects, the outputs are assessed. If at this point any of the subjects produced problematic results (i.e., the connectivity or cluster labels file is empty due to an error during processing), CBPtools will halt further processing and instead produce a log file (at `log/validate_cluster_labels.log` within the project directory) with information about the subject-ids and reason of the problematic results. The processing can be resumed manually once all problems have been addressed. If there are no problems at this point, the workflow will resume with the next tasks.

2.6.5 Grouping

This task receives all subject-wise cluster labels per requested number of clusters, k, and uses them to obtain a group consensus parcellation per k. The cluster-ids used to assign voxels to a cluster are arbitrary (i.e., they can be permuted), preventing a direct comparison between subject-wise cluster labels. To permit comparison the cluster-ids must be reassigned (i.e., relabelled) such that the most similar clusters between subjects get assigned the same cluster-id. For an illustrated explanation of the relabelling strategy, see Sect. 2.7.3 [p. 62], as the following paragraphs only describe the procedure.

First, a reference clustering is obtained by performing hierarchical clustering using the SciPy package (`spatial.distance.pdist` and `cluster.hierarchy`) with Hamming distance. Hamming distance is insensitive to cluster-id permutations as it measures the minimum number of substitutions required to change the set of

cluster labels from one subject to that of another. The pairwise Hamming distance y is calculated on the matrix x. Hierarchical clustering is then performed on x using the linkage algorithm specified in the configuration file, resulting in a linkage matrix, z. The cophenetic correlation is now calculated between z and y. The tree is cut to obtain a reference clustering with k clusters.

```
1. y = pdist(x, metric='hamming')
2. z = hierarchy.linkage(y, method=linkage, metric='hamming')
3. coph = hierarchy.cophenet(z, y)
4. reference_labels = hierarchy.cut_tree(z, n_clusters=len(np.unique(x)))
```

Next, each subject-wise set of cluster-ids is relabelled by obtaining all permutations of the cluster-ids and comparing each to the reference clustering representative to all subjects. The permutation most similar to the reference is used to reassign cluster-ids for that subject. Once all subject-wise labels have been relabelled to best match the reference clustering, they become comparable.

The most frequent assignment for each voxel is then obtained by taking the `mode` (using the SciPy `stats.mode` function) over the relabelled cluster-ids and used as the group consensus parcellation. Alternatively, the reference clustering may instead be selected as the group consensus parcellation if specified in the configuration file.

Lastly, the group consensus parcellations are mapped upon the ROI for each k and stored as a NIfTI image. The seed coordinates, created in the masking task (see Sect. 2.6.1 [p. 52]), are used to obtain the x, y, and z coordinates of each voxel in the cluster labels, as the ordering of the seed coordinates file coincides with the ordering of the cluster labels file.

2.6.6 Validity

This task uses the connectivity matrix and cluster labels for each participant to compute the requested validity metrics. The scikit-learn package (`sklearn.metrics`) is used to obtain the Silhouette index and Calinski-Harabasz index, whereas the Davies-Bouldin index is implemented in CBPtools. The requested validity metrics are each computed per subject and per k clustering granularity using the subject's connectivity matrix as a feature array, and the subject's predicted labels. Note that for the Silhouette index, the metric for calculating distance between instances in the feature array is Euclidean. The scores obtained from this task are merged into a tab-delimited file, used in the report (see Sect. 2.6.8 [p. 59]), to create plots.

2.6.7 Similarity

This task computes subject-to-subject, subject-to-group, and the (optional) reference-to-group similarity scores using the similarity metric defined in the configuration file. The scikit-learn package (`sklearn.metrics`) is used for all available metrics, consisting of: the adjusted Rand index, adjusted mutual information score, and the V measure score.

The subject-to-subject similarity matrix contains the pairwise similarity scores between the cluster labels of each subject for each cluster granularity k. Using the same method, the subject-to-group similarity matrix computes the similarity between the cluster labels of each subject and the group consensus parcellation for each cluster granularity k. If reference images are provided, the labelled voxels are extracted from the images in C-contiguous order. The similarity scores are then computed between each reference and each group consensus parcellation (for each k).

2.6.8 Report

Plots are now generated for the various statistics obtained from the clustering, grouping, validity, and similarity tasks. This part of the procedure consists of multiple small tasks, split in such a way to enable faster processing by making optimal use of the available resources.

All ROI NIfTI images obtained from the grouping task are plotted in various views (i.e., right, left, superior, inferior, posterior, and anterior views) and color-coded by cluster. The matplotlib package is used for generating 3D voxel plots and various optimizations are performed to reduce the size of the figure files when the output file type is set to vector graphics. Regardless, depending on the size of the ROI these figure files can still be relatively large. The same figures can optionally be generated for the clustering results of each subject, but this option is not enabled by default.

The validity scores created in the validity task (see Sect. 2.6.6 [p. 58]) are plotted as box plots, using the seaborn package which provides a high-level interface for matplotlib. The colour palette used for the figures is colour blind safe for the most common variant of colour blindness that leads to difficulties distinguishing reds from greens. The similarity scores generated in the similarity task (see Sect. 2.6.7 [p. 59]) are plotted as heat- and clustermaps, likewise using the seaborn package. The cophenetic correlation and relabelling accuracy,

computed during the grouping task (see Sect. 2.6.5 [p. 57]), are plotted as a line plot with the score on the y-axis and the number of clusters on the x-axis, and a box plot, respectively.

Once all plots have been generated the workflow procedure is complete. All the interim and final output can be viewed in the project folder. Note that CBPtools only deletes temporary interim data when the data is merged, ensuring that there is no loss of information.

2.7 Methods Explanations

2.7.1 Median Filtering

Median filtering is an optional procedure which can be applied to 'clean-up' artifacts in a seed mask. Sometimes, a seed mask contains small holes, single-voxel strands that protrude from the mask, or sharp borders. These artifacts usually arise in hand-drawn ROIs. Median filtering may be a useful tool to get rid of such artifacts. The reason why median filtering is particularly useful for hand-drawn masks is that they usually lack continuity in the 'depth' direction when drawing in 2D. When such artifacts are not expected, e.g., for atlas-derived ROIs, this is not a recommended option. For each selected voxel within the binary mask ROI, median filtering reassesses its selection based on its neighbourhood. To apply median filtering, the spatial nearest neighbours are taken for each voxel (resulting in a 3x3x3 matrix of the selected voxel and its neighbours) and the median selection value (i.e., median of all the values in the matrix) is assigned as the new selection value for this voxel. **Fig. 8** shows three examples of this procedure simplified to a 2-dimensional space. The first example (**Fig. 8a**) shows a selected voxel that has too few neighbours that are part of the mask. As a result, the voxel is removed from the mask by having its value set to zero (the median). The second example (**Fig. 8b**) instead shows a selected voxel that is not part of the mask but has many neighbours that are part of it. The selected voxel is added to the mask by having its value set to one. Lastly, the third example (**Fig. 8c**) shows a selected voxel that is part of the mask and has many neighbours that are likewise part of the mask. Its value remains unchanged, as the median is the same as its original value.

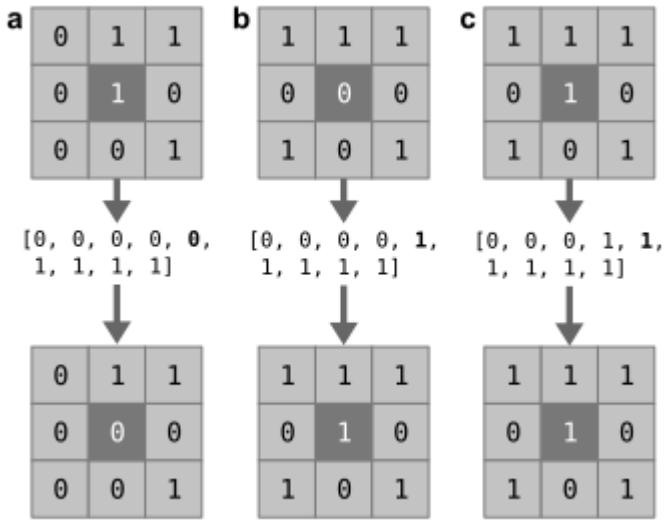


Figure 8 Median filtering example. The top row shows a selected voxel (centre) and its nearest neighbour voxels (off-centre), where 1 means the voxel is part of the mask, and 0 means it is not. The view is reduced to 2D for simplicity. The middle row shows all voxel values ordered incrementally. The median value is highlighted in bold. The bottom row shows the same as the top row after the selected voxel has its value changed with the median of itself and its neighbours. a Median filtering that results in the voxel being set to 0 (i.e., not part of the mask). b Median filtering that results in the voxel being set to 1 (i.e., part of the mask). c Median filtering that does not result in any changes (i.e., the voxel remains part of the mask)

2.7.2 Nuisance Signal Regression

Nuisance signals (i.e., confounds) can optionally be supplied as a tab-separated file per subject (see Appx. 3). This file is expected to have a header on the first row naming each column. The column names can be used to select columns to be used as nuisance regressors (note that if no columns are specified, then all columns are used). Commonly used columns may be white matter, cerebrospinal fluid, grey matter, or global signal, as well as motion regressors to correct for the effects of head motion in the scanner. If required, a linear trend and constant should be added as columns to the files. The nuisance signal removal is applied as a linear regression of the confound time-points on the time-series of the corresponding subject and retaining the residuals as the new signal.

2.7.3 Relabelling Strategy

Relabelling is applied to obtain a group-level parcellation by combining subject-level parcellations. The subject-level cluster labels (per k) are obtained by applying the k-means (or alternatively the agglomerative or spectral) clustering algorithm. This results in a cluster labelling per subject, per k. **Fig. 9a** shows an example set of labels for k = 5. The cluster-ids (represented by numbers and colours) are arbitrary (i.e., they can be permuted), yet in this example all subjects have identical clusterings (the dotted line separates the different clusters). To interpret the parcellations over a population, the parcellations must be combined into a single (group) parcellation per k by computing the most representative cluster assignment for each ROI voxel across subjects. As the cluster-ids per subject are arbitrary, they need to be reassigned such that the most similar clusters between subjects get assigned the same cluster-id. To achieve this, all permutations of the cluster-id (**Fig. 9d**) per subject are checked and the permutation-derived labels to a reference clustering representative to all subjects are compared. The permutation most similar to the reference is used to reassign cluster-ids for that subject. The reference clustering (**Fig. 9b**) is obtained by performing hierarchical clustering (**Fig. 9c**) with Hamming distance on all of the subject labels (Nguyen and Caruana 2007). Hamming distance is insensitive to cluster-id permutations as it measures the minimum number of substitutions required to change the set of cluster labels from one subject to that of another. Once all subject labels have been relabelled to best match the reference clustering, they become comparable. The most frequent assignment for each voxel is then obtained by taking the mode over the relabelled cluster-ids and used as the group level parcellation (**Fig. 9e**). Note that in **Fig. 9**, all subjects clusterings are identical for the assignment of cluster-ids. This is an ideal situation, but not a realistic one. In real-world data, it is common to see differences in relabelled results.

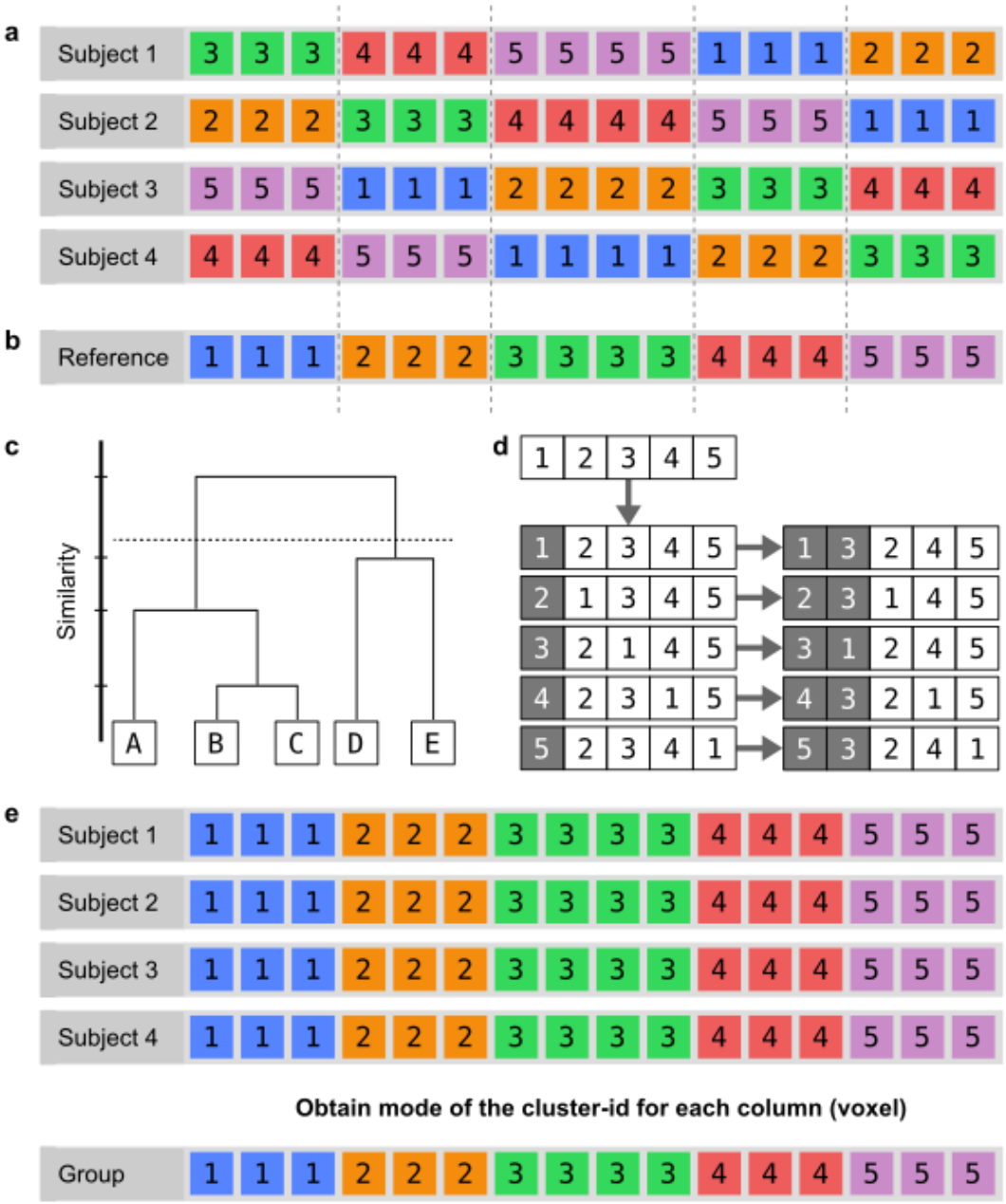


Figure 9 Relabelling strategy to obtain group cluster labels. **a** Four example label sets obtained from clustering where each colour/number represents a cluster-id that is assigned randomly. Each of the subjects has an identical clustering, yet the cluster-ids differ. **b** Reference cluster labels obtained through hierarchical clustering of the subject labels in a. **c** Illustration of hierarchical clustering performed on 5 voxels. **d** Cluster-ids are swapped for each possible permutation of the cluster-ids array. Each permutation is then tested for similarity against the reference clustering from b. **e** The cluster labels for each subject after the relabelling strategy has been applied. The mode is obtained for each voxel, resulting in the group clustering

2.8 Alternative Approaches

There are various ways to configure CBPtools either to process data differently, or to receive different outputs. The primary focus of CBPtools is to obtain group-level parcellation results stemming from various types of connectivity markers. Alternatively, users may be interested in different outputs such as those derived from multi-session data, clustering algorithms other than the default (k-means clustering) or single-subject parcellations.

2.8.1 Cluster Methods

The k-means algorithm is the default clustering algorithm in CBPtools and was used to obtain the results in this work. However, CBPtools also provides agglomerative (hierarchical) and spectral clustering as options to obtain subject-level parcellations. The method to obtain group-level parcellations remains the same irrespective of subject-level parcellation algorithm. All three available algorithms (k-means, agglomerative clustering and spectral) use the scikit-learn package (Pedregosa et al 2011). To be more precise, the implementations are the `KMeans`, `AgglomerativeClustering`, and `SpectralClustering` classes from the `sklearn.cluster` module. The parameters that can be specified for each of these algorithms can be adjusted in the configuration file. Note that not all scikit-learn provided parameters can be used, as some are not relevant for CBP or do not work in conjunction with the other steps in the workflow. Importantly, the clustering results strongly depend on the parameters used. While CBPtools has a set of default parameters, they might not be the best choice for a given ROI. For example, it is possible that the spectral clustering algorithm might not be able to properly compute a (semi-)positive definite similarity matrix when inappropriate kernel parameters are used. In this case the spectral clustering cannot proceed. For such cases, CBPtools will log the issues that occur during the clustering step, perform clustering for all possible subjects, and then halt processing until the issues are resolved. It should be noted that even if clustering succeeds, the clustering results may not necessarily reflect biologically plausible results. For instance, a 2-cluster solution may assign only one voxel to a cluster, and all other voxels to the other cluster. If this happens, the group clustering (when mode is being used as the method) may entirely remove the small cluster, resulting in a 1-cluster solution being posed as a k-cluster solution. It is therefore strongly suggested to investigate the single-subject cluster labels and not only the group-level clustering results. It is highly recommended to examine the log files to

identify any problematic runs and parameters causing them. When using the connectivity input modality with spectral clustering, it is possible to provide adjacency instead of connectivity matrices. When choosing this option, the adjacency matrices are given as input (in the configuration field `data: connectivity`) and the spectral clustering affinity must be set to precomputed (`parameters: clustering: cluster options: kernel`). This is not possible with k-means or hierarchical clustering.

2.8.2 Multi-session Data

Multi-session data (i.e., a data set with multiple runs per subject) can be processed using CBPtools by specifying sessions in the configuration file. Data for each session will be processed separately until the connectivity step (**Fig. 5c**), after which the connectivity matrices for each subject will be averaged across all the sessions. When using multi-session data, the (optional) PCA transformation will be performed after averaging the connectivity matrices, whereas the other transformations (Fisher's Z transform and cubic transform) will be applied before averaging. The averaged connectivity matrices will then be used for the remainder of the procedure.

Multi-session data can be defined in the configuration file as follows:

```
modality: rsfmri
data:
  session: [sess1, sess2, sess3]
  time_series: /path/to/data_set/{participant_id}/{session}/time_series.nii.gz
  confounds:
    columns: ['constant', 'wm.linear', 'csf.linear', 'motion-*']
    delimiter: \t
    file: /path/to/data_set/{participant_id}/{session}/confounds.tsv
  masks:
    seed: /path/to/seed_mask.nii.gz
    target: /path/to/target_mask.nii.gz
    space: standard
  …
```

Note how the `{session}` wildcard is now used to reference the data set files. Assuming two subjects with participant IDs 'sub-001' and 'sub-002', and the sessions defined as in the code-block above, the *time_series* files will become:

- */path/to/data_set/sub-001/sess1/time_series.nii.gz*
- */path/to/data_set/sub-001/sess2/time_series.nii.gz*
- */path/to/data_set/sub-001/sess3/time_series.nii.gz*
- */path/to/data_set/sub-002/sess1/time_series.nii.gz*
- */path/to/data_set/sub-002/sess2/time_series.nii.gz*

- */path/to/data_set/sub-002/sess3/time_series.nii.gz*

All wildcards can be used multiple times and do not have to indicate a folder (i.e., they can also be part of a file name). The data set structure and naming must be consistent, otherwise subjects will be marked as missing data.

2.8.3 Single-subject Parcellation

CBPtools can generate subject-specific reports (i.e., metrics and plots) in addition to the group-level clustering reports if specified in the configuration file. This is done by setting the field `parameters: report: individual_plots` to true. This option is turned off by default as it requires more computation time. CBPtools can also be used to obtain parcellations in the subject's native space by setting the `data: masks: space` field to 'native'. In this case all the data, i.e., the input masks and the fMRI and dMRI data, should be in the native space. Furthermore, the target mask is no longer optional, as the default target mask is in a common reference space rather than native space. Note that group-level parcellations cannot be computed in this scenario. This is because CBPtools does not perform any transformations to bring native data into a common reference space. As a result, only the subject-level parcellations can be computed. When subject-specific input masks are provided, CBPtools will generate all figures for each individual subject, rather than generate output at the group level. Steps F and G in **Fig. 5** are skipped, and step H will provide different output. Note that in any case, seed and target masks must always be in the same space.

2.8.4 Automatically deriving the seed mask from an atlas

To accommodate the use of atlases, an atlas can be provided as a seed mask (i.e., in the `data: masks: seed` field) instead of a binary seed mask. When doing so, the atlas must be a NIfTI image using integers as region-ids, and the region(s) to be used as a seed mask must be specified as one or more region-ids (integers) in the configuration file. A composite binary mask of the specified region(s) will be generated. Since CBPtools does not perform any image warping, the atlas must be in the same space as the input data (i.e., the time-series for the rsfMRI modality, and the bedpostx output for the dMRI modality).

2.8.5 Using reference images

To allow direct comparisons between the CBPtools group-level cluster solutions and a priori parcellations (e.g., the cytoarchitectonically defined preSMA-SMA subdivision as used in this work) one or more reference images can be provided. These images must be in the same space as the seed mask, covering the same voxels, and have at least two clusters. A similarity score (either Cramer's V measure, ARI, or adjusted mutual information score) will be computed between each reference image and each group clustering solution. The output will be provided as a tab-separated file, as well as a heatmap. An example of the latter is shown in **Fig. 10**.

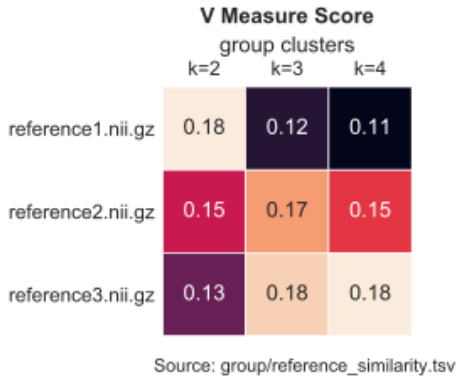


Figure 10 Reference similarity heatmap example. Three different group clustering results (k = [2, 3, 4]) are compared to three different reference NIfTI images using dummy data.

2.8.6 Running multiple CBPtools projects in parallel

Clustering of input data from the rsfMRI and dMRI modalities are run separately (i.e., no multi-modal solution is provided), and the various configuration options can be different between the modalities. CBPtools cannot create a project that processes both rsfMRI and dMRI data within a single workflow. However, it is possible to create multiple projects (using different configuration files) to subsequently run them in parallel by executing snakemake once per project. On a system that is not managed by a scheduler (e.g., SLURM, HTCondor, etc.) it is recommended to specify the number of jobs and amount of memory each snakemake instance can use. If a scheduler is being used, this is not necessary. For example, on a system with 8 threads and 30 GB of memory the following approach can be used to initialize two CBPtools projects (with each having access to half of the available resources):

```
$  cbptools create -c config_r_presma -sma_rsfmri. yaml -w r_presma -sma_rsfmri
```

```
$   cbptools create -c config_r_amygdala_dmri.yaml -w r_amygdala_dmri
```

Two terminal windows can then be opened, to run the following commands (one in each terminal):

```
$   cd r_presma-sma_rsfmri
$   snakemake -j 4 --resources mem_mb =15000
```

and

```
$   cd r_amygdala_dmri
$   snakemake -j 4 --resources mem_mb =15000
```

2.9 Benchmarks

To assess the performance of CBPtools on a cluster, benchmarks were obtained using snakemake's (Köster and Rahmann 2012) benchmarking utilities. Snakemake uses the psutil package to obtain the values for various benchmarking metrics. rsfMRI and dMRI workflows were benchmarked for the preSMA-SMA ROI parcellation as described in Ch. 3 [p. 78]. The benchmarks were performed on a system with 30 total available threads and 100GB total available RAM. The data set was stored on a shared remote storage server, hence competing for file read and write speed (I/O) may have increased the duration of each individual task. The entire procedure for the rsfMRI data took 1 day, 14 hours, 56 minutes, and 1 second. For the dMRI data, it took 3 days, 23 hours, 42 minutes, and 23 seconds. Tables 1 and 2 show the total task duration for the rsfMRI and dMRI data, respectively. Tables 15, 16, 17, 18, and 19 in Appx. 6 show the maximum resident set size (RSS), maximum virtual memory size (VMS), maximum unique set size (USS), maximum proportional set size (PSS), and maximum CPU load in seconds for regional CBP of the preSMA-SMA ROI performed on rsfMRI data. Tables 20, 21, 22, 23, and 24 in Appx. 6 show the same for the dMRI data. The RSS, VMS, PSS, and USS refer to the memory usage during the execution of a task, with each metric providing a different measurement.

Due to overlap between metrics it is not correct to sum them, instead they should be assessed individually. USS is likely the most representative metric for determining how much memory is actually being used by a process (Rodola 2019). Note that the reported benchmarks are per run of a task, where each task ran several times equal to the n jobs column. For example, the connectivity task ran 300 times (once per subject), whereas the k-means clustering task ran 1200 times (4 times per connectivity matrix, once for each requested value of k). Other

metrics, such as I/O in, and I/O out were reported by Snakemake but not included here for the sake of brevity. All reported values are in megabytes (MB), except duration which is in the `hours:minutes:seconds` format. Since benchmarking is recorded in seconds, tasks that took less than half a second are rounded to 0 seconds. Accurately testing runtime and CPU usage is difficult as it depends on many factors that can differ strongly between systems and configuration settings. For example, the size of the input data and ROI, as well as using local or shared computational resources may influence any of the reported metrics. With data sets as large as the HCP data, users will often be limited to using shared resources when processing this data with CBPtools. Total compute time for the entire procedure may therefore vary considerably even between runs on the same system. Thus, all reported benchmarks should only be interpreted as a loose guideline of what can be expected from this software.

Table 1. Duration per task for the rsfMRI preSMA-SMA parcellation

task	mean	std	min	max	n jobs
process masks	00:00:00	-	-	-	1
connectivity	00:07:51	00:02:47	00:02:55	00:13:25	300
kmeans clustering	00:03:07	00:00:52	00:01:25	00:05:14	1200
internal validity	00:00:06	00:00:00	00:00:05	00:00:07	300
group-level clustering	00:00:01	00:00:01	00:00:00	00:00:04	4
group similarity	00:00:07	-	-	-	1
plot individual similarity	00:00:02	00:00:00	00:00:01	00:00:02	4
plot group similarity	00:00:00	-	-	-	1
plot labelled ROI	00:00:04	00:00:00	00:00:03	00:00:04	24
plot internal validity	00:00:00	00:00:00	00:00:00	00:00:00	3

Table 2. Duration per task for the dMRI preSMA-SMA parcellation

task	mean	std	min	max	n jobs
process masks	00:00:00	-	-	-	1
connectivity	00:01:24	00:00:19	00:00:34	00:02:41	300
kmeans clustering	00:07:50	00:02:10	00:03:49	00:12:25	1200
internal validity	00:00:11	00:00:00	00:00:10	00:00:11	300
group-level clustering	00:00:01	00:00:01	00:00:00	00:00:04	4
group similarity	00:00:08	-	-	-	1
plot individual similarity	00:00:02	00:00:00	00:00:01	00:00:02	4
plot group similarity	00:00:00	-	-	-	1

plot labelled ROI	00:00:04	00:00:00	00:00:03	00:00:04	24
plot internal validity	00:00:00	00:00:00	00:00:00	00:00:00	3

2.10 Extending CBPtools

It is possible to extend CBPtools with new processing tasks or to modify existing tasks. The codebase is publicly available, hence the option to modify or supplement the software can be done with relative ease.

2.10.1 Modifying the Repository

The CBPtools repository on the INM7 GitHub page[1] is always a reflection of the latest CBPtools version made available through the Python Package Index (i.e., the latest version installable through pip). Future changes will be committed to a developer branch that will only be merged with the master branch when a new release is available. To contribute or create a personal version of the software this repository can be forked. A fork is a copy of a repository, which can be modified or extended without influencing the original. The forked repository can be cloned to obtain a local copy, which can also be installed and used as is (e.g., using `pip install -e .` in its root directory). It is furthermore recommended to create a virtual environment for this install, separating it from any existing CBPtools installations. **Fig. 11** outlines how to both fork and clone the CBPtools repository. Note that forking (but not cloning) requires a GitHub account, and cloning requires a local install of the git software.

The clone (i.e., local copy) can be modified and tested locally without influencing any existing CBPtools installations. Any changes made can be pushed (i.e., uploaded) to the forked repository. If the clone was made from the fork, then the remote (i.e., the location of the fork) will already be defined, hence a regular push (using `git push origin master`) will upload all changes and make them available on the forked repository. If public, others can then see and contribute to the fork as well.

Another function of forks is to propose changes to the original repository. Any changes made to a fork can be submitted as a pull request (PR), which, if

[1] https://github.com/inm7/cbptools

accepted, will be merged with the original repository. This effectively extends CBPtools. Aside from bug fixes, any future additions to CBPtools will be made in the form of PRs to ensure a proper history is available and any contributor can comment on future changes or propose changes to the PR.

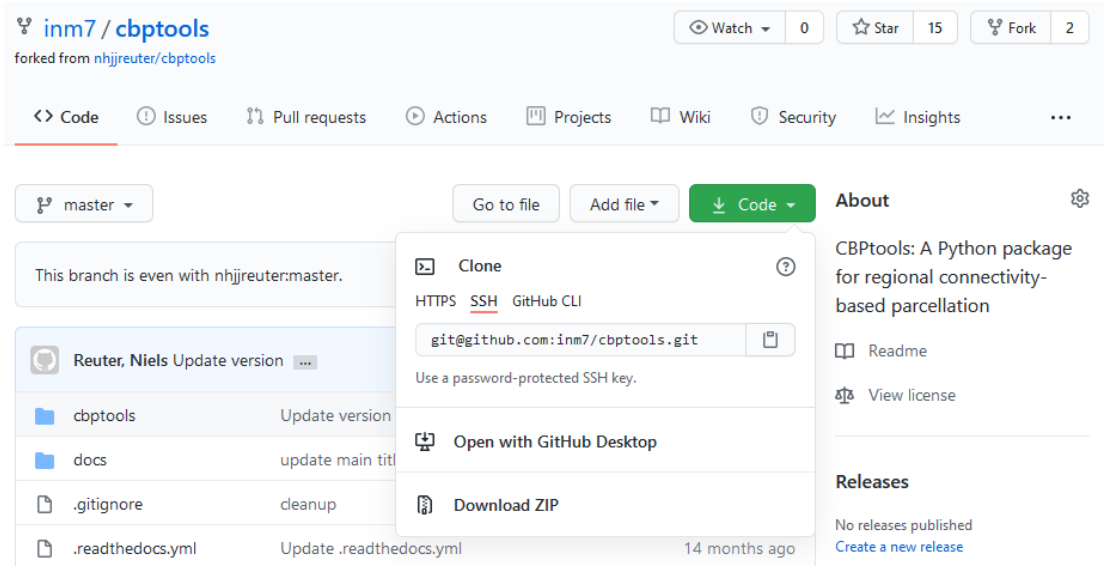


Figure 11 Forking and cloning the CBPtools repository. This website can be reached at https://github.com/inm7/cbptools. After logging in to GitHub, the CBPtools repository can be forked by pressing the "Fork" button in the top right corner. It can be cloned (ideally from the fork) by pressing the green "code" button, and using the SSH address by issuing the `git clone` command from a local terminal using the listed SSH address

2.10.2 Extending a Task

Procedures in CBPtools are contained in the *tasks* module (see Sect. 2.1.3 [p. 42]). Any Python function that is to be executed as part of the pipeline and generates output files from input files can be written as a task. The various tasks employed by CBPtools are outlined in Sect. 2.6 [p. 52].

The location of the *tasks* module in the CBPtools repository is `cbptools/cbptools/tasks/`. Each Python file in this directory represents a group of functions belonging to a particular category (i.e., `clustering.py` contains the functions for applying a clustering algorithm, cluster validation, group-level clustering, and the merging of cluster labels). The contents of these functions can be changed to modify the processing of that particular task. For example, if a user wishes to extend the k-means clustering procedure by first applying another

function to the cluster labels, then the `kmeans_clustering` function in `clustering.py` can be edited, without the need to modify anything else.

The *workflow* module (see Sect. 2.1.2 [p. 41]) is responsible for appending tasks to the workflow. It is located at `cbptools/cbptools/workflow.py`. Each class uses the `name` property to link the relevant function in the *tasks* module. For example, the class `RuleKMeansClustering` has its name property set to "kmeans_clustering", which refers to the identically named function in `cbptools/cbptools/tasks/clustering.py`, exposed by `cbptools/cbptools/tasks/__init__.py`. The Snakemake workflow automatically adopts this name, hence there is consistency in the naming convention making it easier to modify relevant tasks. If the CBPtools pipeline is executed in verbose mode, one can see which function is being used by the workflow, including its input and output files.

2.10.3 Modifying the Workflow

Modifying the workflow is slightly more involved as opposed to modifying a task. The first step in this procedure is to create a new task in the *tasks* module. This is done by either adding a new function to an existing script (i.e., a clustering-related function to `clustering.py`) or creating a new file containing a new function. The function then needs to be added to `cbptools/cbptools/tasks/__init__.py` to expose it to the other modules in CBPtools. This is done by importing the function, and then adding its name to the `__all__` variable.

Next, the task must be attached to the workflow. This is done by adding a new "rule" class to `cbptools/workflow.py`. The class must be prefixed by 'Rule'. For example, if the user wants to extend CBPtools by adding the Ward clustering algorithm, then an appropriate class name would be `RuleWardClustering`. The class must inherit from the `BaseRule` superclass. This ensures that, even if not explicitly defined, the class possesses all methods required to be imported into the workflow. The `name` property is then set to the name of the task function, for example `ward_clustering`. The class consists of various predefined methods, each with a particular purpose for workflow integration.

The `is_active` method assists the workflow generation code in determining whether the task should be added to the workflow or not. For example, if the task should only be executed when a particular configuration parameter is used,

it would be evaluated in this method. The method always returns a boolean; `False` for when the task is to be ignored, `True` when the task should be included.

The `input` and `output` methods define which input is expected and where to find it, and which output is to be expected and where it should be stored, respectively. The `params` method is used to read configuration parameters. All three methods return a dictionary object that is passed to the task, containing the input file paths, output file paths, and parameter values, respectively. This is how configuration parameters as well as file paths used by the workflow manager are made accessible to the task function.

The `log` and `benchmark` methods are optional and define where to store log files or benchmark results, respectively. The `cluster_json` method assists in dynamically generating a cluster file that can be used when initiating the workflow (see Sect. 2.5.4 [p. 50]). The `threads` and `resources` methods define how many threads and resources are going to be used by the task. For threads, an integer declaring how many threads will be used is passed, as snakemake allows for parallelization within a task. However, as of this writing no CBPtools task makes use of this. In resources the memory usage can be defined in megabytes as 'mem_mb', and the I/O tokens can be defined as 'io' (see Sect. 2.5.4 [p. 50]). Lastly, the `run` method returns the location within CBPtools of the newly created task with all necessary arguments (i.e., `tasks.ward_clustering(input, output, params, …)`). This method uses the `name` property to derive the task's name. Alternatively, it is possible to use the `shell` method to use a shell script instead of Python code.

Once this is all set up, the new task will automatically be added to the workflow, provided its activation conditions are met. If the task generates output that is not used by any subsequent tasks, then the output files should be added to the `RuleAll` class. Without this, the directed acyclic graph generated by Snakemake will exclude the newly created function, as its output is not deemed necessary.

2.10.4 Extending the Configuration

When more customizability is required on the side of the end user, more parameters can be added to the configuration file. This is particularly useful if a new task gets added that requires new parameters to function. The `Validator` class evaluates the configuration file using these new parameters (see Sect. 2.1.1 [p. 39]).

The configuration file is divided into three sections: input modality (under the "modality" key), input data (under the "data" key), and processing parameters (under the "parameters" key). These keys are also used in the configuration file itself, for which an example can be generated. In the below code snippet, the parameter is named "binarization". All keys nested within the parameter are predefined rules that perform a particular evaluation.

```
1.  binarization:
2.    type: float
3.    min: 0.0
4.    default: 0.0
5.    dependency: [modality: [rsfMRI, dmri]]
6.    required: false
7.    desc: "Threshold above which voxels in the ROI mask image are ..."
8.    custom: [has_sessions]
```

The rules *required*, *type*, *contains*, *allowed*, *max*, *min*, *maxlength*, and *minlength* are outlined in Appx. 3. The *default* rule contains the default value if no value is entered. The parameter is ignored (i.e., not evaluated) if its *dependency* is not met. In the above code snippet, the modality must be either 'rsfMRI' or 'dMRI' for this parameter to be evaluated. Lastly, the *desc* key contains a description of this parameter which will be added to the CBPtools documentation automatically.

It is possible to make a custom rule for evaluation, which can be attached to the parameter as a value of the *custom* key, as depicted in the code snippet above. Such rules must be prefixed by `_rule_custom_`. The code snippet above, showing "has_sessions" as a custom rule, triggers a custom function defined in `cbptools/validation.py` called `_rule_custom_has_sessions`. This may be necessary if none of the other rules sufficiently evaluates the configuration parameter. Custom functions for rule evaluation should always return a boolean (`True` for when the rule is passed successfully, and `False` if the rule has failed to evaluate). The addition of a new custom evaluation rule is automatically detected by CBPtools, so no further references have to be added anywhere. Custom rules have access to the current class object (through the `self` argument) and the configuration values as entered by the user (through the `this` argument).

Once a custom configuration option is made, it will automatically be evaluated when CBPtools is used next. Furthermore, rebuilding the documentation automatically adds its description to the relevant documentation page.

Part Three

Empirical Work

3	Example Data	78
3.1	Data and pre-processing	78
3.2	Analysis Procedure	80
3.3	SMA Parcellation	83
3.4	Insula Parcellation	86
3.5	Amygdala Parcellation	90
4	Outlier Detection	98
4.1	Approach	98
4.2	Results	99
4.3	Additional Exploratory Analyses	100
4.4	Discussion	102

3 Example Data

To illustrate both the usage of CBPtools and the output it provides, the resting-state and DWI modalities of the HCP data (Van Essen et al 2013) were used to parcellate three regions that have been frequently analysed using the rCBP procedure: the right (R) Insula, R Amygdala, and an ROI comprising R presupplementary motor area (preSMA) and R supplementary motor area (SMA).

3.1 Data and pre-processing

The Right (R) preSMA and SMA, R insula, and R amygdala are prominently featured regions in CBP analyses and were therefore selected as ROIs to evaluate the CBPtools software (see **Fig. 12**). The R preSMA-SMA region (at 972 voxels), known collectively as the medial frontal cortex (MFC), was extracted using the Jülich Cytoarchitectonic Atlas (Eickhoff et al 2005; Ruan et al 2018). Originally obtained in MNI Colin space, the mask NIfTI images were warped to MNI152 space using FSL's FNIRT software for non-linear registration. Both NIfTI images were then merged into one mask for the purpose of parcellation, whereas the separated images were used for cluster validation. The R insula (546 voxels) and R amygdala (280 voxels) regions were both extracted using the FSL distributed Harvard-Oxford Atlas. No further processing on these masks was necessary, as the Harvard-Oxford Atlas is already in the MNI152 common reference space.

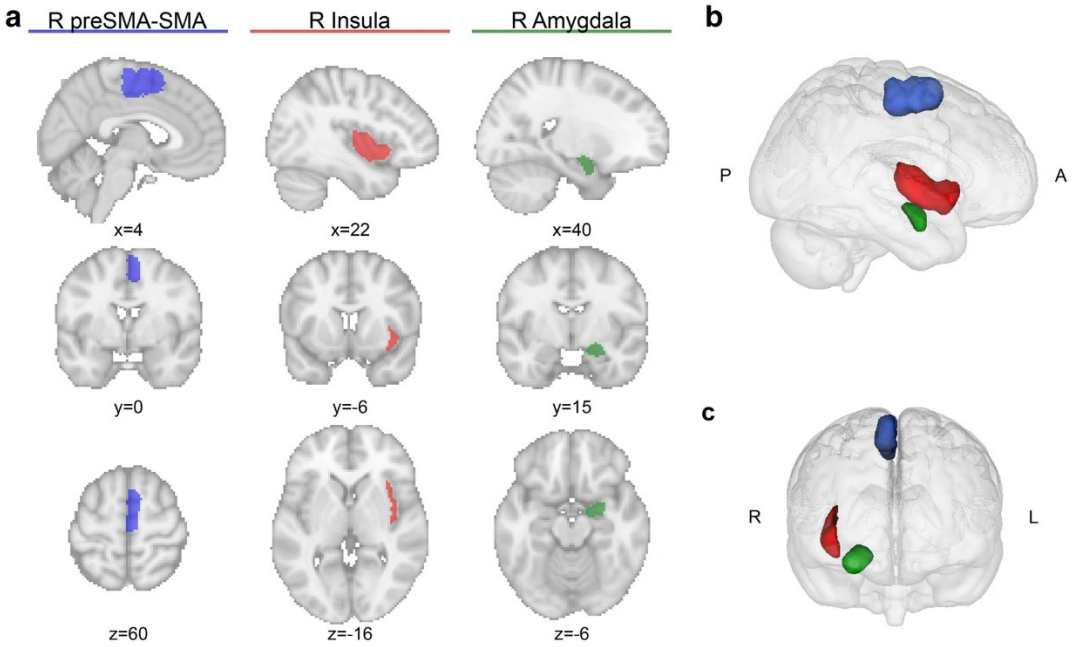


Figure 12 Visualization of the R preSMA-SMA, R insula, and R amygdala. a The three columns highlight the R preSMA–SMA (blue, left), R amygdala (green, middle), and R insula (red, right) in sagittal, coronal, and axial (top to bottom) sections. The figures were generated using Nilearn's plotting tools (Abraham et al 2014). **b** All ROIs shown from a right-sided view with posterior (P) to the left, and anterior (A) to the right. **c** An anterior view of the three ROIs, with right (R) and left (L) flipped to radiological display convention. The 3D representations in b and c were generated using Mango (multi-image analysis GUI; http://ric.uthscsa.edu/mango/)

Prior studies in our institute have made use of the FIX-denoised rsfMRI data and minimally processed (Glasser et al 2013) data of the young adult 1200 data set of the HCP (Van Essen et al 2013) which contains a total of 1206 subjects in 457 unique families, some of which are genetically confirmed monozygotic (149 pairs) or dizygotic (94 pairs) twins. This sample was used due to its large size, high temporal and spatial resolution, and high quality (for an open access data set), as well as having young and healthy subjects. Of the entire data set, only the rsfMRI and dMRI data of 300 healthy unrelated subjects (mean age 28.57, 150 females, no significant age ($t = .71$, $p = .48$) and educational ($t = -.31$, $p = .75$) difference between genders) were used. As the HCP young adult 1200 data set includes monozygotic and dizygotic twins it was important not to include related individuals in our sample, under the assumption that brain structure in twins is more similar than between unrelated singletons. Note that twins face additional challenges during early development (i.e., more adverse pre- and

perinatal events than singletons) causing further divergence from singleton brain structure and thus potentially hampering comparability (Ordaz et al 2010).

Pre-processing of the rsfMRI and dMRI data followed the minimal processing pipelines of the HCP [cf. Glasser et al. (2013)] consisting of structural, functional, and diffusion pipelines. These pipelines had been applied prior to receiving the data through official HCP channels. The structural pipeline obtains an undistorted native structural volume space per subject, aligns T1w and T2w images, performs bias field correction, and finally registers the native structural volumes to MNI space. The functional pipelines consist out of a volumetric and surface-based pipeline, of which only the volumetric pipeline is relevant here as no surface data was used. It removes spatial distortions, performs volume realignment to correct for motion artifacts, registers fMRI data to the structural data, reduces the bias field, normalizes the image to the global mean, and masks the data with a brain mask to remove non-brain tissue. Lastly, using the original diffusion timeseries (from two different phase encoding directions), the diffusion pipeline starts with b0 intensity normalization, susceptibility induced distortion correction, eddy current and subject motion correction, gradient nonlinearity correction, registration of the b0 image to the T1w image, and resampling to native structural space.

After acquisition of the data from official HCP channels, the rsfMRI minimally processed data was FIX-denoised. The dMRI minimally processed data was further pre-processed with FSL's Bayesian estimation of diffusion parameters obtained using sampling techniques (bedpostx) which utilizes Markov chain Monte Carlo sampling to create distributions of diffusion parameters for each voxel, and this is then used to model crossing fibres in the brain. Importantly, it generates all the files necessary for performing probabilistic tractography (probtrackx2), a necessary step for obtaining dMRI connectivity markers. Further (pre-)processing of the data was done using CBPtools.

3.2 Analysis Procedure

CBPtools was used for further processing and analysis, with workflow execution proceeding separately for each ROI and modality, as depicted in **Fig. 5**. For each execution the average MNI152 T1 brain (2mm isotropic) from FSL (Jenkinson et al 2012) was binarized and used as a whole-brain grey matter target mask. For rsfMRI only, the target mask was subsampled (see Sect. 2.6.1 [p. 52]) to improve computational efficiency.

Including the pre-processing of the ROI- and target-masks, all the following steps were done by CBPtools or one of its dependencies, using the configuration parameters outlined below (these are the CBPtools default configuration parameters). The rsfMRI BOLD time-series were 5mm full width at half maximum (FWHM) smoothed, global white matter (WM), global cerebrospinal fluid (CSF), 24 motion parameter signal corrected (including a bias term), and 0.01-0.08 Hz band-pass-filtered (see the green boxes in **Fig. 5c**). Global WM and global CSF nuisance signal regression in addition to FIX-denoising were used as they give the highest reliability for rsfMRI CBP (Plachti et al 2019). The linear correlations between ROI- and target-voxel time-series were then computed to obtain an ROI-to-target connectivity matrix for each subject and Fisher's Z transformed. To derive dMRI connectivity, probabilistic tractography was performed with the following parameters: distance threshold = 5, loop check = True, curvature threshold = 0.2, step length = 0.5, number of samples = 5000, steps per sample = 2000, correct path distribution for pathway length = True. This yielded a high-resolution ROI to low-resolution target (whole-brain) connectivity matrix per subject which was cubic transformed.

Each subject's connectivity matrix was used as input for k-means clustering (with k from 2 to 5, the k-means++ initialization method, 256 initializations [as suggested by Nanetti et al. (2009)], and a maximum of 10,000 iterations; **Fig. 5d**). The range of k was chosen after consulting relevant literature regarding the three ROIs. To maintain the same settings for each ROI and make replication of the example procedure computationally less intensive, it was chosen to keep the range of k consistent between ROIs. To obtain a group-level clustering, hierarchical clustering with complete linkage and Hamming distance was applied (**Fig. 5f**) on individual-level clusterings to obtain a combined reference clustering per k (Nguyen and Caruana 2007). The reference clustering was subsequently used to relabel the individual clusterings. The resulting labels were used to calculate the *mode* for each voxel, serving as the group-level clustering result for each value of k. Cluster validation was performed on the individual clusterings using the Silhouette index, the Calinski-Harabasz index, and the Davies-Bouldin index (**Fig. 5e**). The adjusted Rand index (ARI) was computed as a similarity measure between individual- and group-clusterings (**Fig. 5g**).

3.2.1 Parcellation Results

For each ROI a different aspect of the CBPtools workflow was focused on. For the preSMA-SMA ROI emphasis was placed on the reproducibility of histological parcellations, for the insula attention was directed towards the subdivisions of various k cluster solutions for the group parcellations, and lastly, for the amygdala the cluster validity metrics provided as output by the workflow were evaluated.

3.2.2 Amygdala long-range dMRI connectivity

Due to the results obtained from the R amygdala parcellation on dMRI data at high clustering granularity, as well as the problems inherent to probabilistic tractography and the poor signal to noise ratio of MRI in subcortical regions, it was decided to further investigate the source of this clustering. To determine whether this pattern was driven solely by local connectivity rather than differentiation based on long-range connectivity profiles of the ROI voxels, the ROI and a border around it were excluded from the whole-brain grey matter target mask in a follow-up analysis. The exclusion of the ROI along with borders of 5, 20, and 40mm around it were analysed separately. Next, the ARI was computed between these clustering results and the original R amygdala clustering results to assess their similarity. Lastly, the target voxel contribution to the $k = 2$ clustering of the R amygdala was evaluated. The dMRI connectivity matrices, computed with probtrackx2 (see Sect. 2.6.3 [p. 56]) were averaged for all subjects, resulting in an ROI voxel by target voxel matrix. The ROI voxels were separated by cluster, and the connectivity values of each target voxel to all ROI voxels within a cluster were then averaged. This provided an average connectivity value of each target voxel to each of the two clusters. The Euclidean distance between the connectivity values of both clusters for each target voxel was then calculated. The higher this distance, the larger of a contribution this target voxel provided to separating both clusters. This array of Euclidean distance values per target voxel was then z-scored and values below 1.96 were discarded. The remaining target voxels were considered the most important features as per their contribution to the cluster separation for $k = 2$.

3.3 SMA Parcellation

The MFC is commonly subdivided into the SMA and preSMA, separated by the vertical commissure anterior. Located on the midline surface of the hemisphere, activity in the SMA corresponds to the control of movement as its neurons project directly to the spinal cord. The most anterior part of the SMA is termed the pre-SMA, and both the SMA (posterior) and pre-SMA (anterior) subdivisions are marked by a sharp cytoarchitectonic border making them easily distinguishable with CBP.

3.3.1 Results

The group-clusterings for the 2-cluster solution approximated the R preSMA-SMA cytoarchitectonic differentiation with an ARI of .71 for rsfMRI and .76 for dMRI results (where 0 indicates no similarity at all, and 1 indicates perfect similarity). That is, only 76 (7.82%) and 63 (6.48%) out of all voxels were mismatched for rsfMRI and dMRI, respectively. **Fig. 13a** provides a visual representation of the ROI with the 2-cluster labels mapped onto it for the cytoarchitectonically defined region, and the two rCBP defined subdivisions using rsfMRI and dMRI data. The Silhouette index (**Fig. 13b**), Davies-Bouldin index, and Calinski-Harabasz index all indicate the 2-cluster solution as the best fit to the rsfMRI input data (note that the Davies-Bouldin index indicates a better fit through a lower value). The Silhouette and Calinski-Harabasz indices obtained from the dMRI clusterings both suggested the 2-cluster solution, with only the Davies-Bouldin index suggesting a slightly better fit for the 3-cluster solution. Results are consistent with previous studies regarding functional and structural parcellation of the preSMA-SMA regions (Johansen-Berg et al 2004; Klein et al 2007; Kim et al 2010; Zhang et al 2015).

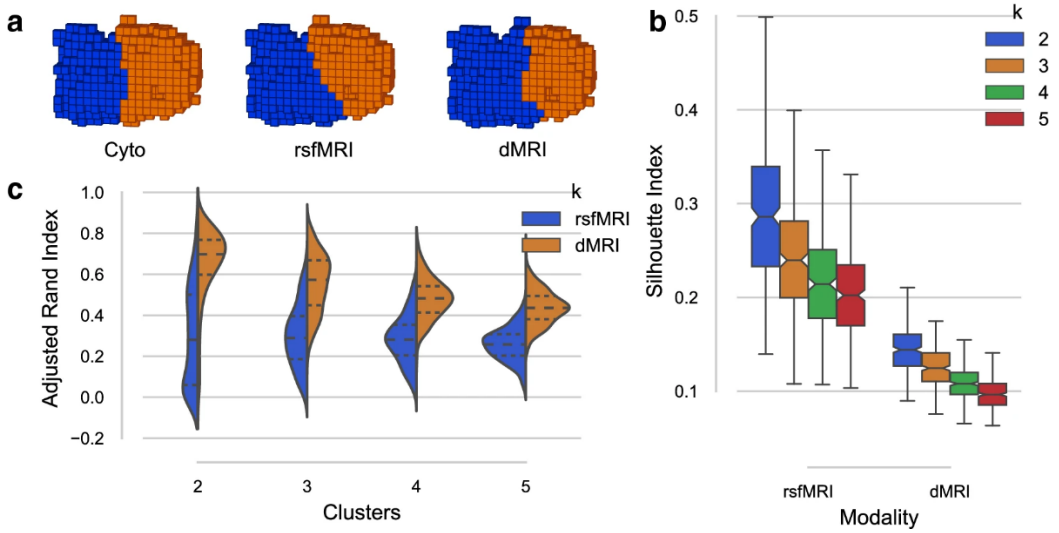


Figure 13 R preSMA–SMA results from the rCBP procedure. a The two-cluster solutions of the combined R preSMA and SMA ROI for the cytoarchitectonically defined (Ruan et al 2018) subdivision from the Jülich histological atlas (Eickhoff et al 2005), and the rsfMRI and dMRI connectivity-based parcels from left to right. The 3D representations were generated using matplotlib's 3D voxel/volumetric plotting and are in the same view as Fig. 12b. **b** ARI scores between the individual subject clustering results and the group-level clustering result for both rsfMRI and dMRI for k=[2,3,4,5]. **c** Silhouette index for all cluster solutions (k=[2,3,4,5]) where a higher Silhouette index indicates a better fit. Here, the two-cluster solution seems to best fit the input data for both rsfMRI and dMRI

By keeping the configuration of CBPtools consistent between ROIs, additional results (i.e., validity metrics and further cluster solutions) were automatically generated from the existing data. These results can be found in **Fig. 14**, which contains all the results provided as output by CBPtools for the preSMA-SMA ROI. While the results in **Fig. 14a** have been shown in full for the R insula and R amygdala ROIs, **Fig. 14b** contains the relabel accuracy as described in Sect. 2.7.3 [p. 62], showing that in particular for the dMRI 2-cluster solution the percentage of overlap is high. Generally, the adjusted rand index (ARI) is more credible than the percentage of overlap as it corrects for chance grouping of voxels within a cluster solution. The cophenetic correlation is displayed in **Fig. 14c**. It is a measure on how well the pairwise distances between the individual cluster labels (i.e., the cluster labels generated from each subject's connectivity matrix) are preserved in the group-level clustering. In this case it is the mode of the relabelled individual clustering results for each value of k, as described in Sect. 3.2 [p. 80]. The result is particularly high for the chosen 2-

cluster solution for both dMRI and rsfMRI. **Fig. 14d** shows the remaining 3, 4, and 5-cluster solutions, although both internal and external validity suggests a 2-cluster solution.

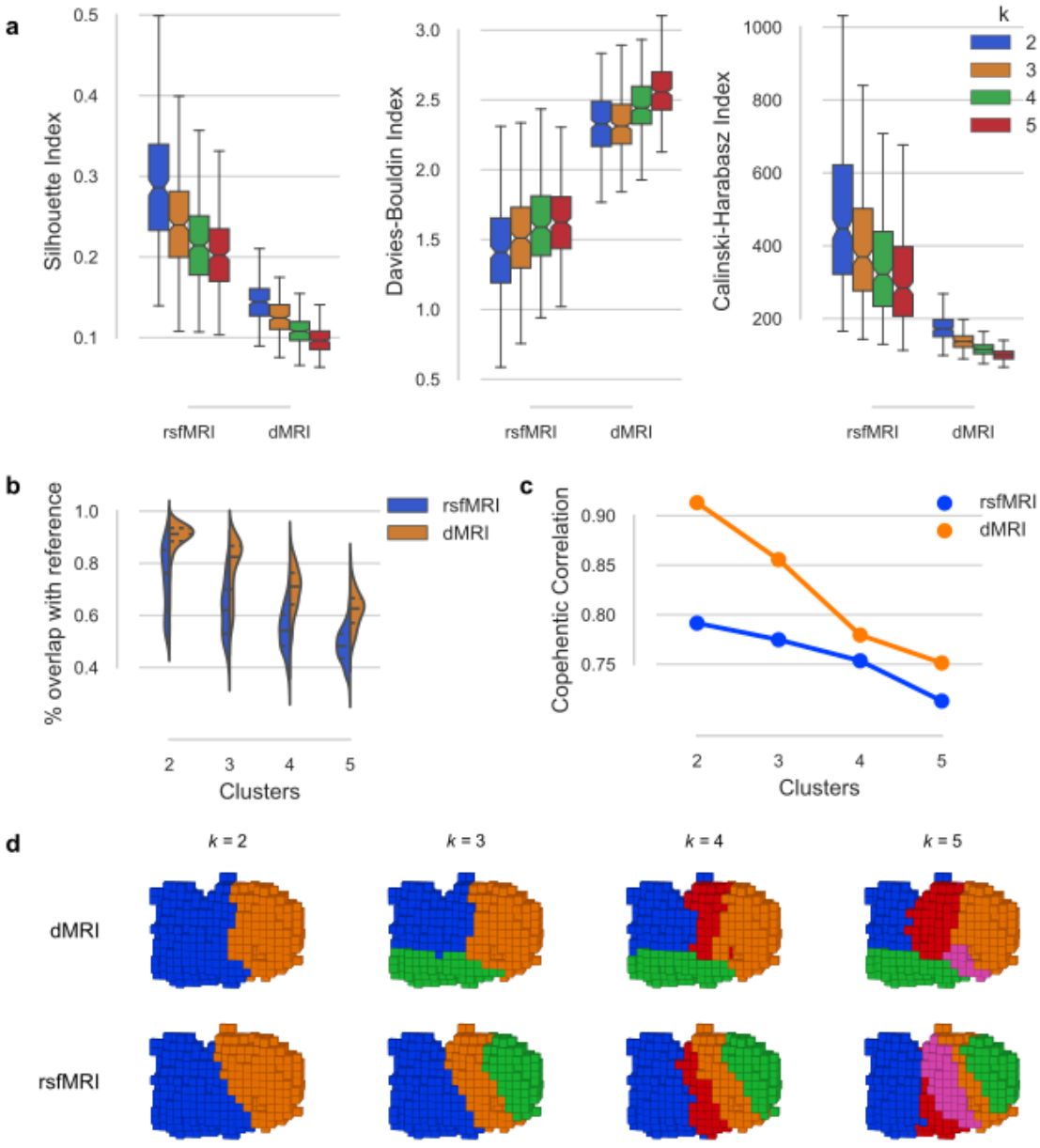


Figure 14 R preSMA-SMA validity metrics and parcels. a Internal validity scores (the Silhouette index, the Davies-Bouldin index, and the Calinski-Harabasz index) for all tested solutions (k = [2, 3, 4, 5]). **b** Accuracy of the relabelling for each individual clustering to the group-clustering with the cluster number k on the x-axis, comparing rsfMRI (blue) to dMRI (orange). **c** Cophenetic correlation scores of the reference clusterings for k = [2, 3, 4, 5] based on the rsfMRI (blue) and dMRI (orange) modalities. **d** Parcels for the k = [2, 3, 4, 5] cluster solutions

for both dMRI (top row) and rsfMRI (bottom row). The view is from the right side of the ROI with the posterior on the left and anterior on the right (the same view as applied in Fig. 12b)

3.3.2 Discussion

Separating the SMA and preSMA is a popular approach to validate rCBP methods (Johansen-Berg et al 2004; Klein et al 2007; Kim et al 2010; Zhang et al 2015) as it provides a gold standard and furthermore high-lights the ability of the rCBP procedure to reproduce histological parcellations. These two neighbouring regions exhibit an abrupt change in connectivity profile where their borders are expected to be, attributed to predominant connections to the motor regions for the SMA and prefrontal connections for the preSMA (Johansen-Berg et al 2004). As voxels are assigned to clusters based on similarity in their connectivity profiles, separating the preSMA-SMA ROI through automated parcellation approaches should therefore be straightforward. By using the cytoarchitectonically defined preSMA and SMA regions as external validation, it was possible to assess histological reproducibility of the preSMA-SMA ROI using CBPtools. This was achieved with a very high similarity (ARI $>$ .7) for both the dMRI and the rsfMRI connectivity driven parcellation to the cytoarchitectonic definition of the ROI.

3.4 Insula Parcellation

Located deep within the lateral sulcus of the Sylvian fissure, which separates the parietal and frontal lobes from the temporal lobe, the insular cortex is a region first described by J. C. Reil (1808), therefore also known by the moniker 'the island of Reil'. It is suggested to be involved in processing sensory, motor, gustatory, olfactory, auditory, and pain information, body representation, as well as modulating attention and emotion (Bamiou et al 2003; Ackermann and Riecker 2004; Brooks et al 2005; Menon and Uddin 2010; Rudenga et al 2010; Gasquoine 2014; Dambacher et al 2015; Avery et al 2015; Preston and Ehrsson 2016; Mazzola et al 2017). As its associations imply, it is one of the most frequently activated regions in functional neuroimaging research (Duncan and Owen 2000; Nelson et al 2010; Yarkoni et al 2011; Chang et al 2013). With the emergence of functional MRI it has been subjected to a diverse range of mapping and parcellation studies, such as cytoarchitectonic mapping (Mesulam and Mufson 1982; Kurth et al 2010a), as well as rCBP studies employing connectivity markers sourced from tractography (Nanetti et al 2009), MACM (Wager and Barrett 2004; Mutschler

et al 2009; Kurth et al 2010b), and resting-state (Nelson et al 2010; Cauda et al 2011; Deen et al 2011).

3.4.1 Results

All internal validity metrics agreed that a 2-cluster separation into an anterior and posterior subdivision fitted the rsfMRI source data best. The 2- to 5-cluster solutions are shown as 3D volumetric/voxel plots in **Fig. 15a**. The 3-cluster rsfMRI solution added a medial parcel (green), extending more into the anterior parcel (blue) rather than the posterior parcel (orange) of the 2-cluster solution. The 4-cluster rsfMRI solution further subdivided the medial and part of the posterior parcel into dorsal-anterior and medial parcels (green and red, respectively), whereas the 5-cluster solution added only a thin parcel (magenta) in between the aforementioned dorsal-anterior and medial parcels.

Likewise, for dMRI data the 2-cluster solution separated the R insula into anterior and posterior subdivisions. However, the Davies-Bouldin index slightly diverged from the other metrics, instead suggesting a 3-cluster solution to best fit the source data (see **Fig. 15b**). The shape of the dMRI clusters also showed a different picture than the rsfMRI results, particularly for the 3- and 5-cluster solutions. The 3-cluster solution added a medial parcel that did not extend as much into the dorsal direction as the rsfMRI 3-cluster solution did. Slightly more agreement between modalities was found in the 4-cluster solution, where the posterior parcel was subdivided into a dorsal (blue) and ventral (red) part, the latter of which was further split in the 5-cluster solution.

Functional parcellation of the 2-cluster rsfMRI solution for the insula were in line with prior parcellations (Kelly et al., 2012). In addition, Nanetti et al. (2009) suggests a common parcellation of the insula along the anterior-posterior axis for dMRI data. The 4-cluster dMRI parcellation furthermore visually resembles the insula's functional differentiation uncovered by (Kurth et al 2010b) using a meta-analytic approach, with only our ventral-anterior parcel (red) extending more anteriorly than theirs.

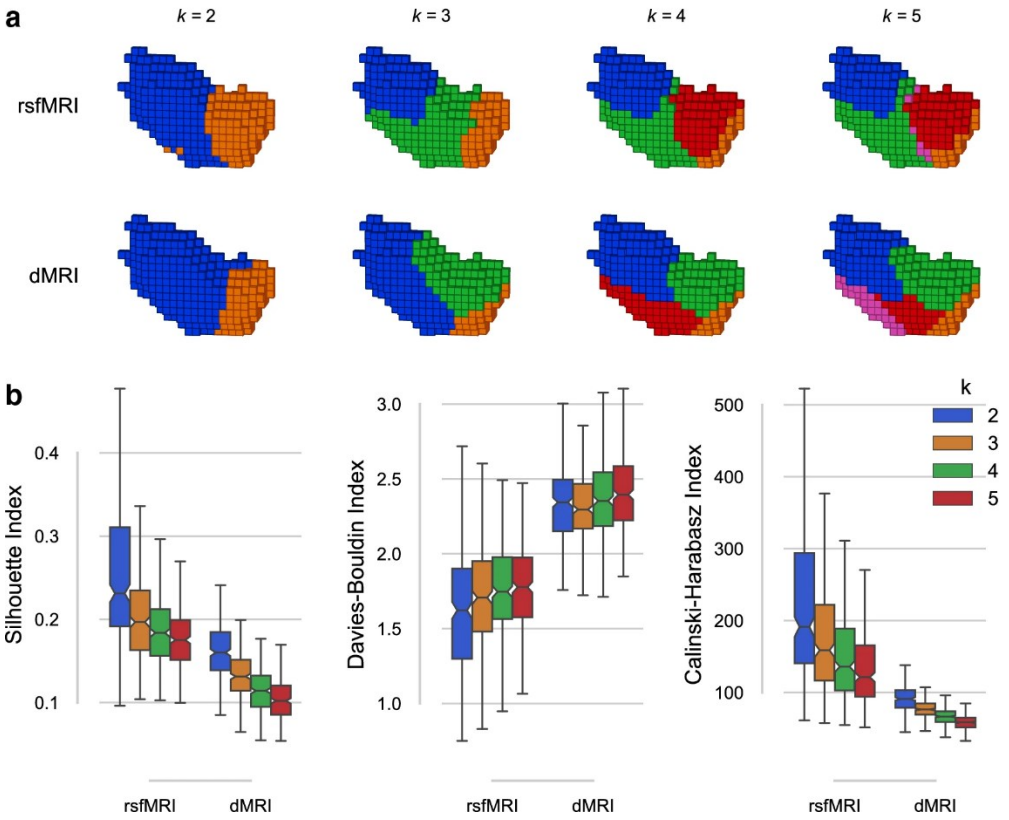


Figure 15 R Insula results from the rCBP procedure. a Insula parcels for the two-, three-, four-, and five-cluster solutions obtained from rsfMRI (top row) and dMRI (bottom row) connectivity. All images are in the same view (right-sided) as Fig. 12b. **b** Internal validity scores for all tested solutions (k=[2,3,4,5]). The Silhouette index (left) and the Calinski–Harabasz index (right) indicate a better fit through a higher score, whereas the Davies–Bouldin index (middle) is better when lower

As was the case for the SMA parcellation, additional results were also obtained for the Insula parcellation by keeping the CBPtools configuration between the ROIs consistent. These results are outlined in **Fig. 16**.

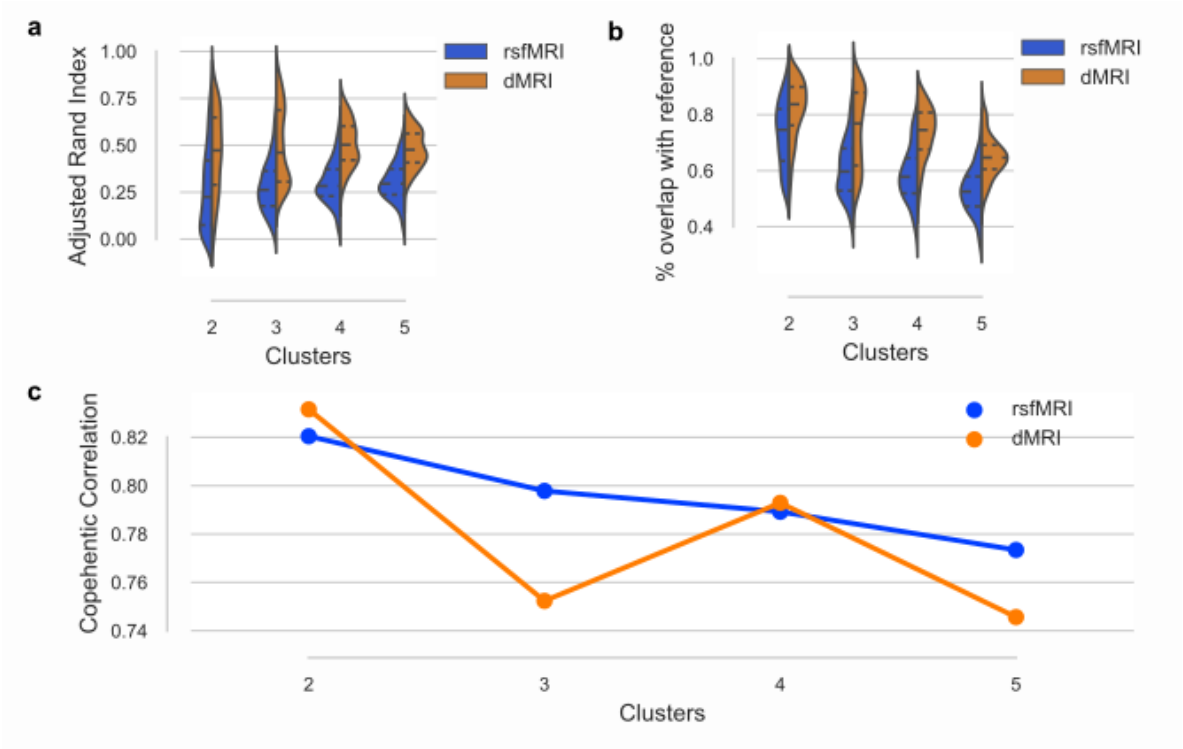


Figure 16 Validity metrics for the R insula. a Group similarity scores (i.e., the similarity of individual clusterings to the group-clustering) using the ARI with the cluster number k on the x-axis, comparing rsfMRI (blue) to dMRI (orange). **b** Accuracy of the relabelling of individual subject cluster labels to a reference clustering, calculated as the percentage of overlap between both clusterings. **c** Cophenetic correlation scores of the reference clustering for k = [2, 3, 4, 5] based on the rsfMRI (blue) and dMRI (orange) modalities

3.4.2 Discussion

While results for the preSMA-SMA parcellation were rather straightforward, this was not the case for the R insula parcellation for which many different suggestions for optimal cluster solutions exist in the literature [2-cluster (Cauda et al 2011), 3-cluster (Deen et al 2011; Chang et al 2013), and 4-cluster (Kurth et al 2010b), as well as various solutions exceeding our *k*-range (Kelly et al 2012)]. These differences may in part be caused by relatively small data sets with different properties, difficulties on account of intersubject alignment when delineating the ROI mask, as well as variability between research groups in their implementation and use of methods and imaging modalities. Our results suggested a 2-cluster solution to best fit both the dMRI and rsfMRI based connectivity markers, although this does not imply it is neurobiologically optimal. Early work on the insula has provided evidence for an anterior (dysgranular) and posterior (granular)

subdivision separated by the central insular sulcus (Brodmann 1909). Cytoarchitectonically the posterior insula can be further subdivided into two dorsal posterior areas and one ventral posterior area (Kurth et al 2010a), but no evidence exists for the anterior insula. Whereas the 2-cluster solution matched well between the dMRI and rsfMRI modalities, the results diverged at the 3- and 5-cluster granularities. The mid-posterior cluster appearing in the dMRI 4-cluster solution (**Fig. 15a**; red) and the mid-anterior cluster appearing in the rsfMRI 4-cluster solution (red) made the solutions at the 4-cluster granularity more similar. However, the rsfMRI mid-posterior cluster (green) extends more dorsally than its dMRI counterpart. Meta-analysis of the insula (Kurth et al 2010b) resembles the 4-cluster solution of the R insula, associating the posterior cluster (blue) with sensorimotor function, the ventral-anterior cluster (orange) with social-emotional functions, the dorsal-anterior cluster (green) with cognitive functions, and the medial cluster (red) with chemical sensory functions. The medial cluster extends further into the posterior direction for the dMRI parcellations than is the case for the meta-analytic results, and in addition extends further dorsally for the rsfMRI parcellation.

3.5 Amygdala Parcellation

Located deep within the temporal lobe of the brain and considered part of the limbic system, the amygdala is associated with the processing and regulation of positive and negative emotional valence (Ball et al 2009) [e.g., fear, anxiety, and social behaviour (Morris et al 1996; Phelps and LeDoux 2005; Adolphs 2010)] both in animals and humans (LeDoux 2007; Janak and Tye 2015), as well as associations to multiple disorders (Hayman et al 1998; Boccardi et al 2002; Wiest et al 2006; García-Martí et al 2008). Animal tract-tracing studies uncovered that the amygdaloid complex consist of multiple nuclei, each uniquely connected to other areas of the brain (Morrison and Salzman 2010). Mapping studies based on cytoarchitecture support a grouping of amygdaloid nuclei into a centromedial, laterobasal, and superficial group (Amunts et al 2005).

3.5.1 Results

Like the other two ROIs, the model that best fitted the data in the R amygdala was bipartite. Nevertheless, the 2-cluster solutions for rsfMRI and dMRI connectivity differed substantially (**Fig. 17d** and **e**). On the one hand, the rsfMRI 2-cluster solution showed a dorsal (superior) and ventral (inferior)

subdivision of the amygdala. On the other hand, the dMRI 2-cluster solution showed a medial and lateral subdivision. At higher clustering granularities, clusters split further among the aforementioned axes rather than finding common ground.

The Silhouette and Calinski-Harabasz indices (**Fig. 17a**) suggested a 2-cluster solution to best fit the rsfMRI source data. However, the Davies-Bouldin index instead suggested a 5-cluster solution to fit better. The 2-cluster solution approximated prior functional parcellations of the amygdala (Mishra et al 2014; Zhang et al 2018) and also prior cytoarchitectonic mapping of the region (Amunts et al 2005). The dorsal cluster (orange) overlapped with the cytoarchitectonic outline of R amygdala centromedial and amygdalostriatal subregions, whereas the ventral cluster (blue) overlapped with the laterobasal and superficial subregions. At the 3-cluster granularity the ventral cluster was divided into a cluster resembling the cytoarchitectonic laterobasal subregion (blue) and one resembling the superficial subregion (green), the latter of which is best seen from a left-sided view (**Fig. 18b**). The 4-cluster solution subdivided mostly the dorsal cluster (orange), with the new cluster resembling the amygdalostriatal subregion. However, it appeared far larger than its cytoarchitectonic counterpart. The 5-cluster solution further subdivided the ventral cluster (blue), but here no further cytoarchitectonic subdivisions exist.

For the dMRI clusterings, all validity indices (**Fig. 17a**) suggested a 2-cluster solution to best fit the source data. As the clustering granularity increased, the R amygdala split further along its medial-lateral axis. Previous parcellation works using dMRI data (Solano-Castiella et al 2010; Saygin et al 2011; Wen et al 2016; Fan et al 2016) also found similar clusters along the medial-lateral axis. ARI similarity of individual clusterings to the group-level clustering (**Fig. 17b**) was higher for clusterings on dMRI data than on rsfMRI data, also reflected by the relabel accuracy (**Fig. 17c**).

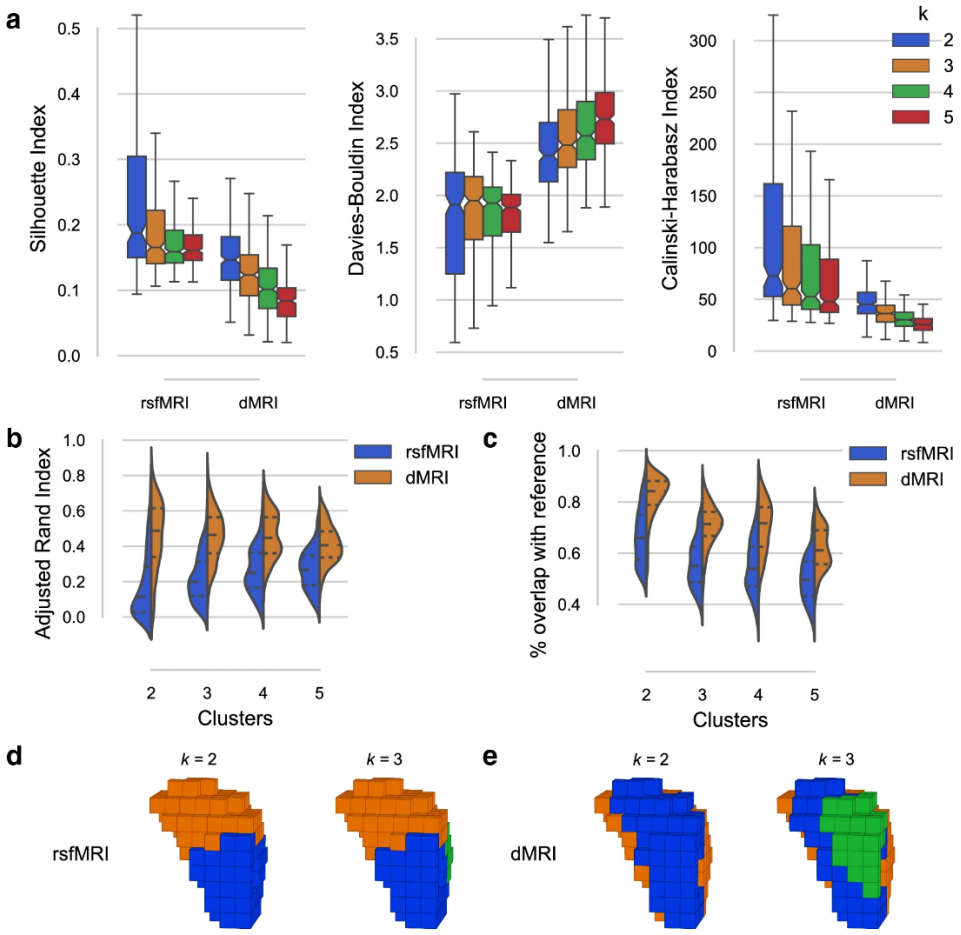


Figure 17 R amygdala results from the rCBP procedure. a Internal validity scores for all tested solutions (k=[2,3,4,5]). The Silhouette index (left) and the Calinski-Harabasz index (right) indicate a better fit through a higher score, whereas the Davies-Bouldin index (middle) is better when lower. **b** Group similarity scores (i.e., the similarity of individual clusterings to the group clustering), with the cluster number k on the x-axis, comparing rsfMRI (blue) to dMRI (orange). **c** Relabel accuracy displayed in a similar format as b. **d** The two- and three-cluster solution of the R amygdala for rsfMRI. **e** the same for dMRI

Due to the strangeness of the layered pattern of clusterings found in the dMRI clusterings along the medio-lateral axis at high clustering granularity, the long-range connectivity profiles of R amygdala voxels were assessed in isolation. While the visual representations of the clusters mapped onto the R amygdala ROI (**Fig. 19**) look relatively similar, the ARI scores reveal that this is not the case as more local connections are removed from the target features (**Fig. 20**). The $k = 2$ cluster solution for the 5mm run had an ARI of .9, whereas the 20mm and 40mm solutions had .8 and .46, respectively. At higher clustering granularities, the 4mm solution remained at low similarity to the original clustering solution. However, while the 5mm and 20mm results each revealed less similarity at the k

$= 3$ granularity, they became more similar to the original solution again at $k = 4$ and 5. Nonetheless, two of the three assessed validity indices, the Silhouette index and the Calinski-Harabasz index, indicated that the 2-cluster solution best fitted the data as per the original clustering results, as well as the 5mm, 20mm, and 40mm results. The Davies-Bouldin index suggested a best fitting 3-cluster solution for the original results as well as the 5mm results. For the 20mm and 40mm results, however, the Davies-Bouldin index instead suggested a better fitting 5-cluster solution. Taken together with a decrease in similarity as a larger area of local connectivity was excluded from the target features, this may hint that a more strongly coherent 2- or 3-cluster solution was driven predominantly by patterns in local connectivity.

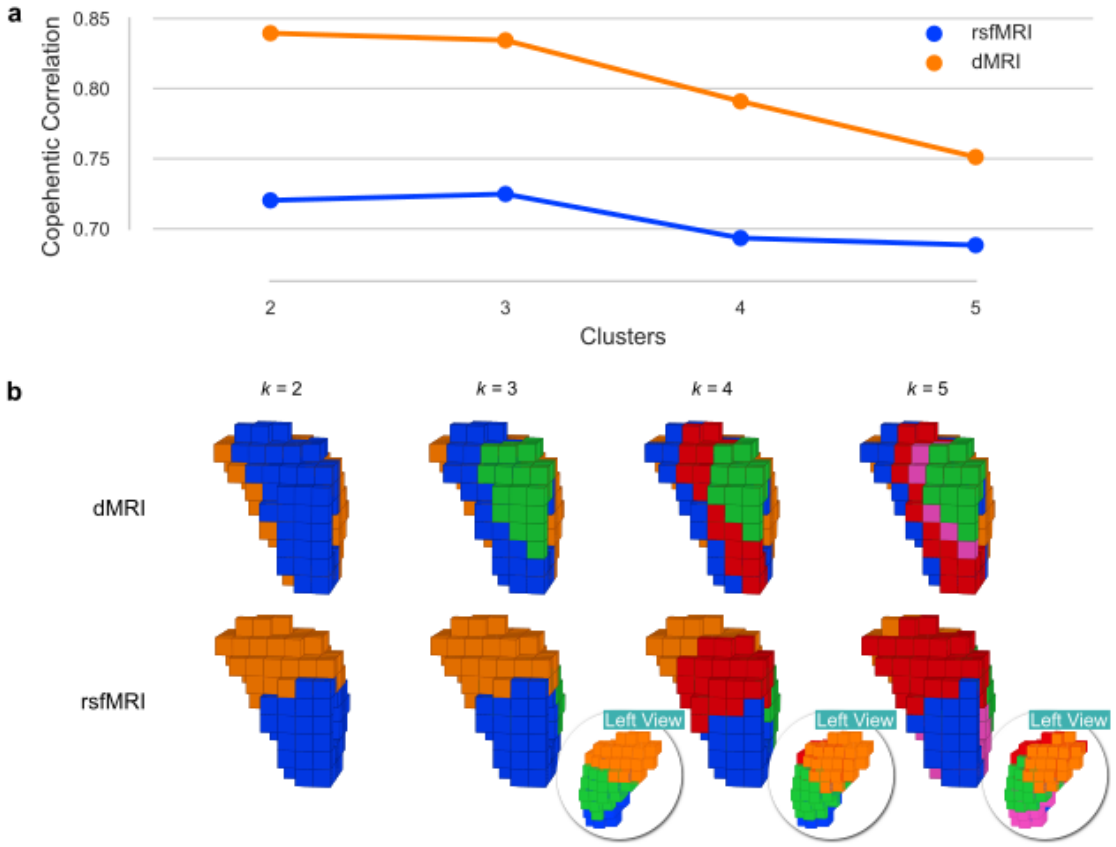


Figure 18 R amygdala metrics and parcels for the 2, 3, 4, and 5-cluster solutions. a Cophenetic correlation scores of the reference clustering for both rsfMRI (blue) and dMRI (orange). **b** Cluster solutions for all investigated values of k mapped onto the original ROI image, for both dMRI (top row) and rsfMRI (bottom row)

Fig. 21 is a visual representation of the target voxels that most contributed to the separation of clusters of the original 2-cluster solution. These voxels are mostly in the same hemisphere as the ROI, with a handful of voxels located in

the left hemispheric amygdala. However, more interhemispheric connections may be expected to the left amygdala, as the medial amygdala is suggested to be strongly connected with its interhemispheric counterpart. Only a handful of voxels being present here may have been caused by lower sensitivity in probabilistic tractography on account of the thresholds used to improve specificity and counteract false positives.

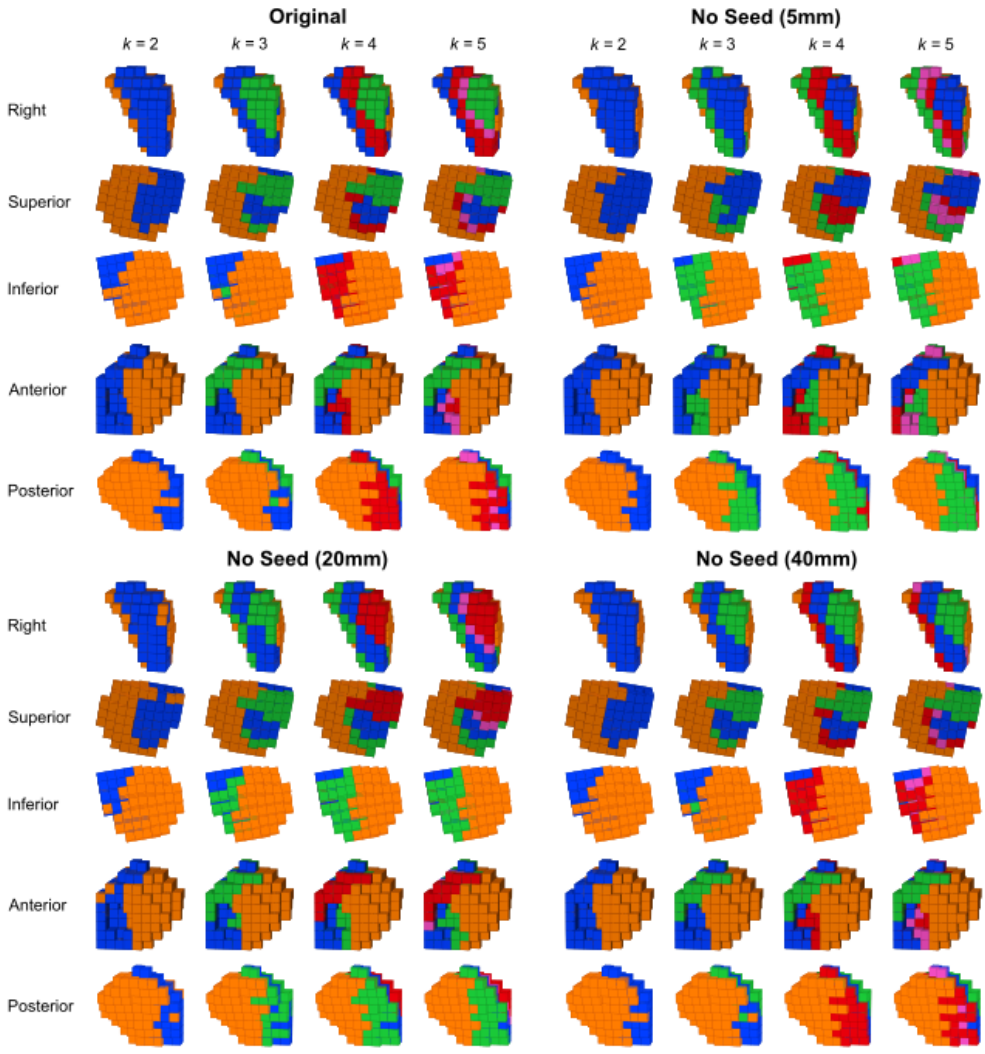


Figure 19 R amygdala parcels for k = [2, 3, 4, 5] using different target features. The top left shows the parcels when ROI voxels have a connectivity profile containing all whole-brain grey matter voxels. The top right shows the parcels that result from extracting the ROI voxels from the target features including a border of 5 mm around the ROI. The bottom left and right show the same, but with a border of 20 and 40 mm respectively

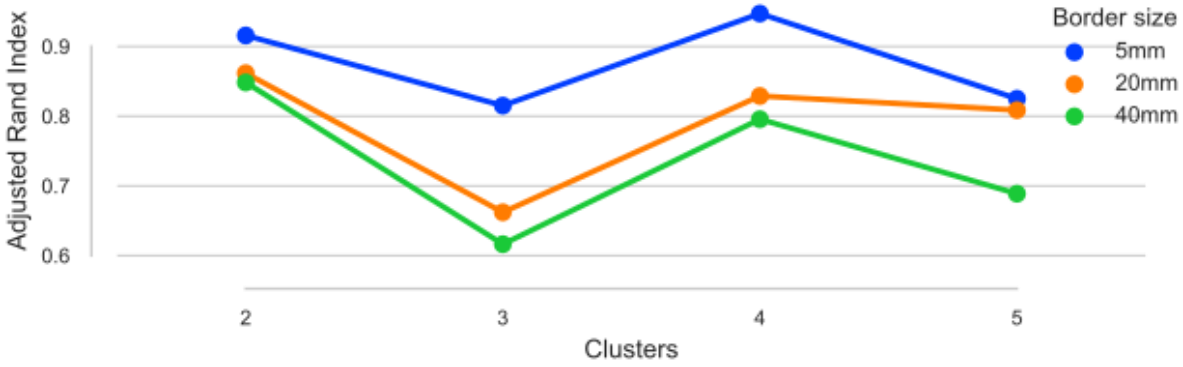


Figure 20 Similarity between R amygdala clustering results with different target features. Local connectivity was excluded from the target features based on a border size of 5 (blue), 20 (orange), 40 (green) mm. The ARI was calculated between the original (all features included) for k = [2, 3, 4, 5]

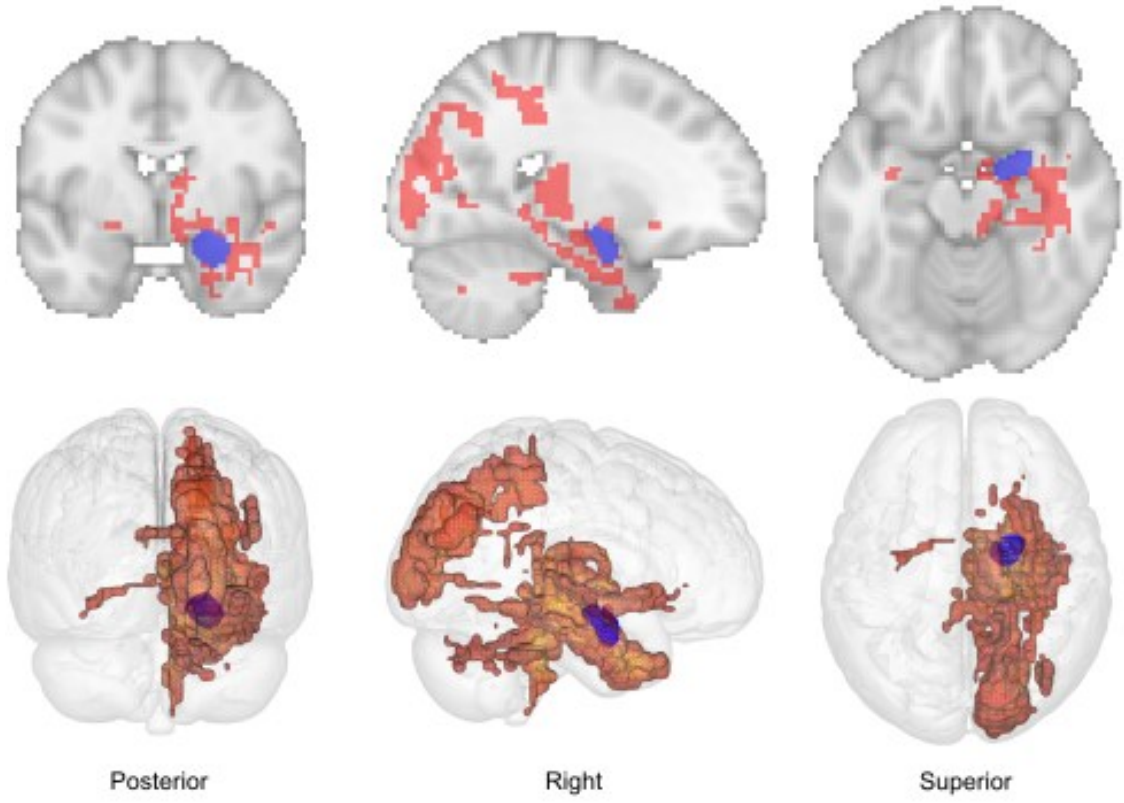


Figure 21 Mapping of target voxels that contributed most to the R amygdala 2-cluster solution. The target voxels are displayed in red, whereas the R amygdala is displayed in blue. The top row of figures was generated using Nilearn's plotting tools (Abraham et al 2014), whereas the bottom row was generated using Mango (multi-image analysis GUI; http://ric.uthscsa.edu/mango/)

3.5.2 Discussion

The 2-cluster solution best represented the data for both the dMRI and rsfMRI modalities. The rsfMRI parcellation of the R amygdala that best fit the source data was a bipartite dorso-ventral subdivision. These results match earlier findings of Mishra et al. (2014), likewise a dorso-ventral (superior-inferior) subdivision in the 2-cluster solution using functional connectivity markers, and a similar dorsal, ventral, and medial subdivision for the 3-cluster solution. The same 3-cluster solution was found by Zhang et al. (2018). Our validity metrics indicated a best fitting 2-cluster solution, but as this does not necessarily imply neurobiological accuracy a 3-cluster solution is likewise viable. Furthermore, the parcellations visually correspond to the cytoarchitectonic mapping of the R amygdala (Amunts et al 2005) up to the 4-cluster solution.

Where a bipartite dorso-ventral subdivision of the amygdala best fit the rsfMRI data, the dMRI data instead best fit within a bipartite medio-lateral subdivision. This pattern resembles the 2-cluster solution found by (Solano-Castiella et al 2010) and Fan et al. (2016) in that the solution divided the amygdala into a medial and lateral cluster. Tract-tracing of the rat amygdaloid complex shows that the medial amygdala is related to connections between both intrahemispheric amygdalae (Pikkarainen and Pitkänen 2001). The lateral amygdala is instead found to be connected to somatosensory cortical areas (Jolkkonen and Pitkänen 1998). Solano-Castiella et al. (2010) note the possible existence of a third cluster in between the medial and lateral clusters which resembled the pattern of clusters found for the CBPtools-derived 3-cluster solution.

The amygdala is a peculiar region on account of its spatial location, which may explain the differences between the rsfMRI and dMRI results. Whereas the rsfMRI parcellations resemble cytoarchitectural subdivisions, the dMRI results may instead be driven by spatial artefacts on account of false positives in probabilistic fibre bundle tracking (Descoteaux et al 2016; Zalesky et al 2016) of subcortical areas. As the region gets split at higher granularities, it is possible that instead of creating subdivisions based on neurobiologically relevant signals, instead the subdivisions are driven by noise in the signal on account of methodological idiosyncrasies. Investigating why such subdivisions occur at higher granularities is beyond the scope of this work but is nonetheless and important consideration when investigating clusters with dMRI data. Further investigation was done to determine whether parcellations of the R amygdala were driven by

within-ROI and short-range connections. dMRI data were parcellated after excluding ROI-to-ROI connectivity (excluding a 5mm, 20mm, and 40mm border around the ROI, see **Fig. 19**), resulting in mostly unchanged parcellations, exhibiting the same medio-lateral pattern of subdivisions.

4 Outlier Detection

This chapter outlines the analysis procedure and preliminary results obtained from a side-project investigating outliers (and automated detection thereof) in a sample to which the rCBP method will be applied.

4.1 Approach

The aim of rCBP is to find biologically meaningful parcels within an ROI, achieved by clustering the voxels in the ROI based on their connectivity profiles. Using a large sample (see Sect. 3.1 [p. 78]), it was found that deviant connectivity profiles substantially influence group-based clustering results. Such outliers can arise due to various reasons and investigation was done into one possible reason for high dimensional data: difference in intrinsic dimensionality. For this analysis the primary focus was on the R insula ROI using rsfMRI data (see Sect. 3.4 [p. 86]), but included further exploratory results using the dMRI data and the R preSMA-SMA and R amygdala in Sect. 4.3 [p. 100].

To identify outliers in the rCBP procedure, the connectivity matrices and subject-wise clustering results provided as interim output by CBPtools were used. For each subject a nearest-neighbour subject was identified using Euclidean distance between connectivity matrices, with the resulting vector d being Z-scored (**Fig. 22a**). K-means (k = 2) clustering of d revealed a separation around 0, providing a conservative threshold (**Fig. 22b**). Two additional thresholds of 1.69 (.95 left tail area on a standard normal distribution) and a liberal 2.5 were added to the list of evaluated thresholds. Group parcellations for each k using hierarchical clustering with average linkage and Hamming distance were

calculated after excluding outliers based on these thresholds. The ARI between k-means clustering results of all subjects was computed, retaining the highest values per subject as a similarity vector a (**Fig. 22c**). Principal component analysis was performed on the connectivity matrices, noting the number of components retaining 95% of variance. Correlation the component numbers to d uncovers whether there is a relationship between intrinsic dimensionality of the connectivity matrices and their distances to one another.

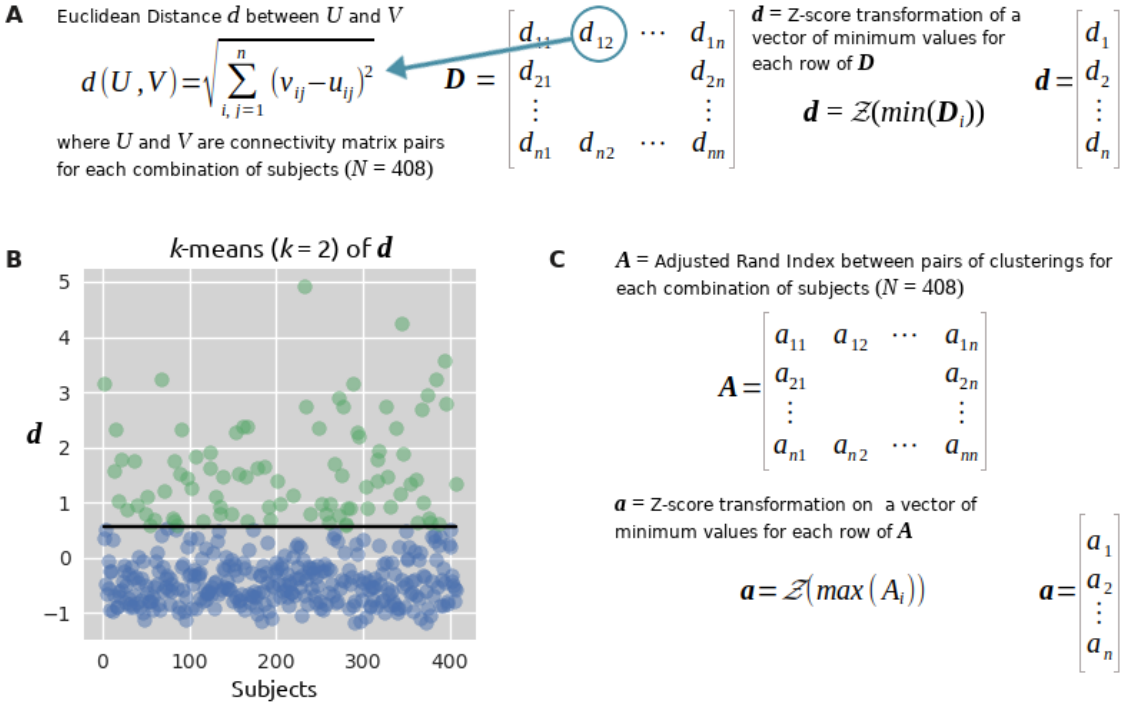


Figure 22 Methods for detecting deviant connectivity profiles (outliers). a Computing the nearest-neighbour Euclidean distance for each subject as vector ***d*** and Fisher's Z-transforming it. **b** Derivation of an estimate for the conservative threshold using k-means (k=2) to cluster deviant and non-deviant connectivity matrices by their distances ***d***. The black horizontal line shows the separation, which was rounded to 0 for usage as a threshold value. **c** ARI between the cluster solutions of each combination of pairs for each individual subject. A similar approach to figure a was used to generate vector ***a***.

4.2 Results

Applying the thresholds of 0, 1.69, and 2.5 removed 134, 32, and 14 subjects, respectively. When correlating distances d (**Fig. 22a**) to the similarity vector a (**Fig. 22c**), correlations of -.38, -.41, -.49, and -.53, for k = *2, 3, 4,* and *5*, were

found, respectively. This result suggests outliers cluster differently, thus including them into a group-level consensus might be detrimental. Accordingly, differences were found between group-level parcellations (**Fig. 23b**). For instance, when comparing the liberal 2.5 threshold-removed group parcellation (**Fig. 23b**, column two) with a group parcellation without outlier removal (**Fig. 23b**, column one), there was only an 81% overlap, ARI = .55 for $k = 3$ (ARI=.67, and .71 for $k = 4,\ 5$, respectively). Further comparisons are illustrated in **Fig. 23b**. The distances $\boldsymbol{d}$ were related to the number of principal components retaining 95% of variance with correlation of -.79 (**Fig. 23a**). That is, if intrinsic dimensionality was low for a subject, the associated connectivity matrix would be more distant to the rest of the sample.

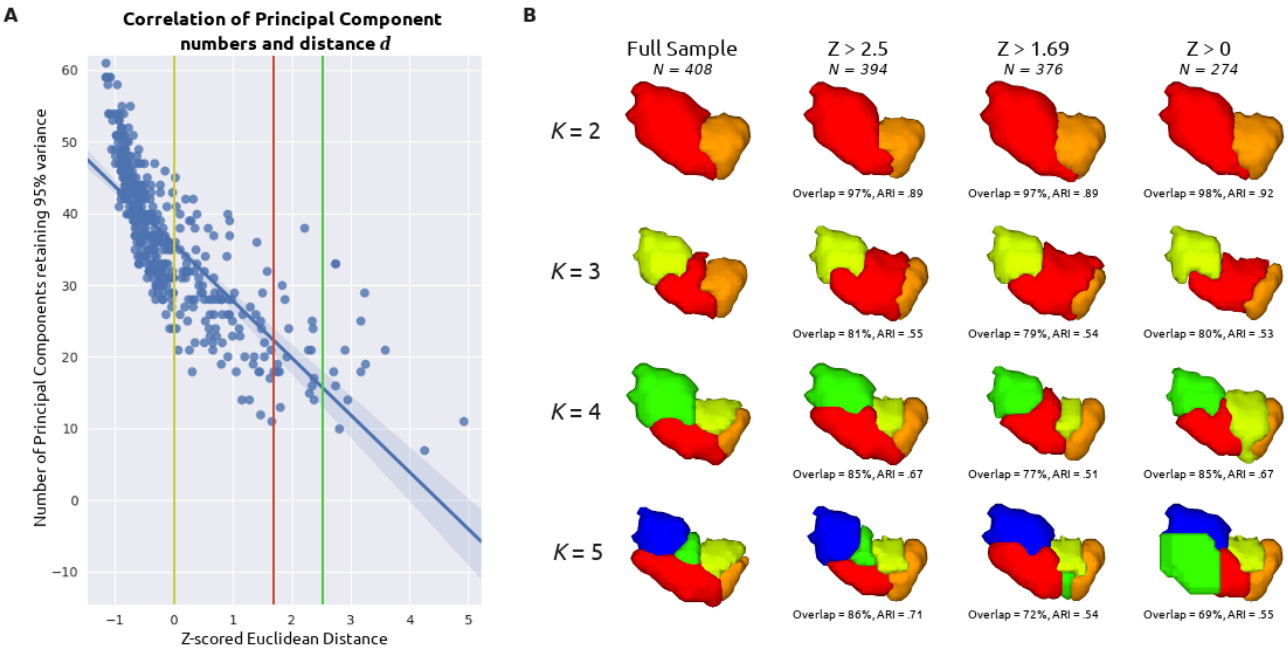


Figure 23 Group-level clustering of R insula with and without outlier-removal. a Correlation between distances $\boldsymbol{d}$ (Fig. 22a) and ARI $\boldsymbol{a}$ (Fig. 22c). The vertical lines represent outlier thresholds for 0, 1.69, and 2.5 as yellow, red, and green, respectively. **b** Group-level clusterings of the R-insula ordered by k clusters and outlier threshold. The overlap and ARI values portray the similarity of the clustering to the clustering without outliers removed. Visualized with the BrainNet Viewer (Xia et al 2013).

4.3 Additional Exploratory Analyses

While the primary analysis focuses on the rsfMRI data for the R Insula, a preliminary assessment of the other two regions, the R Amygdala and R preSMA-

SMA, was also performed (**Fig. 24**). Furthermore, an exploratory analysis was performed to uncover what may cause subjects to be outliers by assessing whether the deviant index relates to age, gender, framewise displacement, and acquisition quarter (i.e., time at which the subject data was acquired). Note that all the data presented in this section is exploratory, requiring further investigation.

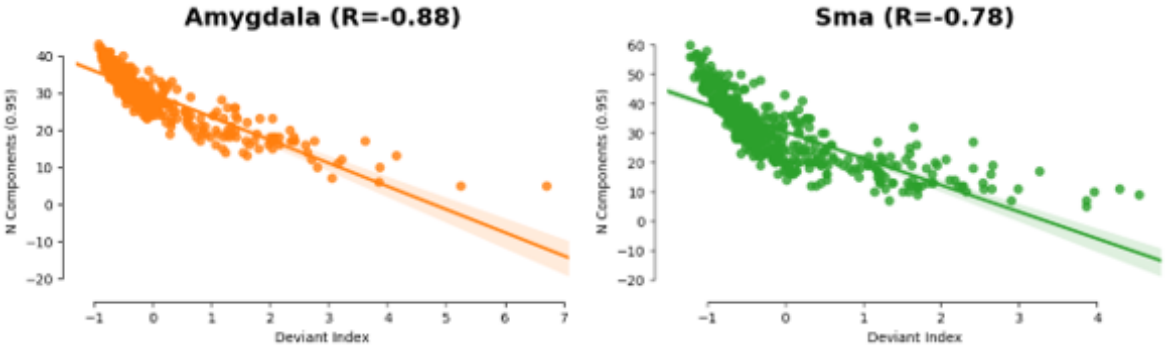


Figure 24 Correlation between the deviant index and component number. On the left, the correlation between the deviant index and the number of components explaining 95% of the variance of connectivity is shown for the R amygdala. On the right, the same is shown for the R preSMA-SMA region. Both regions exhibit a strong correlation between these two metrics.

Subjects were divided into 4 equally sized groups based on the number of components explaining 95% of variance for each region (in ascending order, from low component number to high). Age and gender distributions, as well as framewise displacement values were then compared between groups, finding no remarkable differences. Lastly, the number of components was correlated to the acquisition quarter for each region using Spearman correlation. A small but significant effect was found for all regions (**Fig. 25**). Subjects were again divided into four equally sized groups, this time based on acquisition quarter, to assess the gender distribution per group, again finding no imbalance.

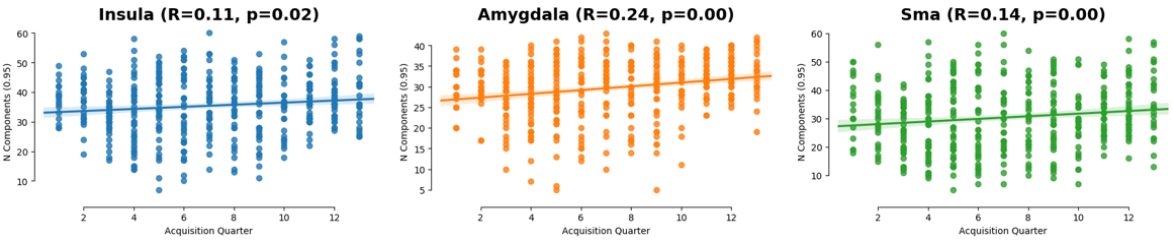


Figure 25 Spearman correlation components to acquisition quarter. From left to right component numbers derived from the connectivity matrices of the Insula, Amygdala, and preSMA-SMA regions are each significantly correlated to the acquisition quarter. Each dot represents a subject in the data set.

4.4 Discussion

The differences in clusterings highlight the influence of outliers. While assessment of the group-level parcellations reveals that clustering results were relatively stable across thresholds for $k = 2$, ample evidence suggests more than 2 clusters in the R insula (Kurth et al 2010b; Cauda et al 2011; Deen et al 2011). As linkage algorithms in hierarchical clustering as well as k-means clustering are sensitive to outliers (Duda et al 2001), it is important to remove them by using a proper identification threshold. In the future focus will be placed on automatic identification of parameters that lead to biologically meaningful parcellations.

Part Four

Discussion

5 General Discussion 106
5.1 CBPtools 106
5.2 Example Data 112
6 Conclusion 115

5 General Discussion

Regional connectivity-based parcellation is a widely used procedure for investigating the structural and functional differentiation within a region-of-interest based on its long-range connectivity. A Python-based software package has been developed for applying the rCBP procedure with resting-state or diffusion MRI data.

5.1 CBPtools

CBPtools, an open-source Python package for performing the rCBP procedure, was outlined and introduced. It is capable of computing two commonly used measures of brain connectivity (i.e., resting-state and diffusion-weighted MRI), can apply three different clustering algorithms to derive connectivity-based parcellations (i.e., k-means, spectral, and agglomerative clustering), and uses various metrics for cluster validation (i.e., the Silhouette coefficient, Davies-Bouldin index, and Calinski-Harabasz index). Furthermore, it can compute group clustering results by aggregating clustering results from all subjects and provide detailed reports on every aforementioned computation.

The architectural design of CBPtools enables an easy to use, flexible, and reproducible approach to applying the rCBP procedure to large data sets. At the same time, the software is structured such that additional processing tasks as well as compatibility features (e.g., compatibility with data types, external processing software such as FSL, or (Python) packages for data representation) can easily be supplemented through git PRs via the GitHub repository (see Sect. 2.10 [p. 70]).

Factors contributing to its ease of use are the command line utility (i.e., using the 'cbptools' entry point for setting up and running a project), using the YAML format (on account of it being a human-readable data-serialization language) for the

configuration file allowing manual and automated configuration, compatibility with workload managers, validation of input, extensive usage documentation, and the fully automated workflow for applying rCBP. Several 'quality of life' features have been added, such as the automatic extraction of an ROI from an atlas foregoing the need of manual ROI delineation (see Sect. 2.8.4 [p. 66]), and an automated comparison of clustering results to predefined reference images (see Sect. 2.8.5 [p. 67]). Validation of input data and configuration files is performed such that most common mistakes are caught at the start of the procedure. Therefore, problems that would normally arise later in the processing pipeline are recognized early, preventing wasteful use of processing resources. Furthermore, the structure of the CBPtools workflow splits all processing tasks into small parts (i.e., jobs in a workload manager). This optimizes parallelization and ensures preferential treatment over larger and resource heavier jobs in workload managers that make use of a priority queueing system.

The procedure is flexible due to its customizability through a configuration file, allowing for fine-tuned processing for each ROI. This allows for a wide array of applications. For example, not only can a choice be made between various clustering techniques, but each technique can also be further customized with options, such as selecting an initialization method and the number of repetitions for the k-means clustering algorithm. Connectivity and clustering methods have been carefully chosen to both reflect the most popular and the most widely evaluated approaches in the brain mapping community.

The core dependencies of the CBPtools software (see Sect. 2.3 [p. 44]) are all open source, continuously developed, widely used, and widely supported packages within the (neuro)scientific Python community. Hence, their continued future support is a realistic expectation, although CBPtools will continue functioning with the package versions it was built with (i.e., the versions noted in its setup file). Note that FSL is not a Python package but considered an external dependency (that the user must install themselves) only required for processing dMRI data in the pipeline. Nevertheless, the FSL software also enjoys continuous development and is well known (and well used) in the neuroscience community.

The CBPtools pipeline applies user-defined pre-processing techniques to the input masks (i.e., ROI and target mask) and the rsfMRI or dMRI images prior to computing the ROI-to-target voxel-wise connectivity matrices. These matrices are subsequently used to obtain a clustering, using one of the three currently available clustering techniques (i.e., k-means, spectral, and hierarchical clustering). The given clustering solutions are then grouped (provided the input data is in the same reference space and

the software is configured to return group, rather than individual, clustering results). All the obtained results are then subjected to one or more user-defined validation metrics, ultimately resulting in a report of all outputs represented as plots, tables, and NIfTI images. Various alternative approaches are implemented, such as different clustering and cluster validation techniques, the ability to meaningfully use multi-session input data, as well as the option to perform single-subject parcellations in native space.

By providing or specifying input as well as parameters given to CBPtools, any parcellation work can be reproduced with relative ease and, importantly, can be compared to other works using this tool. To illustrate the efficiency of the procedure, benchmarks were provided (see Sect. 2.9 [p. 68] and Appx. 6) as a guideline for what can be expected when executing CBPtools on a similar data set, with similar settings, on the average computational cluster. With the CBPtools output, a user will be able to quickly generate parcellations and validity metrics that can either be used directly or used to inform a more detailed post-hoc analysis. For instance, the selection of clustering granularity as well as multi-modal integration of cluster solutions may require further fine-grained and region-specific analyses. It was opted to provide all k cluster solutions with guidance for the user to choose the optimal solution, as there likely is no 'one true parcellation', but instead biologically relevant maps at different granularities.

5.1.1 Future Extensions and Refactoring

Due to CBPtools being open source, anyone can make changes and extend the code to suit their needs. Such changes may be committed to the master branch of the software making them readily available for any CBPtools user or kept in a separate repository as an alternative to CBPtools (e.g., for feature additions that fall outside of the scope of CBPtools). Recommended features that build upon existing functionality to be added in future version of CBPtools include the integration of other modalities, multi-modal analysis of parcellation results, additional validity metrics, and different clustering approaches. It is proposed to integrate the MACM and structural covariance modalities, as they are found to be valuable for studying the brain and adding an additional layer of information to multimodal CBP (Eickhoff et al 2015; Plachti et al 2019). The currently implemented cluster validity metrics evaluate which k cluster solution fits the data best (where $k > 1$). When the question arises whether an ROI can be clustered at all (i.e., has more than one cluster) the currently implemented validity metrics are not enough. Various metrics exist that may answer this question,

such as the gap statistic and the Hopkins statistic. The gap statistic performs a hypothesis test of clustered data (e.g., a 2-cluster solution) against a null hypothesis of random noise (equivalent to not clustering the data). If the resulting score is significantly lower for the k-cluster solution than the no-cluster solution, the k-cluster solution fits the data worse than a no-cluster solution. The Hopkins statistic assesses the data for cluster tendency, indicating whether the data is clusterable (i.e., it is likely there are clusters in the data, although the metric does not reveal how many) or whether it is unlikely there are any statistically significant clusters in the data. Since this method is performed on the connectivity markers instead of the clustering results, it can be implemented as an early warning system to prevent unnecessary occupation of computational resources. Lastly, other clustering methods such as fuzzy c-means clustering and ICA may be introduced, as well as alternatives to clustering such as probabilistic, graded, or boundary mapping. Care must be taken, however, that the addition of such methods does not exceed the scope of the CBPtools project (see Sect. 2.1 [p. 39]).

Further improvements to the usability of CBPtools can be made. Currently users are expected to set up their parameters using a configuration YAML file (see Sect. 2.5.2 [p. 48]). This file must be formatted according to the YAML specifications, reference specific parameter keys, and use proper expected value types. None of these are referenced or validated inside the document itself, having users crucially depend on documentation. While YAML is a very human-readable format, it is not comparable to a user interface (UI) in terms of efficiency for manual data entry. A UI deals with both parameter keys and document specifications, while also providing information about what kind of values can be expected (e.g., through a 'combobox' providing an exhaustive list of values, or a checkbox for Boolean values). The *npyscreen* library is an excellent fit, as it provides a framework for developing a *terminal* UI. This allows users to set up their configuration file through a UI that is displayed inside of a terminal – thus benefiting from a UI even on systems that do not allow graphical interfaces (common on computational clusters, as graphical interfaces as resource intensive). Setting up a CBPtools project using the configuration YAML file is, however, an important feature for users wishing to automate the entire setup process. Therefore, a terminal UI should only assist in creating the configuration YAML file, not replace it.

Currently CBPtools has three processing steps that, for the sake of coherence, should be separated from the primary processing pipeline: Dataset validation (see Sect. 2.5.3 [p. 49]), region extraction from an atlas (see Sect. 2.8.4 [p. 66]), and mask pre-processing (see Sect. 2.6.1 [p. 52]). Dataset validation is a lengthy and resource

intensive procedure applying various compatibility tests to an existing dataset, including (but not limited to): NIfTI image assessment, inferring (and double-checking with manual configuration parameters) the repetition time from NIfTI image headers, evaluation of subject-wise data (i.e., to discover whether there are subjects with missing data, and whether all input time-series images are in 4D sharing the same shape, affine, and number of timepoints), and an assessment of low-variance voxels in subject-wise time-series (see Sect. 2.6.2 [p. 54]). Separating this procedure into its own CBPtools subcommand (as "cbptools validate-dataset …") makes the procedure optional, as it does not have to be executed when the same dataset is used multiple times. This validation procedure should be very explicit in its warnings and errors and provide hints to potential solutions (e.g., removing subjects with missing, incomplete, or incompatible data). The primary processing pipeline should still perform a simplified validation approach aimed at validating configuration parameters, but should otherwise gracefully exit the procedure upon encountering compatibility issues with the dataset.

Automatically deriving a region from an atlas is currently available as part of the setup procedure, but the application is rather arcane (i.e., the region-id must be manually derived and entered, and as a non-default parameter it lacks discoverability). A preferable approach separates region extraction from the setup procedure into a new CBPtools command that uses a terminal UI. Users can then select and download predefined atlases and select regions by name, as most atlases contain a complementary meta-data file tying region-ids to region names. This can be combined with mask pre-processing, currently implemented in the masking task. Pre-processing a mask is a very fast and resource inexpensive procedure. Many things can go wrong when pre-processing mask images, especially if there is a mix-up of masks stored in neurological and radiological convention, or if the NIfTI header is not properly set up. The latter is common for values that cannot be automatically derived from the NIfTI header, such as repetition time (Brett and Poline 2016). It is impossible to address all potential issues, hence separating the masking task from the primary pipeline allows for manual verification of the input masks prior to performing connectivity-based parcellation.

CBPtools is agnostic to the structure of the dataset, as file paths to data are manually defined in the configuration file. The brain imaging data structure (BIDS) is a specification for how neuroimaging datasets should be structured (Gorgolewski et al 2016). This provides many advantages, but one key advantage is that it enables automatic derivation of dataset specific properties through its clearly defined metadata. Implementation of BIDS compatibility in the setup procedure allows CBPtools to exploit this advantage and significantly simplify the configuration and setup

procedures. Along with this change, CBPtools should adhere to the BIDS derivatives specification when storing its output. This allows users to easily append CBPtools output to their datasets for subsequent release.

The CBPtools workflow was originally built upon the Snakemake workflow management system. Nipype is a promising and feature-rich alternative aimed towards managing pipelines for neuroimaging data, whereas Snakemake is more popular in the bioinformatics community. Not only is nipype better known in the neuroimaging community, it 'out-of-the-box' provides bindings for other popular neuroimaging software such as SPM and FSL. Workflow tasks are currently designed using a custom set of methods that format tasks into Snakefiles, a Snakemake-specific syntax for writing workflows. Snakemake lacks the Python bindings for automatic workflow generation (and lacks documentation on its internal workings), preventing a pure Python solution. Nipype, on the other hand, allows tasks to be written entirely in Python in a well-documented format. Any researcher can follow nipype tutorials and use this information to extend or modify the CBPtools workflow to meet their own unique requirements.

To further increase the coherence, applicability, and extendibility of CBPtools various refactoring changes should be performed. Refactoring is a technique for restructuring the codebase without changing the outward-facing behaviour of the software. This includes rewriting code to make its structure more interpretable (i.e., improving coherence) by separating modules and submodules, using better suited design patterns and external modules to optimize execution time and efficiency, and use abstract and data classes to enhance readability and extendibility of the codebase. Design patterns, such as the template pattern, can be used to separate parts of the code such that improvements and modifications minimally impact unrelated or tangentially related functionality. Abstract base classes complement further development by defining mandatory methods and properties. They are then used as blueprints for other classes that interact with existing CBPtools functionality. For example, refactoring image processing functionality into "Image" classes (i.e., *Mask* and *TimeSeries* classes) will make the image processing steps more coherent by separating each individual step (e.g., smoothing, band-pass filtering, etc.). Together with identifiable naming conventions for methods, the code can then more easily be used as a reference for identifying processes involved in each task and in which order they are applied. Abstract base classes ensure that outward facing functionality remains consistent throughout the CBPtools software. All this should be accompanied by more explicit docstrings and type declarations to improve the automatic generation

of documentation, as well as dataclasses specifically designed to interpret configuration parameters and manage various parts of the dataset.

Lastly, thorough integration testing of the CBPtools codebase should be implemented. CBPtools has been extensively tested using exploratory testing, benchmarking, and its output has been compared to the output of external tools and research. However, with continued development of the pipeline exploratory testing does not suffice. Integration testing allows future commits to the CBPtools repository to be tested automatically against a predefined set of assertions. This significantly decreases the chance of software bugs when changes are made.

5.2 Example Data

The most important aim of this work is the provision of a functional rCBP procedure that results in biologically relevant and meaningful output. To illustrate usability and to analyse the CBPtools output, three functionally and spatially different ROIs were selected that underwent processing in the CBPtools workflow. ROI targets were selected because of their popularity in rCBP literature (e.g., the MFC is considered a gold standard for testing rCBP procedures, as it has a clear functional and structural distinction at the borders between what is known as the preSMA and SMA), making them easier to compare to external findings. To maintain consistency between the ROIs, the same CBPtools configuration parameters and data (including pre-processing procedures on said data) were used. The HCP data (Van Essen et al 2013) provided a sufficiently large sample that would not only highlight the ability of CBPtools to deal with many large input files, but also allowed for more relevant benchmarking.

Only recommended standard pre-processing techniques were applied to the HCP data (see Sect. 3.1 [p. 78]) prior to its application in CBPtools, to ensure that the resulting parcellations were based on methods implemented in CBPtools, rather than those used outside of it. This ascertained that the same methods would be readily available to those intending to use CBPtools for applying the rCBP procedure. The CBPtools configuration was set up such that methods most popular in prior literature (e.g., k-means as the clustering algorithm, and the Silhouette, Calinski-Harabasz, and Davies-Bouldin indices for cluster validity) were used, and the parameter values reflected default or recommended values where possible. All configuration values used in the example data are also defined as the CBPtools default values, excepting those

that require user definition. For example, a value that requires user definition is the range of k clusters to compute, as this depends crucially on the input data and ROI.

The effectiveness of CBPtools in procuring resting-state functional and diffusion MRI connectivity-derived parcellations on three functionally and spatially different ROIs has been demonstrated. As outlined in the discussion sections of each ROI (see Sect. 3.3.2 [p. 86], 3.4.2 [p. 89], and 3.5.2 [p. 96]) similar results were found in existing literature. The preSMA-SMA results showcase the ability of CBPtools, and by extension the rCBP approach, to reproduce histological parcellations. Although results at higher cluster granularities are added for completeness, their validity and relevance are not further investigated as the 2-cluster solution is the most accepted and widely reported solution in existing literature. The insula results highlight the subdivisions of various k cluster solutions, showing commonalities between the results and existing literature at several levels of clustering granularity. Despite differences between data modalities, overlap can be found between subdivisions even at higher clustering granularities. While this was not at all the case for the amygdala, revealing a stark difference between modalities, existing literature has found similar results. The peculiarity of the amygdala in terms of its spatial location, as well as the shape, form, and location of its clusters, led to further investigation of the validity of the results. These post-hoc analyses are not included in the CBPtools software as the most viable and relevant methods used to further investigate may differ considerably between ROIs. While these analyses did not contradict initial results, they do highlight the importance of thoroughly investigating parcellation results even beyond the scope of what CBPtools has to offer.

Divergence between validity indices (i.e., the Davies-Bouldin index from the other validity metrics) highlights the importance of choosing a proper validity metric, each of which assesses cluster validity in a unique way. Note that comparisons of validity scores outside of the sample are meaningless, hence rsfMRI and dMRI validity scores cannot be directly compared. For the R insula the divergence is not necessarily surprising, as it shows transitional changes in cytoarchitecture (Kurth et al 2010a), rather than sharp cytoarchitectonic borders present between the preSMA and SMA, making it difficult to define stable hard borders. Similarity of cluster labels between subject-wise clusterings may vary considerably. It is yet unclear what factors contribute to the high dissimilarity between subjects on some cluster solutions and for some ROIs. For instance, overlap between relabelled clusters for the individual clustering results to the group-level clustering results on the 2-cluster preSMA-SMA solution are high (see **Fig. 14b**), which may imply that the regions have strongly divergent connectivity patterns that are stable between subjects. However, regions

such as the amygdala show lower overlap values. This may in part be due to poor signal to noise ratio with MRI in subcortical regions (Noble et al 2017). Nevertheless, the solutions showcased here can be found in previously published works. However, no studies performing a data-driven clustering using dMRI data for the R amygdala at higher granularities was found.

Outliers in the subject-wise connectivity profiles were investigated and found to cluster differently. Hence, when subsequent clustering results are used in computing a group-level clustering, the ensuing clusters may not be representative of a general population. Further research is necessary to identify the reason behind the differences in connectivity profile and why they induce such marked changes in clustering. One potential reason may be the change in multiband reconstruction algorithm in the third quarter of the HCP data collection (Elam 2015), for which confound regressors can be created. However, the extent to which this confound can be removed is unknown. Another speculative candidate may be scanner drift (Dam 2017), as the data that was used in this work was collected over a long period and the metric used to identify outliers correlated significantly with the subject's acquisition time (see **Fig. 25**). As the constraints to data collection time cannot be circumvented for such large data sets, it is possible similar effects can be found in other large samples. However, a more in-depth analysis is necessary to make any statements beyond speculation at this point. Regardless of the underlying reason, it is wise to identify abnormalities within the data set prior to including subjects for the rCBP procedure and assess their influence on ensuing clustering results. CBPtools includes no facilities for outlier detection, as there is no gold standard means of their detection that is applicable, trustworthy, and relevant for each data set.

6 Conclusion

In this work the rCBP procedure is described within a historical context and modern uses of the approach are outlined. Indications have been made as to why the procedure is relevant to neuroscientists interested in brain mapping yet lacks publicly available tools to perform the approach from start to finish. This has culminated in the creation of the CBPtools Python software, used as an all-in-one package for performing rCBP from an rsfMRI or dMRI data set up until the clustering and validation results presented through figures and tables. Its procedure is outlined, including a thorough description of its architecture, scope, implementation, and usage. In addition, further quality of life features are highlighted. Lastly, its efficacy is demonstrated using three commonly parcellated ROIs on a substantial data set.

By using CBPtools, any scientist interested in performing the rCBP procedure can now easily and efficiently do so. Prior to its release, scientists would have to develop their own processing scripts which is not only a challenging, but also a time-consuming endeavour. The flexibility of existing features as well as the open-source and modular nature of the software allow for an abundance of applications. Future developments by first- or third-party developers may extend the software with suggestions for extensions listed in Sect. 5.1.1 [p. 108]. While further development is encouraged, the software can be used as-is to perform the most applied rCBP methods.

In summary, an openly distributed package has been provided for performing rCBP for which, to our knowledge, there is currently no alternative. By introducing CBPtools researchers are provided with the means to conduct reproducible, data-driven rCBP analyses on multiple neuroimaging modalities and large amounts of subject data.

Bibliography

Abraham A, Pedregosa F, Eickenberg M, et al (2014) Machine learning for neuroimaging with scikit-learn. Front Neuroinformatics 8:14. doi: 10.3389/fninf.2014.00014

Ackermann H, Riecker A (2004) The contribution of the insula to motor aspects of speech production: a review and a hypothesis. Brain Lang 89:320–328. doi: 10.1016/S0093-934X(03)00347-X

Adolphs R (2010) What does the amygdala contribute to social cognition? Ann N Y Acad Sci 1191:42–61. doi: 10.1111/j.1749-6632.2010.05445.x

Amunts K, Kedo O, Kindler M, et al (2005) Cytoarchitectonic mapping of the human amygdala, hippocampal region and entorhinal cortex: intersubject variability and probability maps. Anat Embryol (Berl) 210:343–352. doi: 10.1007/s00429-005-0025-5

Avants BB, Tustison NJ, Song G, et al (2011) A reproducible evaluation of ANTs similarity metric performance in brain image registration. Neuroimage 54:2033–2044. doi: 10.1016/j.neuroimage.2010.09.025

Avery JA, Kerr KL, Ingeholm JE, et al (2015) A common gustatory and interoceptive representation in the human mid-insula. Hum Brain Mapp 36:2996–3006. doi: 10.1002/hbm.22823

Baarsch J, Celebi ME (2012) Investigation of Internal Validity Measures for K-Means Clustering. Proceedings of the International MultiConference of Engineers and Computer Scientists I:14–16.

Bajada CJ, Jackson RL, Haroon HA, et al (2017) A graded tractographic parcellation of the temporal lobe. Neuroimage. doi: 10.1016/j.neuroimage.2017.04.016

Ball T, Derix J, Wentlandt J, et al (2009) Anatomical specificity of functional amygdala imaging of responses to stimuli with positive and negative emotional valence. J Neurosci Methods 180:57–70. doi: 10.1016/j.jneumeth.2009.02.022

Bamiou D-E, Musiek FE, Luxon LM (2003) The insula (Island of Reil) and its role in auditory processing. Brain Res Rev 42:143–154. doi: 10.1016/S0165-0173(03)00172-3

Barron DS, Eickhoff SB, Clos M, Fox PT (2015) Human pulvinar functional organization and connectivity. Hum Brain Mapp 36:2417–2431. doi: 10.1002/hbm.22781

Behrens TEJ, Johansen-Berg H, Woolrich MW, et al (2003) Non-invasive mapping of connections between human thalamus and cortex using diffusion imaging. Nat Neurosci 6:750–757. doi: 10.1038/nn1075

Bezdek JC, Ehrlich R, Full W (1984) FCM: The fuzzy c-means clustering algorithm. Comput Geosci 10:191–203. doi: 10.1016/0098-3004(84)90020-7

Biswal B, Yetkin FZ, Haughton VM, Hyde JS (1995) Functional connectivity in the motor cortex of resting human brain using echo-planar MRI. Magn Reson Med 34:537–541. doi: 10.1002/mrm.1910340409

Boccardi M, Pennanen C, Laakso MP, et al (2002) Amygdaloid atrophy in frontotemporal dementia and Alzheimer's disease. Neurosci Lett 335:139–143. doi: 10.1016/s0304-3940(02)01169-2

Brett M, Markiewicz CJ, Hanke M, et al (2019) nipy/nibabel: 2.4.0. Zenodo. doi: 10.5281/zenodo.2620614

Brett M, Poline JB (2016) Sometimes, the NIfTI image stores the TR in the header — Functional MRI methods. https://bic-berkeley.github.io/psych-214-fall-2016/tr_and_headers.html. Accessed 11 May 2021

Brodmann K (1909) Vergleichende Lokalisationslehre der Grosshirnrinde. Leipzig: Johann Ambrosius Barth

Brooks JCW, Zambreanu L, Godinez A, et al (2005) Somatotopic organisation of the human insula to painful heat studied with high resolution functional imaging. Neuroimage 27:201–209. doi: 10.1016/j.neuroimage.2005.03.041

Bzdok D, Heeger A, Langner R, et al (2015) Subspecialization in the human posterior medial cortex. Neuroimage 106:55–71. doi: 10.1016/j.neuroimage.2014.11.009

Bzdok D, Laird AR, Zilles K, et al (2013) An investigation of the structural, connectional, and functional subspecialization in the human amygdala. Hum Brain Mapp 34:3247–3266. doi: 10.1002/hbm.22138

Calinski T, Harabasz J (1974) A dendrite method for cluster analysis. Communications in Statistics - Theory and Methods 3:1–27. doi: 10.1080/03610927408827101

Caspers S, Moebus S, Lux S, et al (2014) Studying variability in human brain aging in a population-based German cohort-rationale and design of 1000BRAINS. Front Aging Neurosci 6:149. doi: 10.3389/fnagi.2014.00149

Cauda F, D'Agata F, Sacco K, et al (2011) Functional connectivity of the insula in the resting brain. Neuroimage 55:8–23. doi: 10.1016/j.neuroimage.2010.11.049

Chang LJ, Yarkoni T, Khaw MW, Sanfey AG (2013) Decoding the role of the insula in human cognition: functional parcellation and large-scale reverse inference. Cereb Cortex 23:739–749. doi: 10.1093/cercor/bhs065

Chase HW, Clos M, Dibble S, et al (2015) Evidence for an anterior-posterior differentiation in the human hippocampal formation revealed by meta-analytic parcellation of fMRI coordinate maps: focus on the subiculum. Neuroimage 113:44–60. doi: 10.1016/j.neuroimage.2015.02.069

Clos M, Amunts K, Laird AR, et al (2013) Tackling the multifunctional nature of Broca's region meta-analytically: co-activation-based parcellation of area 44. Neuroimage 83:174–188. doi: 10.1016/j.neuroimage.2013.06.041

Cohen AL, Fair DA, Dosenbach NUF, et al (2008) Defining functional areas in individual human brains using resting functional connectivity MRI. Neuroimage 41:45–57. doi: 10.1016/j.neuroimage.2008.01.066

Cox RW (1996) AFNI: software for analysis and visualization of functional magnetic resonance neuroimages. Comput Biomed Res 29:162–173. doi: 10.1006/cbmr.1996.0014

Dale AM, Fischl B, Sereno MI (1999) Cortical surface-based analysis. I. Segmentation and surface reconstruction. Neuroimage 9:179–194. doi: 10.1006/nimg.1998.0395

Dam EB (2017) Simple Methods for Scanner Drift Normalization Validated for Automatic Segmentation of Knee Magnetic Resonance Imaging-with data from the Osteoarthritis Initiative. arXiv preprint arXiv:1712.08425

Dambacher F, Sack AT, Lobbestael J, et al (2015) Out of control: evidence for anterior insula involvement in motor impulsivity and reactive aggression. Soc Cogn Affect Neurosci 10:508–516. doi: 10.1093/scan/nsu077

Davies DL, Bouldin DW (1979) A Cluster Separation Measure. IEEE Trans Pattern Anal Mach Intell PAMI-1:224–227. doi: 10.1109/TPAMI.1979.4766909

de Reus MA, van den Heuvel MP (2013) The parcellation-based connectome: limitations and extensions. Neuroimage 80:397–404. doi: 10.1016/j.neuroimage.2013.03.053

Deco G, Jirsa VK, McIntosh AR (2011) Emerging concepts for the dynamical organization of resting-state activity in the brain. Nat Rev Neurosci 12:43–56. doi: 10.1038/nrn2961

Deen B, Pitskel NB, Pelphrey KA (2011) Three systems of insular functional connectivity identified with cluster analysis. Cereb Cortex 21:1498–1506. doi: 10.1093/cercor/bhq186

Descoteaux M, Sidhu J, Garyfallidis E, et al (2016) False positive bundles in tractography.

Devlin JT, Poldrack RA (2007) In praise of tedious anatomy. Neuroimage 37:1033–41; discussion 1050. doi: 10.1016/j.neuroimage.2006.09.055

Duda RO, Hart PE, Stork DG (2001) Pattern Classification. J of Classification 24:305–307. doi: 10.1007/s00357-007-0015-9

Duncan J, Owen AM (2000) Common regions of the human frontal lobe recruited by diverse cognitive demands. Trends Neurosci 23:475–483. doi: 10.1016/s0166-2236(00)01633-7

Eickhoff SB, Bzdok D, Laird AR, et al (2012) Activation likelihood estimation meta-analysis revisited. Neuroimage 59:2349–2361. doi: 10.1016/j.neuroimage.2011.09.017

Eickhoff SB, Laird AR, Fox PT, et al (2016) Functional segregation of the human dorsomedial prefrontal cortex. Cereb Cortex 26:304–321. doi: 10.1093/cercor/bhu250

Eickhoff SB, Stephan KE, Mohlberg H, et al (2005) A new SPM toolbox for combining probabilistic cytoarchitectonic maps and functional imaging data. Neuroimage 25:1325–1335. doi: 10.1016/j.neuroimage.2004.12.034

Eickhoff SB, Thirion B, Varoquaux G, Bzdok D (2015) Connectivity-based parcellation: Critique and implications. Hum Brain Mapp 36:4771–4792. doi: 10.1002/hbm.22933

Eickhoff SB, Yeo BTT, Genon S (2018) Imaging-based parcellations of the human brain. Nat Rev Neurosci 19:672–686. doi: 10.1038/s41583-018-0071-7

Elam J (2015) Ramifications of Image Reconstruction Version Differences. In: Connectome Data Public - HCP Wiki. https://wiki.humanconnectome.org/display/PublicData/Ramifications+of+Image+Reconstruction+Version+Differences. Accessed 30 Aug 2020

Esteban O, Markiewicz CJ, Blair RW, et al (2019) fMRIPrep: a robust preprocessing pipeline for functional MRI. Nat Methods 16:111–116. doi: 10.1038/s41592-018-0235-4

Fan L, Li H, Zhuo J, et al (2016) The human brainnetome atlas: A new brain atlas based on connectional architecture. Cereb Cortex 26:3508–3526. doi: 10.1093/cercor/bhw157

Fecher B, Friesike S (2014) Open Science: One Term, Five Schools of Thought. In: Bartling S, Friesike S (eds) Opening Science. Springer International Publishing, Cham, pp 17–47

Fischl B, Sereno MI, Dale AM (1999) Cortical surface-based analysis. II: Inflation, flattening, and a surface-based coordinate system. Neuroimage 9:195–207. doi: 10.1006/nimg.1998.0396

Fox MD, Zhang D, Snyder AZ, Raichle ME (2009) The global signal and observed anticorrelated resting state brain networks. J Neurophysiol 101:3270–3283. doi: 10.1152/jn.90777.2008

Fox PT, Lancaster JL (2002) Opinion: Mapping context and content: the BrainMap model. Nat Rev Neurosci 3:319–321. doi: 10.1038/nrn789

García-Martí G, Aguilar EJ, Lull JJ, et al (2008) Schizophrenia with auditory hallucinations: a voxel-based morphometry study. Prog Neuropsychopharmacol Biol Psychiatry 32:72–80. doi: 10.1016/j.pnpbp.2007.07.014

Gasquoine PG (2014) Contributions of the insula to cognition and emotion. Neuropsychol Rev 24:77–87. doi: 10.1007/s11065-014-9246-9

Genon S, Li H, Fan L, et al (2017) The right dorsal premotor mosaic: organization, functions, and connectivity. Cereb Cortex 27:2095–2110. doi: 10.1093/cercor/bhw065

Genon S, Reid A, Li H, et al (2018) The heterogeneity of the left dorsal premotor cortex evidenced by multimodal connectivity-based parcellation and functional characterization. Neuroimage 170:400–411. doi: 10.1016/j.neuroimage.2017.02.034

Glasser MF, Sotiropoulos SN, Wilson JA, et al (2013) The minimal preprocessing pipelines for the Human Connectome Project. Neuroimage 80:105–124. doi: 10.1016/j.neuroimage.2013.04.127

Gorgolewski K, Burns CD, Madison C, et al (2011) Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. Front Neuroinformatics 5:13. doi: 10.3389/fninf.2011.00013

Gorgolewski KJ, Auer T, Calhoun VD, et al (2016) The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. Sci Data 3:160044. doi: 10.1038/sdata.2016.44

Gorgolewski KJ, Varoquaux G, Rivera G, et al (2015) NeuroVault.org: a web-based repository for collecting and sharing unthresholded statistical maps of the human brain. Front Neuroinformatics 9:8. doi: 10.3389/fninf.2015.00008

Halchenko YO, Hanke M (2012) Open is Not Enough. Let's Take the Next Step: An Integrated, Community-Driven Computing Platform for Neuroscience. Front Neuroinformatics 6:22. doi: 10.3389/fninf.2012.00022

Halchenko YO, Hanke M, Poldrack B, et al (2019) datalad/datalad 0.12.0rc6. Zenodo. doi: 10.5281/zenodo.3512712

Hardwick RM, Lesage E, Eickhoff CR, et al (2015) Multimodal connectivity of motor learning-related dorsal premotor cortex. Neuroimage 123:114–128. doi: 10.1016/j.neuroimage.2015.08.024

Hastie T, Tibshirani R, Friedman J (2013) The Elements of Statistical Learning: Data Mining, Inference, and Prediction, illustrated. Springer Science & Business Media

Hayman LA, Rexer JL, Pavol MA, et al (1998) Klüver-Bucy syndrome after bilateral selective damage of amygdala and its cortical connections. J Neuropsychiatry Clin Neurosci 10:354–358. doi: 10.1176/jnp.10.3.354

Janak PH, Tye KM (2015) From circuits to behaviour in the amygdala. Nature 517:284–292. doi: 10.1038/nature14188

Jenkinson M, Beckmann CF, Behrens TE, et al (2012) FSL. Neuroimage 62:782–790. doi: 10.1016/j.neuroimage.2011.09.015

Johansen-Berg H, Behrens TEJ, Robson MD, et al (2004) Changes in connectivity profiles define functionally distinct regions in human medial frontal cortex. Proc Natl Acad Sci USA 101:13335–13340. doi: 10.1073/pnas.0403743101

Jolkkonen E, Pitkänen A (1998) Intrinsic connections of the rat amygdaloid complex: projections originating in the central nucleus. J Comp Neurol 395:53–72. doi: 10.1002/(sici)1096-9861(19980525)395:1<53::aid-cne5>3.0.co;2-g

Kalavathi P, Prasath VBS (2016) Methods on Skull Stripping of MRI Head Scan Images-a Review. J Digit Imaging 29:365–379. doi: 10.1007/s10278-015-9847-8

Kanwisher N (2010) Functional specificity in the human brain: a window into the functional architecture of the mind. Proc Natl Acad Sci USA 107:11163–11170. doi: 10.1073/pnas.1005062107

Kelly C, Toro R, Di Martino A, et al (2012) A convergent functional architecture of the insula emerges across imaging modalities. Neuroimage 61:1129–1142. doi: 10.1016/j.neuroimage.2012.03.021

Kim J-H, Lee J-M, Jo HJ, et al (2010) Defining functional SMA and pre-SMA subregions in human MFC using resting state fMRI: functional connectivity-based parcellation method. Neuroimage 49:2375–2386. doi: 10.1016/j.neuroimage.2009.10.016

Klein JC, Behrens TEJ, Robson MD, et al (2007) Connectivity-based parcellation of human cortex using diffusion MRI: Establishing reproducibility, validity and observer independence in BA 44/45 and SMA/pre-SMA. Neuroimage 34:204–211. doi: 10.1016/j.neuroimage.2006.08.022

Köster J (2019) Snakemake — Snakemake 5.23.0 documentation. https://snakemake.readthedocs.io/en/stable/. Accessed 30 Aug 2020

Köster J, Rahmann S (2012) Snakemake--a scalable bioinformatics workflow engine. Bioinformatics 28:2520–2522. doi: 10.1093/bioinformatics/bts480

Kurth F, Eickhoff SB, Schleicher A, et al (2010a) Cytoarchitecture and probabilistic maps of the human posterior insular cortex. Cereb Cortex 20:1448–1461. doi: 10.1093/cercor/bhp208

Kurth F, Zilles K, Fox PT, et al (2010b) A link between the systems: functional differentiation and integration within the human insula revealed by meta-analysis. Brain Struct Funct 214:519–534. doi: 10.1007/s00429-010-0255-z

LeDoux J (2007) The amygdala. Curr Biol 17:R868–74. doi: 10.1016/j.cub.2007.08.005

Maulik U, Bandyopadhyay S (2002) Performance evaluation of some clustering algorithms and validity indices. IEEE Trans Pattern Anal Mach Intell 24:1650–1654. doi: 10.1109/TPAMI.2002.1114856

Mazzola L, Royet J-P, Catenoix H, et al (2017) Gustatory and olfactory responses to stimulation of the human insula. Ann Neurol 82:360–370. doi: 10.1002/ana.25010

Menon V, Uddin LQ (2010) Saliency, switching, attention and control: a network model of insula function. Brain Struct Funct 214:655–667. doi: 10.1007/s00429-010-0262-0

Mesulam MM, Mufson EJ (1982) Insula of the old world monkey. I. Architectonics in the insulo-orbito-temporal component of the paralimbic brain. J Comp Neurol 212:1–22. doi: 10.1002/cne.902120102

Mishra A, Rogers BP, Chen LM, Gore JC (2014) Functional connectivity-based parcellation of amygdala using self-organized mapping: a data driven approach. Hum Brain Mapp 35:1247–1260. doi: 10.1002/hbm.22249

Morris JS, Frith CD, Perrett DI, et al (1996) A differential neural response in the human amygdala to fearful and happy facial expressions. Nature 383:812–815. doi: 10.1038/383812a0

Morrison SE, Salzman CD (2010) Re-valuing the amygdala. Curr Opin Neurobiol 20:221–230. doi: 10.1016/j.conb.2010.02.007

Muhle-Karbe PS, Derrfuss J, Lynn MT, et al (2016) Co-Activation-Based Parcellation of the Lateral Prefrontal Cortex Delineates the Inferior Frontal Junction Area. Cereb Cortex 26:2225–2241. doi: 10.1093/cercor/bhv073

Mutschler I, Wieckhorst B, Kowalevski S, et al (2009) Functional organization of the human anterior insular cortex. Neurosci Lett 457:66–70. doi: 10.1016/j.neulet.2009.03.101

Nanetti L, Cerliani L, Gazzola V, et al (2009) Group analyses of connectivity-based cortical parcellation using repeated k-means clustering. Neuroimage 47:1666–1677. doi: 10.1016/j.neuroimage.2009.06.014

Nelson SM, Cohen AL, Power JD, et al (2010) A parcellation scheme for human left lateral parietal cortex. Neuron 67:156–170. doi: 10.1016/j.neuron.2010.05.025

Nguyen N, Caruana R (2007) Consensus Clusterings. Seventh IEEE International Conference on Data Mining (ICDM 2007). IEEE, pp 607–612

Noble S, Spann MN, Tokoglu F, et al (2017) Influences on the Test-Retest Reliability of Functional Connectivity MRI and its Relationship with Behavioral Utility. Cereb Cortex 27:5415–5429. doi: 10.1093/cercor/bhx230

Ogawa S, Lee TM, Kay AR, Tank DW (1990) Brain magnetic resonance imaging with contrast dependent on blood oxygenation. Proc Natl Acad Sci USA 87:9868–9872. doi: 10.1073/pnas.87.24.9868

Ordaz SJ, Lenroot RK, Wallace GL, et al (2010) Are there differences in brain morphometry between twins and unrelated singletons? A pediatric MRI study. Genes, Brain and Behavior 9:288–295. doi: 10.1111/j.1601-183X.2009.00558.x

Pedregosa F, Varoquaux G, Gramfort A, et al (2011) Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research

Phelps EA, LeDoux JE (2005) Contributions of the amygdala to emotion processing: from animal models to human behavior. Neuron 48:175–187. doi: 10.1016/j.neuron.2005.09.025

Pikkarainen M, Pitkänen A (2001) Projections from the lateral, basal and accessory basal nuclei of the amygdala to the perirhinal and postrhinal cortices in rat. Cereb Cortex 11:1064–1082. doi: 10.1093/cercor/11.11.1064

Plachti A, Eickhoff SB, Hoffstaedter F, et al (2019) Multimodal parcellations and extensive behavioral profiling tackling the hippocampus gradient. Cereb Cortex 29:4595–4612. doi: 10.1093/cercor/bhy336

Preston C, Ehrsson HH (2016) Illusory obesity triggers body dissatisfaction responses in the insula and anterior cingulate cortex. Cereb Cortex 26:4450–4460. doi: 10.1093/cercor/bhw313

Pruim RHR, Mennes M, van Rooij D, et al (2015) ICA-AROMA: A robust ICA-based strategy for removing motion artifacts from fMRI data. Neuroimage 112:267–277. doi: 10.1016/j.neuroimage.2015.02.064

Raichle ME, MacLeod AM, Snyder AZ, et al (2001) A default mode of brain function. Proc Natl Acad Sci USA 98:676–682. doi: 10.1073/pnas.98.2.676

Reil JC (1808) Untersuchungen über den Bau des grossen Gehirns im Menschen. Arch Physiol 9:136–208.

Reuter N, Genon S, Kharabian Masouleh S, et al (2020) CBPtools: a Python package for regional connectivity-based parcellation. Brain Struct Funct 225:1261–1275. doi: 10.1007/s00429-020-02046-1

Richie-Halford A, Keshavan A, Joseph M, et al (2020) dMRIPrep: a robust preprocessing pipeline for diffusion MRI. Zenodo. doi: 10.5281/zenodo.3609201

Riehle D, Ellenberger J, Menahem T, et al (2009) Open Collaboration within Corporations Using Software Forges. IEEE Softw 26:52–58. doi: 10.1109/MS.2009.44

Rodola G (2019) psutil documentation. In: psutil documentation. https://psutil.readthedocs.io/en/latest/. Accessed 18 Dec 2019

Rokem A, Trumpis M, Perez F (2009) Nitime: time-series analysis for neuroimaging data. Proc. 8th Python Sci. Conf. p 68

Rousseeuw PJ (1987) Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics 20:53–65. doi: 10.1016/0377-0427(87)90125-7

Ruan J, Bludau S, Palomero-Gallagher N, et al (2018) Cytoarchitecture, probability maps, and functions of the human supplementary and pre-supplementary motor areas. Brain Struct Funct 223:4169–4186. doi: 10.1007/s00429-018-1738-6

Rudenga K, Green B, Nachtigal D, Small DM (2010) Evidence for an integrated oral sensory module in the human anterior ventral insula. Chem Senses 35:693–703. doi: 10.1093/chemse/bjq068

Salimi-Khorshidi G, Douaud G, Beckmann CF, et al (2014) Automatic denoising of functional MRI data: combining independent component analysis and hierarchical fusion of classifiers. Neuroimage 90:449–468. doi: 10.1016/j.neuroimage.2013.11.046

Saygin ZM, Osher DE, Augustinack J, et al (2011) Connectivity-based segmentation of human amygdala nuclei using probabilistic tractography. Neuroimage 56:1353–1361. doi: 10.1016/j.neuroimage.2011.03.006

Schaefer A, Kong R, Gordon EM, et al (2018) Local-Global Parcellation of the Human Cerebral Cortex from Intrinsic Functional Connectivity MRI. Cereb Cortex 28:3095–3114. doi: 10.1093/cercor/bhx179

Smith SM, Beckmann CF, Andersson J, et al (2013) Resting-state fMRI in the Human Connectome Project. Neuroimage 80:144–168. doi: 10.1016/j.neuroimage.2013.05.039

Solano-Castiella E, Anwander A, Lohmann G, et al (2010) Diffusion tensor imaging segments the human amygdala in vivo. Neuroimage 49:2958–2965. doi: 10.1016/j.neuroimage.2009.11.027

Triarhou LC (2006) The signalling contributions of Constantin von Economo to basic, clinical and evolutionary neuroscience. Brain Res Bull 69:223–243. doi: 10.1016/j.brainresbull.2006.02.001

Van Essen DC, Smith SM, Barch DM, et al (2013) The WU-Minn Human Connectome Project: an overview. Neuroimage 80:62–79. doi: 10.1016/j.neuroimage.2013.05.041

van den Heuvel MP, Hulshoff Pol HE (2010) Specific somatotopic organization of functional connections of the primary motor network during resting state. Hum Brain Mapp 31:631–644. doi: 10.1002/hbm.20893

Von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17:395–416. doi: 10.1007/s11222-007-9033-z

von Economo CF, Koskinas GN (1925) Die cytoarchitektonik der hirnrinde des erwachsenen menschen. Die cytoarchitektonik der hirnrinde des erwachsenen menschen

Wager TD, Barrett LF (2004) From affect to control: Functional specialization of the insula in motivation and regulation. Published online at PsycExtra.

Wang J, Yang Y, Fan L, et al (2015) Convergent functional architecture of the superior parietal lobule unraveled with multimodal neuroimaging approaches. Hum Brain Mapp 36:238–257. doi: 10.1002/hbm.22626

Wen Q, Stirling BD, Sha L, et al (2016) Parcellation of Human Amygdala Subfields Using Orientation Distribution Function and Spectral K-means Clustering. Comput Diffus MRI (2016) 2016:123–132. doi: 10.1007/978-3-319-54130-3_10

Wiest G, Lehner-Baumgartner E, Baumgartner C (2006) Panic attacks in an individual with bilateral selective lesions of the amygdala. Arch Neurol 63:1798–1801. doi: 10.1001/archneur.63.12.1798

Wright IC, Sharma T, Ellison ZR, et al (1999) Supra-regional brain systems and the neuropathology of schizophrenia. Cereb Cortex 9:366–378. doi: 10.1093/cercor/9.4.366

Xia M, Wang J, He Y (2013) BrainNet Viewer: a network visualization tool for human brain connectomics. PLoS One 8:e68910. doi: 10.1371/journal.pone.0068910

Yarkoni T, Poldrack RA, Nichols TE, et al (2011) Large-scale automated synthesis of human functional neuroimaging data. Nat Methods 8:665–670. doi: 10.1038/nmeth.1635

Zalesky A, Fornito A, Cocchi L, et al (2016) Connectome sensitivity or specificity: which is more important? Neuroimage 142:407–420. doi: 10.1016/j.neuroimage.2016.06.035

Zhang X, Cheng H, Zuo Z, et al (2018) Individualized Functional Parcellation of the Human Amygdala Using a Semi-supervised Clustering Method: A 7T Resting State fMRI Study. Front Neurosci 12:270. doi: 10.3389/fnins.2018.00270

Zhang Y, Caspers S, Fan L, et al (2015) Robust brain parcellation using sparse representation on resting-state fMRI. Brain Struct Funct 220:3565–3579. doi: 10.1007/s00429-014-0874-x

Zilles K, Palomero-Gallagher N, Grefkes C, et al (2002) Architectonics of the human cerebral cortex and transmitter receptor fingerprints: reconciling functional neuroanatomy and neurochemistry. Eur Neuropsychopharmacol 12:587–599. doi: 10.1016/s0924-977x(02)00108-6

Abbreviations

ARI	Adjusted Rand Index
b0	Maximum intensity acquisition
BET	Brain Extraction (an FSL tool)
BOLD	Blood-Oxygen Level Dependent
CBP	Connectivity-Based Parcellation
CSF	Cerebro-Spinal Fluid
dMRI	Diffusion MRI
DTI	Diffusion Tensor Imaging
DWI	Diffusion-Weighted Imaging
FIX	FMRIB's ICA-based X-noiseifier
FWHM	Full Width at Half Maximum
HCP	Human Connectome Project
I/O	Input/Output (or read/write)
ICA	Independent Component Analysis
ICA-AROMA	ICA-based Automatic Removal Of Motion Artifacts
k	The number of clusters in k-means clustering
MACM	Meta-Analytic Connectivity Modelling
MB	Megabytes
MFC	Medial Frontal Cortex
MNI	Montreal Neurological Institute
MRI	Magnetic Resonance Imaging
NaN	Not a Number
PCA	Principal Component Analysis
preSMA	Presupplementary Motor Area
PSS	Proportional Set Size
R	Right
RAM	Random-Access Memory
rCBP	Regional CBP
ROI	Region of Interest
rsfMRI	Resting-State Functional MRI
RSS	Resident Set Size
SC	Structural Covariance
SMA	Supplementary Motor Area
T1w	T1-weighted
T2w	T2-weighted
USS	Unique Set Size
VMS	Virtual Memory Size
WM	White Matter
YAML	YAML Ain't Mark-up Language

Appendices

1. Built-in rules for input validation 132
2. Participants file example 135
3. Confounds file example 136
4. List of configuration parameters 137
5. Task input, output, and parameters 147
6. Benchmark results 151
7. Software Mentions 155
8. Python Package Mentions 157
9. Usage Example 157

Appendix 1: Built-in rules for input validation

Here all built-in rules and requirements are listed for input validation. All standard rules have the rule_ prefix, whereas custom rules (that only apply to one configuration parameter) are prefixed with rule_custom_. If a rule fails to validate, commonly a *RuleError* is raised (although other errors may be possible as well). This error is caught and logged, rather than expressed by halting further code execution. This is done so that all parameters may be assessed, and a complete log can be given of all problems in the configuration file.

`rule_required` This rule assesses both whether the entry is required (and therefore may not be left blank), and whether the entry has a dependency on another entry and if this dependency is met (e.g., the `time_series` parameter is only used if the modality is set to 'rsfmri', hence `time_series` depends on the modality field, and `modality = 'rsfmri'` is the requirement). If the dependency's value (or its default value in case it is left blank) does not meet the requirements, a *DependencyError* is raised. If the entry is not required and left blank, the default value will be used. In other cases, validation will pass without any further changes.

`rule_type` Checks if an entry has the correct object type (e.g., integer, string, boolean, float, or list). Iterable object types (except strings, which are iterable by character) will furthermore have the object type of their contents assessed. For example, a `list[string]` type declaration in the configuration schema first assesses whether the object is a list, and then whether all items in the object are of the string type. In case of floats, scientific notation may be used for very small numbers (excessively large numbers are not used). By default, these entries are typed as strings, hence why this rule also performs a conversion to float to ensure the entry meets its requirements.

`rule_contains` This rule checks whether the entry's value contains a must-have value. For example, a substring that must be present in a string (such as the `{participant_id}` wildcard substring that is required in some input file paths).

`rule_allowed` This rule will be used if an entry is only allowed to consist of values that match a list of predefined allowed values. The use of expansions (i.e., the * character) can be used. This rule differentiates iterables from non-iterables and ensures that *all* values in an iterable entry match the allowed

criteria. For example, a predefined list of allowed values may look like: 'kmeans', 'spectral', and 'agglomerative'. In this case, the entry must contain one of these values.

`rule_max` This rule ensures the entry is not larger than a given maximum value. This means that the entry can have a value up to and including the maximum value, but not beyond it.

`rule_min` The same as rule_max, except it ensures the entry is not smaller than a predefined minimum value.

`rule_maxlength` This rule ensures the entry does not contain more items than a predefined maximum length. For strings, it counts the number of characters. For iterables, it counts the number of objects in the iterable.

`rule_minlength` The same as rule_maxlength, except the entry may not contain fewer items than a predefined minimum length.

`rule_custom_bandpass` This is a custom rule for the bandpass filtering parameter. It ensures that the low-pass entry is larger than the high-pass entry.

`rule_custom_voxdim` This is a custom rule for the voxel dimensions parameter. It ensures that either exactly 1 or 3 values are entered in a list. It is common that voxels (3-dimensional) are equivalent in size on all dimensions, which is expressed by entering only one value. In cases where the sizes differ, three values can be entered. However, two values or more than three values cannot be interpreted and are hence not allowed.

`rule_custom_tr` A custom rule that logs a warning, rather than raising an error. The repetition time (TR) entry must be entered in seconds. It may mistakenly be entered in milliseconds. Hence, if the entry is larger than 100 (a TR at or above 100 is extremely unlikely) the user is warned that they may have made a mistake.

`rule_custom_agglomerative_linkage` A custom rule for the linkage field of agglomerative clustering. If the selected linkage is 'ward', then the distance metric *must* be 'Euclidean'. Since the default distance metric is 'Euclidean', this rule also passes when the distance metric entry is left blank.

`rule_custom_has_sessions` A custom rule for the sessions field. If multi-session input data is to be used, then the input data file paths must contain the `{session}` wildcard substring. This substring is replaced with the different session identifiers to find the data. Conversely, if the wildcard substring is given

but no sessions are specified, then there are no values to replace the wildcard. This will subsequently raise a *RuleError* as the file paths appear invalid.

`rule_custom_space_match` In CBPtools it is possible to provide masks per subject in their native space. This prevents group analysis, as there is no common reference space that can be used to compare images. When using a standard space ROI, it is possible to use the default MNI152 whole-brain grey matter mask as a target mask. In native space, a target mask must be defined per subject as there is no default. Furthermore, this rule ensures that the mask input file paths contain the `{participant_id}` wildcard substring, as the masks are now subject-specific. It is also not possible to use multi-session data with native masks, as the former requires data to be merged which may not be straightforward with multi-session data in native space.

`rule_custom_benchmarking` This rule ensures the psutil package is installed when benchmarking is requested. Benchmarking is not a feature that is enabled by default, hence foregoing installing psutil as a dependency, instead requiring a manual install. Note that snakemake uses psutil for benchmarking, and CBPtools uses snakemake's benchmarking utilities.

`rule_custom_spectral_kernel` When selecting the 'precomputed' kernel for spectral clustering, the input data must consist of adjacency matrices. This can be done by setting the modality to 'connectivity' and, instead of regular connectivity matrices, provide adjacency matrices as input. Note that the distinction between the two matrices cannot reliably be made, hence this is not validated.

`rule_custom_references` Median filtering cannot be used when using reference images, because reference images must match the ROI precisely in terms of voxel coordinates and number of voxels. Median filtering alters the ROI and cannot be applied to a non-binary image. Since the reference images contain at least two clusters, they are not binary and cannot be median filtered.

`rule_custom_resample` Resampling of subject-specific (i.e., native) masks cannot be used on the ROI mask if the xfm and inv_xfm entries are provided for dMRI data. When using these input files, the transformations will instead take place using the probtrackx2 tool.

`rule_custom_has_inv_xfm` If the xfm entry is defined, then the inv_xfm entry must also be defined (and vice versa).

Appendix 2: Participants file example

Table 3 is an example of what a 'participants' file may look like in its tabular format. Note that the file contents itself must use a consistent separator (e.g., a tab or a comma). CBPtools only uses the `participant_id` column (regardless of its location) and requires a header that names one column in the file as such. The age and gender columns are added to illustrate that a 'participants' file may contain more data than just the participant-id, as this will not obstruct CBPtools in its functioning.

Table 3. Participants file example

participant_id	age	gender
sub-001	23	M
sub-002	28	M
sub-003	36	F
sub-004	24	M
sub-005	23	F
…	…	…

Appendix 3: Confounds file example

Table 4 is an excerpt from a real confounds file. A typical confounds file consists of linear (*1) and quadratic (*2) grey matter (GM), white matter (WM), cerebrospinal fluid (CSF) and global signal time-series. Additionally, 24 motion parameter time-series are added for pitch, roll, yaw, motion in the x, y, and z direction, as well as their quadratic components and of all these the derivatives. If a linear trend or constant should be added (which is recommended), they should be added as additional columns. The number of rows in the confounds file must be equivalent to the number of timepoints in the rsfMRI time-series (excluding the header row).

Table 4. Confounds file example

gm1	wm1	csf1	global1	gm2	...	motion1	...	motion24
-1.10036	-0.82036	-1.47967	-1.10691	0.174062	...	1.916362	...	-0.64127
-1.3962	-1.06832	-1.32441	-1.3171	0.781524	...	-0.44798	...	-0.51575
-2.21392	-1.54627	-2.02818	-2.03058	3.209518	...	1.818313	...	-0.05313
-0.64252	-0.73947	-0.75407	-0.71124	-0.48225	...	-0.42983	...	1.477089
-0.64055	-0.59776	-0.78717	-0.67019	-0.48433	...	-0.42332	...	-0.50371
...	...	...	...	...		...		...

Appendix 4: List of configuration parameters

Below is a list of all configuration parameters, divided into two sections: a section for input file paths and parameters, and a section for processing parameters. The configuration file is in the YAML format which allows nesting of key/value pairs. The arrow (→) indicates that the key on the right is nested in the key to the left. Each parameter will have its value type, default value, and dependencies listed. These indicate the type(s) of validation performed on the parameter. More about validation can be read in Sect. 2.1.1 [p. 39] and Appx. 1.

Input data parameters

participants → file:

`string, required, allowed=['*.tsv', '*.csv', '*.xls', '*.xlsx']`

Path to a tabular file containing a column with participant-ids. When a {participant_id} wildcard substring is requested, then the file path must contain this template at the place where otherwise the participant-id would go. This wildcard will be replaced by the actual participant-ids during execution of the pipeline.

participants → delimiter:

`string`

Delimiter to use (e.g., '\t' for tab-, or ',' for comma-delimited files).

participants → index_column:

`string`

Name of the column containing the participant-ids.

session:

`list[string]`

If multiple sessions are to be used for each subject, enter the sessions as partial paths here. For example, ['sess1', 'sess2'] will replace the {session} wildcard substring used in time_series and confounds with both 'sess1' and 'sess2', similar to how {participant_id} is replaced with the subject-id.

time_series:

`string, required, allowed=['*.nii', '*.nii.gz'], dependency(modality == 'rsfmri')`

Path to a 4D NIfTI image containing the resting-state time-series (x, y, z, timepoints). The time-series shape and affine must match that of the

seed and target masks. This field must contain the {participant_id} wildcard substring, as there should be one time-series image per subject.

confounds → file:

`string, allowed=['*.tsv', '*.csv'], dependency(modality == 'rsfmri')`

Path to a delimited (e.g., .tsv for tab-delimited) file with a confound signal per columns and a 1-line header. Columns can be selected using the columns parameter. The column length (i.e., number of rows) must match the length of the timepoints in the signal time-series.

confounds → delimiter:

`string, dependency(modality == 'rsfmri')`

Delimiter to use (e.g., '\t' for tab-, or ',' for comma-delimited files).

confounds → columns:

`list[string], dependency(modality == 'rsfmri')`

List of columns that should be used. If left empty, all columns are used. Otherwise only the selected columns will be used in nuisance signal regression. A glob pattern (*) can be used to select multiple columns that match the expression (e.g., 'motion-*' includes 'motion-x', 'motion-y', 'motion-z', etc.)

bet_binary_mask:

`string, required, allowed=['*.nii', '*.nii.gz'], dependency(modality == 'dmri')`

File path to a BET binary mask file. This field must contain the {participant_id} wildcard substring.

xfm:

`string, allowed=['*.nii', '*.nii.gz'], dependency(modality == 'dmri')`

Transform taking seed space to DTI space (either FLIRT matrix or FNIRT warpfield). This field must contain the {participant_id} wildcard substring.

inv_xfm:

`string, allowed=['*.nii', '*.nii.gz'], dependency(modality == 'dmri')`

Transform taking DTI space to seed space. This field must contain {participant_id} wildcard substring.

samples:

`string, required, dependency(modality == 'dmri')`

Merged samples derived from bedpostx output. This field must contain the {participant_id} wildcard substring. Note that this is the same file path that would otherwise be entered in FSL. It selects all files that

start with the entered file path (e.g., /path/to/samples/merged will take all files starting with merged in the /path/to/samples folder).

connectivity:

`string, required, allowed=['*.npy', '*.npz'], dependency(modality == 'connectivity')`

The path to a seed by target connectivity matrix. The number of seed voxels on the first dimension must match the number of seed voxels in the seed mask image. The order in which the seed voxels are listed along the first axis depends on the order that was used to extract the voxels from the mask (i.e., F- or C-contiguous order). CBPtools connectivity matrices have the seed voxels in C-order. This field must contain the {participant_id} wildcard substring. The extension must be either .npy or .npz. If the compressed .npz format is used, the array in the archive must be named connectivity.npy.

seed_coordinates:

`string, required, allowed=['*.npy'], dependency(modality == 'connectivity')`

The path to a 2D NumPy array file with shape of the number of seed voxels by 3. The file contains the 3D coordinates of each seed voxel in the order that the seed voxels appear in the connectivity matrix.

masks → seed:

`string, required, allowed=['*.nii', '*.nii.gz']`

Path to a binary region-of-interest NIfTI image in the same space as the time-series and target mask.

masks → region_id:

`list[integer], dependency(modality in ['rsfmri', 'dmri'])`

If an atlas is used as the seed, specify a list of integers containing the IDs of the regions that should be merged into the seed mask. If only one ID is given, the voxels carrying that ID in the atlas will become the seed mask. If multiple IDs are given, a composite binary mask will be generated of all selected regions in the atlas.

masks → target:

`string, allowed=['*.nii', '*.nii.gz'], dependency(modality in ['rsfmri', 'dmri'])`

Path to a binary NIfTI image covering a target region (e.g., the whole brain). If left empty, the MNI152 2mm grey-matter mask will be used as the default target mask.

masks → space:

`string, allowed=['standard', 'native']`

If native is used, then CBPtools assumes that both seed- and target masks are in the native space of each individual subject. This requires the {participant_id} wildcard substring to be present in the file path to the seed and target masks. Note that group results cannot be computed in native space and are therefore skipped. If standard is used, then the seed and target masks are assumed to be in the same group template space (e.g., MNI152 2mm space).

masks → resample:

`boolean`

Resample the seed and target masks to the space of the input data. This option will use the NiBabel function 'nibabel.processing.resample_to_from' with mode='nearest' and order=0. It is only used for single-subject parcellations (data.masks.space='native') when one seed and target mask are given rather than one seed and target mask per subject. For dMRI data it is only used on the seed image if the xfm and inv_xfm are not given.

masks → references:

`list[string], allowed=['*.nii', '*.nii.gz']`

Paths to one or more reference images. These images must be in the same space as the seed mask, cover the exact same voxels, and have at least 2 clusters. The reference images will be compared to the group clustering results. The comparison results will be provided as figures.

Processing parameters

masking → seed → binarization:

`float, default=0.0, dependency(modality in ['rsfmri', 'dmri'])`

Threshold above which voxels in the ROI mask image are defined as 1's. This is only applied if the mask is not binary.

masking → seed → median_filtering → apply:

`boolean, default=False, dependency(modality in ['rsfmri', 'dmri'])`

Apply median filtering to the ROI mask.

masking → seed → median_filtering → distance:

`integer, default=1, dependency(modality in ['rsfmri', 'dmri'])`

Median filtering distance in mm (i.e., the size of the area to compute the median from for each voxel).

masking → seed → upsample_to → apply:

`boolean, default=False, dependency(modality == 'dmri')`

Upsample the seed mask to the specified voxel size (e.g., from [3, 3, 3] as 3mm isotropic to [1, 1, 1] as 1mm isotropic). If left empty or as null, no upsampling will be done.

masking → seed → upsample_to → voxel_dimensions:

`list[float], default=[1, 1, 1], dependency(modality == 'dmri')`

The voxel dimensions to which the seed mask should be upsampled.

masking → target → binarization:

`float, default=0.0, dependency(modality in ['rsfmri', 'dmri'])`

Threshold above which voxels in the target mask image are defined as 1's. This is only applied if the mask is not binary.

masking → target → remove_seed → apply:

`boolean, default=False, dependency(modality in ['rsfmri', 'dmri'])`

Remove the seed voxels from the target mask.

masking → target → remove_seed → distance:

`integer, default=0, dependency(modality in ['rsfmri', 'dmri'])`

Expand the border around the seed mask (in millimetre) for removal from the target mask. This should only be applied if the input time-series data is smoothed, using the smoothing kernel as a value for this parameter.

masking → target → subsampling:

`boolean, default=True, dependency(modality == 'rsfmri')`

Apply subsampling to the target mask to improve computational efficiency at minimal loss of specificity. This removes every second voxel from the mask and is only recommended if the data has been smoothed.

masking → target → downsample_to → apply:

`boolean, default=False, dependency(modality == 'dmri')`

Downsample the target mask to the specified voxel size, similar to how upsample_seed_to works.

masking → target → downsample_to → voxel_dimesnions:

`list[float], default=[3, 3, 3], dependency(modality == 'dmri')`

The voxel dimensions to which the target mask should be downsampled.

connectivity → low_variance_error → apply:

`boolean, default=True, dependency(modality == 'rsfmri')`

When more than the specified ratio of voxels within the seed has extremely low or no variance over the entire time course, the processing of this participant will not continue. A detailed error report is provided

once all connectivity is processed and further processing is halted until the problems are resolved.

connectivity → low_variance_error → in_seed:
`float, default=0.05, dependency(modality == 'rsfmri')`
Ratio of allowed low-variance voxels occurring within the seed region.

connectivity → low_variance_error → in_target:
`float, default=0.1, dependency(modality == 'rsfmri')`
Ratio of allowed low-variance voxels occurring within the target region.

connectivity → band_pass_filtering → apply:
`boolean, default=False, dependency(modality == 'rsfmri')`
Perform band-pass filtering on the signal time-series.

connectivity → band_pass_filtering → band:
`list[float], default=[0.01, 0.08], dependency(modality == 'rsfmri')`
High- and low-pass value (respectively) for the band-pass filter. Note that if this value is set, TR should also be defined.

connectivity → band_pass_filtering → tr:
`float, default=None, dependency(modality == 'rsfmri')`
Repetition time in seconds

connectivity → smoothing → apply:
`boolean, default=False, dependency(modality == 'rsfmri')`
Apply smoothing to the signal time-series.

connectivity → smoothing → fwhm:
`integer, default=5, dependency(modality == 'rsfmri')`
FWHM kernel value for smoothing.

connectivity → arctanh_transform → apply:
`boolean, default=True, dependency(modality == 'rsfmri')`
Apply an arctanh transform to the connectivity matrix.

connectivity → pca_transform → apply:
`boolean, default=False, dependency(modality in ['rsfmri', 'dmri'])`
Apply a PCA transform to the connectivity matrix.

connectivity → pca_transform → components:
`float or integer, default=0.95, dependency(modality in ['rsfmri', 'dmri'])`
Number of components to keep (if an integer above or at 1) or amount of explained variance (if a float below 1). This value is equivalent to n_components in sklearn.decomposition.PCA.

connectivity → dist_thresh:

`float, default=5.0, dependency(modality == 'dmri')`

Discards samples shorter than this threshold in millimetre (see probtrackx2 documentation).

connectivity → loop_check:

`boolean, default=True, dependency(modality == 'dmri')`

Perform loop checks on paths – slower, but allows curvature threshold (see probtrackx2 documentation).

connectivity → c_thresh:

`float, default=0.2, dependency(modality == 'dmri')`

Curvature threshold (see probtrackx2 documentation).

connectivity → step_length:

`float, default=0.5, dependency(modality == 'dmri')`

Step length in millimeter (see probtrackx2 documentation).

connectivity → n_samples:

`integer, default=5000, dependency(modality == 'dmri')`

Number of samples (see probtrackx2 documentation).

connectivity → n_steps:

`integer, default=2000, dependency(modality == 'dmri')`

Number of steps per sample (see probtrackx2 documentation).

connectivity → correct_path_distribution:

`boolean, default=True, dependency(modality == 'dmri')`

Correct path distribution for the length of the pathways (see probtrackx2 documentation).

connectivity → cubic_transform → apply:

`boolean, default=True, dependency(modality == 'dmri')`

Apply a cubic transformation on the connectivity matrix.

connectivity → cleanup_fsl:

`boolean, default=True, dependency(modality == 'dmri')`

Remove all files created by probtrackx2 (except fdt_matrix2.dot) after the connectivity matrix has been extracted.

clustering → method:

`string, default='kmeans', allowed=['kmeans', 'spectral', 'agglomerative']`

Clustering method to be used, either k-means, spectral clustering, or agglomerative (hierarchical) clustering.

clustering → n_clusters:

`list[integer], default=[], required`

A list of cluster numbers to be evaluated (entered as [2, 3, 8] to receive a 2, 3, and 8-cluster solution).

clustering → cluster_options → algorithm:

`string, default='auto', allowed=['auto', 'full', 'elkan']`

k-means algorithm to use (see sklearn.cluster.KMeans).

clustering → cluster_options → init:

`string, default='k-means++', allowed=['k-means++', 'random']`

Method for initialization (see sklearn.cluster.KMeans).

clustering → cluster_options → max_iter:

`integer, default=10000`

Maximum number of iterations of the k-means algorithm for a single run (see sklearn.cluster.KMeans).

clustering → cluster_options → n_init:

`integer, default=256`

Number of times the k-means algorithm will be run with different centroid seeds (see sklearn.cluster.KMeans or sklearn.cluster.SpectralClustering).

clustering → cluster_options → kernel:

`string, default='nearest_neigbors', allowed=['additive_chi2', 'chi2', 'linear', 'polynomial', 'rbf', 'laplacian', 'sigmoid', 'cosine', 'nearest_neighbors', 'precomputed', 'precomputed_nearest_neighbors']`

Kernel to be used (see the affinity parameter in sklearn.cluster.SpectralClustering).

clustering → cluster_options → gamma:

`float, default=None`

Kernel coefficient for rbf, poly, sigmoid, Laplacian, and chi2 kernels. Ignored when kernel='nearest_neighbors' (see sklearn.cluster.SpectralClustering).

clustering → cluster_options → n_neighbors:

`integer, default=10`

Number of neighbours to use when constructing the affinity matrix using the nearest neighbours method (see sklearn.cluster.SpectralClustering).

clustering → cluster_options → assign_labels:

`string, default='kmeans', allowed=['kmeans', 'discretize']`

The strategy to use to assign labels in the embedding space (see sklearn.cluster.SpectralClustering).

clustering → cluster_options → degree:

`float, default=3.0`

Degree of the polynomial kernel (see sklearn.cluster.SpectralClustering).

clustering → cluster_options → coef0:

`float, default=1.0`

Zero coefficient for polynomial and sigmoid kernels (see sklearn.cluster.SpectralClustering).

clustering → cluster_options → eigen_tol:

`float, default=1e-10`

Stopping criterion for eigendecomposition of the Laplacian matrix when eigen_solver='arpack' (see sklearn.cluster.SpectralClustering).

clustering → cluster_options → eigen_solver:

`string, default=None, allowed=[None, 'arpack', 'lobpcg', 'amg']`

The eigenvalue decomposition strategy to use. AMG requires pyamg to be installed (see sklearn.cluster.SpectralClustering).

clustering → cluster_options → distance_metric:

`string, default='euclidean', allowed=['euclidean', 'l1', 'l2', 'manhattan', 'cosine']`

Metric to compute the linkage. If linkage is 'ward', only 'euclidean' is accepted (see sklearn.cluster. AgglomerativeClustering).

clustering → cluster_options → linkage:

`string, default='ward', allowed=['ward', 'complete', 'average', 'single']`

Which linkage criterion to use (see sklearn.cluster.AgglomerativeClustering).

clustering → grouping → linkage:

`string, default='complete', allowed=['complete', 'average', 'single']`

The linkage algorithm to use (see scipy.cluster.hierarchy.linkage).

clustering → grouping → method:

`string, default='mode', allowed=['mode', 'agglomerative']`

Method for grouping the clustering results of all subjects.

clustering → validity → internal:

`list[string], default=['silhouette_score'], allowed=['silhouette_score',`

`'davies_bouldin_score', 'calinski_harabasz_score']`

List of internal validity metrics to use for cluster validity assessment.

clustering → validity → similarity:

`string, default='adjusted_rand_score', allowed['adjusted_rand_score', 'adjusted_mutual_info_score', 'v_measure_score']`

Similarity metric to use to generate between-subject cluster comparisons and subject to group-level cluster comparisons.

clustering → report → figure_format:

`string, default='png', allowed['png', 'svg', 'pdf', 'ps', 'eps']`

Format of the output figures generated for the summary.

clustering → report → individual_plots:

`boolean, default=False`

Generate cluster-labelled ROI voxel plots for each individual subject (in addition to the group clustering results).

clustering → report → benchmark:

`boolean, default=False`

Benchmark the execution of each workflow task. The psutil package must be installed for benchmarking.

clustering → report → compress_output:

`boolean, default=True`

Compress interim output (i.e., NIfTI images and NumPy arrays) to reduce the file size. This comes at the cost of slower processing speed.

Appendix 5: Task input, output, and parameters

Below is a list of all tasks, detailing their parameters, input files, output files, as including logging and benchmarking output files. Some task input file paths are defined in the configuration file (e.g., `data.masks.seed` in the masking task), hence their input is listed in the parameters section. The output sections always refer to the relative (to the project directory) file path of the file that is created. Parameters refer to the key:value pairs in the YAML configuration file where each dot indicates that the key on the right side is nested into the key on the left side.

Wildcards (the words between curly brackets, such as `{participant_id}` and `{session}`) are placeholders for variable content. For example, `{participant_id}` is replaced with the participant-id in the data set (such as for file path completion) and `{session}` is replaced with the strings in the `data.session` field. This applies to the output, as well as the logging and benchmark output files.

Table 5. Masking task parameters, input, and output

Parameters	data.masks.region_id (optional) parameters.masking.seed.binarization (optional) parameters.masking.seed.median_filtering (optional) parameters.masking.seed.upsample_to (optional, dMRI only) parameters.masking.target.binarization (optional) parameters.masking.target.remove_seed (optional) parameters.masking.target.subsampling (optional, rsfMRI only) parameters.masking.target.downsample_to (optional, dMRI only)
Input	data.masks.seed data.masks.target (optional)
Output	*seed_mask.nii.gz* *target_mask.nii.gz* *seed_coordinates.npy* *highres_seed_mask.nii.gz* (optional, dMRI only)
Logging	*log/process_masks_rsfmri.log* (rsfMRI only) *log/process_masks_dmri.log* (dMRI only)
Benchmarking	*benchmarks/process_masks_rsfmri.log* (rsfMRI only) *benchmarks/process_masks.dmri.log* (dMRI only)

Table 6. rsfMRI Connectivity task parameters, input, and output

Parameters	data.session (optional) parameters.report.compress_output (optional) parameters.connectivity.smoothing (optional) parameters.connectivity.low_variance_error parameters.connectivity.band_pass_filtering (optional) parameters.connectivity.pca_transform (optional) parameters.connectivity.arctanh_transform (optional)
Input	*seed_mask.nii.gz*

	target_mask.nii.gz data.time_series data.confounds (optional)
Output	*individual/{participant_id}/connectivity.npz* *individual/{participant_id}/connectivity_{session}.npz* (temporary file; multi-session)
Logging	*log/{participant_id}.connectivity_rsfmri.log* (single-session) *log/{participant_id}.{session}.connectivity_rsfmri.log* (multi-session) *log/{participant_id}.merge_sessions.log* (multi-session)
Benchmarking	*benchmarks/{participant_id}.connectivity_rsfmri.log* (single-session) *benchmarks/{participant_id}.{session}.connectivity_rsfmri.log* (multi-session) *benchmarks/{participant_id}.merge_sessions.log* (multi-session)

Table 7. dMRI Connectivity task parameters, input, and output

Parameters	parameters.connectivity.dist_thresh parameters.connectivity.c_thresh parameters.connectivity.step_length parameters.connectivity.n_samples parameters.connectivity.n_steps parameters.connectivity.loop_check parameters.connectivity.correct_path_distribution parameters.connectivity.cubic_transform (optional) parameters.connectivity.pca_transform (optional) parameters.report.compress_output (optional) parameters.connectivity.cleanup_fsl (optional)
Input	*seed_mask.nii.gz* *highres_seed_mask.nii.gz* (optional) *target_mask.nii.gz* data.bet_binary_mask data.xfm data.inv_xfm data.samples
Output	*individual/{participant_id}/probtrackx2* (temporary folder) *individual/{participant_id}/probtrackx2_{session}* (temporary folder; multi-session) *individual/{participant_id}/connectivity.npz* *individual/{participant_id}/connectivity_{session}.npz* (temporary file; multi-session)
Logging	*log/{participant_id}.connectivity_dmri.log* (single-session) *log/{participant_id}.{session}.connectivity_dmri.log* (multi-session) *log/{participant_id}.merge_sessions.log* (multi-session)
Benchmarking	*benchmarks/{participant_id}.probtrackx2.log* (single-session) *benchmarks/{participant_id}.{session}.probtrackx2.log* (multi-session) *benchmarks/{participant_id}.connectivity_dmri.log* (single-session) *benchmarks/{participant_id}.{session}.connectivity_dmri.log* (multi-session) *benchmarks/{participant_id}.merge_sessions.log* (multi-session)

Table 8. Clustering (k-means) task parameters, input, and output

Parameters	parameters.clustering.n_clusters parameters.clustering.cluster_options.algorithm parameters.clustering.cluster_options.init parameters.clustering.cluster_options.max_iter parameters.clustering.cluster_options.n_init

Input	*individual/{participant_id}/connectivity.npz*
Output	*individual/{participant_id}/{n_clusters}cluster_labels.npy*
Logging	*log/{participant_id}.k{n_clusters}.kmeans_clustering.log*
Benchmarking	*benchmarks/{participant_id}.k{n_clusters}.kmeans_clustering.log*

Table 9. Clustering (spectral) task parameters, input, and output

Parameters	parameters.clustering.n_clusters parameters.clustering.cluster_options.n_init parameters.clustering.cluster_options.kernel parameters.clustering.cluster_options.gamma parameters.clustering.cluster_options.n_neighbors parameters.clustering.cluster_options.assign_labels parameters.clustering.cluster_options.degree parameters.clustering.cluster_options.coef0 parameters.clustering.cluster_options.eigen_tol parameters.clustering.cluster_options.eigen_solver
Input	*individual/{participant_id}/connectivity.npz*
Output	*individual/{participant_id}/{n_clusters}cluster_labels.npy*
Logging	*log/{participant_id}.k{n_clusters}.spectral_clustering.log*
Benchmarking	*benchmarks/{participant_id}.k{n_clusters}.spectral_clustering.log*

Table 10. Clustering (agglomerative) task parameters, input, and output

Parameters	parameters.clustering.n_clusters parameters.clustering.cluster_options.distance_metric parameters.clustering.cluster_options.linkage
Input	*individual/{participant_id}/connectivity.npz*
Output	*individual/{participant_id}/{n_clusters}cluster_labels.npy*
Logging	*log/{participant_id}.k{n_clusters}.agglomerative_clustering.log*
Benchmarking	*benchmarks/{participant_id}.k{n_clusters}.agglomerative_clustering.log*

Table 11. Grouping task parameters, input, and output

Parameters	parameters.clustering.grouping.linkage parameters.clustering.grouping.method
Input	*individual/{participant_id}/{n_clusters}cluster_labels.npy* (for all subjects at once) data.seed_coordinates (connectivity only)
Output	*group/{n_clusters}clusters/labels.npz* *group/{n_clusters}clusters/labeled_roi.nii.gz*
Logging	*log/k{n_clusters}.group_level_clustering.log*
Benchmarking	*benchmarks/k{n_clusters}.group_level_clustering.log*

Table 12. Validity task parameters, input, and output

Parameters	parameters.clustering.validity.internal

Input	*individual/{participant_id}/connectivity.npz* *individual/{participant_id}/{n_clusters}cluster_labels.npy (for all n_cluster solutions)*
Output	*individual/{participant_id}/internal_validity.tsv (temporary file)* *individual/internal_validity.tsv (merged over all subjects)*
Benchmarking	*benchmarks/{participant_id}.internal_validity.log* *benchmarks/merge_internal_validity.log*

Table 13. Similarity task parameters, input, and output

Parameters	data.references (optional) parameters.clustering.validity.similarity parameters.clustering.n_clusters
Input	*individual/{participant_id}/{n_clusters}cluster_labels.npy*
Output	*group/{n_clusters}clusters/individual_similarity.npy* *group/group_similarity.tsv* *group/cophenetic_correlation.tsv* *group/reference_similarity.tsv* (optional)
Benchmarking	*benchmarks/k{n_clusters}.individual_similarity.log* *benchmarks/group_similarity.log* *benchmarks/reference_similarity.log* (optional)

Table 14. Report task parameters, input, and output

Parameters	parameters.report.figure_format parameters.clustering.validity.similarity parameters.clustering.n_clusters
Input	*seed_mask.npy* *seed_coordinates.npy* (rsfMRI and dMRI only) data.seed_coordinates (connectivity only) *group/{n_clusters}clusters/individual_similarity.npy* *group/group_similarity.tsv* *group/cophenetic_correlation.tsv* *group/reference_similarity.tsv* (optional) *individual/internal_validity.tsv*
Output	*individual/internal_validity_{metric}.[png, svg, pdf, ps, eps]* *group/{n_clusters}clusters/individual_similarity_heatmap.png* *group/{n_clusters}clusters/individual_similarity_clustermap.png* *group/group_similarity.[png, svg, pdf, ps, eps]* *group/relabeling_accuracy.[png, svg, pdf, ps, eps]* *group/cophenetic_correlation.[png, svg, pdf, ps, eps]* *group/{n_clusters}clusters/voxel_plot_{view}.[png, svg, pdf, ps, eps]* *individual/{participant_id}/{n_clusters}cluster_voxel_plot_{view}..[png, svg, pdf, ps, eps]* (optional) *group/reference_similarity.[png, svg, pdf, ps, eps]*
Benchmarking	*benchmarks/{metric}.plot_internal_validity.log* *benchmarks/k{n_clusters}.plot_individual_similarity.log* *benchmarks/plot_group_similarity.log* *benchmarks/k{n_clusters}.{view}.plot_labeled_roi.log* *benchmarks/{participant_id}.k{n_clusters}.{view}.plot_individual_labeled_roi.log* (optional) *benchmarks/plot_reference_similarity.log* (optional)

Appendix 6: Benchmark results

All benchmarking results are separated by CBPtools processing task and pertain to the preSMA-SMA parcellation, either for the rsfMRI modality or the dMRI modality.

Table 15. Maximum RSS in MB (rsfMRI)

task	mean	std	min	max	n jobs
process masks	134.43	-	-	-	1
connectivity	468.99	637.46	205.34	3532.4	300
kmeans clustering	393.53	39.5986	318.15	464.35	1200
internal validity	298.86	11.713	283.92	373.33	300
group-level clustering	152.99	2.34687	149.63	155.21	4
group similarity	147.43	-	-	-	1
plot individual similarity	167.63	0.180278	167.5	167.94	4
plot group similarity	134.37	-	-	-	1
plot labelled ROI	158.79	0.244233	158.11	159.27	24
plot internal validity	137.63	2.22257	134.51	139.52	3

Table 16. Maximum VMS in MB (rsfMRI)

task	mean	std	min	max	n jobs
process masks	3191.87	-	-	-	1
connectivity	7465.29	36.2322	6958.66	7556.79	300
kmeans clustering	3644.03	38.031	3599.16	3676.32	1200
internal validity	3495.10	24.345	3459.93	3564.05	300
group-level clustering	3322.00	31.1755	3268	3340.02	4
group similarity	3338.63	-	-	-	1
plot individual similarity	3372.64	0	3372.64	3372.64	4
plot group similarity	3191.89	-	-	-	1
plot labelled ROI	3383.47	0.0128493	3383.45	3383.5	24
plot internal validity	3261.89	49.4904	3191.9	3296.89	3

Table 17. Maximum USS in MB (rsfMRI)

task	mean	std	min	max	n jobs
process masks	95.21	-	-	-	1
connectivity	427.68	642.234	162.85	3490.46	300

kmeans clustering	352.90	37.9777	276.42	418.92	1200
internal validity	257.04	11.8793	240.17	332.17	300
group-level clustering	109.41	2.91085	105.21	112.8	4
group similarity	102.34	-	-	-	1
plot individual similarity	124.50	1.74108	123.23	127.49	4
plot group similarity	90.58	-	-	-	1
plot labelled ROI	114.10	0.109287	113.89	114.25	24
plot internal validity	92.98	1.66349	90.64	94.36	3

Table 18. Maximum PSS in MB (rsfMRI)

task	mean	std	min	max	n jobs
process masks	114.06	-	-	-	1
connectivity	433.62	642.058	173.45	3497.87	300
kmeans clustering	367.71	37.9632	291.49	434.04	1200
internal validity	272.84	11.7443	255.77	346.41	300
group-level clustering	120.80	4.83297	114.68	126.69	4
group similarity	110.40	-	-	-	1
plot individual similarity	140.05	6.08659	133.26	147	4
plot group similarity	98.17	-	-	-	1
plot labelled ROI	116.17	0.71444	115.42	117.78	24
plot internal validity	99.26	1.82353	96.69	100.73	3

Table 19. Maximum CPU load per second in MB (rsfMRI)

task	mean	std	min	max	n jobs
process masks	0.00	-	-	-	1
connectivity	3.36	3.94097	0.54	21.85	300
kmeans clustering	260.10	48.1961	190.88	428.98	1200
internal validity	369.16	25.1319	300.23	449.27	300
group-level clustering	36.26	30.492	0	83.56	4
group similarity	92.30	-	-	-	1
plot individual similarity	111.55	2.65929	107.93	115.21	4
plot group similarity	0.00	-	-	-	1
plot labelled ROI	71.60	9.28152	57.32	86.26	24
plot internal validity	0.00	0	0	0	3

Table 20. Maximum RSS in MB (dMRI)

task	mean	std	min	max	n jobs
process masks	134.91	-	-	-	1
connectivity	1151.82	108.917	859.34	1503.99	300
kmeans clustering	748.99	97.3975	547.84	915.24	1200
internal validity	468.60	31.9433	418.03	613.05	300
group-level clustering	151.87	2.41363	149.03	155.39	4
group similarity	146.80	-	-	-	1
plot individual similarity	164.48	5.83578	154.38	168.08	4
plot group similarity	134.84	-	-	-	1
plot labelled ROI	158.76	0.30115	158.21	159.38	24
plot internal validity	134.50	0.257207	134.14	134.7	3

Table 21. Maximum VMS in MB (dMRI)

task	mean	std	min	max	n jobs
process masks	3191.92	-	-	-	1
connectivity	4410.93	83.9623	4217.11	4737.08	300
kmeans clustering	4030.72	88.4106	3733.02	4098.15	1200
internal validity	3696.22	38.0761	3636.38	3829.43	300
group-level clustering	3304.01	36.005	3267.99	3340.02	4
group similarity	3338.65	-	-	-	1
plot individual similarity	3355.92	28.9974	3305.7	3372.68	4
plot group similarity	3191.91	-	-	-	1
plot labelled ROI	3383.48	0.0495798	3383.25	3383.51	24
plot internal validity	3191.90	0.00816497	3191.89	3191.91	3

Table 22. Maximum USS in MB (dMRI)

task	mean	std	min	max	n jobs
process masks	93.45	-	-	-	1
connectivity	1109.09	109.136	821.17	1463.93	300
kmeans clustering	707.34	97.206	504.75	869.72	1200
internal validity	425.04	31.7783	373.96	570	300
group-level clustering	108.36	1.9292	106.6	111.35	4
group similarity	103.77	-	-	-	1
plot individual similarity	121.44	5.85576	111.75	127.2	4
plot group similarity	91.03	-	-	-	1

plot labelled ROI	114.06	0.187705	113.59	114.66	24
plot internal validity	90.72	0.139603	90.52	90.83	3

Table 23. Maximum PSS in MB (dMRI)

task	mean	std	min	max	n jobs
process masks	113.51	-	-	-	1
connectivity	1128.20	109.084	839.71	1482.42	300
kmeans clustering	723.25	97.2084	519.75	883.56	1200
internal validity	440.07	31.6481	391.82	585.34	300
group-level clustering	118.84	2.76936	115.67	122.74	4
group similarity	111.25	-	-	-	1
plot individual similarity	135.31	9.35706	120.77	146.83	4
plot group similarity	99.00	-	-	-	1
plot labelled ROI	116.08	0.71373	115.54	118.66	24
plot internal validity	97.63	0.417692	97.04	97.95	3

Table 24. Maximum CPU load per second in MB (dMRI)

task	mean	std	min	max	n jobs
process masks	0.00	-	-	-	1
connectivity	98.11	1.71986	85.59	101.79	300
kmeans clustering	208.21	32.7676	162.54	295.77	1200
internal validity	365.97	13.5011	343.32	418.99	300
group-level clustering	32.65	35.3829	0	84.58	4
group similarity	88.92	-	-	-	1
plot individual similarity	91.03	52.8042	0	128.09	4
plot group similarity	0.00	-	-	-	1
plot labelled ROI	68.86	8.37796	49.62	80.61	24
plot internal validity	0.00	0	0	0	3

Appendix 7: Software Mentions

This appendix contains a list of all software that was mentioned in this work. Where possible, the GitHub repository is mentioned for open-source software. For closed source software, or software not available on GitHub (or any other software forges) the website where it can be acquired is linked. All listed software except MATLAB can be used free of charge.

AFNI (Analysis of Functional NeuroImages). Software suite for the analysis and display of anatomical and functional MRI data. https://github.com/afni/afni

ANTS (Advanced Normalization Tools). A toolkit for medical image registration and segmentation. https://github.com/ANTsX/ANTs

BrainMap. A database of published functional and structural neuroimaging experiments with coordinate-based results. http://www.brainmap.org/

dMRIPrep. Pipeline for pre-processing of diverse dMRI data. https://github.com/nipreps/dmriprep

FIX (FMRIB's ICA-based Xnoiseifier). Tool for automatic classification of ICA components for denoising. https://github.com/jelman/FSL_FIX

fMRIPrep. Pre-processing pipeline for fMRI data. https://github.com/poldracklab/fmriprep

FSL. Library of analysis tools for fMRI, MRI, and DTI brain imaging data. https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/
*** probtrackX2**. Tool for performing probabilistic tractography on bedpostx output.
*** bedpostX**. Bayesian estimation of diffusion parameters obtained using sampling techniques

FreeSurfer. Software for the analysis and visualization of neuroimaging data from cross-sectional and longitudinal studies. https://github.com/freesurfer/freesurfer

HTCondor. Distributed high throughput computing system. https://github.com/htcondor/htcondor

ICA-AROMA (ICA-based Automatic Removal Of Motion Artifacts). Method (and software) for identification and removal of motion-related independent components from fMRI data. https://github.com/maartenmennes/ICA-AROMA

JuBrain Anatomy Toolbox. An SPM plugin for combining probabilistic cytoarchitectonic maps and functional imaging data. https://github.com/inm7/jubrain-anatomy-toolbox

Matlab. Software and programming language for matrix and array mathematics. https://www.mathworks.com/products/matlab.html

Neurodebian. Software delivery platform for Neuroscience. https://github.com/neurodebian/neurodebian

NeuroVault. Web database for statistical maps. https://github.com/NeuroVault/NeuroVault & https://www.neurovault.org/

NeuroSynth. Platform for large-scale automated synthesis of fMRI data. https://neurosynth.org/

Python. High-level general-purpose programming language for quick and efficient software integration. https://www.python.org/

Slurm. Highly scalable workload manager. https://github.com/SchedMD/slurm

Appendix 8: Python Package Mentions

This appendix contains a list of all python packages that were mentioned in this work. Note that all first mentions of the packages are underlined in green. All packages can be installed by name using the pip software (i.e., `pip install <name>` or `pip3 install <name>`). Each mention links to the package's GitHub repository, as all packages are open source.

matplotlib. Library for static, animated, and interactive plots.
https://github.com/matplotlib/matplotlib

nibabel. Utility for read/write access to some common neuroimaging file formats.
https://github.com/nipy/nibabel

nitime. Timeseries analysis for neuroimaging data. https://github.com/nipy/nitime

nipype. Workflows and interfaces for neuroimaging packages.
https://github.com/nipy/nipype

numpy. Package for scientific computing in Python. https://github.com/numpy/numpy

pandas. Data analysis and manipulation library based on R data.frame objects.
https://github.com/pandas-dev/pandas

pip. Python package installer. https://github.com/pypa/pip

psutil. Library for process and system monitoring. https://github.com/giampaolo/psutil

pyyaml. YAML parser and emitter for Python. https://github.com/yaml/pyyaml

scipy. Library for mathematics, science, and engineering. https://github.com/scipy/scipy

scikit-learn. Machine learning in Python. https://github.com/scikit-learn/scikit-learn

snakemake. Workflow management system. https://github.com/snakemake/snakemake

seaborn. Statistical data visualization using matplotlib.
https://github.com/mwaskom/seaborn

virtualenv (venv). Virtual Python environment builder.
https://github.com/pypa/virtualenv

Appendix 9: Usage Example

For the following example pre-processed rsfMRI and dMRI data has been supplied for 100 randomly drawn subjects out of the 300 subjects described in

Sect. 3 [p. 78] (mean age 28.46, 50 females, no significant age (t=-1.5, p=0.14) and education (t=-1.04, p=.30) difference between genders). Furthermore included are the three region-of-interest (ROI) NIfTI images used in this work, as well as the CBPtools configuration files used to process the data. The example data set was prepared and uploaded using DataLad version 0.12.0rc6 (Halchenko et al 2019) and has a total size of 243GB. The example data can be downloaded using DataLad, which can be installed using apt-get or pip (note when using pip, the git-annex dependency must be installed manually; see https://www.datalad.org/get_datalad.html). The example data is located on a remote location linked to through GitHub (https://github.com/inm7/cbptoolsexample-data).

```
$  datalad install --get -data --source https://github.com/inm7/cbptools -example
   -data.git
```

Further downloading options (i.e., downloading only parts of the data) are outlined in the online documentation (see External Resources [p. 16]). A CBPtools project can be created using any of the provided configuration files or a custom configuration file. In this example, the preSMA-SMA ROI with the rsfMRI data will be used (i.e., the '*config_r_presmasma_rsfmri.yaml*' configuration file), although it can be substituted by any other configuration file to use different settings and data.

```
$  cd cbptools -example -data
$  cbptools create --config config_r_presma sma_rsfmri.yaml --workdir
   /path/to/workdir
```

The workdir parameter is used to define where the project files (and eventual output data) will be stored. This can be any directory on the file system with read and write access.

Any errors and warnings occurring during the setup will be logged. The log is available either in the current directory (if the setup fails) or in the log folder inside the workdir (if the setup succeeds). If there are any errors, the project will not be created until they are resolved. If there are no validation problems, the project will be created. Change directory to the workdir and execute the workflow (contained in the Snakefile) using Snakemake, which is installed as a dependency of CBPtools.

```
$  cd /path/to/workdir
$  snakemake
```

For more customization of the Snakemake execution, visit the Snakemake guide (Köster 2019) or the execution section of the CBPtools online documentation.

End of Document.