brainGraph User Guide
Version 3.0.0

Christopher G. Watson, Ph.D.

Dept. of Pediatrics, Children's Learning Institute
University of Texas Health Science Center at Houston

September 30, 2020

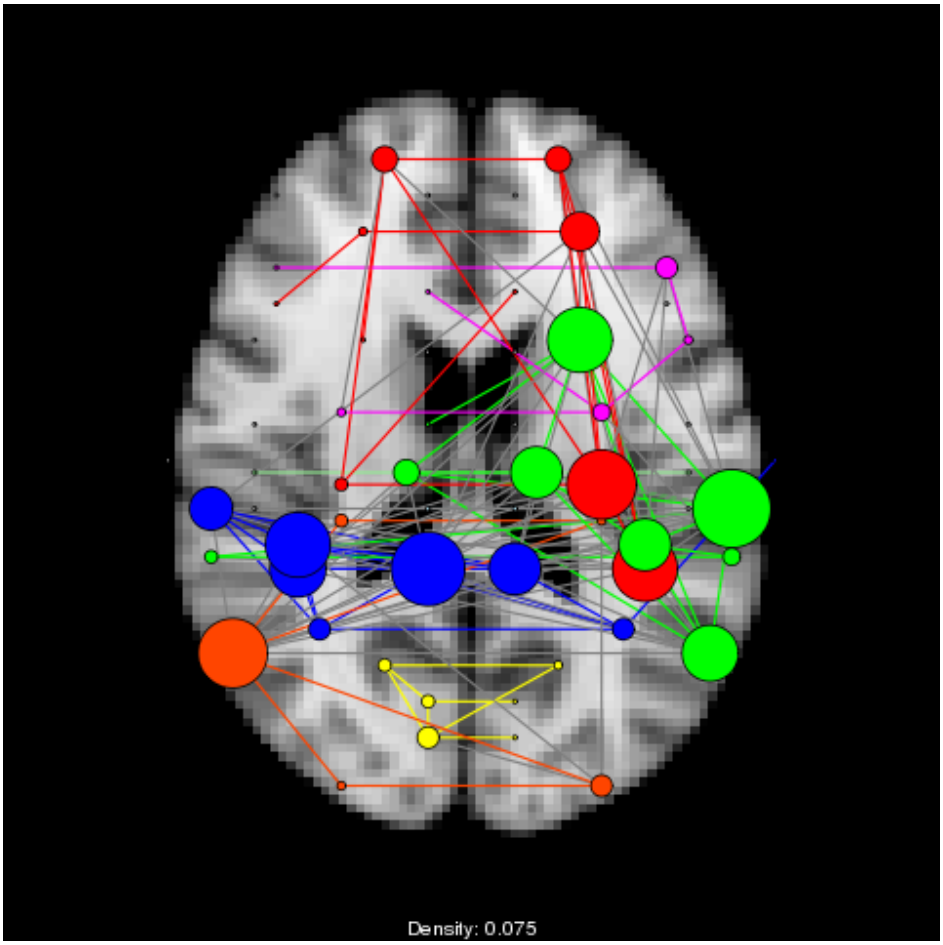

Table of Contents

List of Tables

Preface

`brainGraph` is an `R` package for performing graph theory analysis of brain MRI data. It started out essentially as one long script I wrote while taking a course in Fall 2013 on the statistical analysis of network data. Initially, the functionality was specific to cortical thickness data only (from `Freesurfer`), but I have since extended it to include functionality for DTI tractography (e.g., *fdt_network_matrix* from `FSL`'s *probtrackx2*, and matrices from `PANDA` (21)) and resting-state fMRI (e.g., `DPABI` (132) and `AFNI` (17)). It should work for any data that can be represented as a connectivity matrix.
There is some plotting functionality, but it doesn't look as "polished" as other software. (However, it looks comparable to figures I have seen in publications; see Plotting for some example plots and a function to export the network data, and The GUI for a few screenshots of the GUI).

Organization of this Manual

At the highest level of organization, there are several *Parts*. The general contents of each part are:

Introductory material contains installation information, validity of graph metrics calculated by `igraph` and `brainGraph`, neuroimaging software and brain atlas compatibility, how to get help, other `R` packages that may be of interest to neuroimaging researchers, and some code examples for getting data from `Freesurfer` and `FSL`.

brainGraph basics contains information for starting to use the package. This includes a general overview of the package, a brief introduction to `R` notation/conventions, a recommended workflow/script organization, and an introduction to the package's features and most basic operations.

Graph creation covers the necessary steps for creating graphs from your neuroimaging data. There are separate chapters for *structural covariance networks* and data for which single-subject networks can be created (e.g., DTI tractography or resting-state fMRI). The code blocks in these chapters start with importing your data and end with some example operations you can perform on the graphs.

Group analyses detail the available methods for comparing groups (or performing within-group analyses). These include the standard GLM, the *Network-Based Statistic (NBS)*, and statistical mediation analysis for single-subject graphs. For covariance networks, this includes bootstrapping, permutation/randomization tests, and *individual contributions*. And for both types of network, *random graph* generation, *small world* calculations, and *rich club* analysis.

Visualization describes the components of the `brainGraph` GUI, in addition to other functions for visualizing different aspects of your data, such as adjacency matrix plots, plotting global (graph-level) metrics by density, boxplots of vertex-level metrics, creating three-panel plots of the networks overlaid on a brain MRI slice, and saving a list of graph plots. The chapter closes with description of a function for exporting your data to work with the `BrainNet Viewer` tool.

Appendices list the attributes set by the function `set_brainGraph_attr`, several benchmarks (i.e., runtimes) for various functions/analyses, and the computing environment used in creating this document.

Intended Audience

This User Guide is appropriate for researchers who use brain MRI to study *connectivity*. `brainGraph` is not strictly limited to human MRI data, but it does not contain atlases for animal brains (but these can be provided by the user). It is expected that the user has *some* experience with `R` (and/or other programming languages), but this document should be appropriate for beginners. The user should have some understanding of network/graph theoretical concepts. This User Guide is quite long, but I have attempted to be comprehensive in documenting the functions in the package and the types of analysis that are common in neuroimaging, with extensive code examples and figures. To learn more about the relevant topics (i.e., the mathematics of networks, `R` programming), see some of my suggestions at Getting Help and Other Resources. If you are a non-MRI researcher, there are many functions you can still use. See Functions for generic data for a list.

Rationale

> *"The nice thing about standards is that you have so many to choose from."*
>
> — Andrew S. Tanenbaum

Other tools for performing graph theory analysis of brain MRI data already exist. So why create a new one, and why do it in `R`?

R is Free Software Using `R` does not require an expensive license (like `Matlab`), and does not involve any "red tape" (such as the need to upgrade annually, having to deal with a licensing office, etc.). You can simply download and install it. `R` is also open source, so you can add features, fix bugs, etc. Finally, you can write your own packages, either for personal or public use. **This package is, and will remain to be, free and open source**, in line with the recent push for sharing your code along with publications (see Eglen et al. (32) for discussion).

R was made to do statistics The designers of `R` were statisticians (as are most/all of the `R core` members). Many statisticians use `R` in their work. Additionally, there are currently more than 10,000 packages in the *CRAN* repository (as of Sep. 2020), many of which are created and maintained by experts in statistics. If there is a statistical analysis you would like to perform, there's a good chance it is available in `R`. The `R` community is very extensive, with multiple e-mail lists, blogs, and forums (such as Stack Overflow) available for help.

Package management Package management is very well done in `R`; downloading and updating packages is nearly trivial. I have had a much easier time with `R` compared to dealing with dependencies for e.g., some Python-based software (I know there is `pip`, but I had some issues; see this Stack Overflow question, which I'm sure is now out-of-date). I consider myself tech-savvy, so I imagine it would be even more frustrating for beginners. Downloading and installing `brainGraph` and its dependencies should be very simple, so the user can focus on learning graph theory and how to interact with their data.

Documentation I have found the documentation for other tools/software specializing in graph theory analysis of MRI data to be lacking (and in some cases non-existent), particularly in terms of the first step: getting your data (cortical thickness/volumes, tractography, etc.) into a format that will *just work*. It has been said that software is "only as good as its documentation"; I appreciate that many users just want a tutorial to walk them through the steps, and I hope I have succeeded in that aspect.

Good support for reproducible research It is very easy to generate reproducible reports (or documents such as this one) on-the-fly. For this User Guide, I used `knitr` (131) with LaTeX, and all the code is in a `git` repository for *version control*. This system allows for easy documentation of analysis workflow and any changes in output resulting from parameter changes; (re-)running all the code is essentially automatic (I simply press `\kp` using *Nvim-R* with *tmux* to knit the pdf). I use the same process for generating results

from other analyses I do in `R`, such as reporting summary statistics from DTI analyses of FA/MD/RD.[1] Furthermore, Tables are generated automatically, so I don't have to type all entries by hand, or copy-paste things and worry about formatting (reducing user error). In fact, my dissertation was written with `knitr` and LaTeX because of these features.

Why isn't there a "main" GUI?

Although GUI's can be helpful to beginners, I chose not to create a "point-and-click" GUI for all of the processing/analysis steps (except for exploring the results visually with `plot_brainGraph_gui`) because I think it is of paramount importance to be "closer" to your data, instead of expecting a software package to do all of the work. That said, this User Guide will provide examples for data organization and example code to be placed in scripts that you can then run from the `R` console. So technically, you could "copy-paste" the code from this User Guide and complete your analyses without paying attention, but I don't recommend it. *Inspect your data at every step.* I believe I have provided enough code to do that easily.

Typographical conventions

I use different font styles to indicate different types of objects:

- Software tools/packages, `R` functions/objects (not in `brainGraph`), inline code, data objects (e.g., function arguments or data column names), and filenames are printed in `monospace font`.
- Functions in the `brainGraph` package are `highlighted and in monospace font`.
- `R` code is printed in `monospace font in a box with light gray background`, with syntax highlighting applied.
- Linux command-line code is printed in a separate text box (also with some syntax highlighting).
- Graph metrics are printed in *italicized font*.
- Links to chapters/sections, figures, tables, and footnotes are printed with red font.
- External links (i.e., URL's) are printed with blue font.
- Citations are printed with green font.

Icons and text boxes used

There are several places where you will see a colored text box (occasionally with an exclamation mark):

New in v3.x.x

This box indicates a major change to `brainGraph` that was introduced in v3.0.0 and later.

Warning

This box indicates a *warning*; e.g., operations that will take an extremely long time.

Note

This box replaces a footnote if the note is long.

[1] There are other solutions in `R` that are similar to the `Jupyter` notebook, see for example rNotebook and editR

Release Notes

Including all release notes would extend this document unnecessarily. You can always find the `NEWS.md` file at the `brainGraph` repository: https://github.com/cwatson/brainGraph/blob/master/NEWS.md

Citing `brainGraph`

First, please cite Ref. Csardi and Nepusz (20) and any other relevant references for calculating certain graph theory measures (see function documentation for some references). `brainGraph` is very reliant on `igraph`, and they should be cited for their work.
I do not currently have a manuscript that specifically describes/introduces the package, but you may cite Watson et al. (124). You also can get some citation information with the following code:

```
citation('brainGraph')

##
## To cite package 'brainGraph' in publications use:
##
##   Christopher G. Watson (2020). brainGraph: Graph Theory Analysis
##   of Brain MRI Data. R package version 3.0.0.
##   https://github.com/cwatson/brainGraph
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {brainGraph: Graph Theory Analysis of Brain MRI Data},
##     author = {Christopher G. Watson},
##     year = {2020},
##     note = {R package version 3.0.0},
##     url = {https://github.com/cwatson/brainGraph},
##   }
```

Publications citing `brainGraph`

1. Caligiuri et al. (13)
2. Barbagallo et al. (5)
3. Tanimizu et al. (110)
4. Cinelli et al. (15)
5. Watson et al. (124) – my work in congenital heart disease, analyzing *structural covariance networks* (cortical thickness)
6. Janes et al. (59)
7. Malhotra et al. (75)
8. Mair (73) – a methodology text, *Modern Psychometrics with R*
9. Dajani et al. (22)
10. Watson et al. (125) – my work in pediatric traumatic injury (DTI tractography)

11. Ohashi et al. (82)
12. Homan et al. (52)
13. Ramos-Prats et al. (88)
14. Saba et al. (99)
15. Tiwary (114) – a review article discussing `R` packages for network analysis and/or biomedical modeling
16. Ostachuk (83)
17. Mangangcha et al. (76)
18. Venkatesan and Hillary (120)
19. Richmond et al. (92)
20. King et al. (65)
21. Delgado-Baquerizo et al. (25)
22. Teicher et al. (111)
23. Saqr et al. (101)
24. DellaPosta and Nee (26)
25. MacCormack et al. (72)
26. Rashidi-Ranjbar et al. (89)
27. Feldmann et al. (38)
28. Haqiqatkhah and van Leeuwen (47)
29. Areshenkoff et al. (3)

Part I

Introductory Material

1

Installation and Requirements

1.1 System Requirements

New in v3.0.0

There are several fewer package dependencies in the latest version. This should allow for faster/leaner installs.

There aren't any, specifically (aside from required `R` packages). But I have some recommendations:

Hardware You should have a multi-core CPU and lots of RAM (the more, the better; probably at least 4 GB). Having a large number of cores is very important, particularly if you want to do permutation testing or bootstrapping (or if you are working with very large graphs). Note that you will need a lot of RAM with more CPU cores.

Operating System I 100% recommend using Linux (my personal preference is `CentOS`; `RHEL` and `Scientific Linux` are the same, and `Fedora` is similar). If you have been using `Freesurfer` and/or `FSL`, then you likely have access to a Linux machine. It may be worthwhile to spin up a virtual machine running some flavor of Linux. Almost all testing and developing of this package was done on 64-bit CentOS 6 and CentOS 7. Some (early development) was done on 64-bit Windows 7.

Packages There are several packages that are required/recommended:

- `igraph` (current version is v1.2.4.1)
- `foreach` and `doParallel` (additionally, `doMC`/`doSNOW`, the multicore packages for Linux/Windows)
- `data.table` (excellent general-purpose package)
- `lattice`, `grid` (plotting-related; fewer dependencies compared to `ggplot2`)
- `Matrix`, `abind`, (functionality for matrices and multi-dimensional arrays)
- `permute` (to generate random permutations)
- `MASS` (general-purpose packages for data analysis; supports the Venables & Ripley textbook *Modern Applied Statistics with S*)(119)

Suggested Packages required for 1 or a few functions, and other general packages, include:

- `RGtk2` and `cairoDevice` are mandatory only if you want to use the GUI. Thanks to `@michaelhallquist` for his contribution in making this work!

- `boot` (required for `brainGraph_boot`)
- `ggplot2`, `scales`, `gridExtra`, and `ggrepel` (plotting-related packages)
- `Hmisc`, `ade4` (general-purpose packages for data analysis). Required for `corr.matrix` and `loo`/`aop`, respectively.
- `mediation` (causal mediation analysis)
- `car` (tools for applied regression)
- `oro.nifti` (a general *NIfTI* library)
- `expm` (for `communicability` and `centr_betw_comm`)

R interface For users not comfortable with a full command line interface, I recommend using RStudio. It is a full desktop environment (free), that is *very* similar in look and concept to the `Matlab` desktop.

1.2 GUI-related note

The GUI requires both `RGtk2` and `cairoDevice` to be installed. These can sometimes be difficult to properly install without issue.
If your operating system is *Mac OS* or *Windows*, please see this GitHub Gist. The comments contain more recent information. You may need to install a couple additional packages; the code is at the end of the gist (repeated here). The third package listed (`RGtk2Extras`) might not be required.

```
install.packages('gWidgets', dependencies=TRUE)
install.packages('gWidgetsRGtk2', dependencies=TRUE)
install.packages('RGtk2Extras', dependencies=TRUE)
```

1.3 OS-specific instructions and notes

1.3.1 Preferred method, any OS

Regardless of the operating system, to install the latest *stable* version, you can download it from *CRAN* by simply typing:

```
install.packages('brainGraph')
```

For the latest development version, use `devtools` to install it from *Github*. It should install all dependencies for you.

```
require(devtools)
devtools::install_github('cwatson/brainGraph')
```

1.3.2 Linux

You may download the tarball from the official CRAN page or elsewhere, and in `R` you would install it by typing one of the following two commands:

```
install.packages('brainGraph_${VERSION}.tar.gz', type='source', dependencies=TRUE)
install.packages('brainGraph_${VERSION}.tar.gz', type='source', repos=NULL)
```

The latter command above is equivalent to installing from the command line, and requires that all dependencies have already been installed:

```
[user@host]$ R CMD INSTALL brainGraph_${VERSION}.tar.gz
```

1.3.3 Mac

I haven't tested on Mac yet, but here are the versions that it should work on:

macOS 10.13 High Sierra see the note and link in GUI-related note

macOS 10.12 Sierra passed *CRAN* checks.

OS X 10.11 El Capitan *has not* passed recent *CRAN* checks (specifically, version *10.11.6*), but I have had previous confirmation from a user that it works.

OS X 10.10.5 Yosemite it may be necessary to download the binary `GTK` package (`.pkg` file) from https://r.research.att.com/.

OS X 10.9.2 (13C64) passed *CRAN* checks.

1.3.4 Windows

You should first try to download directly from *CRAN* (using `install.packages()`). If that causes any problems, I recommend using `devtools` and installing directly from *Github*. See also the URL listed in GUI-related note for some instructions when it comes to installing `RGtk2`. You may end up needing to add/edit a couple *environment variables*.
Some additional information:

- To install *from source* on Windows, you first have to download and install Rtools
- You may have to install either the 32- or 64-bit version *only*, in case there is a problem with `GTK+` (required for the plotting GUI). To install the package for just one architecture, run the following command:

```
install.packages('brainGraph', INSTALL_opts='--no-multiarch')
```

1.4 Compatible neuroimaging software

The functions in `brainGraph` should work for any software that returns a text file containing connectivity matrices (unless you are creating covariance/correlation networks). Here is a list of software that I know to work (either first-hand or from a `brainGraph` user):

- Freesurfer (for structural covariance networks)
- FSL (specifically DTI tractography using `probtrackx2`)
- DPARSF (seed-based connectivity from resting-state fMRI; see Ref. (132))
- PANDA (for DTI analyses; see Ref. (21))
- TrackVis (for DTI analyses; see Ref. (123))

Atlas name	R object	Ref.
Desikan-Killiany	`dk`	(27)
Desikan-Killiany-Tourville	`dkt`	(66)
Destrieux	`destrieux`	(28)
FS atlases w/ scgm	`dk.scgm`, `dkt.scgm`, `destrieux.scgm`	
AAL	`aal90`, `aal116`	(115)
AAL2	`aal2.94`, `aal2.120`	(96)
Dosenbach	`dosenbach160`	(29)
Craddock-200	`craddock200`	(18)
Harvard-Oxford	`hoa112`	(74)
LONI probabilistic brain atlas	`lpba40`	(106)
Brainsuite	`brainsuite`	(85, 105)
HCP Multimodal Parcellation	`hcp_mmp1.0`	(42)
Power	`power264`	(86)
Gordon	`gordon333`	(44)
Brainnetome	`brainnetome`	(37)

Table 1.1: **Brain atlases, their corresponding `R` object names, and a reference for the atlas.**

1.5 Compatible atlases

There are a handful of atlases that will work "out-of-the-box". See Table 1.1 for the atlases, the `R` object names, and references for each.

New in v3.0.0

The following atlases are new in `v3.0.0`:

- **hcp_mmp1.0**
- **power264**
- **gordon333**
- **brainnetome**

In the next code block, I show several lines of the `dk` atlas (this object is a `data.table`), along with the *structure* of the object. The `name.full` column is provided (for a handful of atlases) for use in tables for publication (if desired).

```
dk[4:9]

##    name x.mni y.mni z.mni      lobe hemi index                  name.full
## 1: lCUN    -1   -82    20 Occipital    L     4                   L cuneus
## 2: lENT   -16   -10   -29  Temporal    L     5                L entorhinal
## 3: lFUS   -24   -54   -16  Temporal    L     6                  L fusiform
## 4: lIPL   -47   -70    31  Parietal    L     7 L inferior parietal lobule
## 5: lITG   -56   -32   -24  Temporal    L     8  L inferior temporal gyrus
## 6: liCC    -1   -48    25 Cingulate    L     9 L isthmus cingulate cortex

str(dk, strict.width='cut')
```

```
## Classes 'data.table' and 'data.frame': 68 obs. of  8 variables:
##  $ name     : chr  "lBSTS" "lcACC" "lcMFG" "lCUN" ...
##  $ x.mni    : num  -56 -2 -45 -1 -16 -24 -47 -56 -1 -43 ...
##  $ y.mni    : num  -44 21 18 -82 -10 -54 -70 -32 -48 -87 ...
##  $ z.mni    : num  5 27 46 20 -29 -16 31 -24 25 1 ...
##  $ lobe     : Factor w/ 6 levels "Frontal","Parietal",..: 3 6 1 4 3 3 2 3..
##  $ hemi     : Factor w/ 2 levels "L","R": 1 1 1 1 1 1 1 1 1 1 ...
##  $ index    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ name.full: chr  "L bank of the superior temporal sulcus" "L caudal an"..
##  - attr(*, ".internal.selfref")=<externalptr>
```

1.5.1 Using your own atlas: required format

If you have an atlas you would like to use, just follow the data structure shown above and it *should* work without issue. To ensure that your `data.table` conforms to this format, you can use the function `as_atlas`. You can also supply individual columns/variables to the function `create_atlas`, although the former is simpler. The requirements are: [1]

- The object must be a `data.table`
- The object must have, at a minimum, columns: `name`, `{x,y,z}.mni`, `lobe`, `hemi`, `index`
- The `lobe` and `hemi` columns should be *factor* variables.
- The `index` column is simply an integer sequence from 1 to the number of rows (regions).
- You can also include additional columns if you wish. For example, `dosenbach160` contains a `network` column, seen in the code block below. I calculate *assortativity* based on these values, and you can calculate other graph measures based on these network delineations as well.

```
dosenbach160[, table(network)]

## network
##           default   fronto-parietal cingulo-opercular      sensorimotor
##                34                21                32                33
##        cerebellum         occipital
##                18                22
```

I recommend writing the data object as a `rda` file (using `save`); then it can be re-loaded with `load`.

1.5.2 Atlases to be added (potential)

I can add other atlases for future versions, but would need the required information (see above) to do so. Atlases/parcellations that I am aware of include:

- *Shen-268* (107)
- *Von Economo – Koskinas* (103)
- *Willard-499* (91)
- *Schaefer-400* (102)

[1] See the structure of other atlases by typing e.g. `str(dosenbach160)`.

2

Getting Help and Other Resources

2.1 Getting help

The main *CRAN* page for `brainGraph` is https://cran.r-project.org/web/packages/brainGraph/; there you will find the `R` reference manual. The *Github* repository page is https://github.com/cwatson/brainGraph; this contains the package source code for the latest development version.

For general questions, complaints, bug reports, feature requests, criticisms, etc., you have 2 options:

1. Join the *Google Group*, `brainGraph-help`, by going to the Google Group page and clicking *Apply to join group*. The email address for the group is brainGraph-help@googlegroups.com.

2. Open an *issue* on the Github issues page. This requires that you have a *Github* username already. I am emailed any time an issue is opened.

For both of these, there are a few things you can do to aid me in figuring out the problem, including:

- Read this Stack Overflow answer.
- Include your session info using either of the commands `sessionInfo()` or `session_info()` (from the `devtools` package).
- Save the relevant data needed for me to reproduce the problem. Use `save` if there are multiple variables, and `saveRDS` for individual objects/variables.
- Include the exact code/command line that you used when you encountered the problem. Ideally, send me the code in a `.R` script.

2.2 Learning resources

2.2.1 Graph theory

M.E.J. Newman's text is perhaps the best comprehensive textbook on networks, and is appropriate for beginners and experts alike (79). In the literature, there are many extensive/classic review articles on graph theory. For an incomplete list, see the Wiki on my *Github* page. If you want to learn more about network statistics (in general, not domain-specific), I recommend Kolaczyk's text, which requires at least an intermediate level of statistical knowledge (67). To learn more about network statistics using `igraph` specifically, I recommend Kolaczyk & Csardi (2014) (part of the *Use R!* series) (68). Finally, a great neuroimaging-specific text is Fornito et al. (39).

Package	Description	URL
network	General network tools	CRAN
intergraph	Convert between `igraph` and `network`	CRAN
statnet	An integrated suite of packages	CRAN
tnet	Tools for analyzing weighted networks	CRAN; Tore Opsahl's site
sna	Tools for social network analysis	CRAN
Rgraphviz	Interface for `graphviz` library (visualization)	Bioconductor
NetworkToolbox	Network-related tools for the psychological sciences	CRAN
qgraph	Weighted network visualization and analysis (developed for psychometric data)	CRAN

Table 2.1: **Other network-related packages.**

2.2.2 R programming

For help with learning R itself, there are a multitude of tutorials (freely) available, and I can help with code issues specific to `brainGraph`. Some websites that might be of use:

- R-tutor
- Quick-R
- R for Matlab users cheat sheet
- Stack Overflow: questions tagged with R

2.3 Other R packages

2.3.1 Other network-related packages

Besides `igraph`, there are several R packages that deal with networks. First, you can see the graphical models Task View, which lists packages with functionality specific to graphs.

2.3.2 Medical Imaging Task View

There are quite a few R packages for working with MRI data. First, there is a "CRAN Task View", *Medical Imaging*, that is a collection of medical image analysis packages. The website is here.

2.3.3 Neuroconductor

Neuroconductor is a repository of medical image analysis packages as well. The name calls to mind the *Bioconductor* project for molecular biology. More information can be found here.

2.3.4 Implementations of popular MRI software

There are several packages that are re-implementations of other popular tools.

fslr Port to FSL tools

spm12r Port to SPM tools

freesurfer Port to Freesurfer

ANTsR Implementation of ANTs tools

rcamino Port of Camino

hcp Connects to the Human Connectome Project

2.3.5 Other

Finally, there are still more packages for working with MRI data:

RNifti Read and write NIfTI images

tractor Tools for tractography in R

dti DTI processing in R

RAVEL Processing and analysis of MRI data

2.4 Validation of graph metrics

Since I don't expect potential users to blindly trust that my code will do what they want it to do, I have compared results using `brainGraph` with results from `Brain Connectivity Toolbox (BCT)`. There are a few minor differences that appear to be off by just a scalar:

	Measure	
No differences	Degree	Strength (weighted networks)
	Edge betweenness	K-coreness centrality
	Subgraph centrality	Within-module degree z-score
	Participation coefficient	Clustering coefficient
	Transitivity (graph-wise)	Assortativity (graph-wise)
	Global efficiency	Local efficiency
	Connected components	Motif frequency (# of triangles)

Betweenness centrality Results obtained from `BCT` are exactly 2x that of `igraph`.

Eigenvector centrality Results obtained from `BCT` are $\approx$ 3.6x that of `igraph` (for unknown reasons)

Leverage centrality Doesn't exist in `BCT`

Characteristic path length To get the same answer as in `BCT`, use:

```
mean(shortest.paths(g)[!is.infinite(shortest.paths(g))])
```

However, the 2 correlate perfectly, and the largest absolute difference I have seen is ≈ 0.03 (which is less than 1%)

Rich club Number of edges is exactly 2x in `BCT` compared to `brainGraph` (and is incorrect in `BCT`)

Modularity (Louvain) There are differences, but minor. I don't know if they are systematic or not.

If you still would like to convince yourself, or do your own testing (please do so!), download the `R.matlab` package and use the following code. To understand what these variables are, see Chapter 5 (and later). To use these variables in `Matlab`, simply type (in `Matlab`) `load g1.mat`.

```
A <- corrs[[1]][[N]]$r.thresh  # Unweighted (binary) adjacency matrix
diag(A) <- 0
W <- corrs[[1]][[N]]$R  # Weighted adjacency matrix
diag(W) <- 0
C <- V(g[[1]][[N]])$comm
writeMat('g1_N.mat', A=A, W=W, C=C)
```

3

Getting data from Freesurfer and FSL

This chapter describes how to get data that can easily be imported into `R` for use with `brainGraph`. Specifically, I will use cortical thickness data from `Freesurfer` in the first section, and will explain how to use Freesurfer ROI's as seed regions for tractography in the second section.

Note

The code in this chapter is specific to `Bash`; there should be a similar solution for other shells.

3.1 Structural data from Freesurfer

Use `aparcstats2table` to get mean cortical thickness for each region and each subject into a single file. You may also get subcortical gray matter volumes with `asegstats2table`. Replace the `subjects` variable definition with something that makes sense for your data.

Collect regional statistics

```
parc='aparc'        # Or 'aparc.DKTatlas40', or 'aparc.a2009s'
measure='thickness' # Or 'volume', 'area'
subjects=$(ls -d [[:lower:]]*[a-z0-9]*/)
for h in 'lh rh'; do
    aparcstats2table --subjects ${subjects} --hemi ${h}
    --meas ${measure} -p ${parc} --skip fsaverage
        --delimiter=comma
        --tablefile ${parc}_${h}_${measure}.csv
done
asegstats2table --subjects ${subjects} --skip --meas volume --common-segs
    --segno 10 11 12 13 17 18 26 49 50 51 52 53 54 58 --delimiter comma
    --tablefile asegstats.csv
```

3.2 Tractography

In this section, I describe the steps for using one of the `Freesurfer` atlases (along with subcortical gray matter) as seed regions for `probtrackx2` in `FSL`. I have a script for automation, but the code in this section is a good start. It is assumed that `recon-all` has been completed for the current subject (because I use the transformation matrices that are calculated by `TRACULA`). I use the *Desikan-Killiany* atlas for cortical regions, and the subcortical regions are included.

I have more tools in a `Bash` library which can be found at my GitHub repository. The code in this library includes initial preprocessing steps through to tractography.

3.2.1 Create/convert parcellated volume

First, the parcellation volume must be created if it doesn't exist, and converted to `NIfTI`.[1] The DWI volume is called `dwi.nii.gz`. If you need to create the file for the *DKT* atlas, the following code will do so.

Parcellation volume

```
# Replace {subj} with your Subject's ID
if [ ! -e "aparc.DKTatlas40+aseg.mgz" ]; then
    mri_aparc2aseg --s ${subj} --annot aparc.DKTatlas40
    mri_convert aparc.DKTatlas40+aseg.{mgz,nii.gz}
fi
```

3.2.2 Get individual seed ROI's

I created a text file with the names and (`Freesurfer`-specific) indices of each ROI; in this example, it is called `dk.scgm.txt`. The lines of this file look like: (see full file at GitHub)

ROI label file

```
1001 1001_ctx-lh-bankssts
1002 1002_ctx-lh-caudalanteriorcingulate
1003 1003_ctx-lh-caudalmiddlefrontal
...
```

The existence of the file `anatorig2diff.bbr.mat` is from `TRACULA`; if you want to use this file, you must run the first step of `trac-all`. The code example assumes that a `dmrirc` config file for the subject exists.

Get individual ROI's

```
    if [ ! -e "${SUBJECTS_DIR}/${subj}/dmri/xfms/anatorig2diff.bbr.mat" ]; then
        trac-all -c ${subj}.dmrirc -intra -masks
    fi

    labelfile='dk.scgm.txt'
    mkdir -p seeds/dk.scgm && cd seeds/dk.scgm
    while read line; do
        roiID=$(echo ${line} | awk '{print $1}' -)
        roiNAME=$(echo ${line} | awk '{print $2}' -)
        fslmaths
            ${SUBJECTS_DIR}/${subj}/dlabel/diff/aparc+aseg.bbr
            -thr ${roiID} -uthr ${roiID}
            -bin ${roiNAME}
        fslstats ${roiNAME} -V | awk '{print $1}' >> sizes.txt
    done < ${labelfile}

    echo ${PWD}/*.nii.gz | tr " " "\n" >> seeds.txt
    paste sizes.txt seeds.txt | sort -k1 -nr - | awk '{print $2}' - >> seeds_srt.txt

    # Ventricles mask; for use with ``--avoid'' flag
    mri_binarize --i ${SUBJECTS_DIR}/${subj}/dlabel/diff/aparc+aseg.bbr.nii.gz
```

[1] The `mgz` volumes for the *DK* and *Destrieux* atlases are created automatically by `recon-all`.

```
22          --ventricles -o ventricles.nii.gz
```

The final step of the `while` loop calculates the ROI sizes, which you may wish to use when normalizing the connectivity matrices; see Tractography and fMRI. The final lines of the code create a text list of the seed region files and sorts them by size; this is used as input to `probtrackx2` (specifically, the `-x` option).

Part II

brainGraph Basics

4

Overview of the `brainGraph` Package

This chapter provides an overview of `brainGraph` both in terms of concepts/workflow and the functions themselves. This package relies almost entirely on the `igraph` package (20) (hence the creative name for my package), so all operations/functions require this package. A complete list of the functions and their help sections is in the package manual.

4.1 Concepts/workflow

There are a handful of conceptual/workflow-related categories (or "levels") under which functionality in `brainGraph` falls. By "workflow", I mean this is more or less the order of operations for your user scripts. The steps I list below are very generic, and some of them will likely have multiple "sub-steps" (e.g., different types of group analyses).

4.1.1 "Step 0": Setting up data and scripts

This needs to be done before using `brainGraph`, and is not specific to the package (i.e., is common to all data analysis projects). I give some advice in Setting up files for your project. Throughout the User Guide, there are code blocks that can be placed into your scripts, or adapted to work with your specific analysis needs.

4.1.2 Step 1: Import data and create connectivity matrices

The operations/functions for this step differ depending on the type of input data you have, and the networks you aim to create:

Structural covariance The input data need to be `csv` or `tsv` files containing the region-wise brain metrics (e.g., cortical thickness; see Import the data) for all subjects.

DTI tractography The input data should be the connectivity matrices as calculated from the tractography program of your choosing. I personally use FSL's `probtrackx2`. See Setting up for more info.

Resting-state fMRI The input data should be the connectivity matrices as calculated by your software-of-choice, representing the inter-regional (partial) correlations, for example.

Once the data are imported, matrices of structural covariance will be created from partial correlations, linear model residuals, etc. (see Correlation Matrix and Graph Creation). The same function that performs those correlations will threshold the matrices by density or threshold. These networks will typically be *group-level*; i.e., there will be 1 network per group.

For subject-level networks (DTI tractography or resting state fMRI), you threshold the connectivity matrices based on criteria of your choosing (see Import, normalize, and filter matrices for all subjects).

4.1.3 Step 2: Create graphs and calculate metrics

Graph creation differs slightly for covariance networks (Graph creation) and single-subject networks (Graph creation), but the same function (`make_brainGraphList`) will be invoked. In those same sections, I show how to collect graph metrics of interest into a table, as well.

4.1.4 Step 3: Perform group analyses

There are several types of analysis, all of which require more in-depth description. See Part IV and Part V.

4.1.5 Step 4: Visualize results

There is a GUI for quick manipulation and inspection of graphs; furthermore, several separate functions are available for plotting results from specific analyses. See Part VI and the group analysis chapters.

4.2 Functions

4.2.1 Graph creation

Several functions can create graphs for various analyses; the names begin with `make_`, following the naming convention for graph constructors in the `igraph` package. The creation functions and the respective classes/analyses are described in more detail in Box 4.1 and Table 4.1.

New in v3.0.0

All of the following (except the last 2) are new functions, while `make_brainGraph` is now an S3 method.

make_brainGraphList Creates an object containing, in addition to some metadata, a *list* of `brainGraph` objects. This list should contain *all* study subjects, irrespective of group membership, for a single threshold or density.[1] This same function is used to create graphs of the results from `brainGraph_GLM` (see Vertex-wise group analysis (GLM)), `mtpc` (see Multi-threshold permutation correction), and `NBS` (see Network-based statistic (NBS)). Furthermore, you can supply the output of `corr.matrix` directly to this function for *structural covariance networks* (see Graph creation).

make_brainGraph Creates a `brainGraph` graph object, given an `igraph` graph object or matrix. This assigns several attributes that are specific to brain MRI data (mostly related to the brain atlas in use). You most likely should not have to call this function yourself (except when creating graphs of the results from `brainGraph_mediate`; see Graph- and vertex-level mediation analysis). Otherwise, you should use `make_brainGraphList`.

make_auc_brainGraph Creates a single `brainGraphList` object in which each graph contains the *area under the curve (AUC)* of the specified graph- or vertex-level metric(s).

make_intersection_brainGraph Create a graph based on the intersection of vertices meeting some criteria. For example, if you have multiple t- or F-contrast graphs, you can determine vertices showing a significant difference. See `?make_intersection_brainGraph` for examples.

make_empty_brainGraph Creates an *empty* graph (i.e., one with no edges); typically the end user will not need to call this. This is analogous to `igraph`'s `make_empty_graph`.

make_ego_brainGraph Creates a graph of the union of multiple vertex neighborhoods. This extends `igraph`'s `make_ego_graph`.

[1] You can think of these objects as analogous to the 4-D *NIfTI* files that are created by FSL's *tbss* (i.e., the files used as input to *randomise*).

4.2.2 Graph metrics

`igraph` already contains functions for the most common graph metrics (e.g., degree, betweenness, etc.). Functions that I wrote and the graph metrics they calculate include:

- leverage centrality; `centr_lev` (62)
- global, local, and nodal efficiency; `efficiency`
- *Vertex roles*; `gateway_coeff` (118), `part_coeff` and `within_module_degree_z_score` (45)
- Rich club calculations; `rich_club_coeff`, `rich_club_norm`, and `rich_core` (71)
- The *s-core* of vertices; `s_core` (33)
- Weighted shortest path lengths: `mean_distance_wt`
- Vertex "hubness": `hubness` (117)
- Euclidean distances; `edge_spatial_dist` and `vertex_spatial_dist`
- Communicability; `communicability` (19, 34) and `centr_betw_comm` (35)
- Vertex vulnerability; `vulnerability` (1, 51)
- Edge counts; `count_homologous` (counts the number of edges between homologous brain regions) and `count_inter` (counts the number of edges between and within all lobes, hemispheres, etc.)
- Asymmetry index: `edge_asymmetry`

One function in the package, `set_brainGraph_attr`, calculates most of these metrics and more for a given graph (e.g., global efficiency, clustering coefficient, characteristic path length, and many more), and for its vertices (e.g., degree, nodal efficiency, etc.) and edges (e.g., edge betweenness). This is very useful because it removes the nuisance of having to type a separate command for every measure of interest. To see exactly what it calculates, see Attributes created by `set_brainGraph_attr` or check the function help (accessible by typing `?set_brainGraph_attr`) under the heading *Value*. However, you will likely not call this function yourself; it will be done automatically, for all subjects, if you pass `set.attrs=TRUE` to `make_brainGraphList`.

4.2.3 Group comparison

There are several methods for comparing groups:

- Between-group vertex-wise analysis of graph metrics with the *General Linear Model (GLM)*: `brainGraph_GLM`. See Vertex-wise group analysis (GLM) for details.
- The *network-based statistic (NBS)* (134): `NBS`. See Network-based statistic (NBS) for details.
- *Multi-threshold permutation correction (MTPC)* (30) method for inference: `mtpc`. See Multi-threshold permutation correction for details.
- *Mediation analysis*: `brainGraph_mediate`. See Graph- and vertex-level mediation analysis for details.
- Bootstrapping and permutation testing (for structural covariance networks): `brainGraph_boot` and `brainGraph_permute`. Details can be found in Further analysis.
- "Individual contributions" for data in which single-subject graphs are not available (e.g., structural covariance networks); `loo` and `aop`. See Individual contributions for details. (100)
- Targeted attack and failure analyses: `robustness`. See the "Robustness" section in Further analysis for implementation and plotting.

Class name	Creation function	Description
brainGraph	make_brainGraph	Any graph with certain attributes (see text)
brainGraphList	make_brainGraphList	A *list* of graphs (see text)
brainGraph_GLM	make_brainGraphList	Graphs from GLM analysis
brainGraph_NBS	make_brainGraphList	Graphs from NBS analysis
brainGraph_mtpc	make_brainGraphList	Graphs from MTPC analysis
brainGraph_mediate	make_brainGraph	Graphs from mediation analysis
brainGraph_boot	brainGraph_boot	Bootstrapping analysis (non-graph)
brainGraph_permute	brainGraph_permute	Permutation analysis (non-graph)
brainGraph_resids	get.resid	Residuals for covariance networks (non-graph)
corr_mats	corr.matrix	Correlation matrix of residuals (non-graph)
IC	aop, loo	Individual/regional contributions (non-graph)

Table 4.1: Class names and graph creation functions.

4.2.4 Visualization

There is a GUI for plotting the graphs overlaid on a slice of the MNI152 brain (plot_brainGraph_gui). You can visualize up to two brains (single orientation; e.g., axial) at once. The GUI controls vertex/edge color and size, labels, inclusion/exclusion, and more. See The GUI for more details. In addition, there are some functions for plotting various graph metrics; see Other plotting. Finally, there is a plotting `method` for the classes mentioned in Box 4.1. See the respective chapters introducing the classes for details.

4.2.5 Random graphs, small world, and rich club

sim.rand.graph.par will create a number of random graphs in parallel, based on the "standard" method of random graph generation (78). An additional option is to generate random graphs with the same degree distribution *and* transitivity (clustering) as the observed graph. This is based on the algorithm from Bansal et al. (4), and is particularly important for partial correlation networks (e.g., cortical thickness correlations; see Refs. Hosseini and Kesler (53), Zalesky et al. (134)). You may also choose to create random *covariance matrices* with sim.rand.graph.hqs; see Hirschberger et al. (49). This function is meant to be used with structural covariance networks but could also work with resting-state fMRI. Finally, analysis_random_graphs is really a wrapper that will perform all of the steps for getting small-world and rich-club coefficients for all group data. See Random graph generation for more information.

4.2.6 Generic methods

There are several generic methods that will work for different types of brainGraph objects. Most of these are convenience functions used within the package but may be helpful in regular use. These include:

groups Returns a *character vector* of group names for each subject. Works for brainGraphList, brainGraph_resids, and corr_mats objects.

nobs Returns an *integer* of the number of subjects in an object. Works for bg_GLM, NBS, mtpc, brainGraphList, and brainGraph_resids objects.

case.names Returns a *character vector* of the subject ID's in an object. Works for bg_GLM and brainGraph_resids objects.

region.names Returns a *character vector* of region names in an object. Works for data.table, bg_GLM, mtpc, brainGraph_resids, and corr_mats objects.

nregions Returns an *integer* of the number of regions in an object. Works for bg_GLM, NBS, mtpc, corr_mats, and brainGraph_resids objects.

BOX 4.1 Classes and methods

Having classes and methods should simplify package usage in several areas. For example, in `base R`, if you use the `lm` function, the object returned has class `lm`; to view a summary and to plot some results, you simply call `summary` and `plot`, respectively. These "invisibly" run the functions `summary.lm` and `plot.lm` which are specialized for that type of data.

Second, each of these objects will also contain the important input parameters that the user supplies (for example, the significance level `alpha`). This is helpful for bookkeeping purposes and facilitates reproducible research or comparing results with varying inputs.

Here, I list the classes in `brainGraph` and in later chapters give example usage and output. Note that you do not need to remember the full names of these classes; you can simply type the base method name (e.g., `plot(res.glm)`) and `R` will take care of the rest.

brainGraph This class is essentially the same as an `igraph` graph object, but adds several graph-level (atlas, modality, Group, etc.) and vertex-level (lobe, hemi, spatial coordinates, etc.) specific to brain MRI analysis. These are created directly by `make_brainGraph` and indirectly by `make_brainGraphList`.

brainGraphList A class containing various *metadata* (e.g., the package version used to create it and the date), as well as a *list* of graphs. Most of the group analysis functions work on these objects.

bg_GLM Contains results from `brainGraph_GLM` (see Vertex-wise group analysis (GLM)).

NBS Contains results from `NBS` (see Network-based statistic (NBS)).

mtpc Contains results from `mtpc` (see Multi-threshold permutation correction).

bg_mediate Results from `brainGraph_mediate` (see Graph- and vertex-level mediation analysis).

brainGraph_GLM The graph associated with `bg_GLM` objects. It has GLM-specific attributes (for plotting), created by the function `make_brainGraphList`.

brainGraph_NBS The graph associated with `NBS` objects. It has NBS-specific attributes (for plotting), also created by `make_brainGraphList`.

brainGraph_mtpc The graph associated with `mtpc` objects. It is also created by `make_brainGraphList`.

brainGraph_mediate The graph associated with `bg_mediate` objects. It has mediation-specific attributes (for plotting), created by `make_brainGraph`.

brainGraph_boot Returned by `brainGraph_boot`. See Bootstrapping for details.

brainGraph_permute Returned by `brainGraph_permute`. See Permutation testing for details.

brainGraph_resids Returned by `get.resid` for structural covariance networks (see Structural covariance networks).

corr_mats Returned by `corr.matrix` for structural covariance networks

IC Returned by `aop` and `loo`. See Individual contributions for details.

5

Getting started

In this Chapter, I describe the most basic aspects of using `brainGraph`. I begin by suggesting some script/code organization. Then I show the other `R` packages that I load. Next, I show the structure of my *covariates* data. Finally, I introduce graph, vertex, and edge attributes and show how to plot from the terminal. For some information about `R` notation, see Box 5.1.

5.1 Setting up files for your project

5.1.1 Project scripts

I have several scripts, each of which carries out a separate "task"; they are numbered sequentially in a manner that may depend on the imaging modality, project, etc. For example: (incomplete list)

00_packages.R loads required packages

01_load_myProject.R loads/imports the [thickness/tractography/rs-fMRI] data and creates some initial variables. I have a different script for each modality and for each project/study.

02_create_graphs.R creates the graphs, etc. I have a different script for volumetric (covariance networks) data and for tractography/rs-fMRI.

03_random_graphs.R runs `analysis_random_graphs`, and does extra processing if, for example, I create random graphs controlled for clustering.

main.R sources all of the other scripts. I can comment out specific lines if I don't want to re-do a step.[1]

This is similar in philosophy to the top response to this Stack Overflow question. I also recommend that you read Noble (81) which is specific to computational biology but has very good recommendations for project organization. In the future, I would like to move to using *Makefiles* for this kind of data processing workflow.

I keep my `code`, `data`, and `results` in separate directories. Within the `results`, I have sub-directories for different modalities and the date the analysis was performed.

[1] I actually haven't done analyses this way lately because it seems to be slower overall (possibly due to how R handles memory, does garbage collection, or something else).

BOX 5.1 Basic R notation

Here, I *briefly* explain some of the `R` notation. In sections with `R` code, any line beginning with a double hashtag/pound sign (i.e., `##`) signifies code output. Lines beginning with a single hashtag/pound sign are comments written by me, and are ignored by the `R` interpreter.

Assignment Unlike `Matlab`, assignment is usually done with the symbol `<-` . Reasons for this are beyond the scope of this document. However, argument specification within a function call will always use the equals sign.

Lists A *list* is very similar to a *cell array* in `Matlab`. To access list elements in `R`, you must use double square brackets (whereas in `Matlab` you would use curly braces). For example, to access the graphs for group 1 (shown later), you would type `g[[1]]` .

Dollar sign The dollar sign `$` is used to access list or *data frame* elements if they are *named*. In the section on covariates, if I want to access the column for subject `Age`, I would just type `covars$Age` .

data.table assignment In some code using `data tables`, I use the assignment operator `:=` . This allows you to insert/change a column *in-place*, and is very fast and memory efficient.

The *apply functions These functions (`sapply, lapply, mapply, Map, llply`, etc.) all operate on lists/vectors. They are equivalent to a `for` loop in other languages, but require less typing.

Object names Following the Google style guide, I name my *constants* beginning with a `k`, e.g., `kNumDensities` refers to the number of densities. Object names should be as informative as possible; however, some of the ones I use are stupid/bad and were done out of laziness. Most of the `data.table`'s I create begin with `dt` and the graphs I create begin with `g`. I *usually* include a dot/period in object names, which differs from Hadley Wickham's style guide, and also differs from dot notation in Object-Oriented Programming.

5.1.2 Package global options

New in v3.0.0

Global options are new in `v3.0.0`.

There are a few package-specific global options that affect package usage. I will first list the option names, and then describe how you can change them.

bg.subject_id The character string specifying the variable name (in your project) corresponding to subject ID's. The default value is `Study.ID` (this was hard-coded in all previous versions). If your project uses a different name (e.g., if you follow the *BIDS* suggestion of `participant_id`) then you can specify it with this option.

bg.group The character string specifying the variable name (in your project) corresponding to the group variable. The default is `Group` (this was hard-coded in previous versions). One possible alternative (suggested by *BIDS*) is `group`.

bg.session (*not used*) The character string specifying the session/time variable name. The default is `Time`; one alternative (*BIDS*) is `session_id`.

bg.progress A logical indicating whether to always/never show a progress bar (for functions with the option). The default is `TRUE`.

bg.ncpus Integer indicating how many CPU cores to use for parallel operations. The default is `2`. This is only a fallback; it is expected that the user registers the appropriate number themselves (see next section).

Changing any of these options is simple, and can be specified in your `.Rprofile` or at the start of your analysis scripts. For example, to use the *BIDS* defaults, you would include

```
options(bg.subject_id='participant_id', bg.group='group')
```

If you would like to use the defaults, you do not need any code.

5.1.3 Loading required packages

This step will be the same regardless of the imaging modality. See System Requirements for a list of required/recommended packages. The `if-else` part of the OS check below is unnecessary if you will be using the same OS every time. I load the most useful packages (e.g., `data.table`) at the start of every `R` session by including the relevant commands in my `.Rprofile`.

```
suppressMessages(library(brainGraph))
# Check OS version for parallel processing
OS <- .Platform$OS.type
if (OS == 'windows') {
  pacman::p_load(snow, doSNOW)
  num.cores <- as.numeric(Sys.getenv('NUMBER_OF_PROCESSORS'))
  cl <- makeCluster(num.cores, type='SOCK')
  clusterExport(cl, 'sim.rand.graph.par')
  registerDoSNOW(cl)
} else {
  suppressMessages(library(doMC))
  registerDoMC(detectCores())
}
```

Once `brainGraph` is loaded, you can quickly see all its functions and the package help section:

```
ls('package:brainGraph')
help(package='brainGraph')
```

5.1.4 Project data

You will almost certainly want to include covariates for your analyses (e.g., adjust for *age*, *sex*, etc.). Additionally, you may be interested in testing for associations between graph metrics and demographic or neuropsychological variabes. I show a portion of my covariates below. The *Study ID*'s have been changed and will possibly/probably be character strings in your project.

```
covars

##      Study.ID   Group Sex   Age Scanner
##   1:        1 Patient   F 19.92  Site 1
##   2:        2 Control   F 14.92  Site 1
##   3:        3 Control   M 16.08  Site 1
##   4:        4 Control   M 13.42  Site 1
```

```
##   5:        5 Control   F 15.75  Site 1
##  ---
## 137:      137 Control   F 13.75  Site 2
## 138:      138 Patient   M 16.42  Site 1
## 139:      139 Control   F 16.25  Site 1
## 140:      140 Control   M 16.50  Site 2
## 141:      141 Control   M 16.50  Site 2

str(covars)

## Classes 'data.table' and 'data.frame': 141 obs. of  5 variables:
##  $ Study.ID: chr  "1" "2" "3" "4" ...
##  $ Group   : Factor w/ 2 levels "Control","Patient": 2 1 1 1 1 1 1 2 2 2 ...
##  $ Sex     : Factor w/ 2 levels "F","M": 1 1 2 2 1 1 2 2 2 1 ...
##  $ Age     : num  19.9 14.9 16.1 13.4 15.8 ...
##  $ Scanner : Factor w/ 2 levels "Site 1","Site 2": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

One of the nice things about R is that you don't need to change a variable such as *sex* to 0's and 1's; it considers the M and F as *factors* and can handle them easily. The same goes for subject group names (and pretty much any other non-numeric variable).

What I consider to be a smart thing to do (regarding covariates files) is to include some kind of "indicator variable" in your spreadsheet/database of *all* study subjects, where a 1 means the subject has acceptable data for that [MRI sequence, neuropsychological test, experiment, etc.], and a 0 otherwise. You can see this in the first code block of Tractography and fMRI, in which I subset the `covars.all` data table using `tract == 1`. I then only select the first 5 columns, as those contain the only relevant covariates I wanted for that application. Later, if I choose to, for example, test for correlation between vertex betweenness and *full-scale IQ (FSIQ)*, I can access that variable easily:

```
cor.test(btwn, covars.all[tract == 1, FSIQ])
```

5.1.5 Data Compatibility

The only real requirements for your non-MRI data are that they be in a csv file and that they share a `Study.ID` column (or whatever name you specify with the `bg.subject_id` option) with the MRI data. If you keep your patient data in an Excel file, then it is simple enough to save it as a csv. If you use a database, it also should be simple; I know that REDcap has an option to output files for use in R and there are packages for working with SQL.

5.2 Graph object attributes

There are three types of *attributes* that an igraph graph object can have: graph-, vertex-, and edge-level.

5.2.1 The summary method

The `summary` method for objects of class `brainGraph` is different than the method in igraph, but shows similar information. The default behavior is to print all attributes, separated by attribute *type*; to show only some general info, you can include the argument `print.attrs='none'`. The following information is shown; there are some values that will be automatically expanded (e.g., dk will be printed as Desikan-Killiany).

version A *named list* with the versions of R, igraph, and brainGraph at the time of graph creation

date A character string with the datetime for when the graph was created. The format is the *ISO 8601* date standard, uppercase *T*, and the *ISO 8601* time standard.

type Whether the graph is *observed* or a *random* graph

atlas The atlas

modality The imaging modality represented by the data

weighting If the graph is weighted, it will print either the value of `g$weighting` or what the edge weights represent (e.g., if `weighting='sld'` , it will print `Streamline density`)

clust.method The *clustering* (i.e., community detection) method used to calculate communities

density The graph density (percent)

threshold The numeric value used to threshold graphs (if applicable)

Subject ID The `Study.ID` (taken from the `name` attribute), if one was supplied to `make_brainGraph` or `make_brainGraphList`

Group The subject group, if one was supplied

level Either `subject`, `group`, or `contrast`

```
summary(g.ex)

##
## ============================================
## Summary for *observed* subject-level graph:
## ============================================

##
##  Software versions
##         R release:   R version 3.6.0 (2019-04-26)
##        brainGraph:   3.0.0
##            igraph:   1.2.4.1
##  Date created:       2020-09-30 12:31:46
##  Observed or random? Observed
##  Brain atlas used:   Desikan-Killiany
##  Imaging modality:   Cortical thickness
##  Edge weighting:     Unweighted
##  Clustering method:  Louvain (multi-level modularity optimization)
##  Graph density:      16.90%
##  Threshold:          N/A
##  Subject ID:         N/A
##  Group:              Eg
## Graph attributes-----------------------------

##
##  version  Lp           diameter         vulnerability
##  sys      rich         transitivity
##  date     E.global     assort
##  atlas    clust.method assort.lobe
##  level    mod          assort.lobe.hemi
##  type     density      asymm
##  modality conn.comp    spatial.dist
##  Group    max.comp     num.hubs
##  Cp       num.tri      E.local
```

```
## Vertex attributes-----------------------------

##
##  name          degree        k.core
##  lobe          comp          transitivity
##  lobe.hemi     color.comp    E.local
##  hemi          asymm         E.nodal
##  x.mni         dist          vulnerability
##  x             dist.strength eccentricity
##  y.mni         knn           comm
##  y             Lp            color.comm
##  z.mni         btwn.cent     circle.layout.comm
##  z             hubs          GC
##  color.lobe    ev.cent       PC
##  circle.layout lev.cent      z.score

## Edge attributes-----------------------------

##
##  color.lobe color.comp dist btwn color.comm
```

5.2.2 Graph-level attributes

Graph-level attributes can be of any data type (e.g., a character string specifying the atlas used, a numeric specifying the global efficiency, etc.). They are visible when you print the graph (by typing the object name), or by typing `graph_attr_names(g.ex)`
Using the `$` operator, you can access these graph-level attributes; the following example will display the size and count of the graph's connected components.[2]

```
g.ex$conn.comp

##   size number
## 1   68      1
```

Also of interest may be the *rich club coefficient* of a graph (see Colizza et al. (16), Zhou and Mondragón (135)). Briefly, the rich club coefficient is the ratio of edges present to total possible edges in a subgraph with minimum degree k.[3] The following example returns a `data.frame` for (in this specific example) $k = 1, 2, \cdots, 22$; R is the rich club coefficient, N_k is the number of vertices present, and E_k is the number of edges present:

```
g.ex$rich

##      k    phi Nk  Ek
##  1:  1 0.1690 68 385
##  2:  2 0.1690 68 385
##  3:  3 0.1690 68 385
##  4:  4 0.1758 66 377
##  5:  5 0.1788 65 372
##  6:  6 0.1877 62 355
##  7:  7 0.2065 56 318
##  8:  8 0.2261 50 277
##  9:  9 0.2386 46 247
## 10: 10 0.2549 41 209
```

[2] Calculated by the `igraph` function `components`, and set by `set_brainGraph_attr`
[3] See Rich-club Analysis for more details.

```
## 11: 11 0.2746 33 145
## 12: 12 0.3080 24  85
## 13: 13 0.3392 19  58
## 14: 14 0.3333 12  22
## 15: 15 0.4444  9  16
## 16: 16 0.4000  6   6
## 17: 17 0.3333  3   1
## 18: 18 0.0000  2   0
## 19: 19 0.0000  2   0
## 20: 20    NaN  1   0
## 21: 21    NaN  1   0
## 22: 22    NaN  0   0
##      k    phi Nk  Ek
```

If you're working with single-subject graphs, you can use the `subject` argument to `set_brainGraph_attr`. This will give the graph a `name` attribute, which is displayed when you print the graph; the name is on the first line of the output:[4]

```
print(g.tmp, full=FALSE)

## IGRAPH 5fc817a U--- 68 401 -- Watson, Christopher
## + attr: name (g/c)

g.tmp$name

## [1] "Watson, Christopher"
```

5.2.3 Vertex-level attributes

You can also access vertex-level attributes, using both the `V()` function and the `$` operator (the following example will display each vertex's degree).

```
V(g.ex)$degree

##  [1]  8 16 14  8  7 13  8 11 12  4 16 12 16  8 14 17 13 10 18  6 12 12 11
## [24] 11 12  6  9 12  9 11 14 11 12 14  7 12 17  8 15  5  6 10 22  7 15  9
## [47] 12 10 10 20 14 13  7  8  7 11 15 17 13  9 11  4 14 13 10 11  7 14
```

5.2.4 Edge-level attributes

Finally, you can access edge-level attributes, using both the `E()` function and the `$` operator. First, I show a sample of the edges; as you can see, the vertex names are joined by double dashes. Then I display edge betweenness the first several edges (to save space).

```
E(g.ex)[2:6]

## + 5/385 edges from 8c4d360 (vertex names):
## [1] lBSTS--lpreC lBSTS--lSMAR lBSTS--lTT   lBSTS--lINS  lBSTS--rPARH

head(E(g.ex)$btwn)

## [1] 15.221 11.475  9.010  9.796 10.805 10.291
```

[4] You will most likely want to use subject ID's, not real names.

5.3 Community detection

There are multiple community detection algorithms available in `igraph`. You can run community detection on your own or through `make_brainGraph` (or `make_brainGraphList`). Alternatively, you can specify which via the `clust.method` argument. All clustering functions begin with the string `'cluster_'`; the function argument recognizes the remainder of the function name. For example, if you would like to specify the *Walktrap* algorithm, you would pass `clust.method='walktrap'`.
The benefit of letting the package choose one automatically is that the appropriate method will be chosen depending on the type of input graph. There are a few different behaviors:

- By default, the *Louvain* algorithm (see Ref. Blondel et al. (12)) is applied (mainly because it seems to be the most popular one for neuroscience studies)
- If you select `'spinglass'`, but the graph is unconnected, then the *Louvain* algorithm is used.
- The *Walktrap* algorithm is used if there are any negative edge weights, unless you pass `'spinglass'`.
- If `'edge_betweenness'` is selected, the edges are first transformed because this algorithm considers the edges as distances (not connection strengths).

For general help with communities, type `?communities`. For a great demo on community detection (from which you can get useful code) is accessed by typing `demo(community)`. A description of some of the algorithms can be found in this Stack Overflow answer.
To plot the communities with a specific layout, use the following code.[5] Here, the function `layout_with_fr` uses the *Fruchterman-Reingold* method, which is a force-directed layout algorithm (41). This is shown in Figure 5.1.[6]

```
class(g.ex) <- 'igraph'
plot(cluster_louvain(g.ex), g.ex, layout=layout_with_fr, vertex.label=NA,
  vertex.size=5, edge.width=0.5)
class(g.ex) <- c('brainGraph', class(g.ex))
```

5.4 Plotting

Plotting graphs is much simpler when using the GUI `plot_brainGraph_gui`; however, you can achieve almost all of its functionality on the command line. For example, the `subgraph` argument allows you to specify a condition for which vertices to *keep*; if you wanted to plot only vertices with degree greater than 10, this would be `subgraph='degree > 10'`. You can combine multiple conditions, using either `&` (AND) or `|` (OR), e.g., `subgraph='degree > 10 & btwn.cent > 50'`. Plotting a single hemisphere is a "special" subgraph and can be chosen using the `hemi` argument. Additonally, you may choose to show a legend for vertex colors, using the `show.legend` argument.

5.4.1 MNI152 template

By default, graphs will be plotted over a slice from the `mni152` template. I loaded the *NIfTI* volume (from `FSL`) and saved the data in the package. Prior to `v3.0.0`, this was in the form of a `nifti` object. To convert the array with the correct parameters (if you have a reason), use the following code:

[5] To avoid seeing the polygons that highlight each group, include the argument `mark.groups=NULL`.

[6] For more layouts, type the command `?layout_` (include the trailing underscore).

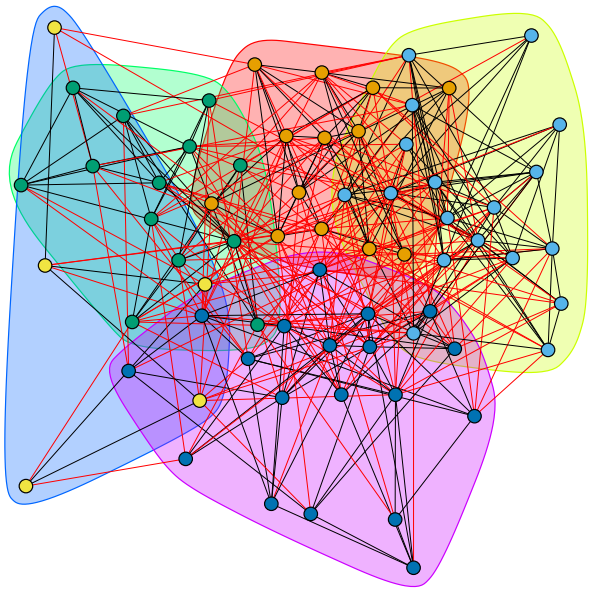

Figure 5.1: Communities plot, Fruchterman-Reingold layout.

```
suppressMessages(library(oro.nifti))
mni152 <- nifti(brainGraph:::mni152_mat, datatype=4, regular='r', reoriented=TRUE,
    scl_slope=1, quatern_c=1, descrip='FSL3.3', sform_code=4, xyzt_units=10,
    srow_x=c(-2, 0, 0, 90), srow_y=c(0, 2, 0, -126), srow_z=c(0, 0, 2, -72),
    pixdim=c(-1, 2, 2, 2, 1, 1, 1, 1), qoffset_x=90, qoffset_y=-126, qoffset_z=-72)
mni152@cal_max <- 8000
mni152@cal_min <- 3000
mni152@qform_code <- 4
mni152@pixdim[1] <- -1
```

5.4.2 Axial

Figure 5.2 shows an example plot of an axial view[7]; here, vertex size is proportional to vertex degree. The vertex color is based on community (module) membership.

```
plot(g.ex, vertex.label=NA, vertex.size='degree',
     vertex.color='color.comm', edge.color='color.comm', main='Toy graph')
```

5.4.3 Sagittal

Figure 5.3 shows an example of left and right sagittal views. These only show the *intra*-hemispheric connections. Here, lobe colors are based on *lobe* membership.

[7] Axial images are always in neurological orientation

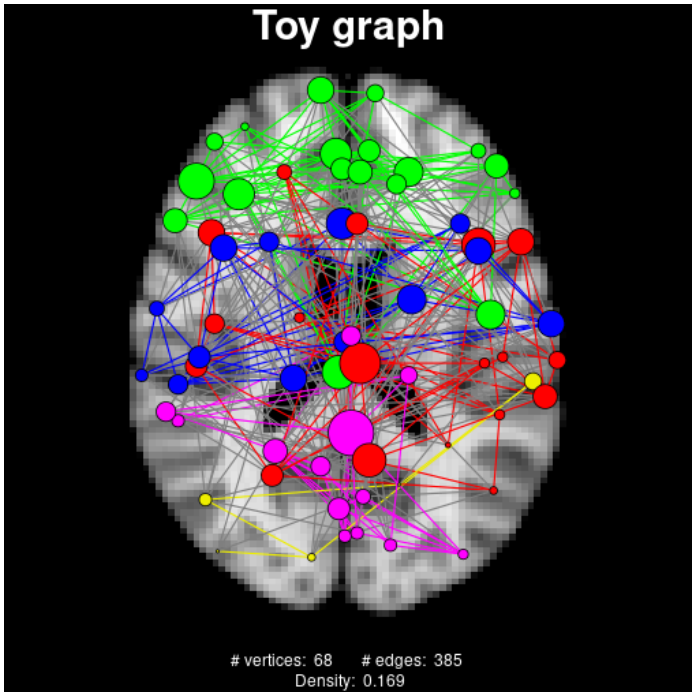

Figure 5.2: Axial view; vertex colors signify community membership.

New in v3.0.0

You can include the `label` argument to include text labels in the figures. Here, I include `A)` and `B)`.

```
plot(g.ex, plane='sagittal', hemi='L', label='A) ',
     vertex.label=NA, vertex.size=10, vertex.color='color.lobe',
     edge.color='color.lobe', edge.width=1, main='Toy graph (L)')

plot(g.ex, plane='sagittal', hemi='R', label='B) ',
     vertex.label=NA, vertex.size=10, vertex.color='color.lobe',
     edge.color='color.lobe', edge.width=1, main='Toy graph (R)')
```

5.4.4 Circular

Figure 5.4 shows a circular plot. As the legend indicates, vertex color indicates lobe membership. Vertices of the left hemisphere are located on the left half of the plot, frontal lobe vertices at the front, etc.

```
plot(g.ex, plane='circular', vertex.label=NA, vertex.size=5,
     vertex.color='color.lobe', edge.color='color.lobe',
     edge.width=1, show.legend=TRUE)
```

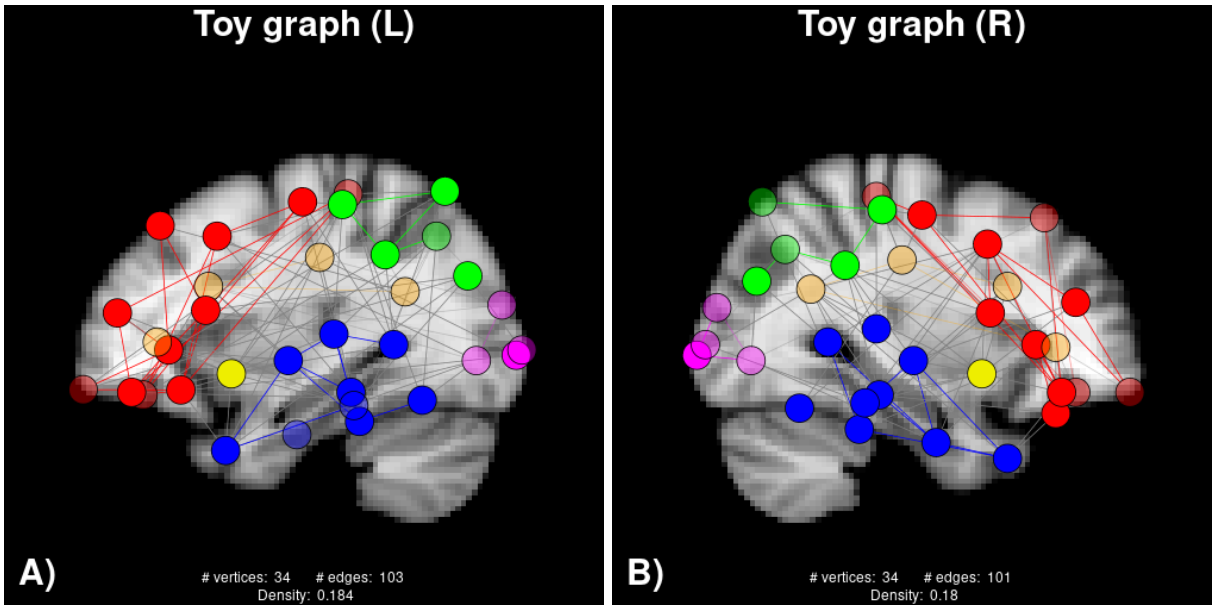

Figure 5.3: Sagittal view; vertex colors signify lobe membership.

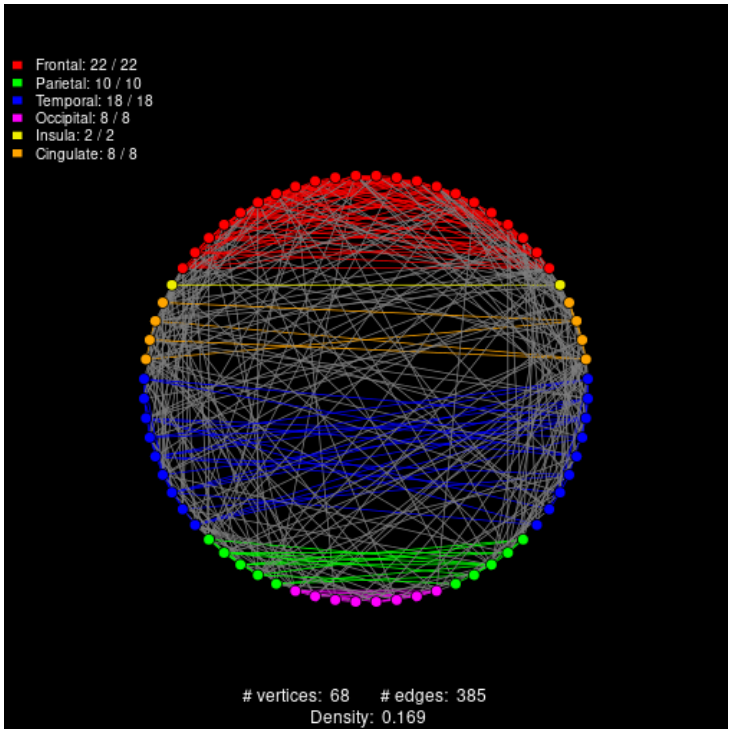

Figure 5.4: Circular view; vertex colors signify lobe membership.

Part III

Graph Creation

6

Structural covariance networks

This chapter shows how to create graphs of *structural covariance networks*. The code is specific to `Freesurfer` and cortical *thickness*, but will also work with *volume*, *surface area*, and *local gyrification index (LGI)*. It is possible that structural MRI data from other software packages are compatible, but (currently) the data files will have to conform to the outputs from Freesurfer.

6.1 Overview

The following is the sequence of steps for structural covariance network analysis. Each of these is discussed in subsequent sections of this chapter.

1. Load data and covariates
2. Generate model residuals (and check the QQ plots) (this is optional; you may also correlate the *raw* volumetric data)
3. Create correlation matrices (of residuals or raw volumetric data) for a number of densities/thresholds
4. Create a set of graphs, one for each group and density/threshold
5. Put the data into tables for easy exploration of graph- and vertex-level metrics (e.g., global efficiency, vertex degree, etc.)

6.1.1 Complete example

The following code block shows an example of a complete script for performing the above steps. Note that there are multiple ways of doing it, but this should get you started.

```
# 1. Import data & set up variables
datadir <- '/home/user/data'
exclude.subs <- c('sub005', 'sub013')
raw_data <- import_scn(datadir, atlas='dk', modality='thickness',
                       exclude.subs=exclude.subs)

# A hack to get the variables into the workspace
invisible(lapply(seq_along(raw_data), function(x)
   assign(names(raw_data)[x], eval(raw_data[[x]]), envir=.GlobalEnv)))

covars <- fread(file.path(datadir, 'covars.csv'))
covars <- covars[!Study.ID %in% exclude.subs]
grps <- covars[, levels(factor(Group))]

# 2. Get and check residuals
myResids <- get.resid(lhrh, covars=covars, exclude.cov='Group')
residPlots <- plot(myResids)
ml <- gridExtra::marrangeGrob(residPlots, nrow=3, ncol=3)
ggsave('residuals.pdf', ml)
system('xdg-open residuals.pdf')

# 3. Correlation matrices
densities <- seq(0.05, 0.20, 0.01)
corrs <- corr.matrix(myResids, densities=densities)

# 4. Create graph lists and calculate graph metrics
g <- lapply(seq_along(densities), function(x)
            make_brainGraphList(corrs[x], modality='thickness'))

# 5. data.tables of the metrics
dt.G <- rbindlist(lapply(g, graph_attr_dt))
dt.V <- rbindlist(lapply(g, vertex_attr_dt))
```

6.2 Import the data

The following code block is what I put in my `01_load_project.R` script; however, this script will have to be project-specific. The main purpose is to import the data through `import_scn`, which will do a few things:

- Changes the first column name to match the subject ID string; i.e., the value of `getOption('bg.subject_id')` .
- Removes the mean thickness (or volume, area, etc.) column(s) (if present)
- For *subcortical gray matter (SCGM)*: keeps only the first 15 columns (i.e., the subject ID column and the 14 SCGM data columns). Requires `asegstats.csv` to be in the same directory as the other files.
- Abbreviates the region names, for the DK and DKT atlases
- Removes subjects to be excluded from the data tables
- For atlases including SCGM: finds "missing" subjects; i.e., those that are not present in *both* the cortical and subcortical data files

You will need to have a few files in the `datadir`:

- `${parcellation}_lh_${modality}.csv`: See Structural data from Freesurfer for creating this file.
- `${parcellation}_rh_${modality}.csv`
- `asegstats.csv`: Only required if you are including subcortical data; e.g., subcortical volumes to be analyzed along with cortical thickness.

```
datadir <- '~/Dropbox/packages/brainGraph.other/data/Patient'
covars <- fread(file.path(datadir, 'covars.csv'), stringsAsFactors=TRUE)
covars[, Study.ID := as.character(Study.ID)]
grps <- covars[, levels(Group)]

# If you need to exclude subjects, specify here
# Best to use their Study.ID's for record-keeping
exclude.subs <- NULL  #c(2, 25, 30)
raw_data <- import_scn(datadir=datadir, atlas='dk', modality='thickness',
                       exclude.subs=exclude.subs)
invisible(lapply(seq_along(raw_data), function(x)
   assign(names(raw_data)[x], eval(raw_data[[x]]), envir=.GlobalEnv)))

# In case one density in particular is of interest, here 10%
densities <- seq(0.05, 0.20, 0.01)
N <- which(abs(densities - 0.10) < 0.001)
```

6.2.1 Custom atlas

To use a custom atlas, you must:

1. Specify `atlas='custom'` as the first argument
2. Supply the name of the R object (the `data.table`) for the custom atlas you would like to use.
3. Make sure the `data.table` is loaded in your R environment and matches the structure of the atlases in `brainGraph` (see Compatible atlases for details)

6.3 Model residuals

The next step is to get the model residuals. You can perform linear models on a per-group basis or across all groups at once. `get.resid` calculates the *studentized* residuals (sometimes called *leave-one-out residuals*).[1]

6.3.1 Function arguments

dt.vol A `data.table` of the structural data; this should be the object `lhrh` which is output by `import_scn`.

covars A `data.table` of covariates. It must have a column for both the subject ID and group, which depend on the value of `getOption('bg.subject_id')` and `getOption('bg.group')`, respectively.

method Here you can select to get residuals from a single linear model (the default; the option `'comb.groups'`) or a separate model for each group (`'sep.groups'`).

[1] In my data, the correlation between these and the *standardized* residuals (for all regions/models) was no lower than 0.9993.

use.mean Logical indicating whether or not to include the mean per-hemispheric structural measure in the models. Default: `FALSE`

exclude.cov A character vector of columns in `covars` that you would like to exclude from the models (if any).

6.3.2 Return value

It returns an object of class `brainGraph_resids`, which has three elements:

data A `data.table` with both the input volumetric data and the covariates

X The design matrix. If you chose to run separate linear models for each group, and/or to include the mean hemispheric structural measure, this will be a list of matrices.

method

use.mean

resids.all A "wide" `data.table` of residuals for each vertex. It is ordered first by `Group` (to more easily work with functions in the next step of the workflow).

Group Character vector with the group names

atlas

6.3.3 Example

Here I accept the default arguments except that I choose to exclude `Group` from the models.

```
resids <- get.resid(lhrh, covars=covars, exclude.cov='Group')
```

If you would like to get the residuals for each group separately, you simply include `method='sep.groups'` .

6.4 Data checking

It is always useful to check the quality of your data (which *should* have been done at a previous step). The `plot` method for structural covariance residuals plots a *qqplot* for the desired region(s) (see the Wikipedia page if you are unfamiliar with qqplots); these are useful for checking normality.

6.4.1 Summary

On visual inspection of my data, for *rSMAR* (right supramarginal gyrus), I saw that there is one sample quantile (i.e., one subject) with a value less than ≈ -3; this is very likely to be an outlier. To check which subject this is, you can look at the `summary` method output. When I looked at the subject's brain MRI, it turns out he/she had a stroke in the right supramarginal/inferior parietal lobe.
If you do not include a value for the `regions` argument, then the function prints outlier information for all regions (not shown).

```
summary(resids, region='rSMAR')

##
## ====================================
## Structural covariance residuals
## ====================================
## # of outliers for region rSMAR:
```

```
## 6

## Subjects that are outliers:
## ------------------

## [1] "029" "075" "098" "103" "125" "131"

## Outliers
## ------------------

##    Study.ID   Group Region resids
## 1:      125 Patient  rSMAR -3.264
## 2:      103 Control  rSMAR -3.173
## 3:      075 Control  rSMAR -2.462
## 4:      029 Control  rSMAR -2.050
## 5:      131 Control  rSMAR  2.073
## 6:      098 Patient  rSMAR  2.080
```

6.4.2 Plot

As an example of the plotting output, Figure 6.1 shows the qqplot for one region; one panel is the "standard" output, and the second shows points colored by group. In both, "outliers" (those further than 2 SD's from the mean) are triangles. The outliers are also marked with their number in the `data.table` of residuals.

```
plot(resids, region='rSMAR')[[1]]
plot(resids, region='rSMAR', cols=TRUE)[[1]]
```

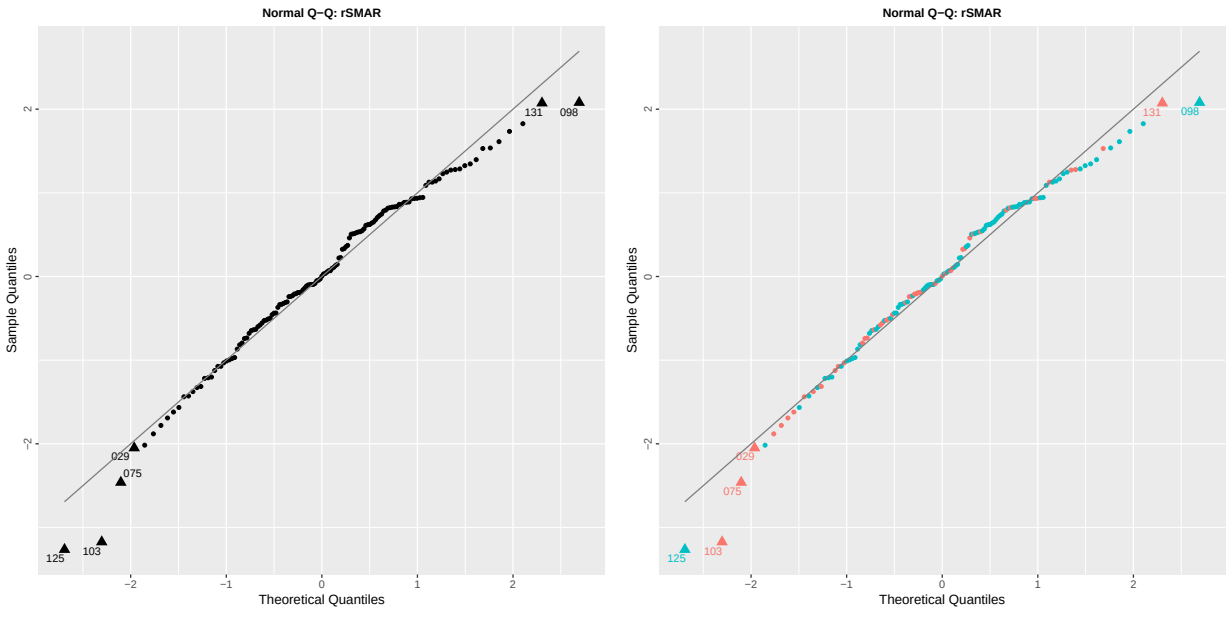

Figure 6.1: QQ plot of model residuals.

For all regions, you should save the plots to a multi-page PDF using the function `marrangeGrob` from the `gridExtra` package:

```
residPlots <- plot(resids)
ml <- gridExtra::marrangeGrob(residPlots, nrow=3, ncol=3)
ggsave('residuals.pdf', ml)
```

6.5 Correlation Matrix and Graph Creation

Next, correlate between pairs of regions for each group using `corr.matrix`. Simply use the output of `get.resid` for the first argument here.

The function `corr.matrix` has an argument, `density`, which indicates the density of the desired graph. So, if you want a graph with 10% of all possible connections, this value should be 0.1. To access the (correlation) threshold used, see code below.

```
corrs <- corr.matrix(resids, densities=densities)
corrs$thresholds

##       Control Patient
##  [1,]  0.5362  0.4846
##  [2,]  0.5192  0.4685
##  [3,]  0.5049  0.4560
##  [4,]  0.4936  0.4380
##  [5,]  0.4800  0.4234
##  [6,]  0.4714  0.4118
##  [7,]  0.4604  0.4030
##  [8,]  0.4520  0.3958
##  [9,]  0.4460  0.3869
## [10,]  0.4353  0.3747
## [11,]  0.4291  0.3667
## [12,]  0.4218  0.3581
## [13,]  0.4126  0.3498
## [14,]  0.4063  0.3409
## [15,]  0.3971  0.3360
## [16,]  0.3901  0.3284
```

Alternatively, you can skip the step in the previous section and just correlate the raw cortical thickness values. I don't recommend this, though, as variables like age and sex have known effects on cortical thickness. Simply include `what='raw'` in the call to `corr.matrix`.

6.5.1 Excluding regions

If you wanted to exclude some regions from your analysis (e.g., if you weren't interested in the L and R *transverse temporal*), use the following code:

```
exclude.reg <- c('lTT', 'rTT')
corrs <- corr.matrix(resids, densities=densities, exclude.reg=exclude.reg)
```

6.5.2 Graph creation

Then create the graphs, which will be in `brainGraphList` objects. The default behavior of `make_brainGraphList` will be to also set all graph-, vertex-, and edge-level attributes; to change the behavior, use `set_attrs=FALSE` .

Note

Compare the new code in the next block with the old code here:

```
g <- lapply(corrs, function(x)
            apply(x$r.thresh, 3,
                  graph_from_adjacency_matrix, mode='undirected', diag=F))
g <- Map(function(x, y) llply(x, set_brainGraph_attr, atlas=atlas,
                              modality=modality, group=y, .progress='text'),
         g, as.list(grps))
```

```
# Create simple, undirected graphs for each group
g <- lapply(seq_along(densities), function(x)
       make_brainGraphList(corrs[x], atlas='dk',
                           modality='thickness', .progress=F))
```

If you prefer that your graphs are weighted by the correlation coefficient, then you simply need to add `weighted=TRUE` to the above function call.

New in v3.0.0

An alternate method of getting a subgraph of e.g., the left hemisphere only, is to use the `subnet` argument:

```
regions.lh <- get(atlas)[hemi == 'L', name]
g.lh <- lapply(seq_along(densities), function(x)
              make_brainGraphList(corrs[x], atlas='dk',
                                  modality='thickness', subnet=regions.lh))
```

6.6 Getting measures of interest

There are a number of graph measures that may be of interest to you, so there are a couple of helper functions to make plotting and exploring easier. `graph_attr_dt` will get graph-level (global) attributes into a `data.table`, ordered by graph density. Similarly, `vertex_attr_dt` will get vertex-level attributes; for multiple densities, they are combined (by row) into a single `data.table`.

```
dt.G <- rbindlist(lapply(g, graph_attr_dt))
dt.V <- rbindlist(lapply(g, vertex_attr_dt))
setkey(dt.V, density, Group)

# Maximum degree for each Group & density
dt.V[density < 0.1, .SD[which.max(degree), .(region, degree)],
     by=.(Group, density)]

##       Group density region degree
##  1: Control 0.05004   rIPL     12
##  2: Patient 0.05004   rSPL     14
```

```
##  3: Control 0.06014   rSTG     14
##  4: Patient 0.06014   rSPL     15
##  5: Control 0.07024   rSTG     16
##  6: Patient 0.07024   lSPL     15
##  7: Control 0.08033  rPCUN     17
##  8: Patient 0.08033   lSPL     16
##  9: Control 0.09043   rSTG     19
## 10: Patient 0.09043   lSFG     17
```

```
# List the regions with the highest participation coefficient at one density
dt.V[abs(density - densities[N]) < .001,
     .SD[order(-PC, -degree)[1:5],.(density, region, lobe, hemi, degree, PC)],
     by=Group]
```

```
##       Group density region     lobe hemi degree     PC
##  1: Control  0.1001  rpTRI  Frontal    R     14 0.6173
##  2: Control  0.1001   rLOF  Frontal    R     10 0.6100
##  3: Control  0.1001  rrMFG  Frontal    R     11 0.5868
##  4: Control  0.1001  rpORB  Frontal    R     15 0.5556
##  5: Control  0.1001 rparaC  Frontal    R     13 0.5503
##  6: Patient  0.1001  rcMFG  Frontal    R     17 0.4394
##  7: Patient  0.1001   lSFG  Frontal    L     20 0.4375
##  8: Patient  0.1001   rSFG  Frontal    R     17 0.4325
##  9: Patient  0.1001  rSMAR Parietal    R     11 0.3884
## 10: Patient  0.1001  rpreC  Frontal    R     13 0.3728
```

```
# Count vertices with betweenness > mean + sd (i.e. 'hub' regions)
dt.V[round(density, 2) == densities[N] & hubs > 0, .N, by=Group]
```

```
##      Group  N
## 1: Control 34
## 2: Patient 34
```

```
# Mean degree by 'Group' and 'lobe', for one density
dt.V[round(density, 2) == densities[N],
     .(mean.deg=mean(degree)),
     by=.(Group, lobe)]
```

```
##       Group      lobe mean.deg
##  1: Control  Temporal    6.833
##  2: Control Cingulate    0.250
##  3: Control   Frontal    6.409
##  4: Control Occipital    5.375
##  5: Control  Parietal   13.000
##  6: Control    Insula    8.500
##  7: Patient  Temporal    5.444
##  8: Patient Cingulate    0.875
##  9: Patient   Frontal    8.318
## 10: Patient Occipital    6.250
## 11: Patient  Parietal   10.800
## 12: Patient    Insula    5.000
```

6.7 Long and wide data tables

To make some plotting functions and data exploration easier, it's important to "melt" the data into "long" format (see Wickham (127)). The "melt" operation is an example of *data reshaping*. This step is usually *optional*, but can make data exploration simpler.

```
# For a given density, vertex-wise network measures
charCols <- names(which(sapply(dt.V, is.character)))
dt.V.long <- melt(dt.V, id.vars=c('density', charCols))

# Number of rows = (number of vertices) * (number of variables)
dt.V.long[seq(1, nrow(dt.V.long), nrow(get(atlas)))]

##      density region     lobe hemi atlas  modality   Group  variable value
##   1: 0.05004  lBSTS Temporal    L    dk thickness Control    degree  2.00
##   2: 0.05004  lBSTS Temporal    L    dk thickness Patient    degree  1.00
##   3: 0.06014  lBSTS Temporal    L    dk thickness Control    degree  2.00
##   4: 0.06014  lBSTS Temporal    L    dk thickness Patient    degree  1.00
##   5: 0.07024  lBSTS Temporal    L    dk thickness Control    degree  2.00
##  ---
## 700: 0.18042  lBSTS Temporal    L    dk thickness Patient threshold  0.18
## 701: 0.19008  lBSTS Temporal    L    dk thickness Control threshold  0.19
## 702: 0.19008  lBSTS Temporal    L    dk thickness Patient threshold  0.19
## 703: 0.20018  lBSTS Temporal    L    dk thickness Control threshold  0.20
## 704: 0.20018  lBSTS Temporal    L    dk thickness Patient threshold  0.20

# Example: melting the global network measures data.table
# Number of rows = (number of densities) * (number of global measures)
char_cols <- names(which(sapply(dt.G, is.character)))
(dt.G.long <- melt(dt.G, c('density', char_cols)))

##      density atlas  modality   Group      variable   value
##   1: 0.05004    dk thickness Control     threshold 0.05000
##   2: 0.05004    dk thickness Patient     threshold 0.05000
##   3: 0.06014    dk thickness Control     threshold 0.06000
##   4: 0.06014    dk thickness Patient     threshold 0.06000
##   5: 0.07024    dk thickness Control     threshold 0.07000
##  ---
## 540: 0.18042    dk thickness Patient vulnerability 0.06416
## 541: 0.19008    dk thickness Control vulnerability 0.02699
## 542: 0.19008    dk thickness Patient vulnerability 0.06290
## 543: 0.20018    dk thickness Control vulnerability 0.02603
## 544: 0.20018    dk thickness Patient vulnerability 0.06613
```

7

Tractography and fMRI

This section will list the code needed to get *fdt_network_matrix* and *waytotal* into `R` so you can create and work with graphs. This example code uses the `dkt.scgm` atlas and 2 subject groups. I used `FSL` for DTI-related processing; see relevant references (8, 9, 60, 61).

Note

The information in this chapter is not specific to `FSL`, nor to DTI tractography. The code is applicable to any set of single-subject graphs. If you use different software with different outputs, just adjust the code (e.g., filenames) accordingly. I call the covariates object `covars.dti` simply because I don't want it to be over-written if I am loading data from multiple modalities concurrently and the covariates set is different for each. In my fMRI scripts, I call it `covars.fmri`.

7.1 Setting up

First, as mentioned in Setting up files for your project, you will need to load the required packages. Then, you set some initial variables that will depend on your project, data, directory structure, etc.

7.1.1 Tractography

The files containing connectivity matrices must contain the `Study.ID`'s for your study in the file names. For example, the directory structure might look like:

```
sub001-fdt_network_matrix
sub001-sizes.txt
sub001-waytotal
sub002-fdt_network_matrix
sub002-sizes.txt
sub002-waytotal
etc.
```

The `*-sizes.txt` files contain the ROI volume (in # of voxels) for each of the 76 ROI's (i.e., it contains a single column vector with 76 elements). The *waytotal* files have the same format. The *fdt_network_matrix* files are output by `probtrackx2` and are 76×76 matrices (in this example).

You can place the following code in a script, e.g., `01_load_DTI.R`. Note again that these files are *not* required; if you use a different software for tractography (or for fMRI), just change the relevant lines in the following code block.

```
#============================================================================
# These variables need to be set correctly before any data analysis is done
#============================================================================
grps <- c('Control', 'Patient')

# Get all relevant filenames
datadir <- '/home/cwatson/probtrackx_results'
covars.all <- fread(file.path(datadir, 'covars.all.csv'))
covars.dti <- covars.all[tract == 1, 1:5, with=F]
covars.dti[, Group := as.factor(Group)]
covars.dti[, Scanner := as.factor(Scanner)]
setkey(covars.dti, Study.ID)

matfiles <-
  list(A=list.files(datadir, pattern='fdt_network_matrix', full.names=T),
       way=list.files(datadir, pattern='waytotal', full.names=T),
       size=list.files(datadir, pattern='sizes.txt', full.names=T))
inds <- lapply(grps, function(x) covars.dti[Group == x, which=TRUE])

# Output directory to save the data
today <- format(Sys.Date(), '%Y-%m-%d')
savedir <- file.path('/home/cwatson/brainGraph', today)
```

7.1.2 fMRI

The code for resting-state fMRI is almost exactly the same, except the matrix files are different. The following code example is using the outputs of `DPABI`:

```
matfiles$A <- list.files(datadir, 'ROICorrelation_[a-z]+.*.txt', full.names=T)
```

7.2 Import, normalize, and filter matrices for all subjects

Now we will load all the relevant data from the files provided, and normalize the connection matrices based on what you want (e.g., divide every entry by the corresponding *waytotal*). For resting-state fMRI, the thresholds will likely be different (e.g., correlation coefficients), or you may choose to threshold in such a way that the graphs have a specific density (but see (116)). Either way, the same function is used. In this section, I describe the inputs and outputs of `create_mats`.

7.2.1 Function arguments

New in v3.0.0

You can also specify directories and/or patterns for the `A.files` and `div.files` arguments. In addition to supplying a *character vector* of filenames, you can:

- Specify a *single character string* naming the directory in which the files exist. This will load *all* files in the directory.
- Specify a *named list* in which the names match arguments to `list.files`. For example, this may be `list(path='/data/probtrack_results', pattern='.*fdt_network_matrix')` . You may also specify `recursive=TRUE` to search all child directories.

A.files A character vector of the filenames containing the connectivity matrices (and see above).

modality A character string; either `dti` (default) or `fmri`.

divisor A character string specifying how to normalize the matrices. Either `none` (default), `waytotal`, `size` (normalize by average size of ROI pairs), or `rowSums` (normalize by the row sums of the connection matrix). Ignored if `modality='fmri'` .

div.files Character vector of the filenames of the files containing the normalization factor (e.g., the *waytotal* files from `FSL`'s `probtrackX2`). Ignored if `divisor='none'` .

threshold.by Character string with 5 possible options. The 3rd and 4th options will enforce the same connections across all study subjects.

consensus Perform "consensus-based" thresholding; i.e., keep connections that are above a given threshold for a certain percentage of subjects *in each group*. The default value for `sub.thresh` of 0.5 means it will keep connections if they are present in at least 50% of subjects. See de Reus and van den Heuvel (24).

density Threshold the matrices such that they result in a specific graph density. The values given to `mat.thresh` must be between 0 and 1. See van den Heuvel et al. (116).

mean You may choose to specify a set of thresholds τ (which you would supply to `mat.thresh`) and keep connections only if

$$mean(A_{ijk}) + 2 \times SD(A_{ijk}) > \tau$$

where A_{ijk} is a connectivity matrix, and k indexes *Subject*. See for example (14, 43).

consistency Perform "consistency-based" thresholding (94). Similar to specifying `density`, you supply the desired graph densities to `mat.thresh`, and the matrices are thresholded to keep the most consistent connections across all study subjects (as determined by the *coefficient of variation*).

raw Threshold each subject individually, irrespective of group membership. Ignores the `sub.thresh` argument.

mat.thresh Numeric vector of the thresholds to apply. See the description for `threshold.by` for how this is applied. Default: `0` . These values may end up being arbitrary, but with *deterministic tractography* you may choose *streamline counts*; for *probabilistic tractography*, this may be some measure of *streamline density* or *connectivity probability*; and with *resting-state fMRI* you may choose to threshold based on *correlation coefficients*.

sub.thresh Numeric (between 0 and 1); only valid if `threshold.by='consensus'` . Default: `0.5`

inds List (number of elements equal to the number of groups) containing integer vectors; the integers should represent the indexes for each group, and the length of each individual vector should equal the number of subjects per group. For example, if you have 3 groups of 12 subjects each, this would be: `inds=list(1:12, 13:24, 25:36)` .

algo Character string specifying the tractography algorithm used; either `probabilistic` (the default) or `deterministic`. Ignored if `modality='fmri'` .

P Integer; the number of samples per voxel for probabilistic tractography (default: `5000`). Only valid if `algo='probabilistic'` .

... Other arguments passed to `symmetrize`. Here you can pass the argument `symm.by` which tells the function how to symmetrize the matrices. The default in `igraph` is to take the *maximum* of $\{A_{ij}, A_{ji}\}$, so the default option is `symm.by='max'` . You may also specify `min` or `avg` .

7.2.2 Return value

`create_mats` returns a list (of lists) of arrays. I use the following abbreviations:

N_v The # of vertices in the graph

N_s The # of subjects (total, across all groups)

N_g The # of groups in the study

N_τ The # of thresholds applied

Most of the elements in the return object are *lists*; otherwise, they are 3D (numeric) *arrays*.

A The raw connection matrices (from e.g., *fdt_network_matrix*). Dimensions of this array are $N_v \times N_v \times N_s$.

A.norm The normalized connection matrices (e.g., `A ÷ waytotal`). Dimensions are the same as for `A`. This is different from `A` only if `modality='dti'` ; further, for *deterministic* tractography only if `divisor='size'` .

A.bin List of binarized matrices based on some *threshold*. The number of elements in the list equals the number of thresholds, i.e., N_τ arrays of size $N_v \times N_v \times N_s$. Only valid if `threshold.by='consensus'`

A.bin.sums A list of 3-d arrays, in which each entry represents the total number of subjects with that connection present (from `A.bin`). This will be used to threshold by % of subjects. The number of elements in this list equals the number of thresholds; the extent of the arrays equals the number of groups. So, there are N_τ arrays of $N_v \times N_v \times N_g$ arrays. Only valid if `threshold.by='consensus'` .

A.inds A list of 3-d binary arrays, in which a *1* indicates that a connection is present *for that group*. The dimensions are the same as for `A.bin.sums`. Only valid for *consensus-* and *consistency-based* thresholding.

A.norm.sub A list of 3-d arrays with the same dimensions as `A.bin`; i.e., N_τ arrays of size $N_v \times N_v \times N_s$. However, the connections which were deemed absent (i.e., if too few subjects have the connection) have been replaced by *0*. All of these matrices will be *symmetrized* based on the value given to `symm.by` . Furthermore, the order along the 3rd dimension matches that of the input matrix files (i.e., the input argument `A.files`), and therefore also the raw matrices and normalized matrices in `A` and `A.norm`, respectively. (Thanks to `@seantma` for pointing out the bug that led to this fix!)

A.norm.mean A list of 3-d arrays with the same dimensions as `A.bin.sums` and `A.inds`. Each matrix in this list is the corresponding group average (based on `A.norm.sub`).

7.2.3 Code example

First, I set the *subject threshold* to 0.5, meaning I will only accept connections which are present in at least 50% of the subjects of a given group. If you do not want to impose any subject constraint, set equal to 0. The *matrix threshold* is determined by the variable `thresholds`, which may end up being arbitrary (i.e., it will depend on the tractography algorithm and the actual values of the matrices, and would be different if you used correlations for fMRI). For *deterministic* tractography, this could be equal to the number of streamlines connecting two ROI's (e.g., an integer between 1 and 10, or higher).

```
modality <- 'dti'
thresholds <- rev(seq(0.001, 0.01, 0.001))
sub.thresh <- 0.5
divisor <- 'waytotal'
my.mats <- create_mats(matfiles$A, modality=modality, divisor=divisor,
                       div.files=matfiles$way, mat.thresh=thresholds,
                       sub.thresh=sub.thresh, inds=inds)
A.norm.sub <- my.mats$A.norm.sub
A.norm.mean <- my.mats$A.norm.mean
```

7.2.4 Applying the same thresholds to other matrices

In the case where you have connectivity matrices in which the entries are from the same subjects but are a different metric (e.g., in DTI tractography *streamline count* and *mean FA*), you can use the function `apply_thresholds` to threshold the second set by the first. See the following code block and Ref. (69).

```
matfiles$W <- list.files(datadir, pattern='.*W.txt', full.names=T)
if (length(matfiles$W > 0)) {
  W.mats <- apply_thresholds(A.norm.sub, A.norm.mean, matfiles$W, inds)
  W.norm.sub <- W.mats$W.norm.sub
  W.norm.mean <- W.mats$W.norm.mean
}
```

7.3 Graph creation

We now create graphs from the matrices from the previous section, and calculate the relevant attributes for these graphs.

In this case, I will use the list of arrays stored in `A.norm.sub` (which contains the arrays thresholded by both the % of subjects with a connection, and by a series of thresholds). The list `g` will have multiple `brainGraphList` objects (1 for each threshold). You may want to place the following code in its own script, e.g., `02_create_graphs_DTI.R`.

Note

Compare the old code in this box to the new code in the block that follows. This is meant to highlight the improvements in `v3.0.0`.

```
g.group <- g <- fnames <- vector('list', length=length(grps))

for (i in seq_along(grps)) {
  for (j in seq_along(thresholds)) {
    foreach (k=seq_along(inds[[j]])) %dopar% {
      g.tmp <- graph_from_adjacency_matrix(A.norm.sub[[j]][, , inds[[i]][k]],
                                           mode='undirected', diag=F, weighted=T)
      g.tmp <- set_brainGraph_attr(g.tmp, atlas, modality='dti',
          weighting='sld', threshold=thresholds[j],
          subject=covars.dti[groups[i], Study.ID[k]], group=grps[i],
          use.parallel=FALSE, A=A.norm.sub[[j]][, , inds[[i]][k]])
      saveRDS(g.tmp, file=paste0(savedir,
                                 sprintf('g%i_thr%02i_subj%03is', i, j, k, '.rds')))
    }
  }

  # Group mean weighted graphs
  g.group[[i]] <- lapply(seq_along(thresholds), function(x)
                         graph_from_adjacency_matrix(A.norm.mean[[x]][[i]],
                                                     mode='undirected', diag=F,
                                                     weighted=T))
  g.group[[i]] <-
    llply(seq_along(thresholds), function(x)
          set_brainGraph_attr(g.group[[i]][[x]], atlas,
                              modality='dti', weighting='sld',
                              threshold=thresholds[x], group=grps[i],
                              A=A.norm.mean[[x]][[i]], use.parallel=FALSE),
          .parallel=TRUE)
}
```

```
atlas <- 'dkt.scgm'
g <- g.group <- vector('list', length(thresholds))
for (j in seq_along(thresholds)) {
  g[[j]] <- make_brainGraphList(A.norm.sub[[j]], atlas, modality=modality,
                                weighting='sld', threshold=thresholds[j],
                                weighted=TRUE, gnames=covars.dti$Study.ID,
                                groups=covars.dti$Group)

  g.group[[j]] <- make_brainGraphList(A.norm.mean[[j]], atlas, modality=modality,
                                      weighting='sld', threshold=thresholds[j],
                                      weighted=TRUE, gnames=grps)
}
saveRDS(g, file=file.path(savedir, 'g.rds'), compress='xz')
saveRDS(g.group, file=file.path(savedir, 'g.group.rds'), compress='xz')
```

7.4 Graph- and vertex-level measures

Just as we did in Getting measures of interest, we will create `data.table`'s of graph- and vertex-level measures for further analysis. (This is also much more compact in the latest version.)

```
dt.V <- rbindlist(lapply(g, vertex_attr_dt))
dt.G <- rbindlist(lapply(g, graph_attr_dt))
dt.V.group <- rbindlist(lapply(g.group, vertex_attr_dt))
dt.G.group <- rbindlist(lapply(g.group, graph_attr_dt))

dt.V.group$sub.thresh <- dt.G.group$sub.thresh <- dt.G$sub.thresh <-
    dt.V$sub.thresh <- sub.thresh
setorderv(dt.V, 'threshold', -1)
setorderv(dt.G, 'threshold', -1)

idvars <- c('modality', 'atlas', 'weighting', 'sub.thresh', 'threshold', 'Group')
dt.G.long <- melt(dt.G, id.vars=c(idvars, 'Study.ID', 'density'))
dt.G.group.long <- melt(dt.G.group, id.vars=idvars)
```

7.5 Example commands

Similar to Getting measures of interest, here are some example commands for looking at your data.

```
# Mean degree for the group-averaged graphs
dt.V.group[, .(mean.deg=mean(degree)), by=.(Group, threshold, lobe)]

##         Group threshold      lobe mean.deg
##    1: Control     0.005      SCGM    16.71
##    2: Control     0.005 Occipital    10.88
##    3: Control     0.005  Temporal    11.14
##    4: Control     0.005    Insula    22.00
##    5: Control     0.005  Parietal    10.70
##   ---
## 206: Patient     0.004  Temporal    10.93
## 207: Patient     0.004    Insula    22.00
## 208: Patient     0.004  Parietal    10.40
## 209: Patient     0.004   Frontal    11.50
## 210: Patient     0.004 Cingulate    13.25

# t-test of nodal efficiency for Frontal lobe vertices
dt.V.group[lobe == 'Frontal',
           .(p=t.test(E.nodal ~ Group)$p.value),
           by=threshold]

##     threshold         p
##  1:     0.005 2.482e-01
##  2:     0.015 2.373e-01
##  3:     0.025 1.735e-01
##  4:     0.035 9.670e-02
##  5:     0.045 1.577e-01
##  6:     0.055 3.196e-02
```

```
##  7:     0.065 1.824e-05
##  8:     0.075 8.523e-03
##  9:     0.085 5.396e-02
## 10:     0.095 2.816e-03
## 11:     0.105 2.775e-01
## 12:     0.001 3.575e-01
## 13:     0.002 2.985e-01
## 14:     0.003 3.049e-01
## 15:     0.004 4.160e-01
```

We can also create a plot of the global graph measures across the thresholds we applied, shown in Figure 7.1. Here I use my function `plot_global`, specifying that I want to plot across `threshold` instead of `density`. This uses the `stat_smooth` function, which creates a line plot along with a smoother.

```
exclude.vars <- c('assortativity.lobe.hemi', 'max.comp', 'diameter.wt',
                  'clique.num', 'num.tri')
plot_global(g.l, xvar='threshold', exclude=exclude.vars)
```

Figure 7.1: Global graph measures vs. density, DTI data.

Part IV

Group Analyses: GLM-based

8

Vertex-wise group analysis (GLM)

Analysis of between-group differences in a given vertex measure (e.g., *degree*, *betweenness centrality*, etc.) is described in this chapter (equivalent to a voxel-wise analysis common in fMRI, DTI, VBM, etc. analyses). This is only possible if you have a graph for each subject (at a given density/threshold). In this chapter, I describe the inputs and outputs of the function `brainGraph_GLM`. Next, I provide a "mini-tutorial" on design matrix coding and provide several examples of common experimental designs. Finally, I show an example using *permutations* (as in FSL's *randomise*).(40, 80, 129)

8.1 Function arguments

```
args(brainGraph_GLM)

## function (g.list, covars, measure, contrasts, con.type = c("t",
##     "f"), outcome = NULL, X = NULL, con.name = NULL, alternative = c("two.sided",
##     "less", "greater"), alpha = 0.05, level = c("vertex", "graph"),
##     permute = FALSE, perm.method = c("freedmanLane", "terBraak",
##         "smith", "draperStoneman", "manly", "stillWhite"), part.method = c("beckmann",
##         "guttman", "ridgway"), N = 5000, perms = NULL, long = FALSE,
##     ...)
## NULL
```

New in v3.0.0

There are a few new arguments in v3.0.0. These are described in more detail below:

outcome If the model outcome differs from the graph metric

perm.method The permutation method, which includes 3 new methods (see under "Optional" arguments)

part.method The model partition method

Furthermore, `con.mat` has been renamed to `contrasts` because *lists* of matrices (for multiple F-contrasts) are allowed in addition to single matrices.

8.1.1 Mandatory

The following list details the *mandatory* function arguments:

g.list A `brainGraphList` object. If your study has multiple groups, this object should contain graphs for all subjects.

covars A `data.table` of covariates which will be used to create the *design matrix*. It should have as its first column `Study.ID` (or whatever the value of `getOption('bg.subject_id')` is); the data in this column must match the `name` graph-level attribute of the graphs in `g.list` (or at least a subset of them). You may include any additional columns of your choosing (e.g., age, sex, etc.).

measure The vertex- or graph-level measure of interest (e.g., *E.nodal.wt*).

contrasts A `list` or numeric `matrix` specifying the contrast(s) of interest. The contrasts can have multiple rows, and you can specify row names. If you provide a list of matrices, then the list element names will be used.

con.type Either `t` or `f`, if you want to calculate t- or F-statistics. F-statistics are by definition "two-sided". Otherwise, this choice only affects whether statistics are calculated for each row of the contrast matrix (`t`) or a single set for the whole matrix or each matrix in the list (`f`). If you pass a `list` of matrices to `contrasts`, then this must be `f`.

8.1.2 Optional

The following are *optional* function arguments:

outcome If you would like the outcome (dependent variable) to be something other than a graph metric, you can specify it here. In this case, the graph metric specified by `measure` will be included in the design matrix as a covariate. At the *vertex-level*, there will be one design matrix per region (since the graph metric differs for each vertex).

X If you wish to provide your own design matrix, you can specify it with this argument. Note that, if you do this, you still need to provide the `covars` data table; this is used for making sure the covariates and the graph metrics are in the same order, and to remove the appropriate data if some are missing for certain subjects. However, the function will assume that you have correctly created your own design matrix, and doesn't do any checking.

> **Note**
>
> If you specify a variable for `outcome`, then passing in a design matrix is ignored. This argument will be deprecated in a future version.

con.name A character vector of the name(s) of the contrast(s); e.g., `'Control > Patient'` . If this argument is not provided, the function first checks if `contrasts` has row names (if it is a matrix) or list names otherwise. You can create these yourself. If names are not supplied, they will be generic; e.g., `'Contrast 1'`

alternative Either `two.sided` (the default), `less`, or `greater` corresponding to (respectively)

$$H_A : \hat{\gamma} \neq 0$$
$$H_A : \hat{\gamma} \leq 0$$
$$H_A : \hat{\gamma} \geq 0$$

alpha The significance level. Default: `0.05`

level Either `vertex` or `graph`, depending on if the measure of interest is vertex- or graph-level.

Permutation-related arguments

The following optional arguments pertain to *permutation/randomization* tests (see Permutation testing):

permute A *logical* indicating whether or not to do a permutation test. Default: `FALSE`

perm.method The permutation method to use. The default is to use the `freedmanLane` method (40). You may also choose `terBraak` (2), `smith` (108), `draperStoneman` (31), `manly` (77), or `stillWhite` (109).

part.method The method used to partition the design matrix. The default is the `beckmann` method (7). You may also choose `guttman` (46) or `ridgway` (93).

N An *integer* indicating the number of permutations to create. Default: `5,000`

perms A numeric matrix of the permutation order, if you would like to provide your own.

long A *logical* specifying whether or not to return the null distribution of the maximum statistics across permutations. Default: `FALSE`

8.1.3 Unnamed: design matrix arguments

The ellipsis (i.e., the `...`) in a function call specifies unnamed arguments that are passed to other functions. In `brainGraph_GLM`, they are passed to `brainGraph_GLM_design`, a function that simply creates a design matrix `X`. Here I list the arguments that you may specify in your function call.

coding A character string, either `dummy` (default), `effects`, or `cell.means`. This determines how your factor variables will be coded in the design matrix. See Tutorial: design matrix coding for more.

factorize A *logical* specifying whether or not to convert *character* columns to *factor*. Default: `TRUE`

binarize A *character* vector of the column name(s) of `covars` to convert from *factor* to *numeric* (binary) vector. If the factor has $k > 2$ levels, it will not be binarized but instead changed to $0, 1, \ldots, k-1$.

int A *character* vector of the column names of `covars` to test for an interaction. The resulting columns will be appended to the *end* of the design matrix. If you provide only one column, nothing will happen. If you provide 3, then the 3-way *and all 2-way interactions* will be added to the design.

mean.center A *logical* specifying whether or not to mean-center numeric variables. Default: `FALSE` . Mean-centering will be done for all subjects, by default, but see the next bullet point. Note that any factor variables that were binarized will also be mean-centered.

center.how A character string specifying whether variables should be centered based on `all` subjects (the default), or `within-groups`. The groups are specified by the `center.by` argument (see next bullet point). Although this may not be the most common choice in neuroimaging, see this article on the *AFNI* site for an excellent discussion.

center.by A character string specifying which variable is the grouping variable (only valid if `center.how='within-groups'`). By default, it will center according to the `Group` variable, if present. If your design has factor interactions (e.g., *Group X Sex*), then you may want to specify both variables here; i.e., `center.by=c('Group', 'Sex')` .

Note

It is recommended that, if you include interaction terms, you should also mean-center the other variables to partially reduce multicollinearity (e.g., see Chapter 8.2 of Kutner et al. (70)).

To summarize, columns in the design matrix `X` will be in a different order than the input `covars`. The first column will be the *Intercept* (a column of all 1's), if `coding` is not set to `cell.means`. The next column(s) will by any numeric variables, followed by *factor* column(s), and finally *interaction* columns (if applicable). You may wish to inspect the design matrix before specifying your contrast matrix by typing, e.g., `head(brainGraph_GLM_design(covars, ...))` .

8.2 Return object

This function returns an object of class `bg_GLM` with several of the input arguments in addition to a `data.table` of the statistics of interest. For details on the statistics calculated, see GLM Statistics.

New in v3.0.0

There are several new elements of the return object in v3.0.0.

outcome Can now be different from (or the same as) `measure`

measure Formerly called `outcome`

DT.Xy A `data.table` with all design matrix and outcome variable data

contrasts Renamed from `con.mat`

removed.subs Renamed from `removed`

runX,runY Regions for which a model can be fit based on the design matrix and outcome variable, respectively

atlas

GLM statistics These are `coefficients`, `rank`, `df.residual`, `residuals`, `sigma`, `fitted.values`, `qr`, `cov.unscaled`, `var.covar`, `se`. See GLM Statistics for details.

perm.order The permutation matrix applied to the data

perm.method

part.method

perm The element `null.dist` is now a 3-D array, and `null.max` is a column matrix with the maximum statistic for each permutation

level Either `vertex` or `graph`

X The *design matrix*

y The *outcome variable*. If `level='graph'` , this is a *numeric vector*; If `level='vertex'` , this is a numeric matrix. The number of rows equals the number of subjects, and the number of columns equals the number of vertices.

outcome The name of the *outcome variable*; this will either be the graph metric chosen by the `measure` function argument or the variable specified by the `outcome` argument.

measure The graph- or vertex-level metric of interest.

covars The data table used to create the design matrix. If any subjects have incomplete data, they will not be included in the table.

con.type Either `'t'` or `'f'`.

contrasts A *contrast matrix* or *list* of matrices (for F-tests only).

con.name The contrast name(s).

alt The *alternative hypothesis*.

alpha The significance level.

DT A `data.table` containing statistics of interest; see the next section for details.

removed.subs A character vector of the subjects removed from the analyses (due to incomplete data). If none were removed, this will be `NULL` .

permute A *logical* indicating whether or not permutations were calculated.

perm.method The permutation method used.

part.method The model partition method used.

N The number of permutations (if applicable).

perm A *list* containing:

- **thresh** The $(1-\alpha)$th percentile of the null distribution.
- **null.dist** A 3-D array of the null distribution of the maximum statistic. The # of rows equals the # of regions, # of columns equals # of permutations, and the length of the 3rd dimension equals the # of contrasts.
- **null.max** A *matrix* of the maximum statistic for each permutation (across regions). The # of rows equals the # of permutations and the # of columns equals the # of contrasts.

atlas The name of the atlas for the input graphs.

8.2.1 Return object: `DT`

The `data.table` object that is returned contains all statistics of interest. Each row is a different vertex (brain region) or a single row for graph-level metrics. Additionally, statistics are listed for each contrast (if more than one is specified).

For *t-contrasts*, the columns are described in the following list. For *F-contrasts*, the only difference is that `ESS` (the *extra sum-of-squares*) replaces `gamma`.

region The region name (abbreviated)

gamma The contrast(s) of parameter estimates

se The standard error of the contrast(s)

stat The *t-* or *F-statistic(s)* associated with the contrast(s)

p The *p-value* associated with the contrast(s)

ci.low The lower confidence limit

ci.high The upper confidence limit

p.fdr The FDR-adjusted p-value

p.perm The permutation p-value, if `permute=TRUE`

Outcome The name of the outcome variable (e.g., `E.nodal.wt`)

Contrast The name of the contrast(s)

contrast Integer indicating the contrast number

8.3 Tutorial: design matrix coding

8.3.1 Suggested reading

A very good website with MRI-focused information on design matrices in the GLM is FSL's GLM site. For more information regarding coding schemes in linear models, see (63, 84, 104, 128). For a complete treatment of this topic, see Kutner et al. (70); they usually use *dummy* coding, but see Chapter 8.4 where they introduce *effects* and *cell means* coding (see the sub-section `Other codings for indicator variables`). For some information regarding how coding is handled in `R`, see this UCLA page. Finally, Anderson Winkler's blog has a great post of common GLM formulas.

8.3.2 Dummy coding

The default parameterization of factors in `R` is known as *reference*, or *dummy*, coding. Let's say we have a simple one-way ANOVA (a factor named *Group*) with two levels (*Control* and *Patient*). In this scheme, instead of having a column in the design matrix `X` for each level, there is only a column for the second level (*Patient*) in addition to the *Intercept*. You can use this type of coding by setting `coding='dummy'` in the call to `brainGraph_GLM`. There is more on dummy coding at this UCLA page.

The parameter estimate for the intercept represents the mean of the first (alphabetically, by default) group, and the second parameter estimate is the mean of the second group *relative to the first group*. In equation form (generalized to a $1 \times k$ ANOVA):

$$\begin{aligned}\hat{\beta}_0 &= \mu_1 \\ \hat{\beta}_1 &= \mu_2 - \mu_1 \\ &\vdots \\ \hat{\beta}_{k-1} &= \mu_k - \mu_1\end{aligned}$$

To compare group means, assume there are $k = 3$ factor levels. Group 1 (the reference group) is coded `+1` for the intercept and `0` for the other parameters; group 2 is coded `+1` for the intercept and second parameter, and `0` for the third:

$$\begin{aligned}H_A &: \mu_1 - \mu_2 > 0 \\ &\Rightarrow (\hat{\beta}_0) - (\hat{\beta}_0 + \hat{\beta}_1) > 0 \\ &\Rightarrow -\hat{\beta}_1 > 0 \\ &\Rightarrow C = (0, -1, 0)\end{aligned}$$

$$\begin{aligned}H_A &: \mu_1 - \mu_3 > 0 \\ &\Rightarrow (\hat{\beta}_0) - (\hat{\beta}_0 + \hat{\beta}_2) > 0 \\ &\Rightarrow -\hat{\beta}_2 > 0 \\ &\Rightarrow C = (0, 0, -1)\end{aligned}$$

For a general between-group comparison (that does not involve the reference group), the contrast has a `+1` in the i^{th} entry and a `-1` in the j^{th} entry:

$$\begin{aligned}H_A &: \mu_i - \mu_j > 0 \\ &\Rightarrow (\hat{\beta}_{i-1} + \hat{\beta}_0) - (\hat{\beta}_{j-1} + \hat{\beta}_0) > 0 \\ &\Rightarrow \hat{\beta}_{i-1} - \hat{\beta}_{j-1} > 0 \\ &\Rightarrow C = (0, 0, \cdots, 1, 0, \cdots, -1, 0, 0, \cdots, 0)\end{aligned}$$

```
# Create a data.table with just Study.ID and Group
X <- env.glm$covars.dti[, 1:2]
setkey(X, Group, Study.ID)
print(X, 4)

##      Study.ID   Group
##   1: 02-115-0 Control
##   2: 02-126-1 Control
##   3: 02-127-2 Control
##   4: 02-128-9 Control
##  ---
## 153: 02-629-8 Patient
## 154: 02-630-6 Patient
## 155: 02-631-5 Patient
## 156: 02-632-2 Patient

# The Control group is chosen to be the reference group
X[, levels(Group)]

## [1] "Control" "Patient"

# There are only 2 columns, with the second indicating membership in Group 2
model.matrix(~ Group, data=X)[c(1:4, 153:156), ]

##     (Intercept) GroupPatient
## 1             1            0
## 2             1            0
## 3             1            0
## 4             1            0
## 153           1            1
## 154           1            1
## 155           1            1
## 156           1            1
```

```
# Example outcome variable, nodal efficiency for one vertex
y <- sapply(g.glm[], function(x) V(x)$E.nodal.wt[1])
Xy <- cbind(X, y)

# Simple linear regression
summary(lm(y ~ Group, data=Xy))$coefficients

##                Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   0.045358  0.0009747   46.53 1.338e-92
## GroupPatient -0.003256  0.0011504   -2.83 5.274e-03

# beta_0 equals the mean of Group 1, and beta_1 equals the difference in Group
# means (Group 2 - Group 1)
Xy[, mean(y), by=Group]

##      Group      V1
## 1: Control 0.04536
## 2: Patient 0.04210

Xy[, mean(y), by=Group][, diff(V1)]

## [1] -0.003256
```

8.3.3 Cell means coding

The values in the second row of the summary in the previous code block would be equivalent to setting a contrast of `c(-1, 1)` in a 2^{nd}-level SPM analysis if the coding scheme were *cell means* coding. Here, the parameters represent the factor level means:

$$\hat{\beta}_1 = \mu_1$$
$$\hat{\beta}_2 = \mu_2$$
$$\vdots$$
$$\hat{\beta}_k = \mu_k$$

If you prefer this kind of parameterization, you can exclude the intercept with the following code. This is the same matrix that results from setting `coding='cell.means'` in a call to `brainGraph_GLM`.

```
# You can manually exclude the intercept to get group means:
model.matrix(~ Group + 0, data=X)[c(1:4, 153:156), ]

##     GroupControl GroupPatient
## 1              1            0
## 2              1            0
## 3              1            0
## 4              1            0
## 153            0            1
## 154            0            1
## 155            0            1
## 156            0            1

summary(lm(y ~ Group + 0, data=Xy))$coefficients

##              Estimate Std. Error t value   Pr(>|t|)
## GroupControl  0.04536  0.0009747   46.53  1.338e-92
## GroupPatient  0.04210  0.0006109   68.91 1.221e-117
```

8.3.4 Effects coding

As detailed in the FSL GLM page, *effects* coding is a better choice for more complicated designs. The coding is almost the same as *dummy* coding except that the reference group is indicated by -1 in the design matrix instead of 0. You can use this type of coding by setting `coding='effects'` in a call to `brainGraph_GLM`. There is more on effects coding on this UCLA page.

The parameter estimate for the intercept represents the *mean of factor level means* ($\mu_\cdot$ in the equations below). The remaining parameter estimates represent the difference between the overall mean and the factor level mean.

Note

This value will not equal the mean of all observations if there are an unequal number of observations in each level.

$$\begin{aligned}\hat{\beta}_0 &= \mu_. \\ \hat{\beta}_1 &= \mu_2 - \mu_. \\ &\vdots \\ \hat{\beta}_{k-1} &= \mu_k - \mu_.\end{aligned}$$

If you are interested in comparing group means under this coding scheme, first assume that there are $k = 3$ factor levels. Then, since group 1 (the reference group) is coded `+1` for the intercept and `-1` for the other parameters, and group 2 is coded `+1` for the intercept and the second parameter, and `0` for the third:

$$\begin{aligned}H_A : \mu_1 - \mu_2 > 0 \\ \Rightarrow &(\hat{\beta}_0 - \hat{\beta}_1 - \hat{\beta}_2) - (\hat{\beta}_0 + \hat{\beta}_1) > 0 \\ \Rightarrow &-2\hat{\beta}_1 - \hat{\beta}_2 > 0 \\ \Rightarrow &C = (0, -2, -1)\end{aligned}$$

The contrast for a difference in groups i and j (where $i \neq j \neq 1$), i.e., $H_0 : \mu_i - \mu_j \neq 0$, is the same as in the dummy coding example.

```
Xdes <- brainGraph_GLM_design(X, coding='effects')
Xdes[c(1:3, 102:105), ]

##          Intercept GroupPatient
## 02-115-0         1           -1
## 02-126-1         1           -1
## 02-127-2         1           -1
## 02-294-3         1            1
## 02-295-3         1            1
## 02-311-4         1            1
## 02-312-4         1            1

# Contrasts
Cmat <- matrix(c(1, -1, 1, 1, 0, -2, 0, 2), nrow=4, ncol=2, byrow=TRUE)
rownames(Cmat) <- c(paste('Mean', grps),
                    'Control > Patient', 'Patient > Control')
Cmat

##                     [,1] [,2]
## Mean Control           1   -1
## Mean Patient           1    1
## Control > Patient      0   -2
## Patient > Control      0    2

# Fit the LM and get statistics for all contrasts
fits <- fastLmBG(Xdes, as.matrix(y))
rbindlist(apply(fastLmBG_t(fits, Cmat), 3, as.data.table), idcol='Contrast')

##             Contrast     gamma        se  stat          p      p.fdr
## 1:      Mean Control  0.045358 0.0009747 46.53  1.338e-92  1.338e-92
## 2:      Mean Patient  0.042102 0.0006109 68.91 1.221e-117 1.221e-117
## 3: Control > Patient  0.003256 0.0011504  2.83  5.274e-03  5.274e-03
## 4: Patient > Control -0.003256 0.0011504 -2.83  5.274e-03  5.274e-03
```

8.4 Examples

This section contains examples of common analysis scenarios. This is the first demonstration of the `summary` method for `brainGraph_GLM` results. See Box 8.1.

In all examples, the `brainGraphList` object `g.glm` contains all the graphs for both groups at a single threshold. The covariates table used in each example (varying the inclusion of certain columns) is:

```
env.glm$covars.dti

##      Study.ID   Group Age.MRI Sex Scanner
##   1: 02-001-4 Patient   16.19   M      3T
##   2: 02-003-2 Patient   16.57   M      3T
##   3: 02-006-8 Patient   16.59   M      3T
##   4: 02-008-7 Patient   18.04   F    1.5T
##   5: 02-013-1 Patient   16.80   M    1.5T
##  ---
## 152: 02-639-2 Control   11.47   F      3T
## 153: 02-643-9 Control   11.13   M    1.5T
## 154: 02-644-0 Control   13.15   M    1.5T
## 155: 02-646-5 Control   11.54   F    1.5T
## 156: 02-652-9 Control   10.71   M    1.5T
```

BOX 8.1 The `summary` method for results

In the *Examples* section, you see a printed summary of the results from `brainGraph_GLM`. First, the output in this document is not how it will appear in your terminal. I use the excellent colorout package for "colorizing" different aspects of the text.

In my terminal, the lines you see with two leading pound signs and black text are displayed in blue; these are printed using R's `message` function. Next are the input parameters supplied to the function. Finally, the statistics are displayed; if there are multiple contrasts, there will be a separate section for each. You may also choose to show results for a single contrast using the `contrast` argument. The `summary` function displays only significant results (based on the supplied α level). You may choose to use the actual P-value or the FDR-adjusted P-value for significance (via the function argument `p.sig`). The argument `digits` adjusts the number of significant digits displayed. Finally, the `print.head` argument controls how many rows to display; by default, at most 10 rows will be printed.

8.4.1 Two-group difference

The following code block tests for between-group difference in weighted nodal efficiency. The `summary` method will be shown for a later example.

```
# Simple between-group comparison, Group 1 > Group 2
con.mat <- matrix(c(0, -2), nrow=1, dimnames=list('Control > Patient'))
summary(with(env.glm,
     brainGraph_GLM(g.glm, measure='E.nodal.wt', covars=covars.dti[, 1:2],
                    coding='effects', contrasts=con.mat, alt='greater')))
```

8.4.2 Two-group difference adjusted for covariate

In the following code block, I adjust for *age at MRI*. I also mean-center the covariate (but it does not change the statistics). It must be noted that the function adds the factor variables *after* any covariates, so you will need to adjust your contrast vectors accordingly.

```
# Between-group comparison adjusted for age, Group 1 > Group 2
con.mat <- matrix(c(0, 0, -2), nrow=1, dimnames=list('Control > Patient'))
summary(with(env.glm,
     brainGraph_GLM(g.glm, measure='E.nodal.wt', covars=covars.dti[, 1:3],
                    coding='effects', mean.center=TRUE, contrasts=con.mat,
                    alt='greater')))
```

8.4.3 Two-group difference with continuous covariate interaction

This is essentially the same as the previous model, except now the covariate `Age.MRI` will be mean-centered and then split into two columns in the design matrix (one for each subject group). This is achieved by including `int=c('Group', 'Age.MRI')` in your function call. I only show the results for the `cell.means` approach.

```
# A few rows of the design matrix
X <- brainGraph_GLM_design(env.glm$covars.dti[, 1:3], coding='cell.means',
                           mean.center=TRUE, int=c('Group', 'Age.MRI'))
X[c(1:4, 153:156), ]

##          GroupControl GroupPatient GroupControl:Age.MRI
## 02-001-4            0            1                0.000
## 02-003-2            0            1                0.000
## 02-006-8            0            1                0.000
## 02-008-7            0            1                0.000
## 02-643-9            1            0               -3.827
## 02-644-0            1            0               -1.812
## 02-646-5            1            0               -3.419
## 02-652-9            1            0               -4.249
##          GroupPatient:Age.MRI
## 02-001-4                1.236
## 02-003-2                1.613
## 02-006-8                1.635
## 02-008-7                3.078
## 02-643-9                0.000
## 02-644-0                0.000
## 02-646-5                0.000
## 02-652-9                0.000
```

```
# Between-group comparison with Age interaction, Group 1 > Group 2
con.mat <- matrix(c(0, 0, 1, -1), nrow=1, dimnames=list('Group X Age'))
summary(with(env.glm,
     brainGraph_GLM(g.glm, measure='E.nodal.wt', covars=covars.dti[, 1:3],
                    X=X, contrasts=con.mat, alt='greater')))
```

8.4.4 Two-way between subjects ANOVA: 2x2

Here is an example of a 2x2 ANOVA. The two factors are `Group` and `Sex`, and the levels for each are `Control, Patient` and `Male, Female`. The first example shows *effects* coding, and the second is *cell-means* coding. Note in the second code block that with *cell-means* coding, the order of the columns is `A1B1, A2B1, A1B2, A2B2`. I divide the contrast vector by 4 to match the results of *effects* coding.[1]

```
# A few rows of the design matrix
X <- brainGraph_GLM_design(env.glm$covars.dti[, c(1:2, 4)], coding='effects',
                           int=c('Group', 'Sex'))
X[c(1:4, 153:156), ]

##          Intercept GroupPatient SexM GroupPatient:SexM
## 02-001-4         1            1    1                 1
## 02-003-2         1            1    1                 1
## 02-006-8         1            1    1                 1
## 02-008-7         1            1   -1                -1
## 02-643-9         1           -1    1                -1
## 02-644-0         1           -1    1                -1
## 02-646-5         1           -1   -1                 1
## 02-652-9         1           -1    1                -1

con.mat <- matrix(c(0, 0, 0, 1), nrow=1, dimnames=list('Group x Sex interaction'))
summary(with(env.glm,
     brainGraph_GLM(g.glm, measure='E.nodal.wt', covars=covars.dti[, c(1:2, 4)],
                    X=X, contrasts=con.mat, alt='greater')))

##
## ====================================
## brainGraph GLM results (vertex-level)
## ====================================

## GLM formula:
##   E.nodal.wt ~ Group * Sex
## based on 156 observations, with 152 degrees of freedom.
##
## Contrast type:  T contrast
## Alternative hypothesis:  C > 0
## Contrasts:
##                          Intercept GroupPatient SexM GroupPatient:SexM
## Group x Sex interaction .         .            .    1

##
## Statistics
## ------------------
## Group x Sex interaction:

##     Region Estimate 95% CI low 95% CI high Std. error t value Pr(>|t|)
##  1:   lENT 0.001186   1.12e-04     0.00226   0.000543    2.18  0.01533
##  2:   lFUS 0.000767  -9.32e-05     0.00163   0.000435    1.76  0.04007
##  3:  lpORB 0.000865  -8.88e-05     0.00182   0.000483    1.79  0.03758
##  4:    lTT 0.000963   2.67e-04     0.00166   0.000353    2.73  0.00351
##  5:  lTHAL 0.001186   1.85e-04     0.00219   0.000506    2.34  0.01024
## ---
##  9:   rENT 0.000962  -6.32e-05     0.00199   0.000519    1.85  0.03284
```

[1] The t-statistics and p-values would be unchanged if you did not divide by 4.

```
## 10:  rPARH 0.001011   1.70e-05     0.00201   0.000503    2.01  0.02313
## 11:  rTHAL 0.000887  -1.52e-04     0.00193   0.000526    1.69  0.04682
## 12:  rHIPP 0.001186  -2.63e-05     0.00240   0.000614    1.93  0.02756
## 13:  rAMYG 0.001195   2.72e-04     0.00212   0.000467    2.56  0.00577
##     Pr(>|t|) (FDR)
##  1:          0.254
##  2:          0.254
##  3:          0.254
##  4:          0.219
##  5:          0.254
## ---
##  9:          0.254
## 10:          0.254
## 11:          0.259
## 12:          0.254
## 13:          0.219
```

For *cell-means* coding, I show part of the design matrix, but omit the `summary`, as it is identical to that for *effects* coding. Note that the contrast matrix is divided by 4 to give equivalent results.

```
X <- brainGraph_GLM_design(env.glm$covars.dti[, c(1:2, 4)],
                           coding='cell.means', int=c('Group', 'Sex'))
X[1:4, ]

##          GroupControl:SexF GroupControl:SexM GroupPatient:SexF
## 02-001-4                 0                 0                 0
## 02-003-2                 0                 0                 0
## 02-006-8                 0                 0                 0
## 02-008-7                 0                 0                 1
##          GroupPatient:SexM
## 02-001-4                 1
## 02-003-2                 1
## 02-006-8                 1
## 02-008-7                 0

con.mat <- matrix(c(1, -1, -1, 1) / 4, nrow=1, dimnames=list('Group X Sex'))
```

8.4.5 Two-way between subjects ANOVA: 2x3

Here is an example of a 2x3 ANOVA. The first example shows *effects* coding, and the second shows *cell-means* coding. I show all of the standard ANOVA contrasts.

```
# A few rows of the design matrix; get rid of 'NA' values first
incomp <- covars2x3[!complete.cases(covars2x3), Study.ID]
X <- brainGraph_GLM_design(covars2x3[!Study.ID %in% incomp],
                           coding='effects', int=c('A', 'B'))
head(X)

##          Intercept A2 B2 B3 A2:B2 A2:B3
## 02-001-4         1 -1 -1 -1     1     1
## 02-003-2         1  1  0  1     0     1
## 02-008-7         1 -1 -1 -1     1     1
## 02-013-1         1  1 -1 -1    -1    -1
```

```
## 02-014-6         1  1  0  1     0     1
## 02-016-1         1 -1 -1 -1     1     1

con.mat <- cbind(0, diag(5))
rownames(con.mat) <- c('Main A', 'Main B (1st)', 'Main B (2nd)',
                       'A X B (1st)', 'A X B (2nd)')
anova2x3 <- brainGraph_GLM(g.glm[45:156], measure='E.nodal.wt', covars=covars2x3,
                           X=X, contrasts=con.mat, alt='greater')
summary(anova2x3)

##
## ====================================
## brainGraph GLM results (vertex-level)
## ====================================

## GLM formula:
##   E.nodal.wt ~ A * B
## based on 105 observations, with 99 degrees of freedom.
##
## Contrast type:  T contrast
## Alternative hypothesis:  C > 0
## Contrasts:
##              Intercept A2 B2 B3 A2:B2 A2:B3
## Main A       .         1  .  .  .     .
## Main B (1st) .         .  1  .  .     .
## Main B (2nd) .         .  .  1  .     .
## A X B (1st)  .         .  .  .  1     .
## A X B (2nd)  .         .  .  .  .     1

## 7 subjects removed due to incomplete data:

##    02-006-8, 02-123-7, 02-161-8, 02-170-3, 02-203-8, 02-287-7, 02-521-5

##
## Statistics
## ------------------
## Main A:
## No signficant results!
## Main B (1st):
## No signficant results!
## Main B (2nd):

##    Region Estimate 95% CI low 95% CI high Std. error t value Pr(>|t|)
## 1:   lFUS  0.00123  -1.70e-04     0.00262   0.000703    1.74   0.0423
## 2:  lPARH  0.00173  -3.60e-05     0.00350   0.000892    1.94   0.0274
## 3:  lpORB  0.00139  -1.62e-04     0.00294   0.000781    1.78   0.0394
## 4:  lrMFG  0.00140  -1.80e-04     0.00297   0.000794    1.76   0.0410
## 5:   lPUT  0.00216   2.06e-05     0.00430   0.001079    2.00   0.0239
## 6:  lPALL  0.00181  -5.69e-05     0.00367   0.000940    1.92   0.0286
## 7:  lHIPP  0.00193  -1.34e-04     0.00399   0.001040    1.86   0.0332
## 8: rparaC  0.00178   8.46e-05     0.00347   0.000854    2.08   0.0199
## 9:  rPALL  0.00171  -1.97e-04     0.00362   0.000963    1.78   0.0391
##    Pr(>|t|) (FDR)
## 1:          0.357
## 2:          0.357
## 3:          0.357
```

```
## 4:          0.357
## 5:          0.357
## 6:          0.357
## 7:          0.357
## 8:          0.357
## 9:          0.357

## A X B (1st):
## No signficant results!
## A X B (2nd):

##     Region Estimate 95% CI low 95% CI high Std. error t value Pr(>|t|)
##  1:  lcMFG  0.00167   2.92e-04     0.00306   0.000697    2.40  0.00907
##  2:   lCUN  0.00211   4.48e-04     0.00377   0.000837    2.52  0.00667
##  3:   lENT  0.00188   8.90e-06     0.00376   0.000945    1.99  0.02447
##  4:   liCC  0.00228   3.82e-04     0.00419   0.000959    2.38  0.00956
##  5:   lLOF  0.00263   5.09e-04     0.00475   0.001068    2.46  0.00780
## ---
## 22:  rpORB  0.00111  -1.84e-04     0.00240   0.000650    1.70  0.04601
## 23:  rpTRI  0.00151   1.37e-04     0.00289   0.000695    2.18  0.01578
## 24:   rPCC  0.00156  -1.24e-04     0.00324   0.000847    1.84  0.03456
## 25:  rrACC  0.00165  -9.17e-05     0.00339   0.000878    1.88  0.03154
## 26:   rSFG  0.00131  -1.53e-04     0.00278   0.000739    1.78  0.03930
##     Pr(>|t|) (FDR)
##  1:         0.0733
##  2:         0.0733
##  3:         0.1123
##  4:         0.0733
##  5:         0.0733
## ---
## 22:         0.1399
## 23:         0.0857
## 24:         0.1194
## 25:         0.1194
## 26:         0.1245
```

For *cell-means* coding, I only show part of the design matrix. Note the order of the columns: all of the `A1` factors come first, and then the `A2`'s. I also show how to construct the contrast matrix. Notice that I divide the matrix by 6; this will give equivalent results to the *effects* coding run.

```
X <- brainGraph_GLM_design(covars2x3[!Study.ID %in% incomp],
                           coding='cell.means', int=c('A', 'B'))
head(X)

##          A1:B1 A1:B2 A1:B3 A2:B1 A2:B2 A2:B3
## 02-001-4     1     0     0     0     0     0
## 02-003-2     0     0     0     0     0     1
## 02-008-7     1     0     0     0     0     0
## 02-013-1     0     0     0     1     0     0
## 02-014-6     0     0     0     0     0     1
## 02-016-1     1     0     0     0     0     0

con.mat <- matrix(c(1, 1, 1, -1, -1, -1,
                    -1, 1, 0, -1, 1, 0,
                    -1, 0, 1, -1, 0, 1,
```

```
                    1, 0, -1, -1, 0, 1,
                    0, 1, -1, 0, -1, 1), nrow=5, byrow=TRUE)
con.mat <- con.mat / 6
rownames(con.mat) <- c('Main A', 'B2-B1', 'B3-B1',
                       'A1B1 - A2B1 - A1B3 + A2B3', 'A1B2 - A2B2 - A1B3 + A2B3')
```

```
summary(brainGraph_GLM(g.glm[45:156], measure='E.nodal.wt', covars=covars2x3,
                       X=X, contrasts=con.mat, alt='greater'))
```

8.4.6 Three-way between subjects ANOVA: 2x2x2

If the `int` vector has three elements, all two-way interactions will be automatically included. In this example, the three factors and their levels are:

- Group (*Patient* and *Control*)
- Sex (*M* and *F*)
- Scanner (*1.5T* and *3T*)

```
X <- brainGraph_GLM_design(env.glm$covars.dti[, !'Age.MRI'],
                           coding='effects',
                           int=c('Group', 'Sex', 'Scanner'))
head(X)

##          Intercept GroupPatient SexM Scanner3T GroupPatient:SexM
## 02-001-4         1            1    1         1                 1
## 02-003-2         1            1    1         1                 1
## 02-006-8         1            1    1         1                 1
## 02-008-7         1            1   -1        -1                -1
## 02-013-1         1            1    1        -1                 1
## 02-014-6         1            1    1        -1                 1
##          GroupPatient:Scanner3T SexM:Scanner3T GroupPatient:SexM:Scanner3T
## 02-001-4                      1              1                            1
## 02-003-2                      1              1                            1
## 02-006-8                      1              1                            1
## 02-008-7                     -1              1                            1
## 02-013-1                     -1             -1                           -1
## 02-014-6                     -1             -1                           -1
```

8.5 Permutation testing

If you also would like to get permutation-based p-values, then you can specify `permute=TRUE` to perform the permutations, calculate the maximum test statistic across vertices (or a single statistic if `level='graph'`), and compare this null distribution of maximum statistics to the *observed* statistics. This is essentially the same as the procedure from `FSL`'s *randomise* and the `PALM` utility (albeit without as much of the functionality).(40, 80, 129) The default matrix partitioning scheme is (like that of `PALM`) called `'beckmann'`.(108)

New in v3.0.0

You can choose from 3 permutation methods: Freedman-Lane (the default), Smith, and Ter Braak. See Winkler et al. (129) for details/discussion of these methods. You may also now explicitly choose the matrix partitioning scheme; either Beckmann (the default), Guttman, or Ridgway.

When `long=TRUE`, an additional element called `perm` is returned in the `bg_GLM` object. See the section *Return object* for details.

8.5.1 Example

```
X <- brainGraph_GLM_design(env.glm$covars.dti, coding='effects',
                           binarize=c('Sex', 'Scanner'))
con.mat <- matrix(c(rep(0, 4), -2), nrow=1, dimnames=list('Control > Patient'))
diffs.perm <- brainGraph_GLM(g.glm, env.glm$covars.dti, 'E.nodal.wt',
                             contrasts=con.mat, X=X, alt='greater',
                             permute=TRUE, N=5000, long=TRUE)
```

Here, I show the structure of the `perm` element.

```
str(diffs.perm$perm)

## List of 3
##  $ thresh   : Named num 3.01
##   ..- attr(*, "names")= chr "Control > Patient"
##  $ null.dist: num [1:76, 1:5000, 1] -0.341 -0.67 -1.23 -2.408 -2.142 ...
##  $ null.max : num [1:5000, 1] 0.416 1.233 0.715 1.986 1.253 ...
```

8.6 Plotting LM diagnostics

The `plot` method for `bg_GLM` objects has the same functionality as `plot.lm` in `base R`: it shows linear model "diagnostics". There are 6 possible plots (for a given region), chosen by the argument `which`:

1. Residuals vs. fitted values
2. QQ plot
3. "Scale-location" plot (standardized residuals vs. fitted values)
4. Cook's distance
5. Residuals vs. leverage
6. Cook's distance vs. leverage

The plots shown in Figure 8.1 were generated by the function call in the following code block. The plot title lists the *outcome variable* (the graph metric of interest) and the *region*; there is a plot sub-title (at bottom) which shows the linear model formula used in `R` (in what is sometimes called "Wilkinson notation").

```
grid.arrange(plot(diffs.perm, region='lHIPP', which=1:6)[[1]])
```

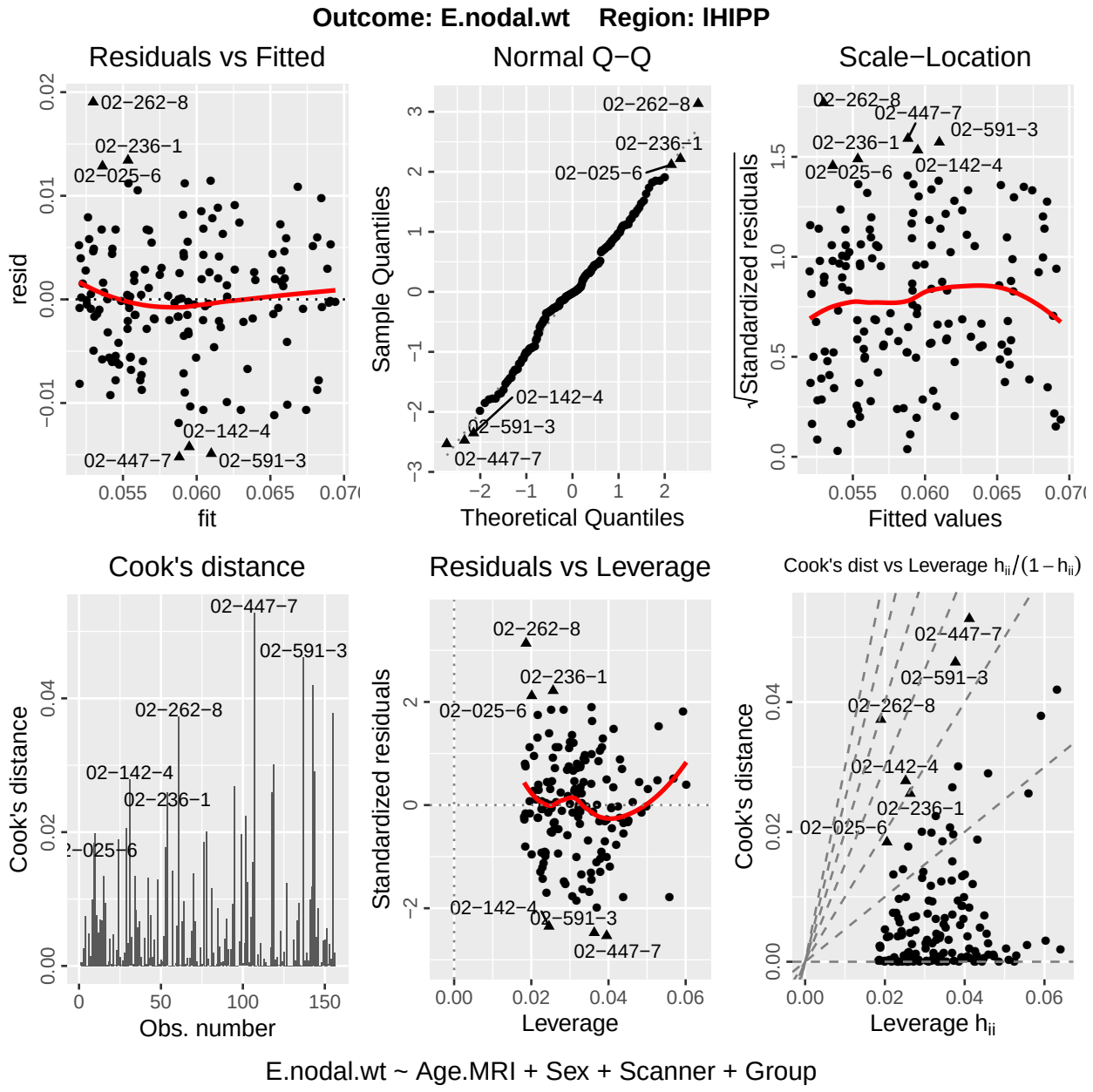

Figure 8.1: **GLM diagnostics.**

If you would like plots for multiple regions, you can supply a character vector, or leave the default (`region=NULL`). The following code block shows how to get the plots for all regions and save each to a page in a PDF. On my system, with 76 regions, it takes 44 seconds for the function call itself, and 38 seconds to save the file to disk. The `glmPlots` object is 60 MB (in memory), and the PDF is 1.1 MB.

```
glmPlots <- plot(diffs.perm, which=1:6)
ml <- marrangeGrob(glmPlots, nrow=1, ncol=1)
ggsave('glmPlots.pdf', ml, width=8.5, height=11)
```

Level	Name	Description
Graph	name	The contrast name
	outcome	The network metric tested
	alpha	The significance level
Vertex	p	1 minus the P-value
	p.fdr	1 minus the FDR-adjusted P-value
	gamma	The contrast of parameter estimates
	se	The *standard error* of the contrast
	size2	The test statistic
	size	The test statistic transformed to a range of 0–20 (for visualization purposes)
	p.perm	1 minus the permutation P-value

Table 8.1: **GLM graph attributes.** The graph- and vertex-level attributes that are added after calling `make_brainGraphList`.

8.7 Create a graph of the results

To create a graph with attributes specific to GLM, use `make_brainGraphList`. The number of graphs equals the number of contrasts, and they will have several additional attributes; see Table 8.1.

```
(g.anova2x3 <- make_brainGraphList(anova2x3, atlas='dkt.scgm'))

##
## ========================================================================
## A "brainGraphList" object of *observed* graphs containing 5 contrasts.
## ========================================================================

##
##  Software versions
##         R release: R version 3.6.0 (2019-04-26)
##        brainGraph: 3.0.0
##            igraph: 1.2.4.1
##  Date created:     2020-09-30 12:32:11
##  Brain atlas used: Desikan-Killiany-Tourville + SCGM
##  Imaging modality: N/A
##  Edge weighting:   Unweighted
##  Threshold:        N/A

## Contrast names:

## [1] "Main A"       "Main B (1st)" "Main B (2nd)" "A X B (1st)"
## [5] "A X B (2nd)"
```

8.8 Plotting a graph of the results

If you would like to plot only significant regions, you can simply call the `plot` method on the graph. The `p.sig` function argument lets you choose which P-value determines significance (the standard P-value, FDR-adjusted, or permutation-based). Below I use the standard P-value, which is the default behavior.

```
# Plot results for the fifth contrast
plot(g.anova2x3[5], vertex.color='color.lobe', vertex.size=10)
```

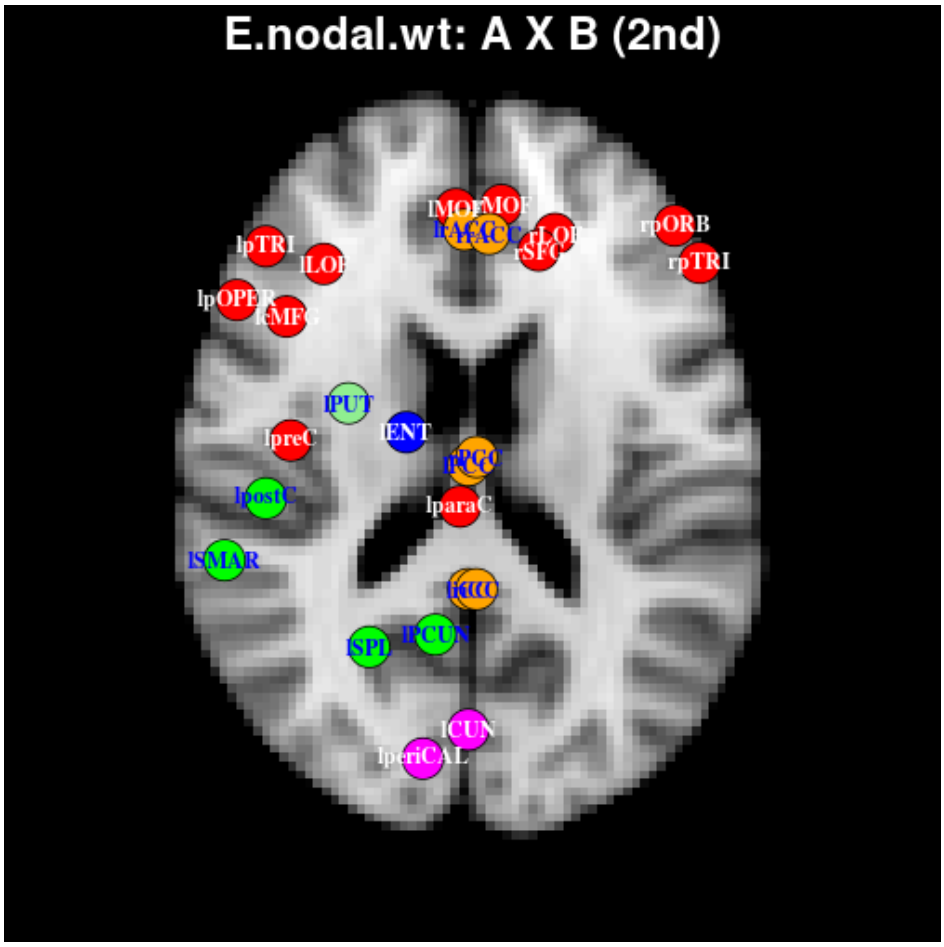

Figure 8.2: **GLM results.**

9

Multi-threshold permutation correction

The *multi-threshold permutation correction (MTPC)* method for statistical inference was introduced by Drakesmith et al.(30) Its goal is to reduce the effects of false positives and the use of multiple thresholds in network analyses. This is because network metrics are not stable across thresholds (sometimes even adjacent thresholds), and there is no real standard for choosing a threshold to examine *a priori*. The `brainGraph` function for performing MTPC is `mtpc`.

9.1 Background

As I have described in previous chapters, it is common to generate sets of *thresholded networks* for a given subject or group; this is necessary because most "raw" brain networks have an unrealistically-high *density* (i.e., the proportion of present to possible connections). In structural covariance and resting-state fMRI networks, it is common to threshold the covariance matrix with a range of correlation values; in DTI tractography, the threshold can be a range of *streamline counts*, connectivity probabilities, or average DTI metric for each tract (e.g., FA). However, there is no *principled* way to choose the "correct" threshold to use for your final results.

9.1.1 MTPC procedure

The steps that are applied in MTPC are:

1. Apply a set of thresholds τ to the networks, and compute network metrics and for all subjects/groups and thresholds (we have already done this in Graph creation and Graph creation).
2. Compute test statistics S_{obs} for all networks and thresholds. This is done from within `mtpc` by calling `brainGraph_GLM` (see Vertex-wise group analysis (GLM)).
3. Permute group assignments and compute test statistics for each permutation and threshold (also done by calling `brainGraph_GLM`).
4. Build the null distribution of the *maximum statistic across thresholds* (and across brain regions, if applicable) *for each permutation*.

5. Determine the *critical value* S_{crit} from the null distribution of maximum test statistics (by taking the top αth percentile).

6. Identify *cluster(s)* where $S_{obs} > S_{crit}$ and compute the *area under the curve (AUC)* for the cluster(s); the AUC(s) are denoted A_{MTPC}. Note that these clusters are *across thresholds*; in `brainGraph`, I consider 3 consecutive significant test statistics as a cluster.

7. Compute a *critical AUC* A_{crit} from the *mean* of the supra-critical AUC's for the permuted tests.

8. Reject the null hypothesis H_0 if the AUC of the significant (observed) clusters is greater than the critical AUC; i.e., if $A_{MTPC} > A_{crit}$.

Note that the procedure is nearly identical for graph- and vertex-level metrics; with vertex-level metrics, step 4 (building the null distribution of maximum test statistics) is the only difference in that the maximum is taken across thresholds *and* brain regions.

9.2 Function arguments

Many arguments are the same as those of `brainGraph_GLM`; any listed below without accompanying information are described in Vertex-wise group analysis (GLM).

```
args(mtpc)

## function (g.list, thresholds, covars, measure, contrasts, con.type = c("t",
##     "f"), outcome = NULL, con.name = NULL, level = c("vertex",
##     "graph"), clust.size = 3L, perm.method = c("freedmanLane",
##     "terBraak", "smith", "draperStoneman", "manly", "stillWhite"),
##     part.method = c("beckmann", "guttman", "ridgway"), N = 500L,
##     perms = NULL, alpha = 0.05, res.glm = NULL, long = TRUE,
##     ...)
## NULL
```

9.2.1 Mandatory

g.list A list of `brainGraphList` objects.

thresholds A numeric vector of the thresholds that were used to create the networks. This can be either the actual threshold values or just an integer vector from 1 to the number of thresholds.

covars

measure

contrasts

9.2.2 Optional

con.type

outcome

con.name

level

clust.size The "cluster size" (i.e., the number of consecutive thresholds for which the observed statistic exceeds the null). Default: `3`

perm.method

part.method

N Default: `500`

perms

alpha

res.glm A list of `bg_GLM` objects, as output by a previous run of `mtpc`. This is useful if you want to assess differences when varying `clust.size` , as it will avoid calculating all of the null statistics over again. Default: `NULL`

long

9.2.3 Unnamed

The unnamed arguments are the same as those in `brainGraph_GLM`; please see the relevant section in Vertex-wise group analysis (GLM) for information.

9.3 Return value

The function returns an object of class `mtpc`. Many of the returned elements are the same as those returned by `brainGraph_GLM`; if they have no accompanying information here, see Vertex-wise group analysis (GLM). The elements are: (τ refers to thresholds, and N_τ is the number of thresholds)

level

X Only present if `outcome=NULL`

outcome

measure

con.type

contrasts

con.name

alt

alpha

removed.subs

atlas

rank

df.residual

qr Only present if `outcome=NULL`

cov.unscaled Only present if `outcome=NULL`

N

res.glm A *list* (with length equal to N_τ) in which each element is the output from `brainGraph_GLM` for a single threshold.

clust.size

DT A `data.table`, which is combined, for all thresholds, from each run of `brainGraph_GLM`.

stats A `data.table` with the statistics calculated by MTPC: τ_{mtpc}, S_{mtpc}, S_{crit}, and A_{crit}.

null.dist A *numeric array* of the maximum statistic for each permutation and threshold. It will have $N_\tau \times N_{rand} \times N_C$ dimensions, where N_{rand} is the number of permutations and N_C is the number of contrasts.

perm.order

9.4 Code example

The following code blocks show how to run `mtpc` for a few different network measures, storing all results in a single list. As with some other functions with long runtimes and/or high processing demands, it would be quite a bit faster to run each individually from a fresh `R` session, but you can let this run without having to repeatedly execute the code while only changing the `measure` argument. There are some benchmarks in Benchmarks.

I have been storing all parameters in a single `data.table`, which makes it easier to loop through without much effort. In the example below, I look at 4 network metrics: 2 graph-level and 2 vertex-level. I show how to change the alternative hypothesis for given metrics. This approach is, of course, optional.

```
mtpcVars <- data.table(level=rep(c('graph', 'vertex'), each=2),
                       outcome=c('E.global.wt', 'Lp', 'E.nodal.wt', 'strength'),
                       alt='greater')

# Change H_A for 'Lp'
mtpcVars[outcome == 'Lp', alt := 'less']
setkey(mtpcVars, level, outcome)

# Different number of permutations based on the level
mtpcVars['graph', N := 1e4]
mtpcVars['vertex', N := 5e3]

# Generate permutation matrices using 'shuffleSet' from the 'permute' package
mtpcPerms <- list(vertex=shuffleSet(n=nrow(covars), nset=mtpcVars['vertex', unique(N)]),
                  graph=shuffleSet(n=nrow(covars), nset=mtpcVars['graph', unique(N)]))

# Create the contrast matrix
mtpcContrast <- matrix(c(0, 0, 0, 0, -2, 0,
                         0, 0, 0, 0, 0, 1),
                       nrow=2, byrow=TRUE,
                       dimnames=list(c('Control vs. Patient', 'Age-squared effect')))

mtpcVars

##     level     outcome     alt     N
## 1:  graph E.global.wt greater 10000
```

```
## 2:  graph          Lp    less 10000
## 3: vertex  E.nodal.wt greater  5000
## 4: vertex    strength greater  5000
```

```
# Loop through the network metrics
# The 1st-level of the list is for each 'level' (i.e., 'graph' and 'vertex')
mtpc.diffs.list <- sapply(mtpcVars[, unique(level)], function(x) NULL)
for (x in names(mtpc.diffs.list)) {  # Loop across 'level'

  # The 2nd-level is for each network metric (for the given level 'x')
  mtpc.diffs.list[[x]] <- sapply(mtpcVars[x, unique(outcome)], function(x) NULL)
  for (y in mtpcVars[x, outcome]) {
    # Print some timing info in the terminal; optional
    print(paste('Level:', x, '; Outcome:', y, ';', format(Sys.time(), '%H:%M:%S')))

    mtpc.diffs.list[[x]][[y]] <-
      mtpc(g, thresholds, covars=covars.dti, measure=y, contrasts=mtpcContrast,
           con.type='t', level=x, N=mtpcVars[.(x, y), N], perms=mtpcPerms[[x]],
           alt=mtpcVars[.(x, y), alt])
  }
}

# Create a single data.table with just the significant regions
mtpc.diffs.sig.dt <-
  rbindlist(lapply(mtpc.diffs.list, function(x)
                   rbindlist(lapply(x, function(y)
                                    y$DT[A.mtpc > A.crit, .SD[1], by=region]))))

# Save the variables created in this script
save(list=ls()[grep('.*mtpc.*', ls())],
     file=paste0(savedir, , grps[2], '_', atlas, '_mtpc.rda'))
```

Here I show what the `summary` method produces. This analysis consisted of 3 groups and 3 t-contrasts. You have the option to choose a single `contrast` (the default is to print results for all contrasts) and to print longer summary tables (the default is to print only the first and last 5 rows).

```
summary(res.mtpc)

##
## ====================================
## MTPC results (vertex-level)
## ====================================

## GLM formula:
##   E.nodal.wt ~
## based on 103 observations, with 97 degrees of freedom.
##
## Contrast type:  T contrast
## Alternative hypothesis:  C != 0
## Contrasts:
##               Intercept age_mri_6w gender ScannerChange GroupPt1 GroupPt2
## Control > Pt1  .         .          .      .            -2       -1
## Control > Pt2  .         .          .      .            -1       -2
```

```
## Pt1 > Pt2      .          .          .      .             1       -1
##
## # of thresholds:  30
## Cluster size (across thresholds):  3

##
## Permutation analysis
## --------------------

## Permutation method:  Freedman-Lane
## Partition method:  Beckmann
## # of permutations:  5,000

##
## Statistics
## --------------------

## tau.mtpc: threshold of the maximum observed statistic
## S.mtpc: maximum observed statistic
## S.crit: the critical (95th percentile) value of the null max. statistic
## A.crit: critical AUC from the supra-critical null AUCs
##
##    contrast tau.mtpc S.mtpc S.crit A.crit      Contrast
## 1:        1     5555  99.86  3.255  13757 Control > Pt1
## 2:        2     5555 119.20  3.186  14520 Control > Pt2
## 3:        3     4920  84.92  3.260  13473     Pt1 > Pt2

## Control > Pt1:

##       Region tau.mtpc S.mtpc S.crit A.mtpc A.crit
##  1:     rLOG    10000  82.65   3.26 151159  13757
##  2:     rLOF     8730  28.17   3.26  39195  13757
##  3:     lLOF    10000  27.72   3.26  19392  13757
##  4:     rFUS     8730  24.85   3.26  32871  13757
##  5:    lCAUD     7460  21.33   3.26  40502  13757
## ---
## 14:     riCC     2622  11.01   3.26  44319  13757
## 15:    lpORB     2156   8.10   3.26  19918  13757
## 16:   lparaC     7460   7.56   3.26  14977  13757
## 17:     rSPL     2778   6.74   3.26  16969  13757
## 18: lperiCAL    10000  -9.72   3.26  21808  13757

## Control > Pt2:

##     Region tau.mtpc S.mtpc S.crit A.mtpc A.crit
##  1:  rHIPP     5555 119.20   3.19  18235  14520
##  2:   rLOG    10000  98.28   3.19 130618  14520
##  3:   lLOG    10000  72.54   3.19  54591  14520
##  4:   lLOF    10000  43.21   3.19  77614  14520
##  5:  lHIPP     2778  24.36   3.19  15823  14520
## ---
## 44:  lPCUN     2000   8.12   3.19  27565  14520
## 45:  rPCUN     2467   7.86   3.19  15457  14520
## 46: lparaC     7460   7.80   3.19  33308  14520
## 47:  lrMFG     1533   6.78   3.19  18280  14520
## 48:   rSFG     8095   6.56   3.19  20308  14520
```

```
## EI > TBI:

##     Region tau.mtpc S.mtpc S.crit A.mtpc A.crit
##  1:  lAMYG     5555  22.67   3.26  36411  13473
##  2:  rBSTS     6190  20.34   3.26  35767  13473
##  3:   lLOF     9365  16.81   3.26  27834  13473
##  4:   lENT     1222  16.31   3.26  18254  13473
##  5:    rFP     8730  13.24   3.26  20976  13473
## ---
## 12:   lCUN     8095   8.54   3.26  15358  13473
## 13:  rrACC      600   8.14   3.26  22893  13473
## 14:  lcACC     8095   7.03   3.26  17747  13473
## 15:  lrMFG     1533   6.63   3.26  14686  13473
## 16:   rMTG     8730  -8.74   3.26  13481  13473
```

9.5 Plotting the statistics

If you would like to see how the (t- or F-) statistic changes with the threshold, in addition to the critical null statistic value, and the null distribution of maximum statistics, you can simply call the `plot` method for your results. This figure is essentially the same as Figure 11 in Drakesmith et al. (30).

```
argsAnywhere(plot.mtpc)

## function (x, contrast = 1L, region = NULL, only.sig.regions = TRUE,
##     show.null = TRUE, caption.stats = FALSE, ...)
## NULL
```

The function arguments are:

contrast You can only plot statistics for one contrast.

region You can choose one or multiple regions; if you choose multiple, the function returns a list of `ggplot2` objects.

only.sig.regions The default is `TRUE` . If you do not supply a `region` , then all significant regions will be returned.

show.null Logical indicating whether or not to show points for the maximum null statistics. Default: `TRUE`

caption.stats Logical indicating whether or not to print the statistics values underneath the plot (default: `FALSE`). These are the values in the `stats` `data.table` from the results.

In Figure 9.1 I show the statistics plots for 4 regions. I use the `grid.arrange` function (from the `gridExtra` package) to plot all 4. As you can see, regions determined to be significant can have very different statistics profiles, so you should always check these.

```
mtpcPlots <- plot(res.mtpc, region=c('rCAUD', 'rLOF', 'lSMAR', 'lFUS'),
                  contrast=1, caption.stats=TRUE)
grid.arrange(grobs=mtpcPlots, nrow=2, ncol=2)
```

To save a plot for every region, use (a variant of) the following code. You can also save the PDF with multiple plots per page by changing the `nrow` and `ncol` values.

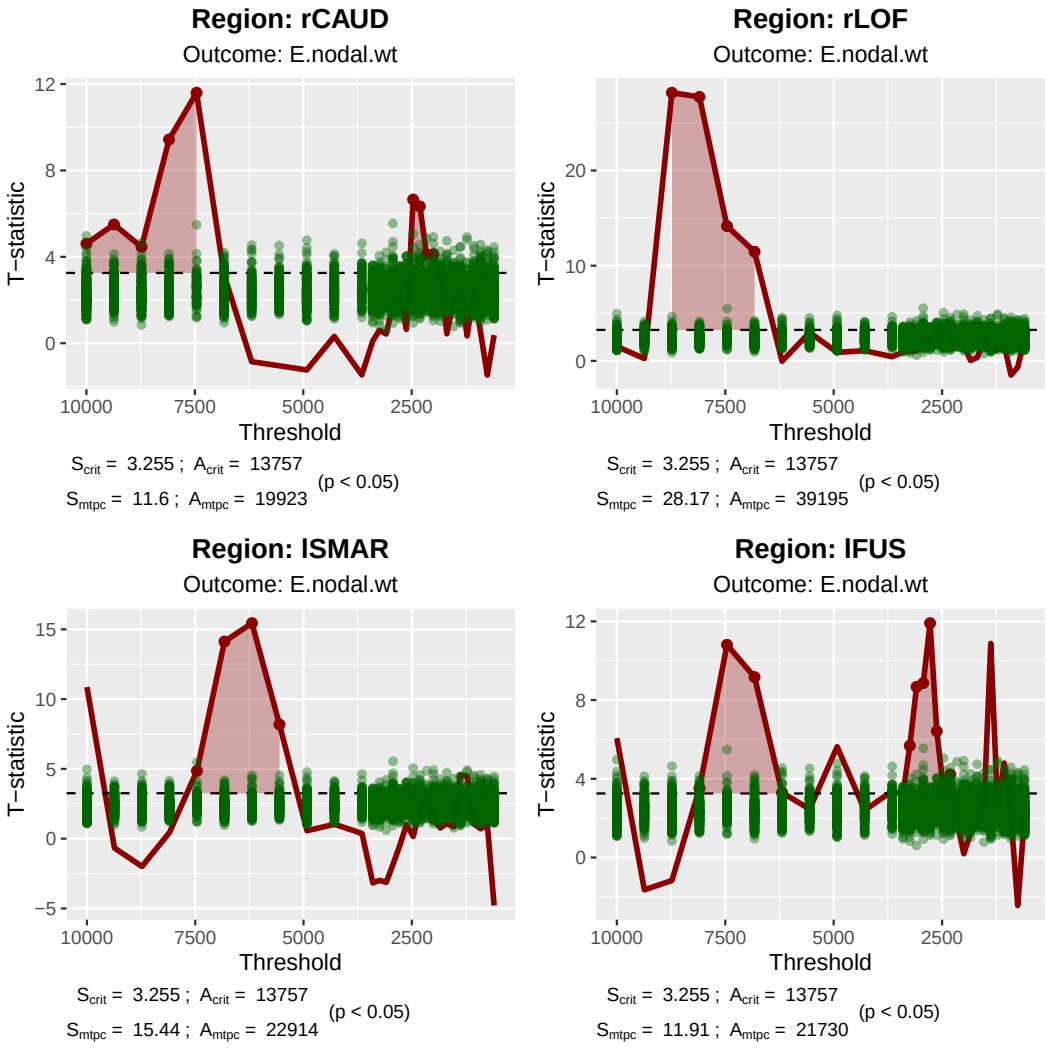

Figure 9.1: **MTPC statistics.** The red shaded areas are those that are above the critical null statistic (dashed line) for at least 3 consecutive thresholds. The green points are the maximum null statistics (per permutation).

```
mtpcPlots <- plot(res.mtpc, contrast=1, only.sig.regions=FALSE)
ml <- marrangeGrob(mtpcPlots, nrow=1, ncol=1)
ggsave('mtpcPlots.pdf', ml, width=8.5, height=11)
```

9.6 Create a graph of the results

To create a graph with attributes specific to MTPC, use `make_brainGraphList`. This will return a list (with length equal to the number of contrasts) of graphs with several additional attributes; see Table 9.1 for the list and a brief description of them.

```
g.mtpc <- make_brainGraphList(res.mtpc, atlas='dk.scgm')
```

Level	Name	Description
Graph	`name`	The contrast name
	`outcome`	The network metric tested
	`tau.mtpc`	Threshold of the maximum observed statistic
	`S.mtpc`	Maximum observed statistic
	`S.crit`	Critical (null) statistic
	`A.crit`	AUC of the null distribution
Vertex	`A.mtpc`	The vertex-wise AUC where $S_{obs} > S_{crit}$
	`sig`	Binary int. indicating whether the vertex was found to be significantly different

Table 9.1: **MTPC graph attributes.** The graph- and vertex-level attributes that are added after calling `make_brainGraphList`.

9.7 Plotting a graph of the results

If you would like to plot only significant regions, you can simply call the `plot` method on the graph. Here I plot the first contrast.

```
# Plot results for the first contrast only
plot(g.mtpc[1], vertex.color='color.lobe', vertex.size=10)
```

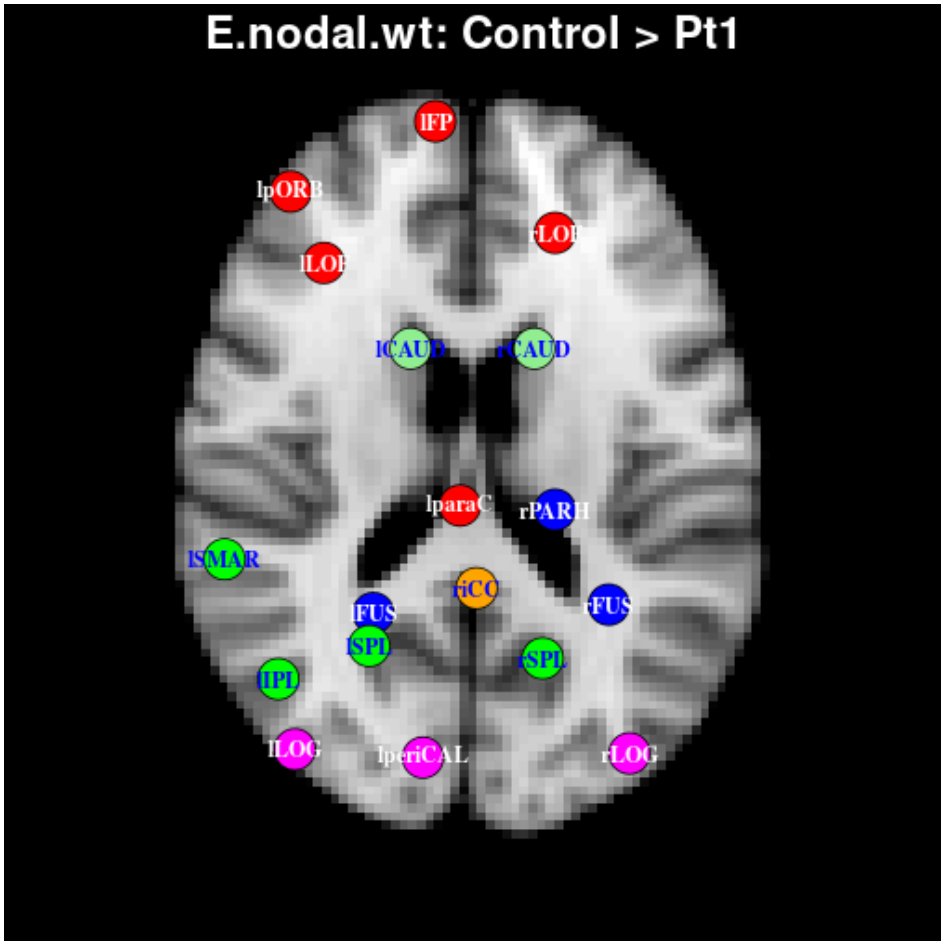

Figure 9.2: **MTPC results.**

10

Network-based statistic (NBS)

The *network-based statistic (NBS)* was introduced by Zalesky et al. for performing FWE control of brain network data in a way that is analogous to cluster-based thresholding in fMRI.(133) The `brainGraph` function for performing NBS is `NBS`.

10.1 Background

Most brain networks are relatively large, in terms of the number of possible pairwise connections: even for a brain atlas with 90 regions (the *AAL* atlas), there are $90 \times (90 - 1)/2 = 4005$ possible connections. This is a *multiple comparisons* problem, but a Bonferroni correction is inappropriate because the data are not independent; a *FDR* adjustment is a better alternative. However, this still treats each data point as isolated; i.e., it does not consider the data as a *network*. The *network-based statistic (NBS)* was thus developed to provide weak FWE control for "mass univariate" testing of all edges in a network.

This algorithm operates directly on the connectivity matrices. A GLM is specified at every matrix element; the model can be as simple as a two-group difference, or as complex as a three-way ANOVA. All designs that are allowed by `brainGraph_GLM` will also work for this function (see Examples). An initial p-value threshold is provided by the user, and the *connected component* size(s) of this graph is (are) calculated. The data are then permuted, and for each permutation the *largest connected component* is recorded, creating a null distribution. A p-value is assigned to the observed connected component size(s) based on the location relative to the null distribution. As in `brainGraph_GLM`, randomization is done through the *Freedman-Lane* procedure.(40)

10.2 Function arguments

As with `mtpc`, many of the arguments are the same as those in `brainGraph_GLM`; see Vertex-wise group analysis (GLM) for more information.

```
args(NBS)

## function (A, covars, contrasts, con.type = c("t", "f"), X = NULL,
##     con.name = NULL, p.init = 0.001, perm.method = c("freedmanLane",
##         "terBraak", "smith", "draperStoneman", "manly", "stillWhite"),
##     part.method = c("beckmann", "guttman", "ridgway"), N = 1000,
##     perms = NULL, symm.by = c("max", "min", "avg"), alternative = c("two.sided",
##         "less", "greater"), long = FALSE, ...)
## NULL
```

A The 3D array of connectivity matrices.

covars

contrasts

con.type

X

con.name

p.init An initial p-value threshold for calculating the observed component sizes. Default: `0.001`

perm.method

part.method

N Default: `1000`

perms

symm.by Character string indicating how to symmetrize the matrices (see Import, normalize, and filter matrices for all subjects). Default behavior is to use the maximum of the off-diagonal elements.

alternative

long

... Arguments that are used for creating the design matrix; see Vertex-wise group analysis (GLM) for more information

10.3 Return value

The function returns an object of class `NBS` . Any values not described here can be found in Vertex-wise group analysis (GLM).

covars

X

p.init The initial p-value.

con.type

contrasts

con.name

alt

p.init

removed.subs

T.mat A 3-D numeric *array* containing the t- or F-statistics. The extent of the array will equal the number of contrasts.

p.mat A 3-D numeric *array* containing the p-values. The extent of the array will equal the number of contrasts.

N

rank

df.residual

qr

cov.unscaled

perm.method

part.method

components A list with two elements: a `data.table` of the observed components and p-values, and a $N_{rand} \times N_C$ matrix of the null distributions of maximum component sizes.

10.4 Code example

Example usage is in the following code block.

```
X <- brainGraph_GLM_design(env.glm$covars.dti, coding='effects',
                           binarize=c('Sex', 'Scanner'))
con.mat <- cbind(0, 0, 0, 0, c(-2, 2))
rownames(con.mat) <- c('Control > Patient', 'Patient > Control')
res.nbs <- NBS(A, env.glm$covars.dti, con.mat,
               coding='effects', binarize=c('Sex', 'Scanner'),
               p.init=0.001, N=1e3, alternative='greater', long=TRUE)
```

Here is what the `summary` method for a `NBS` object returns. First are some details on user-specified inputs. Then it prints a table of the significant components and their associated P-values.

```
summary(res.nbs)

##
## ====================================
## Network-based statistic results
## ====================================

## GLM formula:
##   weight ~ Age.MRI + Sex + Scanner + Group
## based on 156 observations, with 151 degrees of freedom.
```

```
##
## Permutation analysis
## --------------------

## Permutation method:  Freedman-Lane
## Partition method:  Beckmann
## # of permutations:  1,000
## Initial p-value:  0.001
##
## Contrast type:  T contrast
## Alternative hypothesis:  C > 0
## Contrasts:
##                   Intercept Age.MRI Sex Scanner GroupPatient
## Control > Patient  .         .       .   .      -2
## Patient > Control  .         .       .   .       2

##
## Statistics
## --------------------
## Control > Patient:

##      contrast # vertices # edges p-value
## [1,]        1          6       5   0.001 ***
## [2,]        1          4       3   0.001 ***
## [3,]        1          3       2   0.006 **
## [4,]        1          3       2   0.006 **
## [5,]        1          2       1   0.196
## [6,]        1          2       1   0.196
## [7,]        1          2       1   0.196
## [8,]        1          2       1   0.196
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

## Patient > Control:

##      contrast # vertices # edges p-value
## [1,]        2          3       2   0.002 **
## [2,]        2          2       1   0.076 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

10.5 Creating a graph of the results

To create a graph with attributes specific to NBS, use `make_brainGraphList`. This returns a list (length equal to the number of contrasts) of graphs with several attributes (in addition to those created by `set_brainGraph_attr`); see Table 10.1.

```
g.nbs <- make_brainGraphList(res.nbs, atlas=atlas)
```

Level	Name	Description
Edge	`stat`	The t- or F-statistics for each connection
	`p`	1− the P-value for each connection
Vertex	`comp`	Integer index of the connected component that each vertex is a member of
	`p.nbs`	1− the P-value for each connected component

Table 10.1: **NBS graph attributes.** The graph- and vertex-level attributes that are added after calling `make_brainGraphList`.

10.6 Plotting the results

The default behavior of the plotting method for NBS results will only show the component(s) with a significant effect, given some value α. As you can see in Figure 10.1, it will also plot vertices and edges of the same component with the same color and give the plot a title containing the contrast name (this can be over-ridden by changing the `main` argument). You can adjust various other aspects of the plot, as with any other graph; for example, you could change the edge width to scale with the statistic value (by typing `edge.width='stat'` .

```
plot(g.nbs[1], alpha=0.05)
```

10.7 Testing

For some benchmarking info, see Benchmarks. I have done some testing of this function; the t-statistics calculated in my function matched those of the `Matlab` toolbox `NBS`; the connected component sizes differed by only 1, and I assume this is due to their toolbox thresholding by *T-statistic*, whereas I threshold by *p-value*. I encourage you to do your own testing if you wish.

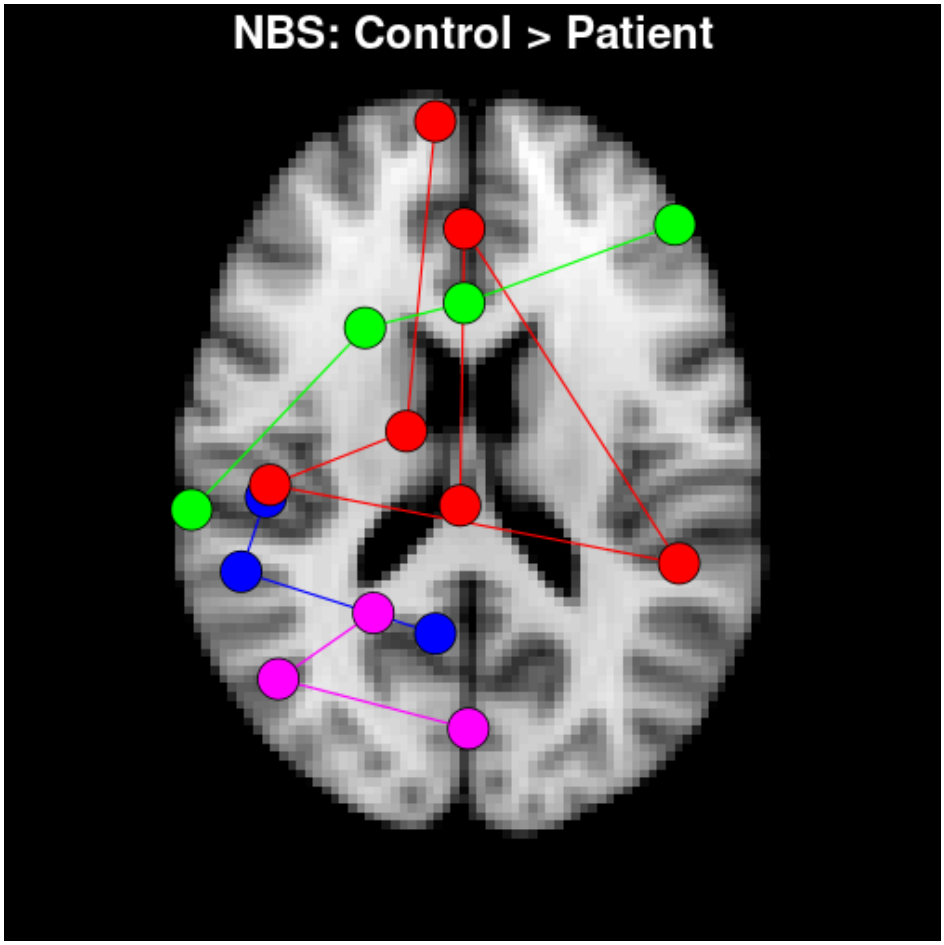

Figure 10.1: **Significant connected components.** Vertices of the same component are plotted in the same color.

11

Graph- and vertex-level mediation analysis

Performing mediation analysis in `brainGraph` follows the *potential outcomes* framework that is implemented in the mediation package (113), as opposed to the more well-known "Baron & Kenny approach" (6). A few advantages of the potential outcomes framework include:

- Allows for mediation-treatment interaction and nonlinearities
- Addresses confounds
- Allows for the decomposition of the total effect into direct and indirect effects

A full treatment of mediation is beyond the scope of this document, but I provide some background here as it can be quite difficult to understand without prior exposure.

11.1 Background

As explained in Preacher (87), the *potential outcomes* framework for causal inference was re-introduced (or popularized) by Donald Rubin in the 1970's (97, 98).[1] In fact, Holland (50) calls it the *Rubin Causal Model*. To re-iterate, for binary treatment variable X, the effect of X on outcome Y for case/subject i can be defined as the difference between two potential outcomes:

1. $Y_i(1)$ – the outcome that would be realized if $X_i = 1$
2. $Y_i(0)$ – the outcome that would be realized if $X_i = 0$

We can say that X *causes* Y if $Y_i(1) \neq Y_i(0)$. However, we can never observe both $X_i = 1$ and $X_i = 0$; Holland (50) calls this the *fundamental problem of causal inference*. Under random assignment, the expected mean between-group difference is equal to the difference in group means; i.e.,

$$E[Y_i(1)] - E[Y_i(0)] = E[Y_i|X_i = 1] - E[Y_i|X_i = 0]$$

[1] The notation had previously been used by Jerzy Neyman in 1923 for randomized experiments.

There are also potential outcomes for the mediator M, only one of which can be observed for a given unit i (i.e., either $M_i(1)$ or $M_i(0)$). The outcomes for unit i can then be defined using the notation $Y_i(X_i, M_i(X_i))$, and the *indirect effect* δ is defined as the "change that would occur in Y when moving from the value of M if $X_i = 0$ to the value of M if $X_i = 1$" (Preacher 2015, p. 834). In equation form,

$$\delta_i(x) = Y_i(x, M_i(1)) - Y_i(x, M_i(0))$$

And under certain assumptions (see below), the average indirect effect $\bar{\delta}(x)$ is simply the difference in expected value: (also known as the *average causal mediation effect (ACME)*)

$$\bar{\delta}(x) = E[Y_i(x, M_i(1)) - Y_i(x, M_i(0))] \tag{11.1}$$

The *average direct effect (ADE)* is he difference in potential outcomes between treatment groups:

$$\bar{\zeta}(x) = E[Y_i(1, M_i(x)) - Y_i(0, M_i(x))] \tag{11.2}$$

And, under the potential outcomes framework, the *total effect* is the sum of the ACME and ADE:

$$\bar{\tau}(x) = \delta_i(x) + \zeta_i(1 - x) \tag{11.3}$$

11.1.1 Suggested reading

An excellent review of mediation analysis can be found in Preacher (87). Specifically, the subsection "Model-Based Tradition" (in the section "Causal Inference for Indirect Effects") provides some historical background in addition to a set of assumptions and implementations that fit with the approach taken in `mediation`. For the earliest works, see Rubin (97); Holland (50); Robins and Greenland (95). Furthermore, Imai and colleagues, who created the `mediation` package, have several papers that should be considered required reading (56, 57, 58, 113); and see also Keele (64).

11.2 Notation

As I mentioned, it is critical that you read Tingley et al. (113). For convenience, I will include some notation here, including the variable names as seen in the `summary` method.[2]

??.acme	The *average causal mediation effect* (see Equation 11.1). This is typically the effect of interest in mediation analysis. Represented by the variables `d0,d1`
??.ade	The *average direct effect* (see Equation 11.2). Represented by the variables `z0,z1`
?.tot	The *total effect* (see Equation 11.3). Represented by the variable `tau`
??.prop	The *proportion mediated*. This is the ACME divided by the total effect. Represented by the variables `n0,n1`
b.*	The effect size; e.g., `b.acme` is the ACME estimate
ci.low.*, ci.high.*	The confidence intervals; e.g., `ci.low.acme`
p.*	The P-value; e.g., `p.acme`

11.3 Function arguments

There are a few arguments that are the same as in `brainGraph_GLM`, so their descriptions are omitted here.

[2] The `?` is a *wildcard* representing any single character.

```
args(brainGraph_mediate)

## function (g.list, covars, mediator, treat, outcome, covar.names,
##     level = c("graph", "vertex"), control.value = 0, treat.value = 1,
##     int = FALSE, boot = TRUE, boot.ci.type = c("perc", "bca"),
##     N = 1000, conf.level = 0.95, long = FALSE, ...)
## NULL
```

11.3.1 Mandatory

g.list As with `brainGraph_GLM`, a "flat" list of graph objects for ≥ 1 subject group at a single threshold/density.

covars Must have columns `'Study.ID'` and several names you supply in the function call: `treat`, `outcome`, `covar.names`

mediator The name of the mediator variable; this must be a valid graph- or vertex-level attribute (e.g., `'E.global.wt'`, `'btwn.cent'`, etc.)

treat The name of the *treatment* variable. A common example would be `'Group'`

outcome The name of the *outcome* variable. This is usually going to be a non-MRI variable you measure in the subject; e.g., IQ some other neuropsychological test/questionnaire score (e.g., WISC processing speed, CVLT immediate recall, etc.)

covar.names Character vector of the nuisance covariates you would like to include in your analyses (e.g., age, sex, etc.)

11.3.2 Optional

level

control.value The value of `treat` to be used as the *control* condition. If your `treat` variable is a *factor*, then you may wish to use the first level. The default, `0`, essentially has this effect.

treat.value The value of `treat` to be used as the *treatment* condition. If your `treat` variable is a *factor*, then you may wish to use the second (or a higher) level. The default, `1`, essentially has this effect.

int Logical indicating whether or not to include a *mediator-treatment* interaction term. Default: `FALSE`

boot Logical indicating whether or not to do bootstrap resampling for estimating confidence intervals (CI) and statistical significance. You should *always* do bootstrap resampling. Default: `TRUE`

boot.ci.type Character string indicating the type of bootstrap confidence intervals. The default is `'perc'` (percentile bootstrap), but you may also choose `'bca'` (bias-corrected and accelerated). However, see Biesanz et al. (11) and Falk and Biesanz (36) which find that the *BCa* approach results in inflated Type I error while the percentile bootstrap performs well.

N Integer; number of bootstrap samples. Default: `1000`

conf.level The confidence level for calculating CI's. Default: `0.95`

long Logical indicating whether or not to also return the statistics for each bootstrap sample. Default: `FALSE`

... Arguments that are used for creating the design matrix. See Vertex-wise group analysis (GLM) for more information; you should not specify the `coding` argument, thus accepting the default coding `'dummy'`

11.4 Return value

The function returns an object of class `bg_mediate` . You will usually not work with the specific elements of this object yourself, but I list them for completeness. Any elements that are simply input arguments are included without description. For more details, see the help file for the `mediate` function in the `mediation` package.

level

removed.subs

X.m The design matrix of the model in which the *mediator* is the outcome variable

y.m A matrix of the mediator variable; the number of columns equals the number of brain regions

X.y A *named list* (the names will be the vertex/region names) of the design matrices of the models in which the mediator is another predictor variable

y.y A numeric vector; the *outcome* variable specified in the function call

res.obs A `data.table` of the *observed* statistics (point estimates)

res.ci A `data.table` of the confidence intervals

res.p A `data.table` of the (two-sided) P-values

boot

boot.ci.type

res.boot A `data.table` of the statistics from all bootstrap samples (if `long=TRUE`)

treat

mediator

outcome

covariates `NULL`

INT The same as input argument `int`

conf.level

control.value

treat.value

nobs The number of observations in the data (i.e., number of rows in the design matrix)

sims The same as input argument `N`

covar.names

11.5 Code example

The following code blocks show how to run `brainGraph_mediate` for a few different network measures. This is basically the same approach I used in Multi-threshold permutation correction. First, I create a table with some of the function argument values.

```
medVars <- data.table(level=c(rep('graph', 2), rep('vertex', 2)),
                      outcome='FSIQ',
                      mediator=c('E.global.wt', 'mod.wt', 'E.nodal.wt', 'strength'),
                      treat='Group',
                      interact=FALSE,
                      N=c(rep(1e4, 2), rep(5e3, 2)))
medVars

##     level outcome    mediator treat interact     N
## 1:  graph    FSIQ E.global.wt Group    FALSE 10000
## 2:  graph    FSIQ      mod.wt Group    FALSE 10000
## 3: vertex    FSIQ  E.nodal.wt Group    FALSE  5000
## 4: vertex    FSIQ    strength Group    FALSE  5000
```

Then you can loop through these values to create a list (of lists) of `bg_mediate` objects. I usually separate graph- and vertex-level analyses into different objects.

Below is code to loop across all thresholds for the graph-level measures. Depending on the number of thresholds, this can take a very long time. The list object `med.list.g` will have 3 "levels":

1. The `outcome` variable(s) (in this case, *FSIQ*)
2. The `mediator` variable(s)
3. An element for each threshold

Note that it is *not recommended* that you perform analyses for every outcome-mediator combination and then choose to report only those with significant results.

```
med.list.g <- sapply(medVars['graph', unique(outcome)], function(x) NULL)
for (x in names(med.list.g)) {
  med.list.g[[x]] <- sapply(medVars[.('graph', x), unique(mediator)], function(x) NULL)
  for (y in names(med.list.g[[x]])) {
    print(paste('Outcome:', x, '; Mediator:', y, ';', format(Sys.time(), '%H:%M:%S')))
    med.list.g[[x]][[y]] <- vector('list', length=length(thresholds))
    for (z in seq_along(med.list.g[[x]][[y]])) {
      med.list.g[[x]][[y]][[z]] <-
        brainGraph_mediate(g.list[[z]], covars.med, mediator=y,
                           treat=medVars[.('graph', x, y), treat],
                           outcome=medVars[.('graph', x, y), outcome],
                           covar.names=c('age', 'gender', 'scanner'),
                           boot=T, boot.ci.type='perc', N=medVars[.('graph', x, y), N],
                           long=TRUE, binarize=c('gender', 'Scanner'))
    }
  }
}
```

11.5.1 Printing a summary

There are 2 ways to use the `summary` method. First, I show the result when setting `mediate=TRUE`, which uses the method from the `mediation` package. For this, you must select a region; if you do not, the first (alphabetically) will be selected.

```
summary(res.med, mediate=TRUE, region='lHIPP')

##
## ====================================
## Vertex-level mediation results
## ====================================

## # of observations:  71

##
## Variables
## ------------------

##
##    Mediator:            E.nodal.wt
##   Treatment:            Group
##      Control condition: Control
##    Treatment condition: Patient
##     Outcome:            brief_gec_raw_6m
##
##  Covariates:           age
##                        gender
##                        scanner
##
## Treatment-mediator interaction?  FALSE

##
## Bootstrapping
## ------------------

## Bootstrap CI type:  Percentile bootstrap
## # of bootstrap replicates:  1,000
## Bootstrap CI level:  [2.5% 97.5%]

## Mediation summary for:  lHIPP
## ------------------

##
## Causal Mediation Analysis
##
## Nonparametric Bootstrap Confidence Intervals with the Percentile Method
##
##                          Estimate % CI Lower % CI Upper p-value
## ACME (control)             1.1049   -10.2795      11.74   0.780
## ACME (treated)             1.1049   -10.2795      11.74   0.780
## ADE (control)             17.4068    -0.5028      33.66   0.062 .
## ADE (treated)             17.4068    -0.5028      33.66   0.062 .
## Total Effect              18.5116     6.5066      29.60   0.006 **
## Prop. Mediated (control)   0.0597    -0.6195       1.03   0.786
## Prop. Mediated (treated)   0.0597    -0.6195       1.03   0.786
## ACME (average)             1.1049   -10.2795      11.74   0.780
```

```
## ADE (average)             17.4068    -0.5028      33.66   0.062 .
## Prop. Mediated (average)   0.0597    -0.6195       1.03   0.786
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Sample Size Used: 71
##
##
## Simulations: 1000
```

The second option will print the entire statistics table (for all regions). Since everything above the "Bootstrapping" section is the same, I only show the summary table.

```
summary(res.med)$DT[]

##       region   Mediator      treat    Outcome  b0.acme ci.low0.acme
##  1:    lACCU E.nodal.wt E.nodal.wt E.nodal.wt  2.14919       -5.788
##  2:    lAMYG E.nodal.wt E.nodal.wt E.nodal.wt -0.04912      -13.119
##  3:    lBSTS E.nodal.wt E.nodal.wt E.nodal.wt -0.29800       -5.046
##  4:    lCAUD E.nodal.wt E.nodal.wt E.nodal.wt  0.61905       -4.795
##  5:     lCUN E.nodal.wt E.nodal.wt E.nodal.wt -0.63081       -5.462
## ---
## 78: rperiCAL E.nodal.wt E.nodal.wt E.nodal.wt  0.45775       -4.196
## 79:   rpostC E.nodal.wt E.nodal.wt E.nodal.wt -2.17657       -7.697
## 80:    rpreC E.nodal.wt E.nodal.wt E.nodal.wt -2.30992       -9.053
## 81:    rrACC E.nodal.wt E.nodal.wt E.nodal.wt -0.77329       -5.903
## 82:    rrMFG E.nodal.wt E.nodal.wt E.nodal.wt -0.26909       -8.318
##     ci.high0.acme p0.acme b0.ade ci.low0.ade ci.high0.ade p0.ade b.tot
##  1:        10.532   0.534  16.36       1.660        31.48  0.024 18.51
##  2:        12.484   0.986  18.56       2.158        37.25  0.028 18.51
##  3:         3.980   0.882  18.81       7.024        30.54  0.000 18.51
##  4:         6.105   0.788  17.89       5.671        30.84  0.004 18.51
##  5:         4.165   0.812  19.14       6.044        31.80  0.004 18.51
## ---
## 78:         5.340   0.836  18.05       5.901        30.66  0.010 18.51
## 79:         1.136   0.228  20.69       9.355        33.13  0.000 18.51
## 80:         1.783   0.306  20.82       8.392        34.05  0.004 18.51
## 81:         4.079   0.750  19.28       6.186        32.49  0.008 18.51
## 82:         8.121   0.974  18.78       5.554        31.60  0.010 18.51
##     ci.low.tot ci.high.tot p.tot   b0.prop ci.low0.prop ci.high0.prop
##  1:      6.999       30.89 0.002  0.116100      -0.3655       0.81121
##  2:      6.843       30.25 0.004 -0.002653      -0.8980       0.81541
##  3:      6.330       30.51 0.002 -0.016098      -0.4248       0.25000
##  4:      7.409       30.74 0.004  0.033441      -0.3170       0.40759
##  5:      6.893       29.39 0.000 -0.034077      -0.3772       0.28401
## ---
## 78:      6.609       30.62 0.002  0.024728      -0.3214       0.39066
## 79:      7.383       30.66 0.000 -0.117578      -0.6691       0.06401
## 80:      6.679       31.45 0.006 -0.124782      -0.6832       0.10843
## 81:      6.523       30.92 0.008 -0.041773      -0.4523       0.25909
## 82:      6.411       29.82 0.004 -0.014536      -0.6550       0.44658
##     p0.prop
##  1:   0.532
##  2:   0.986
```

Level	Name	Description
Graph	`mediator`	The mediator variable
	`treat`	The treatment variable
	`outcome`	The outcome variable
	`nobs`	The number of observations
Vertex	`b.acme`, `p.acme`	The point estimates and (two-sided) P-values for the ACME
	`b.ade`, `p.ade`	The point estimates for the ADE
	`b.tot`, `p.tot`	The point estimates for the total effect
	`b.prop`, `p.prop`	The point estimates for the proportion mediated

Table 11.1: **Mediation graph attributes.** The graph- and vertex-level attributes that are added after calling `make_brainGraph`.

```
##  3:   0.884
##  4:   0.784
##  5:   0.812
## ---
## 78:   0.834
## 79:   0.228
## 80:   0.312
## 81:   0.746
## 82:   0.970
```

11.6 Create a graph of the results

To create a graph with attributes specific to mediation analysis, use `make_brainGraph`. This will return a graph with several additional attributes; see Table 11.1 for the list and a brief description of them.

```
g.med <- make_brainGraph(res.med, atlas='dkt.scgm')
```

11.7 Plot a graph of the results

If you would like to plot only significant regions, you can simply call the `plot` method on the graph. In this example, however, there were no vertices for which $P_{ACME} < \alpha$, so I relax this for display purposes. Note that I use the form `p0.acme > X`, as the P-values are actually subtracted from 1.[3] I also use the default `main` value for printing the plot title (which includes the names of the mediator, treatment, and outcome variables).

```
plot(g.med, subgraph='p0.acme > 0.9', vertex.color='color.lobe', vertex.size=10)
```

[3]Since there is no mediator-treatment interaction, `p0.acme` equals `p1.acme`.

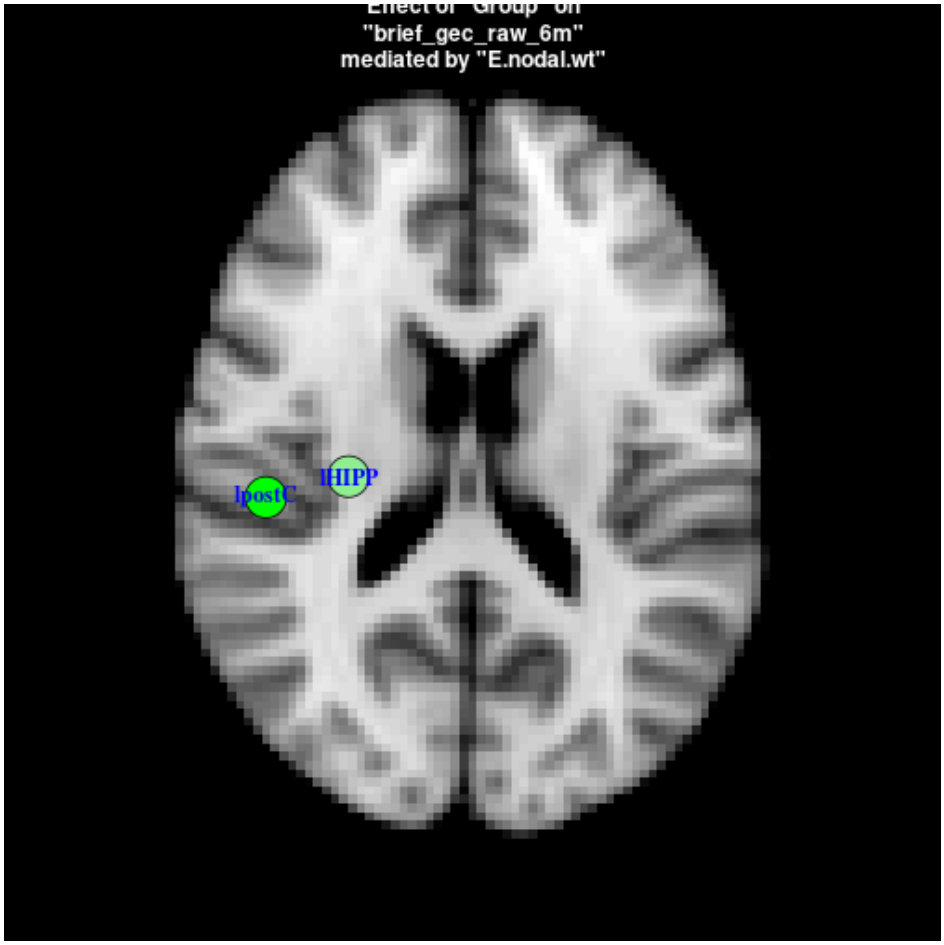

Figure 11.1: **Mediation results.**

11.8 Benchmarks

The `mediate` function in the `mediation` package is quite slow, but very feature-rich (i.e., there are many different types of models you can use in your analyses such as generalized additive models). In `brainGraph_mediate`, only simple linear models are allowed. This is a sacrifice in features but comes with a high speed advantage. Second, `mediate` works with *model objects* (e.g., `R` objects of type `lm`). These objects are convenient for interactive data analysis, but overall slower to work with. Third, `mediate` does not have a parallel or multi-core option, whereas I take advantage of the `foreach` package. For a multi-region analysis in which the same model is used for every brain region (e.g., 82 regions in the `dk.scgm` atlas), and with 1,000 (or more) bootstrap samples calculated, the speed increase can be large (particularly with a higher number of bootstrap samples). Finally, I use `data.table` for fast calculations.

I have seen speed increases of 10x–30x. You can see the actual numbers in Benchmarks.

Part V

Group Analyses: Other

12

Random graphs, small world, and rich-club

12.1 Random graph generation

Random graph generation is a necessary step if you want to calculate a graph's small-worldness.(55, 126) However, generating an *appropriate* random graph is not trivial. This is particularly true when working with data generated from correlations (e.g., structural covariance and resting-state fMRI networks), as graphs generated from this kind of data will necessarily have a higher than expected level of clustering.(53, 134) It is true for other data, as well, because random networks have low clustering in general. For more discussion, see Chapter 12 of Newman (79), and also Telesford et al. (112).

12.1.1 Simple random graph generation

This is the "standard" method of creating random graphs, controlling for the observed *degree distribution*. It relies on the `igraph` functions `rewire` along with `keeping_degseq` (see Viger and Latapy (121) for the method used; see also Milo et al. (78)). `sim.rand.graph.par` generates a number of these in parallel, and calculates *modularity, clustering coefficient, average path length, global efficiency*, and *rich club coefficient* for each random graph. The number of rewiring steps is hard-coded to be the larger of either 10,000 or 10× the graph's edge count (see Ray et al. (90)).

The function `analysis_random_graphs` is a "helper function" (or script) that performs all of the steps that are typically done when you need to create equivalent random graphs. These include:

1. Generate `N` random graphs for each group and density/threshold (and *subject* if you have subject-specific graphs).
2. Write the random graphs to disk (in `savedir`).
3. Read the random graphs back into `R` and combine into *lists*; there will be one *list* object per group and density/threshold combination.
4. Write the combined *list* objects to disk (in a sub-directory named `ALL`); you can delete the individual *.rds* files afterwards (after ensuring that you have all the data).
5. Calculate *small world* parameters, along with a few global graph measures that may be of interest.
6. Calculate *normalized rich-club coefficients* and associated P-values.

The return object is a *list*, with three elements (each one is a `data.table`):

`rich` (normalized) rich-club coefficients and p-values

`small` small-world parameters and related information

`rand` various graph-level measures for the random graphs

```
kNumRand <- 1e2
clustering <- F
outdir <- file.path(getwd(), '../rand')

rand_vars <- analysis_random_graphs(g, N=kNumRand, savedir=outdir,
                                    clustering=clustering)
rich.dt <- rand_vars$rich
rich.dt <- rich.dt[complete.cases(rich.dt)]  # Remove rows w/ NA
small.dt <- rand_vars$small
rand.dt <- rand_vars$rand
```

12.1.2 Control for clustering

`sim.rand.graph.clust` generates equivalent random graphs controlling not only for degree distribution, but also clustering. It uses the algorithm given in Bansal et al. (4). Since you will want to generate a large number of graphs, you should specify `clustering=TRUE` in the call to `sim.rand.graph.par`. Because this step takes quite long, you can limit the number of Markov Chain steps with the `max.iters` argument. The default is 100. Keeping this at 100 is a reasonable limit, but basically defeats the purpose of this step as the observed level of clustering may not be reached with only 100 iterations.

The time required to generate the random graphs with this method can be quite high, and is higher for graphs with high clustering and density. The final line shows how to calculate ω (see next section); this requires that you have already generated simple random graphs and created `small.dt` (done in the previous section).

Warning

This will take a long time. See Benchmarks for example processing times.

```
kNumRandClust <- 1e2  # Create 100 graphs per group/density combination
bgl.rand <- g.rand <- small.clust.dt <- vector('list', length(g))
for (i in seq_along(g)) {
  g.rand[[i]] <- vector('list', length=nobs(g[[i]]))
  for (j in seq_len(nobs(g[[i]]))) {
    g.rand[[i]][[j]] <- sim.rand.graph.par(g[[i]][j], level=g[[i]]$level,
                                           kNumRandClust, clustering=T,
                                           name=g[[i]][j]$name)
  }
  bgl.rand[[i]] <- as_brainGraphList(g.rand[[i]], type='random', level=g[[i]]$level)
  small.clust.dt[[i]] <- small.world(g[[i]], bgl.rand[[i]])
}
small.clust.dt <- rbindlist(small.clust.dt)
```

12.1.3 Random covariance matrices

You can also create random covariance matrices using the *Hirschberger-Qi-Steuer (HQS)* algorithm (49). The function is `sim.rand.graph.hqs`. It generates the observed covariance matrix from residuals (output by `get.resid`), but should work for any other matrix. The random covariance matrices are then scaled to correlation matrices and the random graphs are created. By default, weighted graphs are returned in which edge weights represent correlation values. If you want to return unweighted graphs, you must provide correlation thresholds via the argument `r.thresh` .
To create random graphs for a single group, see the following code block.

```
r1 <- resids[, grps[1]]
thr1 <- corrs[, 1]$thresholds[1]
g.rand.hqs <- sim.rand.graph.hqs(r1, level='group', N=100,
                                 weighted=FALSE, r.thresh=thr1)
```

12.2 Small-worldness

The function `small.world` will calculate small world parameters, including normalized clustering coefficient and characteristic path length, as well as the small world coefficient σ.(55) It returns a `data.table` with those values. Each row corresponds to a group/density combination.

```
head(small.dt)

##    density   N    Lp     Cp Lp.rand Cp.rand Lp.norm Cp.norm sigma   Group
## 1: 0.05004 100 3.252 0.5445   2.633  0.1576   1.235   3.455 2.797 Control
## 2: 0.06014 100 2.888 0.5483   2.522  0.1747   1.145   3.139 2.740 Control
## 3: 0.07024 100 2.868 0.5468   2.452  0.1891   1.170   2.891 2.471 Control
## 4: 0.08033 100 2.699 0.5209   2.353  0.2119   1.147   2.458 2.143 Control
## 5: 0.09043 100 2.571 0.5203   2.283  0.2331   1.126   2.232 1.982 Control
## 6: 0.10009 100 2.408 0.5272   2.194  0.2517   1.098   2.095 1.908 Control
```

To calculate ω (see Telesford et al. (112)), which is a better metric for small-worldness, you will have to:

1. Generate simple random graphs to get the characteristic path length L_{rand} (see previous section).
2. Create random graphs controlling for the level of clustering, and use the clustering coefficient of these random graphs as the value for equivalent *lattice* networks (C_{latt})
3. The values L and C are characteristic path length and clustering coefficient, respectively, for the observed graphs.

The equations for both σ and ω are:

$$\sigma = \frac{C/C_{rand}}{L/L_{rand}} \tag{12.1}$$

$$\omega = \frac{L_{rand}}{L} - \frac{C}{C_{latt}} \tag{12.2}$$

The only downside to this approach is the processing time needed. As I mentioned earlier, a reasonable compromise would be to limit the number of Markov Chain steps (via the `max.iters` function argument), as even with only 100 iterations, the resultant graphs will be much closer to a lattice than a simple random graph controlling only for degree distribution.

The `R` code for calculating ω is:

```
setkey(small.dt, Group, density)
setkeyv(small.clust.dt, key(small.dt))
small.dt[, omega := small.dt[, Lp.rand / Lp] - small.clust.dt[, Cp / Cp.rand]]
small.dt[, Lp.latt := small.clust.dt$Lp.rand]
small.dt[, Cp.latt := small.clust.dt$Cp.rand]
small.long <- melt(small.dt, id.vars=c('density', 'Group', 'N'))
```

To show that the random graph generation approach from the last section *does* result in networks with higher clustering, we can plot these values, shown in Figure 12.1. As you can see, the random graphs generated by the Markov Chain process (dotted lines) have a similar clustering level as the observed networks (solid lines). As expected, the "simple" random graphs have much lower clustering. This is also reflected in the much higher normalized clustering coefficients in calculating σ.

If you refer back to Equation 12.1 and Equation 12.2, the normalized path length will roughly equal 1; i.e., $L/L_{rand} \approx L_{rand}/L \approx 1$. It is also the case that $C/C_{rand} \gg 1$ and $C/C_{latt} \approx 1$. So we should see $\sigma \gg 1$ and $\omega \approx 0$, which is confirmed in Figure 12.2.

```
ggplot(small.dt.m, aes(x=density, y=value, col=interaction(Group, variable),
                       lty=interaction(Group, variable))) +
  geom_line(size=1.25) +
  scale_linetype_manual(name='Group, type',
                        labels=mylabels.sm,
                        values=rep(1:3, each=2)) +
  scale_color_manual(name='Group, type',
                        labels=mylabels.sm,
                        values=rep(c('red', 'cyan3'), 3)) +
  labs(x='Density', y='Clustering coefficient')
```

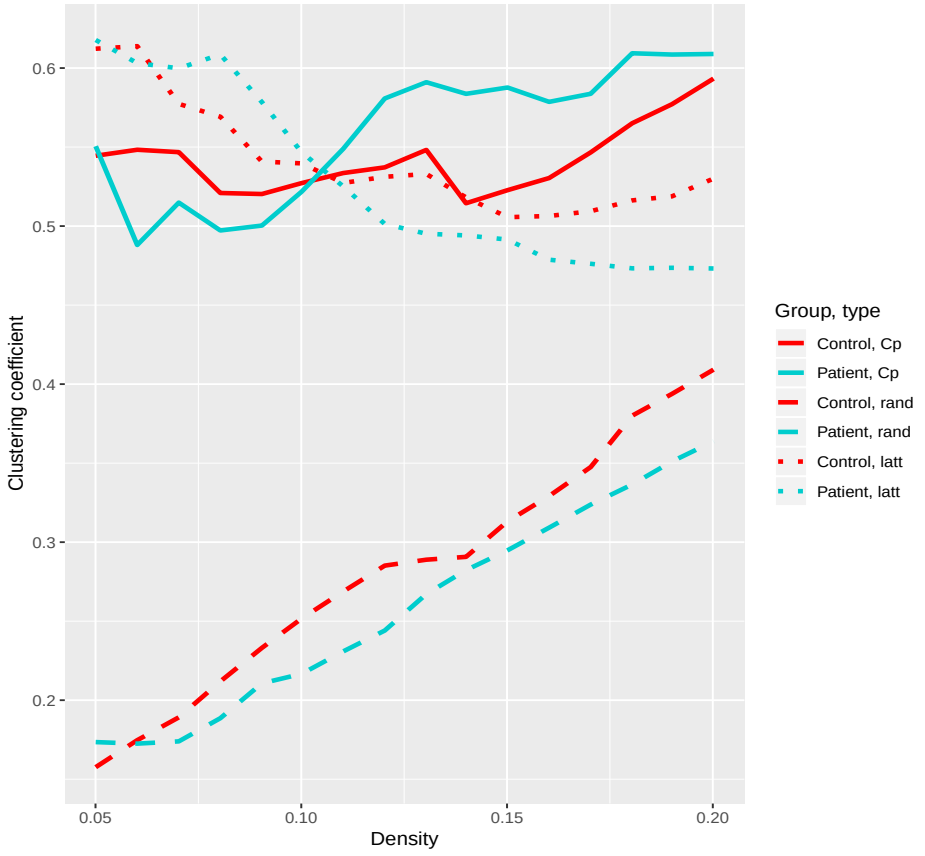

Figure 12.1: **Clustering coefficients for random networks.** The solid lines are clustering for the observed networks. The dotted line are values from networks generated by the Markov Chain approach, and the dashed lines are values from networks generated by the "simple" approach.

We can plot both σ and ω in the same plot, shown in Figure 12.2. As stated in Telesford et al. (112), networks with ω closer to 0 indicate more balance between high clustering and low path length; networks with ω closer

to −1 are more similar to a lattice network. The results of this analysis indicate that the Patient group's networks at higher density are closer to a lattice than the Control networks.

```
small.long[, yint := 1]
small.long[variable == 'omega', yint := 0]
ggplot(small.long[variable %in% c('sigma', 'omega')],
       aes(x=density, y=value, col=Group, group=Group)) +
  geom_line() +
  geom_hline(aes(yintercept=yint), lty=2) +
  facet_wrap(~ variable, scales='free_y', ncol=1) +
  theme(legend.position=c(1, 1), legend.justification=c(1, 1)) +
  ylab('Small-worldness')
```

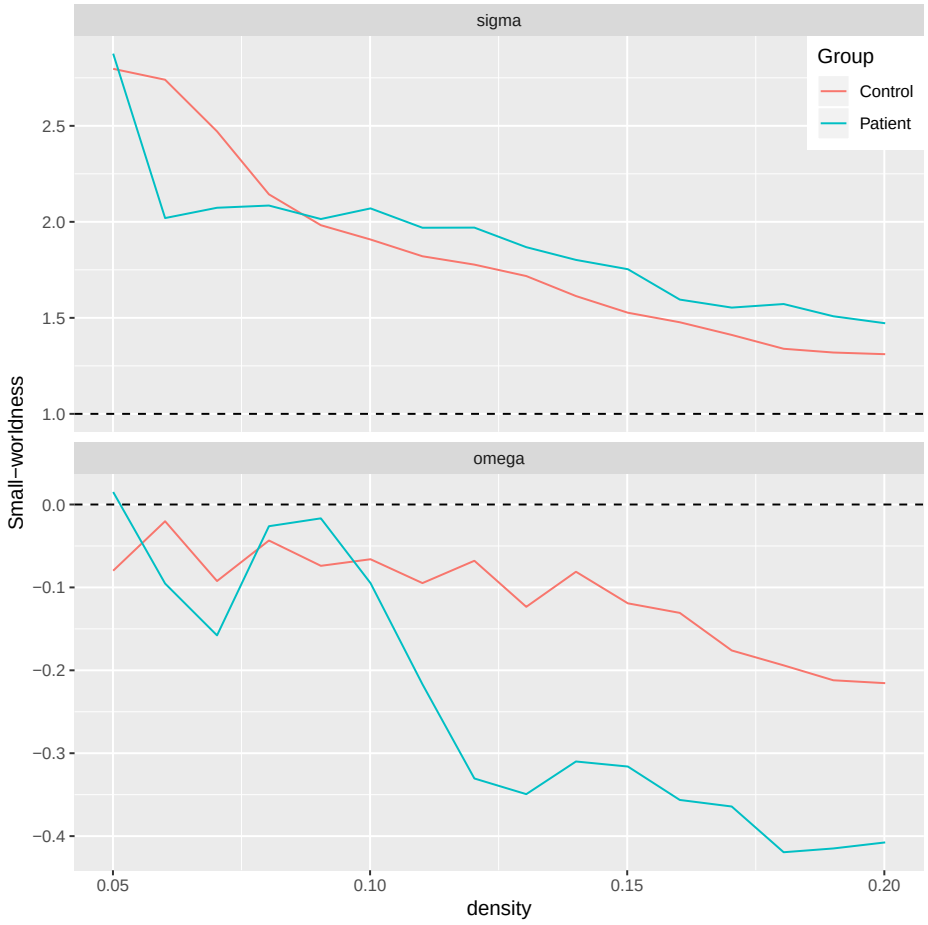

Figure 12.2: **Small-world indexes.** (top) The classic small-world index, σ; the dashed horizontal line at $y = 1$ is included to show the minimum value for a network to be considered "small-world" (126) (bottom) The small-world index ω introduced by Telesford et al. (112); the dashed horizontal line at $y = 0$ indicates the value at which the network displays a balance between clustering coefficient and characteristic path length.

Level	Name	Description
Vertex	`rich`	Binary value indicating whether or not the vertex is in the rich club
	`color.rich`	Either `red` (rich-club vertices) or `gray` (non-rich-club vertices)
	`size.rich`	Either `3` (non-rich-club vertices), or `15` (rich-club vertices)
Edge	`type.rich`	Either `rich-club` (edges connecting 2 rich-club vertices), `feeder` (edges connecting 1 rich-club and 1 non-rich-club vertex), or `local` (edges connecting 2 non-rich-club vertices)
	`color.rich`	Either `red` (rich-club connections), `orange` (feeder connections), or `green` (local connections)
	`size.rich`	Either `3.5` (rich-club connections), `1.5` (feeder connections), or `0.5` (local connections)

Table 12.1: **Rich-club graph attributes.** The and vertex- and edge-level attributes that are added after calling `rich_club_attrs`.

12.3 Rich-club Analysis

The *rich club* is a set of vertices that have a high degree themselves and which also have a high probability of being connected to one another (see Colizza et al. (16), Zhou and Mondragón (135)).
When creating random graphs with `analysis_random_graphs`, normalized rich-club coefficients and P-values were also calculated (now in `rich.dt`). This will be used later for plotting.

12.3.1 Rich-core

A recent paper provided an algorithm for determining the cut-off degree for inclusion in the rich club (71). The function to calculate this is called `rich_core`. The degree value is given in the output data frame, with column name `k.r` (`k` stands for *degree* and the `r` is for *rank*). You can calculate the rich-core for both binary and weighted graphs. In the next code section, I show how to plot a shaded region that starts at the maximum cut-off value from 2 groups which is automatically calculated by the plotting function.

12.3.2 Rich-club plots

Then plot the data using the function `plot_rich_norm`. A plot of the normalized rich club coefficients for each group and 2 example densities is shown in Figure 12.3.[1] The shaded region is based on the *rich-core* calculation; this option is selected if you provide a list of graph objects to the argument `g`. You can also choose to specify significance based on the regular P-values or FDR-adjusted P-values (the default), via the `fdr` function argument).

```
plot_rich_norm(rich.dt, facet.by='density', densities[11:12], g.list=g)
```

12.3.3 Rich-club attributes

The function `rich_club_attrs` will assign vertex- and edge-level attributes based on membership in the rich-club. The function requires a range of degrees in which $\phi_{norm} > 1$ was determined to be significant. See Table 12.1 for a list and description of the attributes assigned by the function.
An example plot is shown in Figure 12.4. This isn't the best solution, but clearly highlights the rich-club vertices. The code can easily be adjusted to increase the size of other vertices/edges.

[1] These probably don't look like the "familiar" plots from the literature; this is likely because I chose to generate a small number of random graphs.

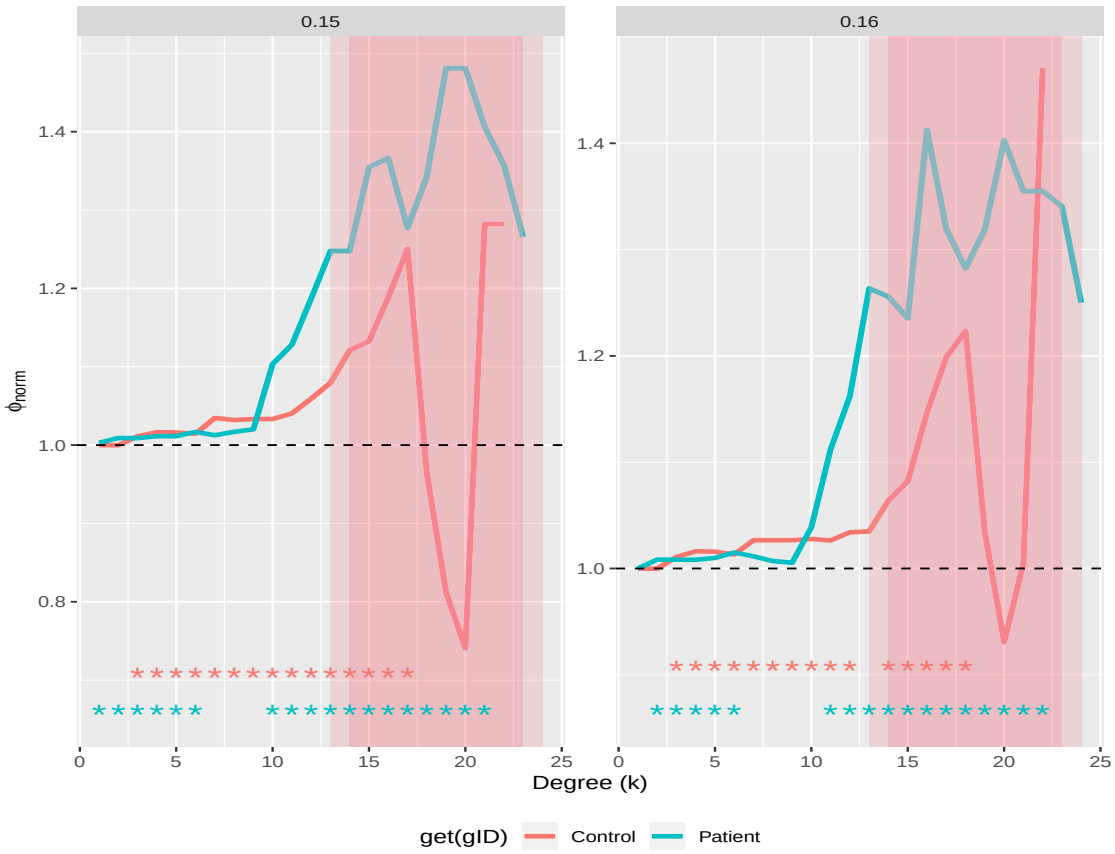

Figure 12.3: Normalized rich club coefficient vs. degree

```
g[[N]]$graphs[[1]] <- rich_club_attrs(g[[N]][1], c(rich_core(g[[N]][1])$k.r, 12))

# Alternatively, the degree range could use the following command, but in the
# current example, the number of random graphs was very low
#rich.dt[density == densities[N] & Group == grps[1] & p.fdr < 0.05, range(k)]

plot(g[[N]][1], vertex.label=NA, vertex.color='color.rich', edge.color='color.rich',
     edge.width='size.rich', vertex.size='size.rich', show.legend=TRUE)
```

12.4 Single-subject networks

To generate random graphs in the case of single-subject data, the code is largely the same as in Random graph generation. Just substitute `g.norm` for `g` (or whatever your graph list object happens to be named). The function `analysis_random_graphs` will return a `data.table` with the normalized rich club coefficients and P-values. The code in the following block may be necessary if you later want to plot these results.

```
rich.dt[, Group := as.factor(Group)]
rich.dt <- rich.dt[complete.cases(rich.dt)]
```

For single-subject data (e.g., DTI tractography or rs-fMRI), you will have to change the function call of `plot_rich_norm` to `facet.by='threshold'` (assuming this is how you generated the connectivity matrices

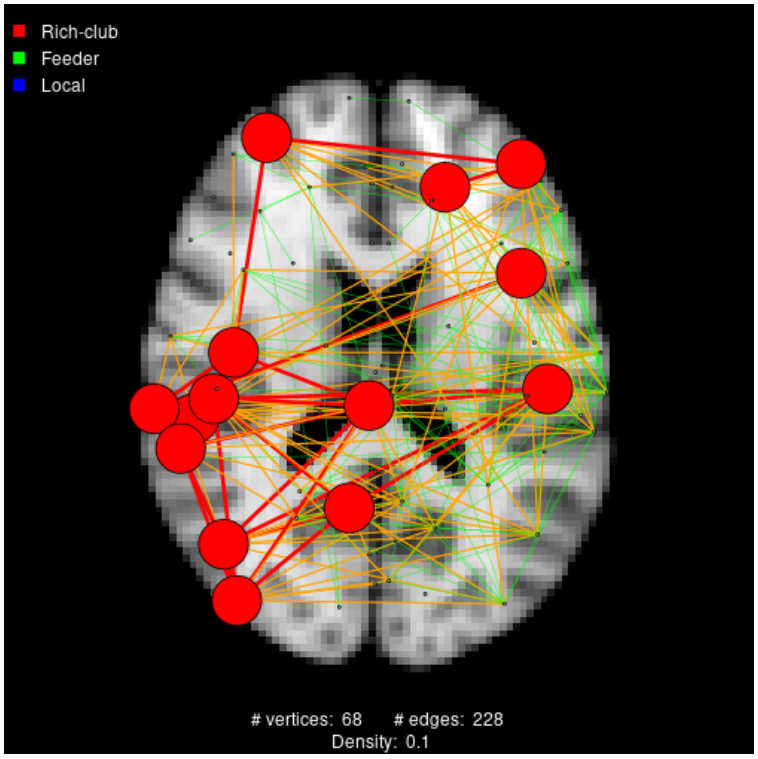

Figure 12.4: Rich-club attributes.

to begin with). Furthermore, the default behavior is to plot a *loess smoother*; if you would like to see an individual line for each subject, then you must specify `smooth=FALSE` .

```
plot_rich_norm(rich.dt, facet.by='threshold', thresholds[1:5])
```

13

Bootstrapping and permutation testing

13.1 Bootstrapping

In structural covariance analyses, since there is only one graph per group (at a given density)—as atlas-based brain regions typically are different sizes—it is not possible to directly compute the within-group variability of graph measures (e.g., *modularity*). In this case, *bootstrapping* is necessary. The function `brainGraph_boot` will perform bootstrap resampling in this case; the `boot` package does all of the resampling.(23)

13.1.1 Function arguments

```
args(brainGraph_boot)

## function (densities, resids, R = 1000, measure = c("mod", "E.global",
##     "Cp", "Lp", "assortativity", "strength", "mod.wt", "E.global.wt"),
##     conf = 0.95, .progress = getOption("bg.progress"), xfm.type = c("1/w",
##         "-log(w)", "1-w", "-log10(w/max(w))", "-log10(w/max(w)+1)"))
## NULL
```

densities The (numeric) vector of network densities (since the function creates networks of the same density for bootstrap samples).

resids The `brainGraph_resids` object output by `get.resid`.

R Integer; the number of bootstrap samples to generate (default: `1000` .

measure Character string indicating which *global* graph metric to test (default: `'mod'` ; i.e., modularity)

conf The confidence level; the default, `0.95` , returns 95% confidence intervals.

.progress Logical indicating whether or not to print a progress bar (default: `getOption, bg.progress` .

There are several graph-level metrics you can choose from; currently, the options are: *modularity (weighted and unweighted)*, *global efficiency (weighted and unweighted)*, *clustering coefficient*, *characteristic path length*, *(degree) assortativity*, and *mean graph strength*. Your data may also have an arbitrary number of groups (i.e., not just 2).

13.1.2 Return value

The returned object is of class `brainGraph_boot`, containing:

measure The global graph measure

densities The vector of densities

Group A character vector of the group names

conf The confidence level

boot A *list* (with length equal to the number of groups). Each element of the list is an object of class `boot`. See the help file for more information (by typing `help(boot)`).

13.1.3 Code example

In the following code block, I show how to estimate the standard error and confidence intervals for *modularity*.

```
# For modularity, the Louvain algorithm is used
kNumBoot <- 1e3
bootmod <- brainGraph_boot(densities, resids, R=kNumBoot, measure='mod',
                           .progress=FALSE)
```

The `summary` method prints some analysis-specific information, and then a `data.table` with the observed values, standard errors, and confidence intervals for each group and density tested.

```
summary(bootmod)

##
## ====================================
## Bootstrap analysis
## ====================================

## Graph metric:  Modularity
## Number of bootstrap samples generated:  1000
## 95 % confidence intervals
##
##       Group density Observed      se ci.low ci.high
##  1: Control    0.05   0.5621 0.05262 0.5112  0.7175
##  2: Control    0.06   0.5410 0.05273 0.4935  0.7002
##  3: Control    0.07   0.5292 0.05373 0.4906  0.7012
##  4: Control    0.08   0.4938 0.05408 0.4401  0.6521
##  5: Control    0.09   0.4696 0.05347 0.4107  0.6203
##  6: Control    0.10   0.4405 0.05359 0.3693  0.5793
##  7: Control    0.11   0.4166 0.05247 0.3397  0.5454
##  8: Control    0.12   0.3983 0.05187 0.3188  0.5221
##  9: Control    0.13   0.3803 0.05086 0.2982  0.4976
## 10: Control    0.14   0.3816 0.04987 0.3154  0.5109
## 11: Control    0.15   0.3664 0.04873 0.2991  0.4901
## 12: Control    0.16   0.3541 0.04737 0.2885  0.4742
## 13: Control    0.17   0.3477 0.04620 0.2888  0.4699
## 14: Control    0.18   0.3191 0.04501 0.2432  0.4196
## 15: Control    0.19   0.3119 0.04372 0.2404  0.4118
## 16: Control    0.20   0.2959 0.04254 0.2201  0.3869
## 17: Patient    0.05   0.5420 0.05245 0.4350  0.6406
```

```
## 18: Patient    0.06   0.5249 0.04938 0.4313  0.6249
## 19: Patient    0.07   0.5268 0.04577 0.4643  0.6438
## 20: Patient    0.08   0.5073 0.04340 0.4503  0.6204
## 21: Patient    0.09   0.4921 0.04187 0.4412  0.6053
## 22: Patient    0.10   0.4875 0.03993 0.4518  0.6083
## 23: Patient    0.11   0.4710 0.03834 0.4376  0.5879
## 24: Patient    0.12   0.4799 0.03681 0.4739  0.6182
## 25: Patient    0.13   0.4559 0.03530 0.4429  0.5813
## 26: Patient    0.14   0.4364 0.03465 0.4176  0.5534
## 27: Patient    0.15   0.4158 0.03395 0.3910  0.5241
## 28: Patient    0.16   0.3970 0.03233 0.3692  0.4959
## 29: Patient    0.17   0.3812 0.03139 0.3507  0.4737
## 30: Patient    0.18   0.3748 0.03065 0.3501  0.4702
## 31: Patient    0.19   0.3657 0.02972 0.3439  0.4604
## 32: Patient    0.20   0.3452 0.02906 0.3141  0.4280
##       Group density Observed      se ci.low ci.high
```

13.1.4 Plotting the results

The shaded regions in the top of Figure 13.1 are ±1 standard error, and in the bottom represent the 95% confidence region (calculated using the normal approximation). I use the function `grid.arrange` to plot both in the same plot device.

```
bootmod.p <- plot(bootmod)
p1 <- bootmod.p$se + theme(legend.position=c(1, 1), legend.justification=c(1, 1))
p2 <- bootmod.p$ci + theme(legend.position=c(1, 1), legend.justification=c(1, 1))
gridExtra::grid.arrange(p1, p2)
```

13.2 Permutation testing

Bootstrapping can give you an estimate of the *variability* of a group measure (e.g., modularity). In order to determine the *significance* of a between-group difference, you need to do permutation testing.

13.2.1 Function arguments

```
args(brainGraph_permute)

## function (densities, resids, N = 5000, perms = NULL, auc = FALSE,
##     level = c("graph", "vertex", "other"), measure = c("btwn.cent",
##         "coreness", "degree", "eccentricity", "clo.cent", "communicability",
##         "ev.cent", "lev.cent", "pagerank", "subg.cent", "E.local",
##         "E.nodal", "knn", "Lp", "transitivity", "vulnerability"),
##     .function = NULL)
## NULL
```

densities

resids

N The number of permutations. Default: `5000`

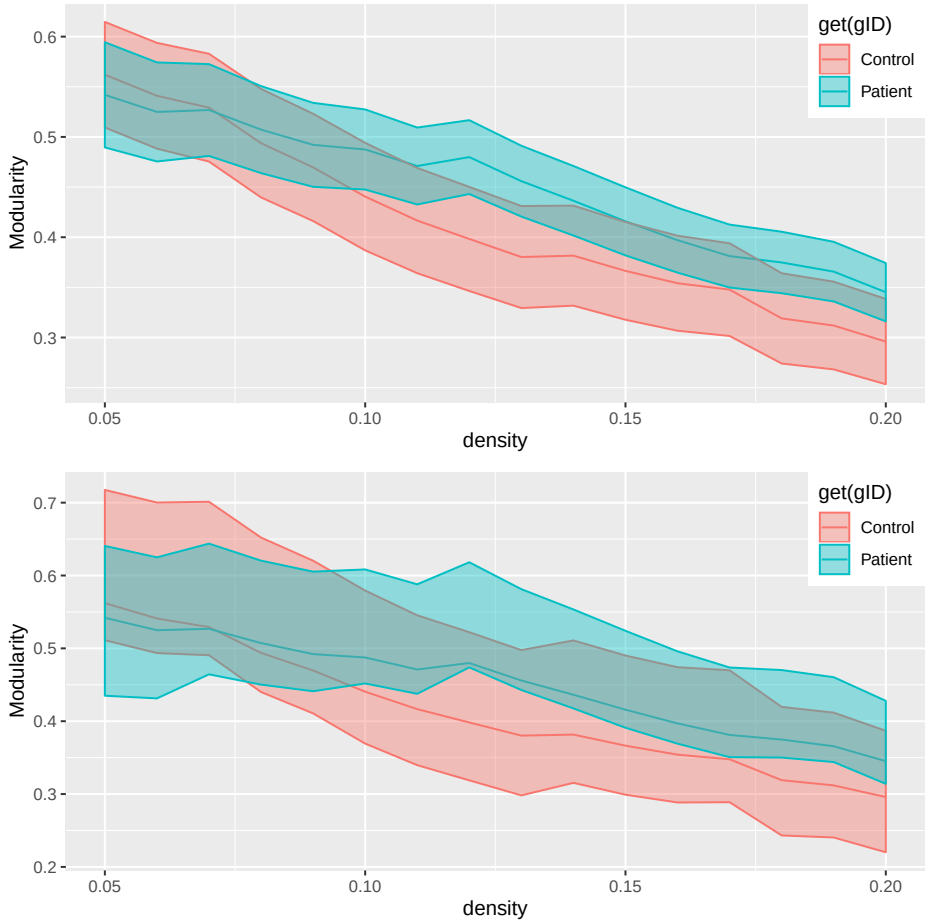

Figure 13.1: **Bootstrap analysis.** Modularity plotted across densities

perms A permutation matrix, if you would like to supply your own

auc Logical indicating whether or not to calculate differences in the *area-under-the-curve (AUC)* of the graph metrics. Default: `FALSE`

level Character string indicating which level the network measure is (either graph, vertex, or "other"). If `level='other'` , then you must supply a custom function via `.function` (see below)

measure Character string indicating the name of the network measure to calculate

.function A custom function if you would like to calculate permutation differences for a network measure that is not hard-coded.

13.2.2 Return value

The function returns an object of class `brainGraph_permute` , with elements:

atlas

auc

N

level

measure

densities

resids

DT A `data.table` with the permutation differences for each density (if `auc=FALSE`), and each region (there are multiple if `level='vertex'`).

obs.diff A `data.table` of the observed group difference. The number of rows equals the number of densities. If `level='vertex'` , the number of columns equals the number of regions.

Group Character vector of group names

13.2.3 Graph-level

Several graph-level measures are calculated: *modularity, clustering coefficient, average path length, assortativity (degree and lobe), asymmetry*, and *global efficiency*.

```
kNumPerms <- 1e3
myPerms <- permute::shuffleSet(n=nobs(resids), nset=kNumPerms)
perms.all <- brainGraph_permute(densities, resids, perms=myPerms,
                                level='graph')
```

For the `summary` method, if `level='graph'` , then you must choose which network measure to summarize. You additionally specify an `alternative` hypothesis, `alpha` level, and which P-value to use for determining significance (either the standard P-value, or FDR-adjusted). Here, I show the summary for `Lp` . In this case, the group difference was significant for only one density.

```
summary(perms.all, measure='Lp', alt='less')

##
## ====================================
## Permutation analysis
## ====================================

## # of permutations: 1,000
## Level:  graph
## Graph metric:  Characteristic path length
## Alternative hypothesis:    Control - Patient < 0
## Alpha:  0.05
##
##    densities region Lp.Control Lp.Patient obs.diff perm.diff 95% CI low
## 1:      0.05  graph      3.042      4.754  -1.7123  -0.22302    -1.4956
## 2:      0.06  graph      2.896      4.024  -1.1281  -0.20345    -0.9930
## 3:      0.07  graph      2.985      3.831  -0.8461  -0.16363    -0.8095
## 4:      0.09  graph      2.579      3.053  -0.4743  -0.04932    -0.4431
##    95% CI high       p  p.fdr
## 1:      2.1486 0.02697 0.1718
## 2:      1.8582 0.03297 0.1718
## 3:      1.7304 0.04296 0.1718
## 4:      0.6577 0.03996 0.1718
```

Plotting: graph-level

The `plot` method has the same arguments as the `summary` method, and returns a list with two `ggplot` objects:

1. A line plot of the *observed* graph-level measure across densities, with an asterisk added if $p < \alpha$; a blue asterisk is added if $\alpha < p < 0.10$ (i.e., a "trend" towards significance). This is shown in Figure 13.2.

2. A line plot of the observed group *difference* across densities. Also shown are dashed lines of the $(1-\alpha)\%$ confidence interval based on the permutation distribution (see Figure 13.3 and the following paragraph).

```
permPlot <- plot(perms.all, measure='Lp', alt='less')
print(permPlot[[1]])
```

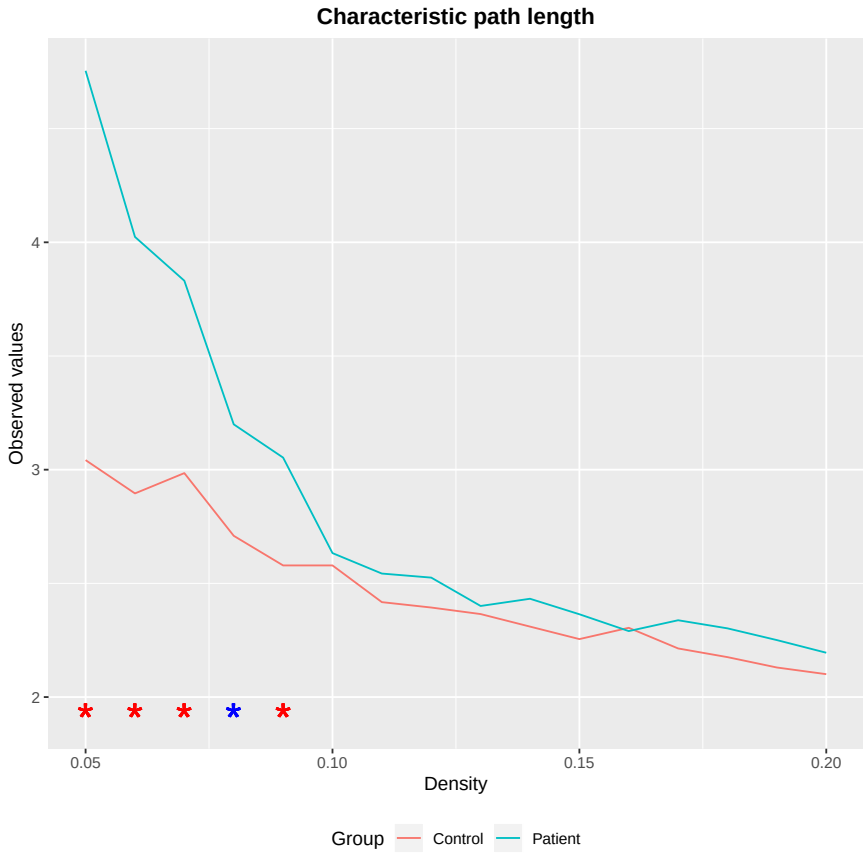

Figure 13.2: **Permutation testing, graph-level.** Observed avg. path length across densities

Instead of plotting the observed value for both groups, you can also plot the group *differences* along with a $(1-\alpha)\%$ confidence interval. In Figure 13.3, the red line indicates observed between-group differences, the central dashed line is the mean permutation difference, and the outer dashed lines are upper and lower bounds. In this example, I used a one-sided test, assuming group 1 would be lower than group 2.

```
print(permPlot[[2]])
```

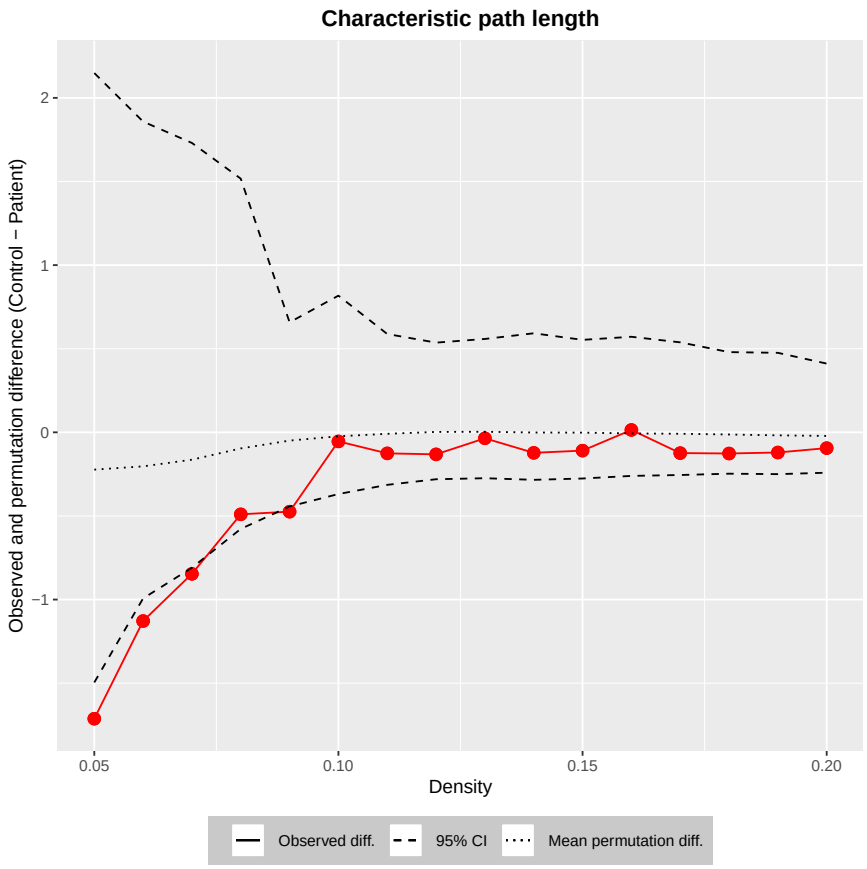

Figure 13.3: **Permutation testing, graph-level.** Observed and permuted differences in avg. path length across densities

13.2.4 Vertex measures

You can also do permutation testing for *vertex*-level measures (e.g., *betweenness centrality*) to look for group differences. See Wang et al. (122) for an example application.

Since a calculation is required at each vertex for each permutation and density, this may take considerably longer than in the previous section. Other vertex-level measures that are hard-coded are: *degree, k-nearest neighbor degree, nodal efficiency, transitivity*, and *vulnerability*.

```
perms.btwn <- brainGraph_permute(densities[N:(N+5)], resids,
                                 perms=myPerms, level='vertex')
```

The `summary` method has the same type of output as with graph-level measures:

```
summary(perms.btwn)

##
## ====================================
## Permutation analysis
## ====================================

## # of permutations: 1,000
## Level:  vertex
## Graph metric:  Betweenness centrality
```

```
## Alternative hypothesis:    Control - Patient != 0
## Alpha:  0.05
##
##     densities region btwn.cent.Control btwn.cent.Patient obs.diff
##  1:      0.10   lMTG            27.476            10.136   185.30
##  2:      0.10  lPARH             0.000             0.000    59.00
##  3:      0.10  lpTRI             0.000             0.000   192.84
##  4:      0.10   rFUS           114.362           194.472   295.43
##  5:      0.10   rINS            24.391             1.982   106.11
##  6:      0.10   rMOF             0.000            11.374   124.60
##  7:      0.10    rTT             0.000             0.000   116.50
##  8:      0.11   rSTG           299.871             5.214   301.48
##  9:      0.12   lMTG            16.904            22.490    95.09
## 10:      0.12  rpreC             2.387            31.713   162.09
## 11:      0.13   rIPL           108.111            12.938   205.15
## 12:      0.13  rPCUN            73.254             4.451   209.69
## 13:      0.13    rTT             0.000             7.167    32.04
## 14:      0.14   lENT             3.918             1.020    29.61
## 15:      0.14   rSFG           205.756           306.672  -238.63
## 16:      0.14   rSTG           217.811            14.467   266.79
## 17:      0.15   lLOF            70.983           249.158  -178.17
##     perm.diff 95% CI low 95% CI high        p  p.fdr
##  1:    8.2428     -53.54       99.48 0.002997 0.2038
##  2:    3.3085       0.00       59.00 0.027972 0.3804
##  3:   14.9290     -47.19      137.43 0.008991 0.2038
##  4:   -1.7625    -282.64      281.35 0.036963 0.3979
##  5:   13.9375     -66.62      111.92 0.040959 0.3979
##  6:    5.5597     -78.84      114.94 0.014985 0.2547
##  7:    3.4196     -10.62       43.72 0.005994 0.2038
##  8:    1.5148    -200.50      222.59 0.008991 0.6114
##  9:    8.0423     -44.79       82.35 0.012987 0.8831
## 10:    9.1818    -122.28      172.66 0.041958 0.9689
## 11:    8.1207    -144.67      172.07 0.017982 0.6114
## 12:   10.9281    -139.43      183.18 0.017982 0.6114
## 13:    3.7009     -15.06       56.03 0.043956 0.9510
## 14:    4.2643     -11.32       44.12 0.046953 0.8067
## 15:  -37.8341    -258.53      180.65 0.043956 0.8067
## 16:   -1.0917    -197.70      196.71 0.008991 0.6114
## 17:    0.4554    -146.28      170.13 0.030969 0.8967
```

Plotting: vertex-level

The `plot` method for vertex-level measures is slightly different. There is only one type of plot, a *barplot* of the permutation differences at those vertices for which $P_{sig} < \alpha$ (depending on the function arguments). The output is shown in Figure 13.4; the horizontal red line segments represent the *observed* between-group differences.

```
plot(perms.btwn)
```

13.2.5 Area-under-the-curve (AUC)

You can, alternatively, calculate the between-group difference in the *area-under-the-curve (AUC)* of a given graph- or vertex-level measure across all densities (see e.g., He et al. (48), Hosseini et al. (54)). The usage is

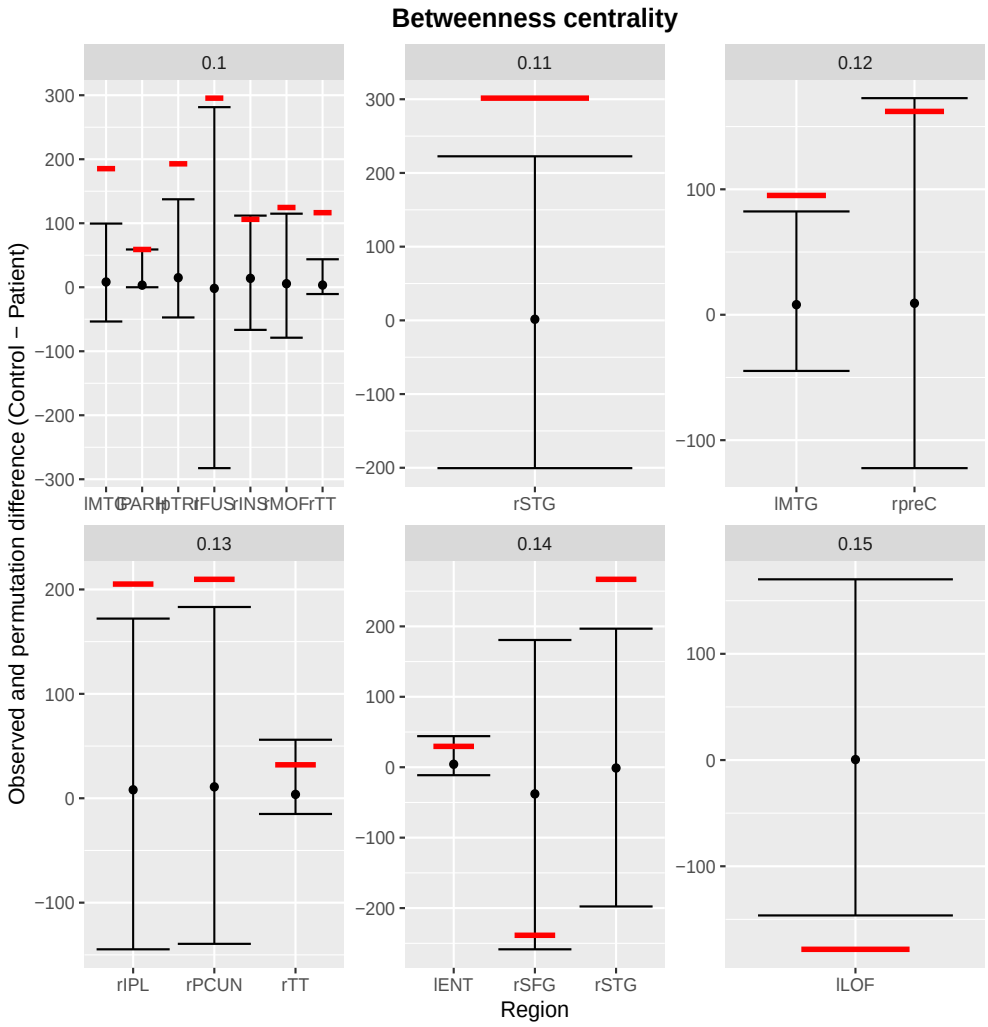

Figure 13.4: **Permutation testing, vertex-level.** Group differences in vertex betweenness centrality (observed difference in red)

identical to that of the previous sections, except you add `auc=TRUE` to the function call.

```
kNumPerms.auc <- 1e2
myPerms.auc <- permute::shuffleSet(n=nobs(resids), nset=kNumPerms.auc)
perms.all.auc <- brainGraph_permute(densities, resids, perms=myPerms.auc,
                                    level='graph', auc=TRUE)
```

Similarly, this can be done for vertex-level measures. The `plot` method will only work if `level='vertex'` .

```
perms.btwn.auc <- brainGraph_permute(densities, resids, perms=myPerms.auc,
                                     level='vertex', auc=TRUE)
plot(perms.btwn.auc, alt='less')
```

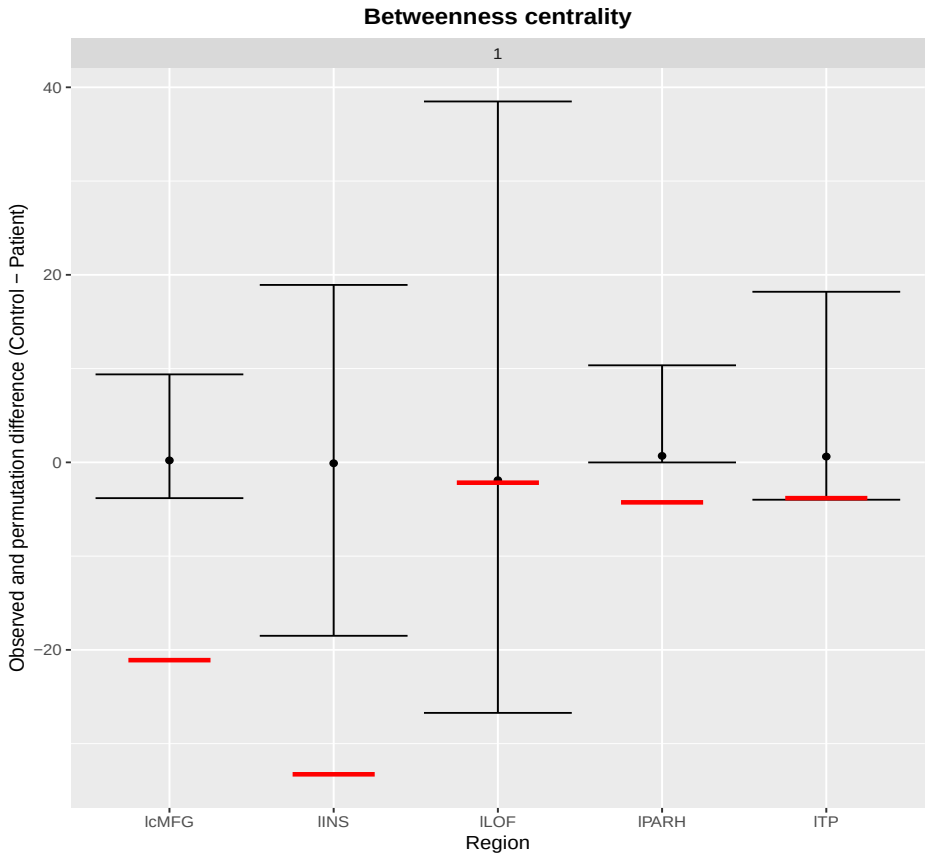

Figure 13.5: **Permutation testing, vertex-level.** Group differences in AUC of vertex betweenness centrality (observed difference in red)

13.2.6 Custom function

Perhaps the nicest feature of `brainGraph_permute` is one that mimics the `boot` function: you can choose to pass a custom function if you want to calculate permutations for a graph measure that I didn't hard-code. Your custom function must take 2 arguments:

g A list (of lists) of graph objects

densities The (numeric) vector of densities

The following example shows the code necessary to calculate the difference in AUC for *closeness centrality*. This example only does 100 permutations per density, which only takes 40 seconds (total, for all 36 densities).

```
# Custom function: determine difference in closeness centrality
clocent.diffs.perm <- function(g, densities) {
  meas <- lapply(g, function(x) t(sapply(x, function(y) centr_clo(y)$res)))
  meas.diff <- sapply(seq_along(V(g[[1]][[1]])), function(x)
      brainGraph:::auc_diff(densities, cbind(meas[[1]][, x], meas[[2]][, x])))
  tmp <- as.data.table(t(meas.diff))
  setnames(tmp, 1:ncol(tmp), V(g[[1]][[1]])$name)
  return(tmp)
}

perms.clocent <- brainGraph_permute(densities, resids, perms=myPerms[1:1e2, ],
```

```
                                    level='other', .function=clocent.diffs.perm,
                                    auc=TRUE)
```

Since `auc=TRUE`, the `summary` method only shows results from one "density":

```
summary(perms.clocent, alt='less')

##
## ====================================
## Permutation analysis
## ====================================

## # of permutations: 100
## Level:  vertex
## Graph metric:
## Area-under-the-curve (AUC) calculated across 16 densities:
##  0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.2
## Alternative hypothesis:    Control - Patient < 0
## Alpha:  0.05
##
## No significant results!
```

14

Further analysis

There is of course much more you can do with network data. In this Chapter I will briefly describe some of what is available in my package, and how to perform these analyses.

14.1 Robustness

14.1.1 Targeted attack

A targeted attack analysis can give further insight into a network's connectivity. In this analysis, you first sort vertices in order (from high to low) of a measure of "strength" of some kind. In my function, `robustness`, the choices for strength are either *degree* or *betweenness centrality*. Vertices are successively removed, and the maximal component size is calculated. (Edges may also be removed in order of *edge betweenness*). See Albert et al. (1), Bernhardt et al. (10) for more information.

Here, I will successively remove vertices ordered both by their degree and their betweenness centrality, and plot, for each group (at a single density), the ratio of the remaining maximal component size to the initial maximal component size. The result is shown in Figure 14.1.

```
attack.vertex.degree <- rbindlist(lapply(g[[N]][], robustness, 'vertex', 'degree'))
attack.vertex.btwn <- rbindlist(lapply(g[[N]][], robustness, 'vertex', 'btwn.cent'))
attack.vertex <- rbind(attack.vertex.degree, attack.vertex.btwn)

attack.edge <- rbindlist(lapply(g[[N]][], robustness, 'edge'))

mylabels <- sub('\\.', ', ', attack.vertex[, levels(interaction(Group, measure))])
attack <- ggplot(data=attack.vertex,
                 aes(x=removed.pct, y=comp.pct, col=interaction(Group, measure),
                     linetype=interaction(Group, measure))) +
  geom_line() +
  geom_abline(slope=-1, intercept=1, col='gray', lty=2) +
  scale_color_manual(name='Group & type',
                     labels=mylabels,
                     values=rep(c('red', 'cyan3'), times=2)) +
  scale_linetype_manual(name='Group & type',
                     labels=mylabels,
                     values=c(1, 1, 2, 2)) +
  theme(legend.position=c(1, 1), legend.justification=c(1, 1))
print(attack)
```

If you are interested in *vulnerability*, see the function `vulnerability`; also, each graph is given both global- and vertex-level attributes called `vulnerability` through `set_brainGraph_attr`.

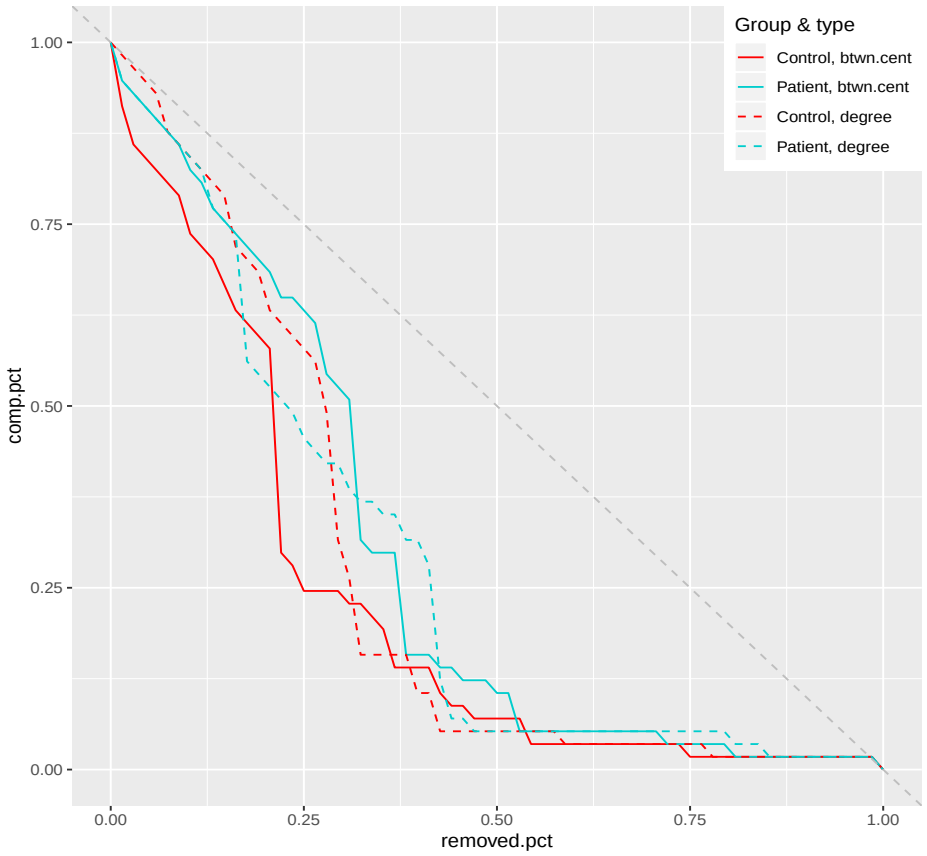

Figure 14.1: Maximal component size as a function of vertices removed.

14.1.2 Random failure

In a *random failure* analysis, you choose a vertex/edge at random, remove it, and calculate the maximum component size until all elements have been removed. This is repeated for many iterations (I only use 100 to save time).

Many graphs should be relatively robust to this kind of removal for both vertices and edges. The plot is shown in Figure 14.2, and includes the targeted attack plots.

```
failure.vertex <- rbindlist(lapply(g[[N]][], robustness, measure='random', N=1e2))
failure.edge <- rbindlist(lapply(g[[N]][], robustness, 'edge', 'random', 1e2))
failure.dt <- rbind(failure.edge, failure.vertex)
robustness.dt <- rbind(failure.dt, attack.vertex.btwn, attack.edge)
```

```
ggplot(robustness.dt, aes(x=removed.pct, y=comp.pct, col=Group)) +
  geom_line() +
  facet_wrap(~ type) +
  geom_abline(slope=-1, intercept=1, col='gray', lty=2) +
  theme(legend.position=c(0, 0), legend.justification=c(0, 0)) +
  labs(x='% edges/vertices removed', y='% of max. component remaining')
```

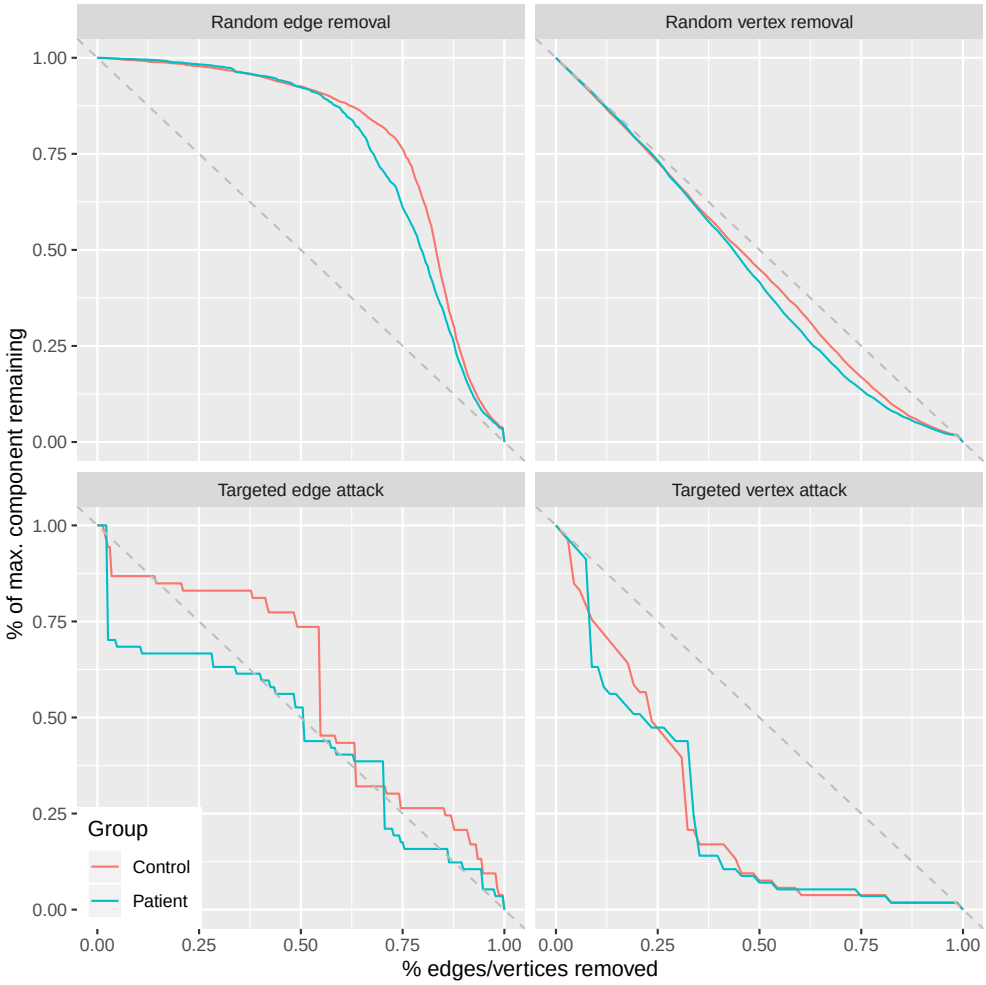

Figure 14.2: Maximal component size as a function of vertices (edges) removed.

14.2 Euclidean distance

It's very easy to get the spatial distance of edges. However, note that this distance is *Euclidean* (i.e., it doesn't follow a geodesic along the cortical surface), and it is based on coordinates that represent (roughly) the median coordinates of the atlas in MNI space.

You will notice that in my code, I use a *Kruskal-Wallis* test. This is an extension of the *Wilcoxon* test allowing for an arbitrary number of subject groups. If you only have 2 groups, then it is equivalent to a Wilcoxon test.

```
create_dists_dt <- function(g.list) {
  rbindlist(lapply(g.list[], function(x)
      data.table(density=x$density, dists=E(x)$dist, Group=x$Group)))
}

dists.dt <- rbindlist(lapply(g, create_dists_dt))
dists.dt[, Group := as.factor(Group)]
setkey(dists.dt, density, Group)
dists.dt[, med.dist := median(dists), by=.(density, Group)]
```

```
# Do a Kruskal-Wallis test at each density
dists.dt[, p := kruskal.test(dists ~ as.numeric(Group))$p.val, by=density]
p.dt <- dists.dt[, .(density=unique(density), p.fdr=p.adjust(unique(p), 'fdr'))]
dists.dt <- merge(dists.dt, p.dt, by='density')
dists.dt$sig <- ifelse(dists.dt$p.fdr < .05, '*', '')
dists.dt$trend <- with(dists.dt,
                       ifelse(p.fdr > .05 & p.fdr < .1, '*', ''))
```

The following code example and Figure 14.3 show group-wise histograms of edge distances for 2 densities. In this case, the control group tends to have more long-range connections.

```
# Plot at a couple densities
plot_dists_density <- function(dists.dt, densities) {
  dat <- dists.dt[density %in% densities]
  meandt <- dat[, .(avg=median(dists)), by=c('density', 'Group')]
  distplot <- ggplot(dists.dt[density %in% densities], aes(x=dists)) +
    geom_histogram(aes(y=..density.., fill=Group),
                   binwidth=10, alpha=0.4, position='dodge') +
    geom_vline(data=meandt, aes(xintercept=avg, col=Group), lty=2, size=0.5) +
    facet_grid(. ~ density) +
    geom_density(aes(col=Group), size=0.8) +
    xlab('Edge distance (mm)') +
    theme(legend.position=c(1, 1), legend.justification=c(1, 1),
          legend.background=element_rect(size=0.5))
    return(distplot)
}
print(plot_dists_density(dists.dt, densities[N:(N+1)]))
```

You can also plot the average edge distance across all densities, with a shaded region signifying the confidence interval (here I plot the 99th %ile). This is shown in Figure 14.4.

```
plot_dists_mean <- function(dists.dt) {
  ggplot(dists.dt, aes(x=density, y=dists, col=Group)) +
    stat_smooth(level=.99) +
    geom_text(aes(y=51, label=sig), col='red', size=7) +
    geom_text(aes(y=51, label=trend), col='blue', size=7) +
    ylab('Edge distance (mm)') +
    theme(legend.position=c(1, 0.25), legend.justification=c(1, 0),
        legend.background=element_rect(size=0.5))
}
print(plot_dists_mean(dists.dt))
```

14.3 Individual contributions

Saggar et al. (2015) recently introduced two methods for estimating the individual contributions of subjects to a group graph (100). I wrote two functions to calculate these measures: `aop` ("add-one-patient") and `loo` ("leave-one-out"). This is useful for structural covariance networks in which there is just a single graph for each group. For both functions, you can calculate either a "global" or a "regional" metric. For the global metrics, these can easily be merged with other data so you can correlate the individual contribution (`IC`) with some variable of interest (e.g., *full-scale IQ (FSIQ)*).

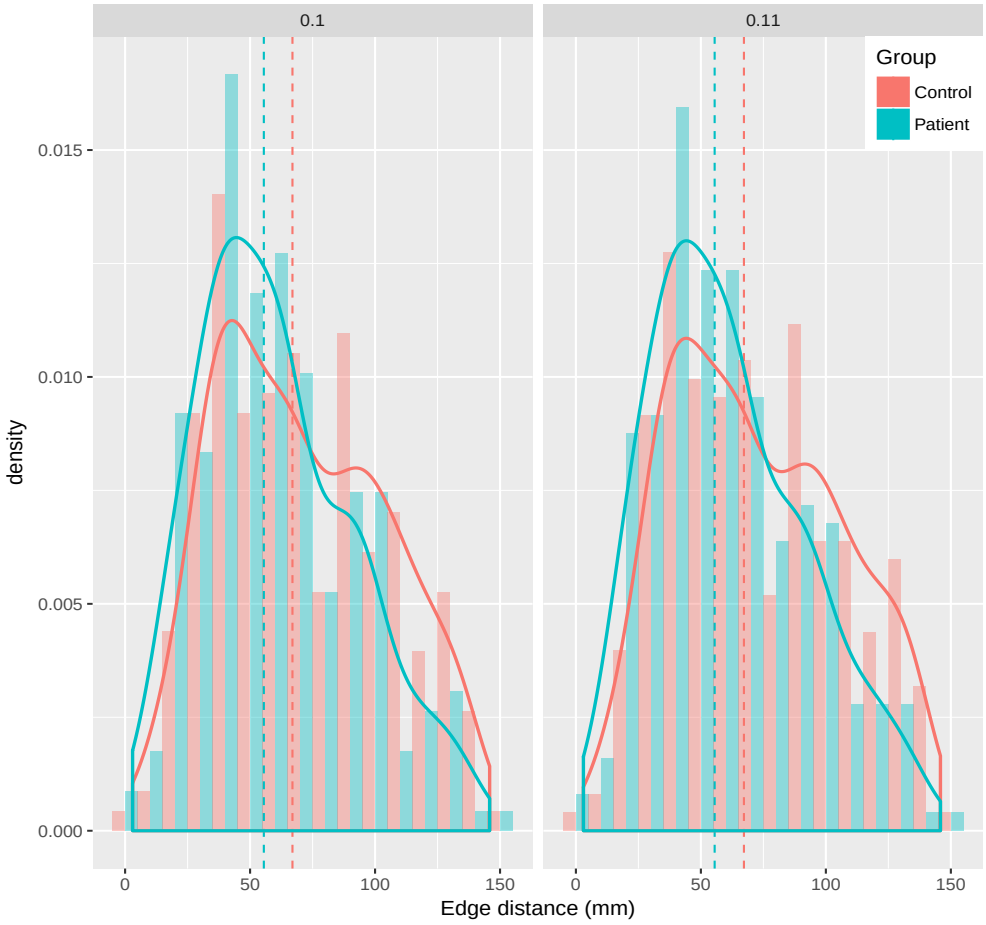

Figure 14.3: Edge distances (in mm)

14.3.1 Add one patient

With this method, a comparison is made between the correlation matrix of the entire control group and a correlation matrix of the control group *with a single patient added*. In the code sample below, it is assumed that the control group is *group 1* (this can be changed by the argument `control.value` . The code will work with any number of groups (i.e., if you have one control group and multiple patient groups).

```
IC.aop <- aop(resids, corrs)
RC.aop <- aop(resids, corrs, level='regional')
summary(RC.aop)

##
## Individual contributions
## -------------------------------------

## Method:  Add one patient
## Level:  regional
##
## Number of outliers per region: (sorted in descending order)
##    rLING     rMOF    rpTRI     lFUS   lparaC     lSPL      lTP    rPARH
##        8        8        7        6        6        6        6        6
##     rPCC     rSFG     lCUN     lIPL     liCC    lLING    lpreC    lrACC
```

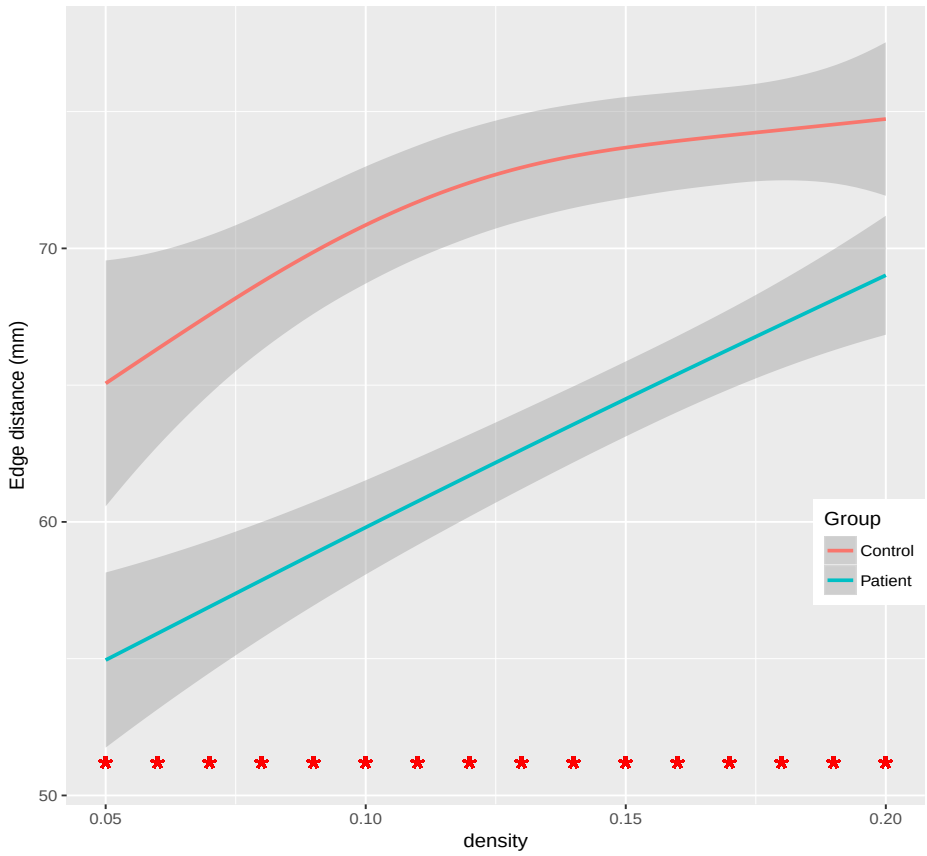

Figure 14.4: Average edge distances (in mm) with 99% confidence intervals

```
##        6        6        5        5        5        5        5        5
##     lSFG     rMTG   rparaC    rpORB   rpostC    rpreC    rPCUN      rFP
##        5        5        5        5        5        5        5        5
##     lITG     lLOF     lMOF    lpORB    lpTRI     lPCC    lPCUN    lrMFG
##        4        4        4        4        4        4        4        4
##      lFP      lTT    rcACC    rcMFG     rFUS     rIPL     rITG     rLOG
##        4        4        4        4        4        4        4        4
##     rLOF rperiCAL    rrMFG     rSTG      rTP    lcACC    lcMFG     lENT
##        4        4        4        4        4        3        3        3
##     lMTG lperiCAL   lpostC    lSMAR     lINS    rBSTS     rENT   rpOPER
##        3        3        3        3        3        3        3        3
##    rrACC      rTT     rINS    lBSTS    lPARH   lpOPER     lSTG     rCUN
##        3        3        3        2        2        2        2        2
##     riCC     rSPL    rSMAR     lLOG
##        2        2        2        1
##
##       Group region    avg  stdev      se
##  1: Patient  lBSTS 0.9419 0.8990 0.09373
##  2: Patient  lcACC 0.9684 0.6882 0.07175
##  3: Patient  lcMFG 1.0546 0.9803 0.10221
##  4: Patient   lCUN 0.8047 0.6834 0.07124
##  5: Patient   lENT 0.9493 0.7151 0.07456
## ---
```

```
## 64: Patient  rSMAR 0.7490 0.5057 0.05272
## 65: Patient    rFP 0.8888 0.6171 0.06434
## 66: Patient    rTP 0.8278 0.5661 0.05902
## 67: Patient    rTT 0.8771 0.5938 0.06191
## 68: Patient   rINS 0.8708 0.7015 0.07314
```

We can also plot the regional contribution for the group using the `plot` method (not shown, as it is almost the same as Figure 14.5).

```
plot(RC.aop)
```

14.3.2 Leave one out

With this method, a comparison is made between the correlation matrix of the entire group, and a correlation matrix of the group *excluding a single subject*. Function use is simple:

```
IC.loo <- loo(resids, corrs)
RC.loo <- loo(resids, corrs, 'regional')
summary(IC.loo)

##
## Individual contributions
## ------------------------------------

## Method:  Leave one out
## Level:  global
##
## "Outliers"
##      Group Study.ID       IC
## 1: Control      131 0.015283
## 2: Control      103 0.012979
## 3: Control        4 0.011464
## 4: Patient      108 0.014513
## 5: Patient       47 0.009209
```

We can also plot the regional contribution for the groups using the `plot` method (see Figure 14.5):

```
plot(RC.loo)
```

And a plot of the *individual* contributions is shown in Figure 14.6.

```
plot(IC.loo)
```

14.3.3 Plot types

When plotting data for *regional* contributions, you can now subset by *region*. Furthermore, you can choose from 3 plot types:

mean The default, which plots a shaded area of values within 2 standard errors of the mean

smooth Plot the result of applying a *loess* smoother to the data. This is much less useful, particularly when plotting a large number of regions

boxplot Plot the data as boxplots. This will also be a very "busy" plot if there are a lot of regions.

Figure 14.5: Regional contribution; leave-one-out

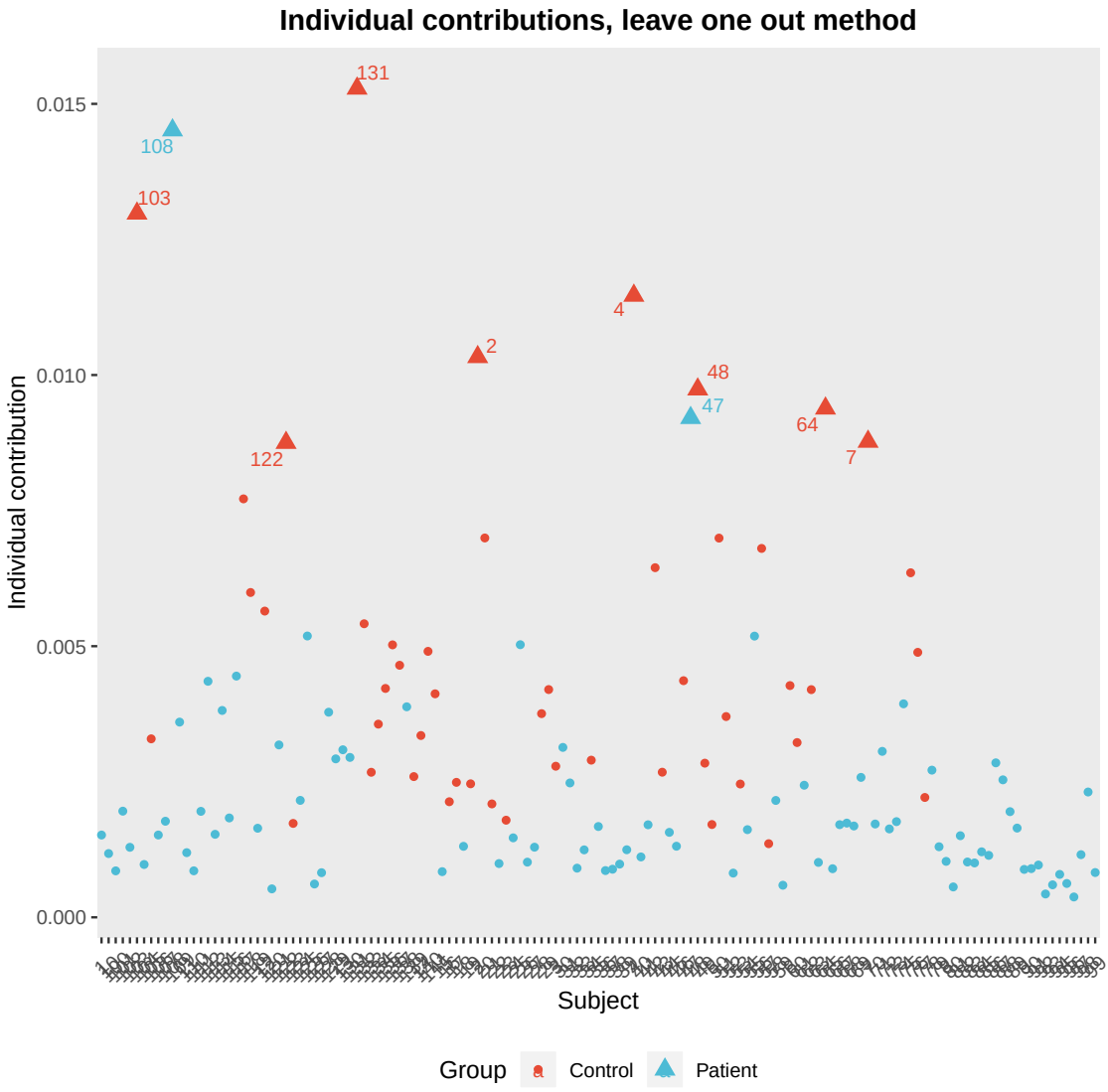

Figure 14.6: Individual contribution; leave-one-out

Part VI

Visualization

15

GUI and other plotting functionality

15.1 The GUI

The purpose of the `brainGraph` GUI is to quickly and flexibly compare graphs (up to two at a time). The function to open it is `plot_brainGraph_gui`. See Figure 15.1 for a screenshot.

The most obvious feature of the GUI is that there are 1–2 plotting sections for showing 1–2 graphs (of subjects, groups, etc.) side-by-side. If you have a single graph you would like to inspect, you can enter the same graph object twice to look at different densities, orientations, etc.

The # of vertices, # of edges, and the graph's density are printed at the bottom of each figure window if "Display subtitle" is checked. At the top of each figure window, the graph's `name` attribute is printed (in this case, group) if "Display main title" is checked.

15.1.1 Orientation

You can choose one of: *axial*, *sagittal (L)*, *sagittal (R)*, or *circular*. The sagittal plots necessarily show only *intra*-hemispheric edges. If you choose *circular*, there is a slider widget beneath each plot that controls the edge curvature.

15.1.2 Hemi/Edges

This option lets you show a subset of edges based on vertex classification/grouping. The default is to show all edges. Other options are:

- Individual hemispheres
- *Inter*-hemispheric edges only
- Edges between *homologous* regions
- Inter- and intra-*community* edges
- Inter- and intra-*lobar* edges
- Inter- and intra-*network* edges; these refer to the *network* attribute for the `power264`, `gordon333`, and `dosenbach160`
- Inter- and intra-*area* edges; here, `area` is an attribute of the `hcp_mmp1.0` atlas
- Inter- and intra-*Yeo 7* or *Yeo 17* edges; this is only applicable for the `brainnetome` atlas

15.1.3 Annotations

There are 4 checkboxes that can be (de-)selected to annotate the plot areas:

- Main title
- Subtitle
- Vertex labels
- Legend

There is also a "combo box" for changing vertex colors. The options are the same as those listed in the previous sections, in addition to `class` (for the *Destrieux* atlases) and *connected components*. There is no legend if vertex colors are based on communities or components. The numbers in the legend box represent the # of vertices present out of the total # of vertices for that lobe/network/etc.

15.1.4 Vertex "decorations"

- *Vertex labels*: you can show or hide the labels. The label placement and text color isn't very "smart", but I have attempted to code it such that labels won't be placed on top of one another. Showing labels would not be feasible for atlases with long vertex names (e.g., the *Destrieux* atlas).
- *Vertex size*: vertices will be scaled to reflect a number of vertex attributes (e.g., degree, betweenness centrality, etc.). The dropdown menu lists them all. Unfortunately, the same scaling is used for most of the vertex measures; the default I chose is to be between 0 and 15. Vertices with size larger than 15 (which is unitless) can end up "blocking" other vertices. For large graphs, it would be too cluttered to be useful.
- There is also an entry for `Other`, which is useful if your graph has vertex attributes I haven't hard-coded. For example, if you have a *P-value* attribute for each vertex, you can type that in the `Other` entry.
- You can remove vertices with values below a certain minimum (see `Min. 1` and `Min. 2`).
- Finally, there are text boxes for writing a simple logical equation. For example, to keep vertices with *degree* above 10 *AND Frontal lobe regions*, you would write `degree > 10 & lobe == 'Frontal'` .

15.1.5 Edge "decorations"

- *Edge width*: edge widths will be scaled to reflect edge betweenness, edge distance (Euclidean), or edge weight (if the graph is weighted); there is also an entry for `Other` (as with vertices) for a custom edge attribute.
- You can specify both minimum and maximum values, below/above which all edges will be removed. These values can be different for the two displayed graphs.
- *Edge color* is tied to vertex color; edges are colored gray if they are connecting vertices with different memberships (community, lobe, etc.).

15.1.6 Vertex groups

Under the *Plot* menu item, you can choose to plot based on combinations of vertex groups. Most of these options are the same as those for *Hemi/edges* above. For example, if you want to see the Frontal and Parietal lobes only, you would choose "Entire graph" and select the lobes at the bottom. To select multiple lobes, you can press `ctrl` and click, or hold `shift` and use the arrow keys.
If you choose "Neighborhood graph", you can plot the *neighborhoods* of combinations of vertices. An example screenshot is in Figure 15.2; in this case, I am looking at the union of neighborhoods of 3 vertices. The main title of the plot lists the vertex names, which are colored in yellow (default) for convenience.
Figure 15.3 shows an example when choosing the 2 largest (*red* is for the largest, *green* for the second-largest) communities. The numbers in the scrolled window at the bottom are ordered from largest to smallest.
Figure 15.4 shows an example when viewing specific Yeo-17 networks.

15.1.7 Keyboard shortcuts

There are a number of keyboard shortcuts available. Note that some shortcuts are accessed with the *Ctrl* key, and others with the *Alt* key.

Alt+f Opens the *File* dropdown menu

Ctrl+o Select new graph objects (same as clicking *Pick new graphs*)

Ctrl+1 Save figure 1 (left)

Ctrl+2 Save figure 2 (right)

Ctrl+q Exit (Quit) the GUI

Alt+p Opens the *Plot* dropdown menu

Ctrl+e Plot Entire graphs (optionally selecting combinations of lobes)

Ctrl+n Plot Neighborhoods

Ctrl+c Plot Communities

Ctrl+f Plot Functional networks (i.e., the `network` attribute)

Ctrl+a Plot Cortical Areas (`hcp_mmp1.0` only)

Ctrl+y Plot Yeo-7 networks (`brainnetome` only)

Ctrl+7 Plot Yeo-17 networks (`brainnetome` only)

Ctrl+g Plot Gyri (`brainnetome` only)

Alt+o Same as clicking *Ok*

Alt+n Same as clicking *Pick new graphs*

Alt+m Toggle the *Display main title* checkbox

Alt+s Toggle the *Display subtitle* checkbox

Alt+l Toggle the *Display vertex labels* checkbox

Alt+g Toggle the *Show legend* checkbox

Alt+d Toggle the *Show diameter* checkbox

Alt+e Toggle the *Show edge differences* checkbox

15.2 Other plotting

15.2.1 Adjacency matrix plots

It is very easy to plot the adjacency matrices using the `plot` method for objects of type `corr_mats`. I haven't found these to be particularly useful, except when viewing the vertices in a specific order (e.g., by community membership). By default, vertices will be ordered by *lobe*. You can also plot the *raw* correlation (adjacency) matrices by passing `type='raw'` in the function call. Figures 15.5 and 15.6 show them clustered into communities, and Figures 15.7 and 15.8 show them clustered into lobes (zoom in to read the axis labels).

```
matplot.comm <- plot(corrs, thresh.num=10, order.by='comm', graphs=g[[10]], grp.names='')
matplot.lobe <- plot(corrs, thresh.num=10)

# To plot both in the same device, uncomment the following line
#grid.arrange(grobs=matplot.comm, ncol=2)
print(matplot.comm[[1]])
```

```
print(matplot.comm[[2]])
```

```
print(matplot.lobe[[1]])
```

```
print(matplot.lobe[[2]])
```

15.2.2 Plot global graph measures

Global graph measures can be plotted with `plot_global` (see Figure 15.9). There are several options to control the output. You can include a dashed vertical line at a density/threshold of interest (e.g., if your main analysis focuses on one density, but you want to report results for multiple in a compact way). You can also include data from a permutation analysis (i.e., `brainGraph_permute`) to add asterisks for significance. Currently, it will only plot red asterisks for $P < 0.05$ and blue for $0.05 \leq P < 0.10$. You also need to provide a character vector of the *alternative hypotheses* in the order of the columns of the permutation data table (excluding `density`). Finally, there is an option to change the facet label names (e.g., from `Cp` to `Clustering coeff.`).

```
# mod, Cp, Lp, assortativity, E.global, assortativity.lobe, asymm
alt <- c('less', 'two.sided', 'less', 'less', 'two.sided', 'less', 'less')
print(plot_global(g, vline=densities[N], exclude='assort.lobe.hemi',
                  perms=perms.all, alt=alt))
```

15.2.3 Save a three-panel plot of the brain graphs

The function `plot_brainGraph_multi` will save a *PNG* file of the left sagittal, axial, and right sagittal views for one or more groups. I recommend this only for group-level graphs, or for subject-level graphs with a small number of subjects (up to ≈ 5). You have the option of specifying conditions for subsetting the graphs (via the `subgraph` argument): you can specify a single condition to be applied to all graphs, or multiple conditions. If you would like to have a single graph and multiple conditions in the same figure, then "repeat" the graph in the input list (see Examples on the function's help page). An example is shown in Figure 15.10. To control the size of the panel titles, supply the argument `cex.main` ; values greater than 1 will increase the size. All other arguments accepted by `plot.brainGraph` can be used here.

```
plot_brainGraph_multi(g, subgraph='coreness > 5', filename='kcore5.png',
  main='k-core 5', vertex.color='color.lobe', edge.color='color.lobe')
```

15.2.4 Save a single view for multiple brain graphs

The function `slicer` (similar to the `FSL` program of the same name) will save a *PNG* file of a single view (axial, sagittal) for multiple graphs. For example, you can quickly plot for all subjects in a `brainGraphList` object. The same function arguments to the `plot.brainGraph` method apply here. Additionally, you must supply one (or both) of the `nrows` and/or `ncols` argument(s).
An example is shown in Figure 15.11.

```
slicer(g.glm[1:15, g=1, drop=FALSE], nrows=3, filename='slicerExample.png',
       vertex.color='color.comm.wt', edge.color='color.comm.wt',
       vertex.label=NA, subtitle=NULL)
```

15.2.5 Plot vertex-level measures

You can use the function `plot_vertex_measures` to plot a series of boxplots for a single vertex-level measure (e.g., *betweenness centrality*), grouped by *lobe* (the default), *network* (in the `dosenbach160` atlas), *hemi*, etc. This requires having a "long" dataset (see Long and wide data tables). An example is shown in Figure 15.12.

```
plot_vertex_measures(g[[N]], facet.by='lobe',
                     'btwn.cent', show.points=TRUE,
                     ylabel='Betweenness centrality')
```

15.2.6 Plot group-wise volumetric data for ROI's

It is very easy to plot group-wise volumetric data either in the form of histograms or violin plots. The function `plot_volumetric` will do this.You may also choose an integer vector as the second argument; in that case, the regions with those indices will be determined by the ordering in the input. You can choose as many regions at a time as you like, but more than 9 at a time might get cluttered.

15.2.7 Save a list of graph plots

The `plot` method for `brainGraphList` objects will save a series of graph plots to a multi-page *PDF* file. This could be a quick way to "scroll" through all of the graphs at a given threshold/density to check for potential outliers/issues.[1] There is an option to highlight edge differences between subsequent graphs, and to color the vertices (either all the same color (default), or by lobe/community membership). You may also pass arguments that will go to `plot.brainGraph`, such as `vertex.label=NA` to omit vertex labels. The basic command is:

```
plot(g[[1]], filename.base='group1_dk', diffs=TRUE)
```

There is also an option to view only a subgraph of vertices. The `subgraph` argument accepts logical expressions joined by `&` and/or by `|` for multiple conditions. For example, if you only want to plot vertices with degree greater than 10 and nodal efficiency greater than 0.5, you type:

```
plot(g[[1]], 'group1', subgraph='degree > 10 & E.nodal > 0.5')
```

After this, you can convert this series of *PNG*'s to either a *GIF* or to a video file. Converting to a *GIF* is done using `ImageMagick`, a powerful command-line utility for general image processing. You may then view the *GIF* in any image viewer, or, on Linux, `gifview` lets you step through each frame. Converting to a video will require a tool called `ffmpeg` (available on Linux).

Convert to gif

```
# The delay argument means show each frame for 0.5 seconds (i.e. 50/100)
convert -delay 50 -loop 0 group1.pdf animation.gif

# Each frame will be displayed for 0.5 seconds
#TODO: not sure if this would work for a single PDF
ffmpeg -framerate 2 -i group1_dk_%03d.png -c:v libx264 -r 30 \
    -pix_fmt yuv420p out.mp4
```

15.2.8 Other visualization tools

There are, of course, other visualization tools that are certainly more polished than this one. `igraph` can write graph objects that will work for several, e.g., Pajek. For a brain-specific tool, `write_brainnet` will save the `.node` and `.edge` files necessary for visualization with *BrainNet Viewer* (130). You can choose to color vertices by *community*, *lobe*, or *connected component* membership (or any other grouping/classification, provided it is a vertex attribute). You can scale vertex size by any vertex attribute the graph contains (e.g., *degree*, *betweenness centrality*, etc.). Finally, you can choose an edge attribute to write out a weighted adjacency matrix (e.g., *t.stat* from the `NBS` function). This tool requires `Matlab`, and while it is relatively slow, produces great figures.

```
write_brainnet(g[[1]][[N]], vcolor='comm', vsize='degree',
               edge.wt='t.stat', file.prefix='group1')
```

[1] I plan on including some sort of outlier detection in the future.

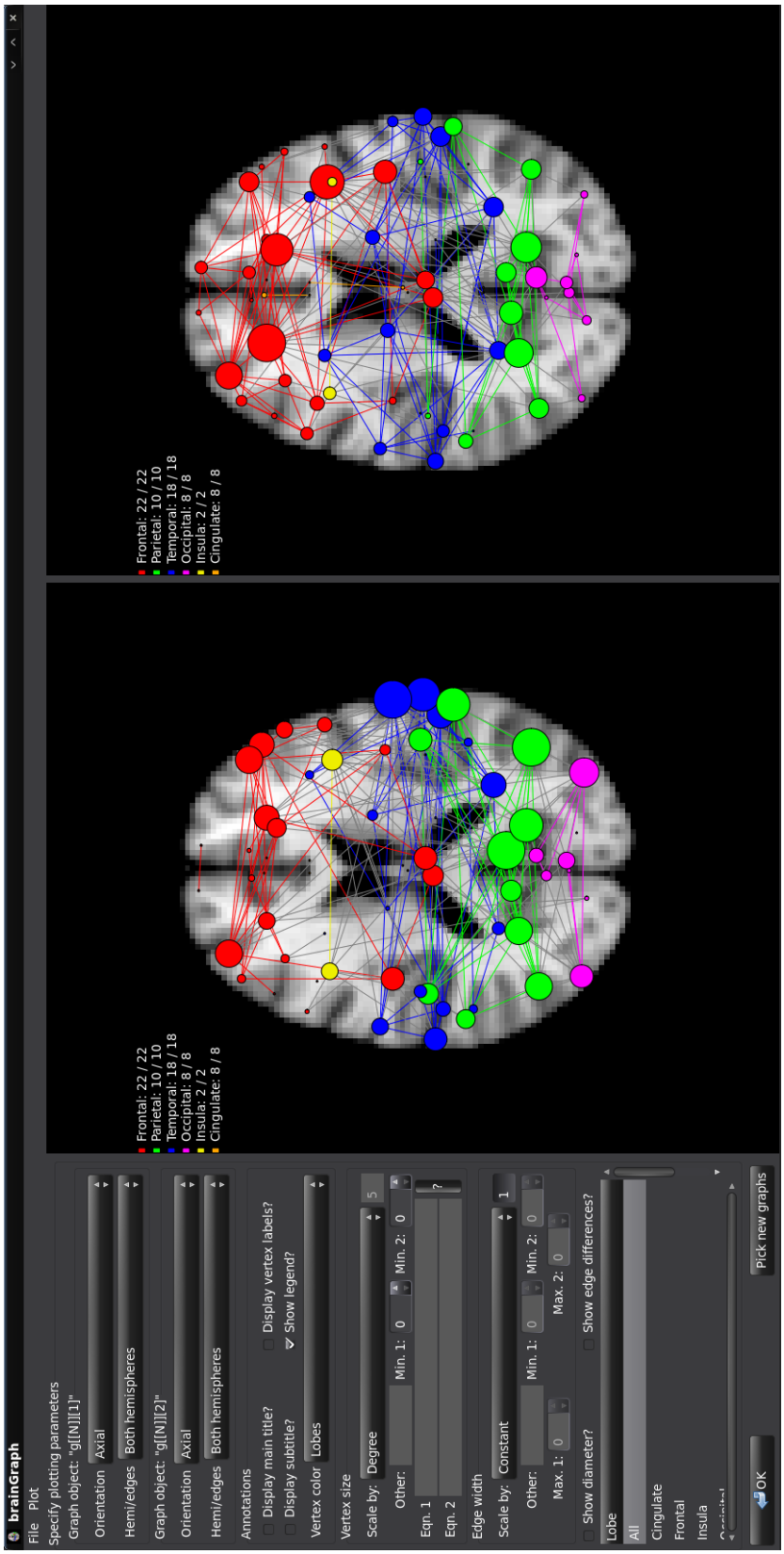

Figure 15.1: The `brainGraph` plotting GUI.

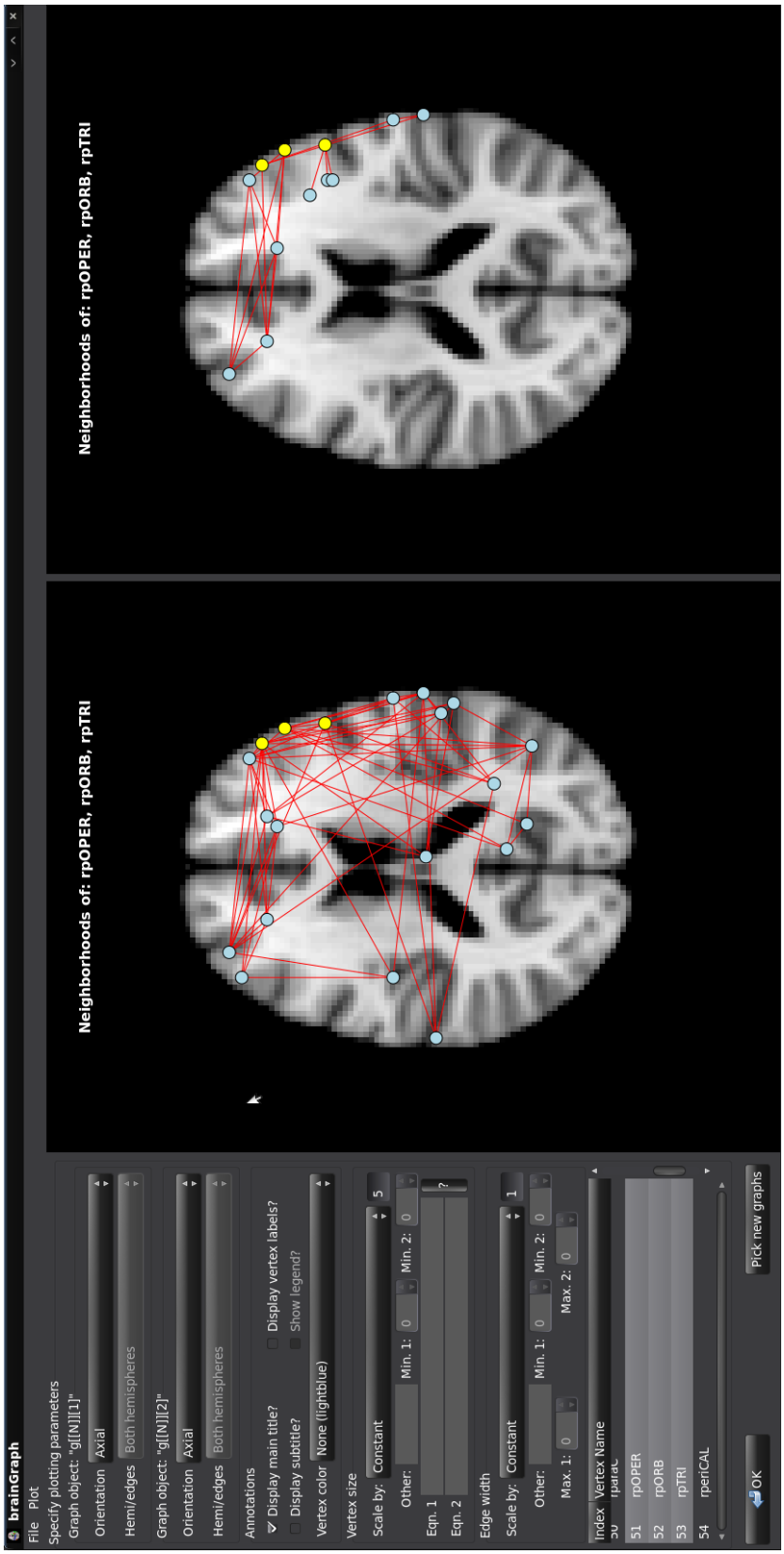

Figure 15.2: The `brainGraph` plotting GUI; neighborhoods.

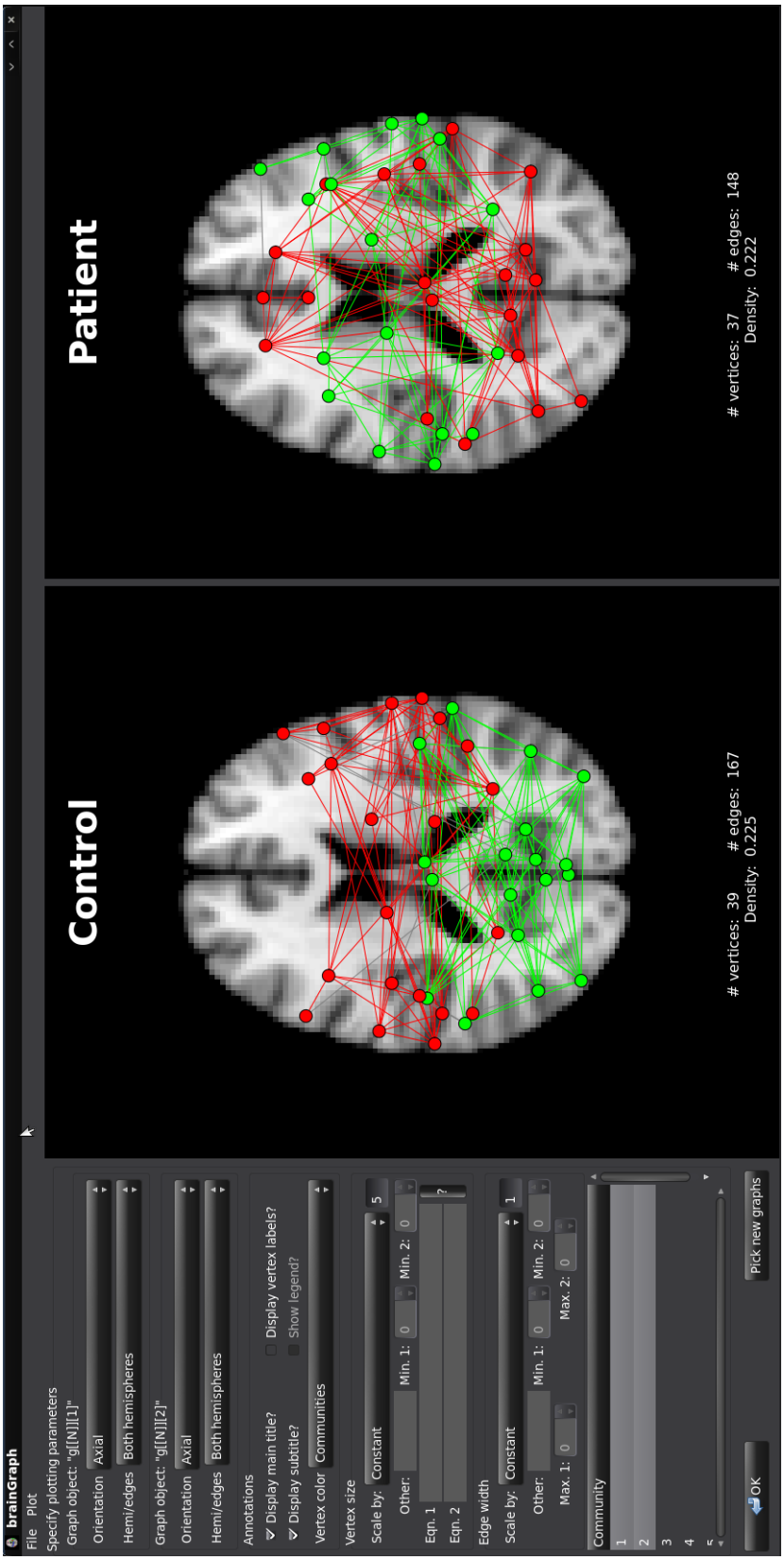

Figure 15.3: The `brainGraph` plotting GUI; communities.

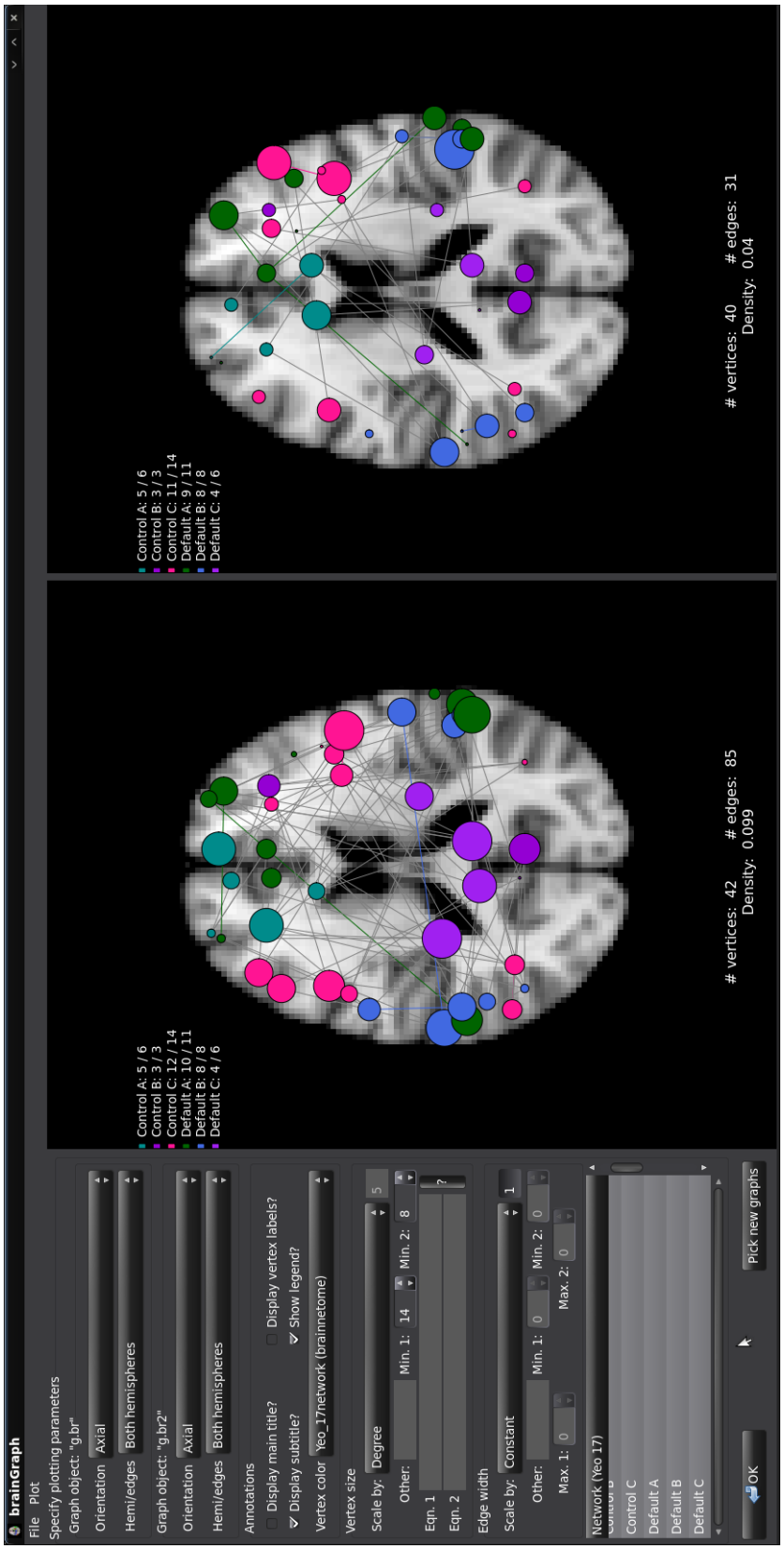

Figure 15.4: The `brainGraph` plotting GUI; Yeo-17 networks.

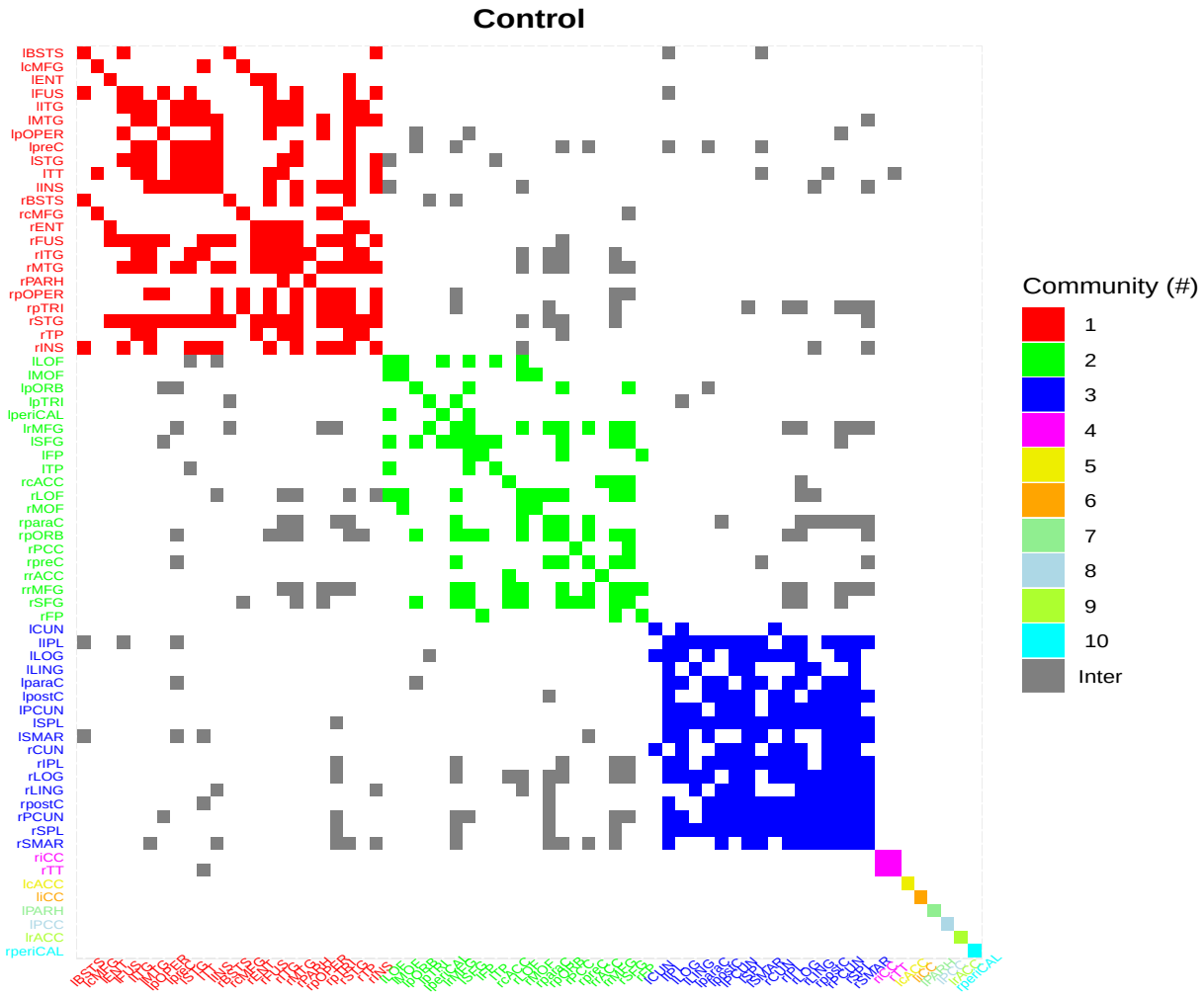

Figure 15.5: Adjacency matrix, group 1.

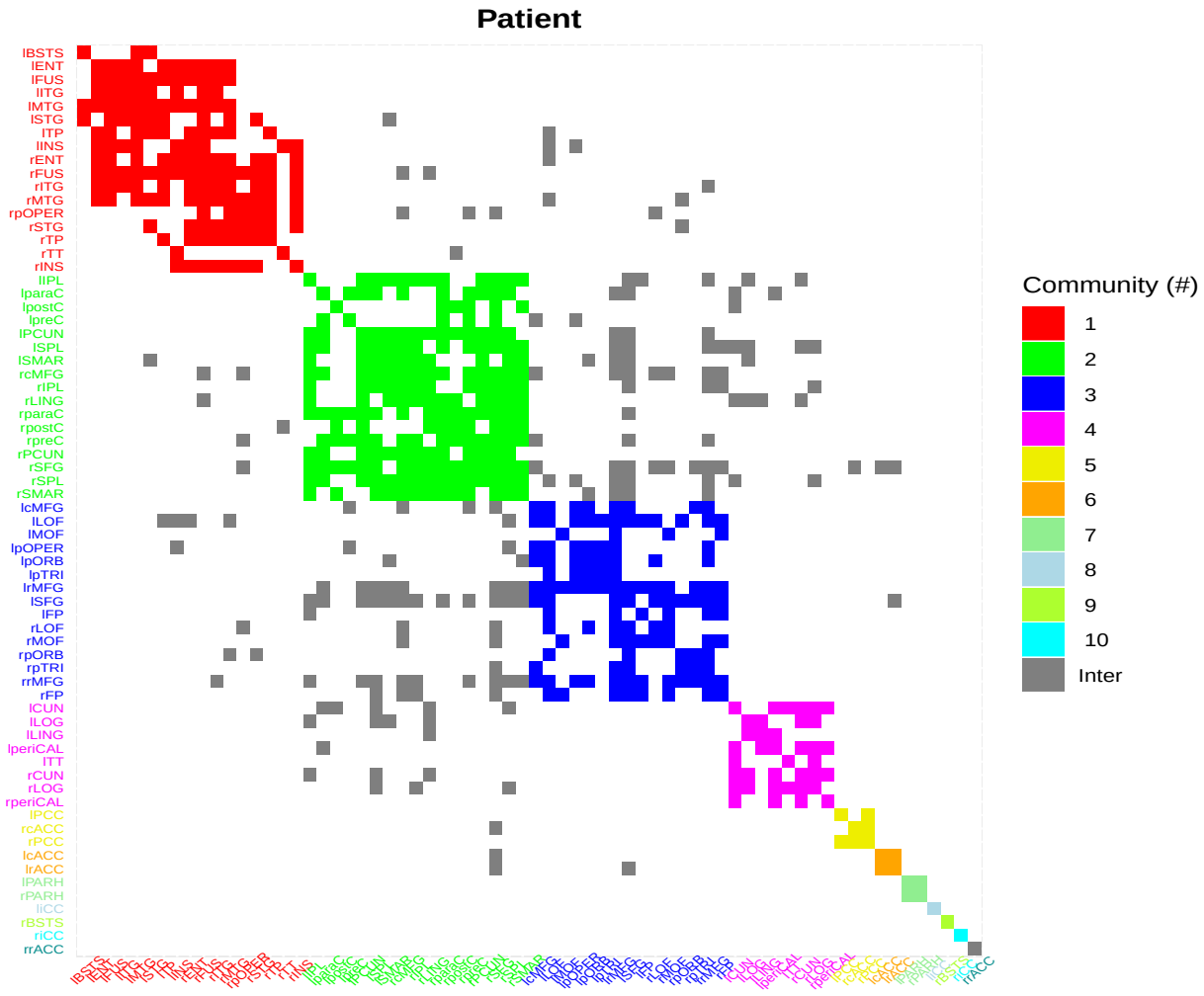

Figure 15.6: Adjacency matrix, group 2.

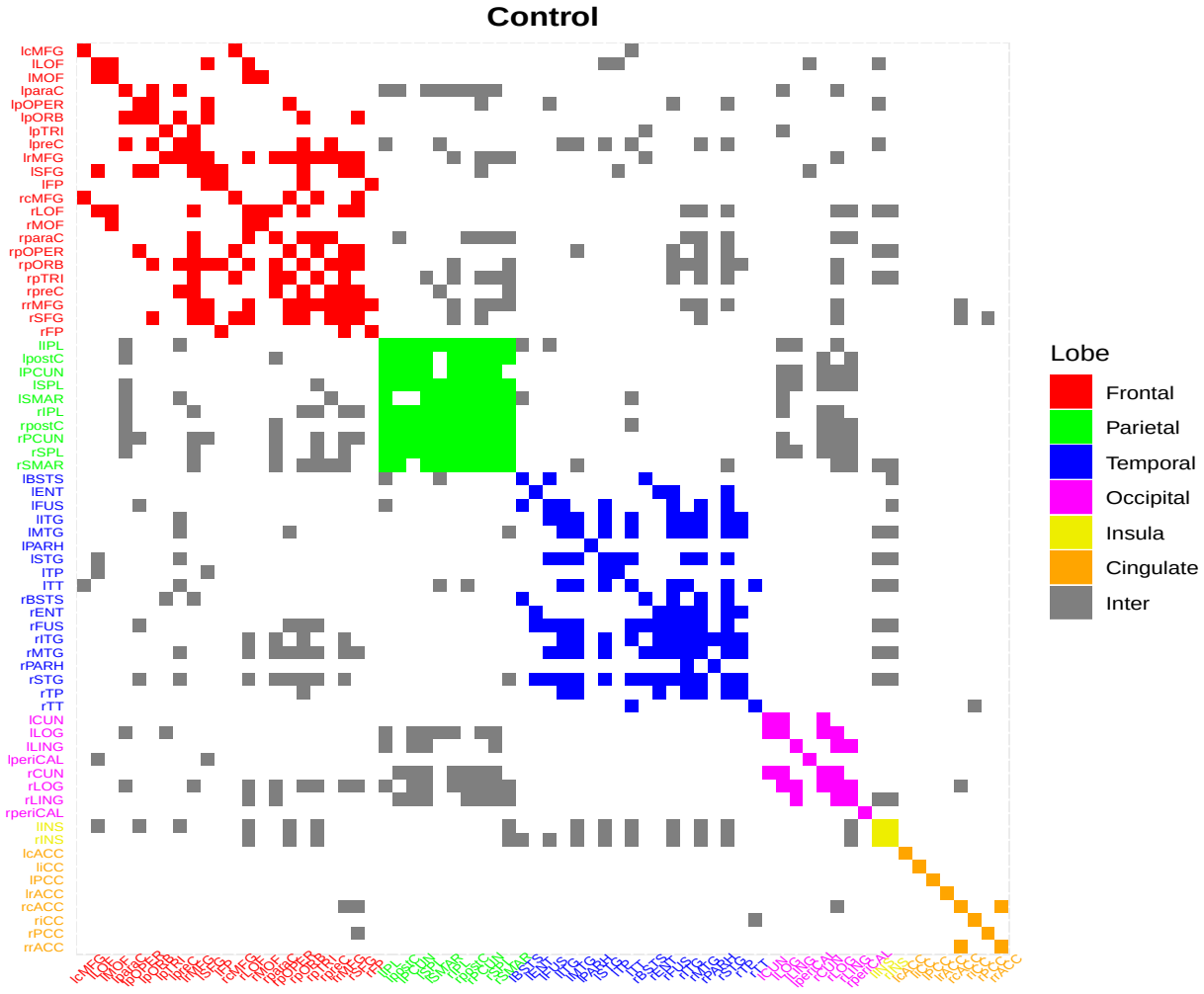

Figure 15.7: Adjacency matrix, group 1.

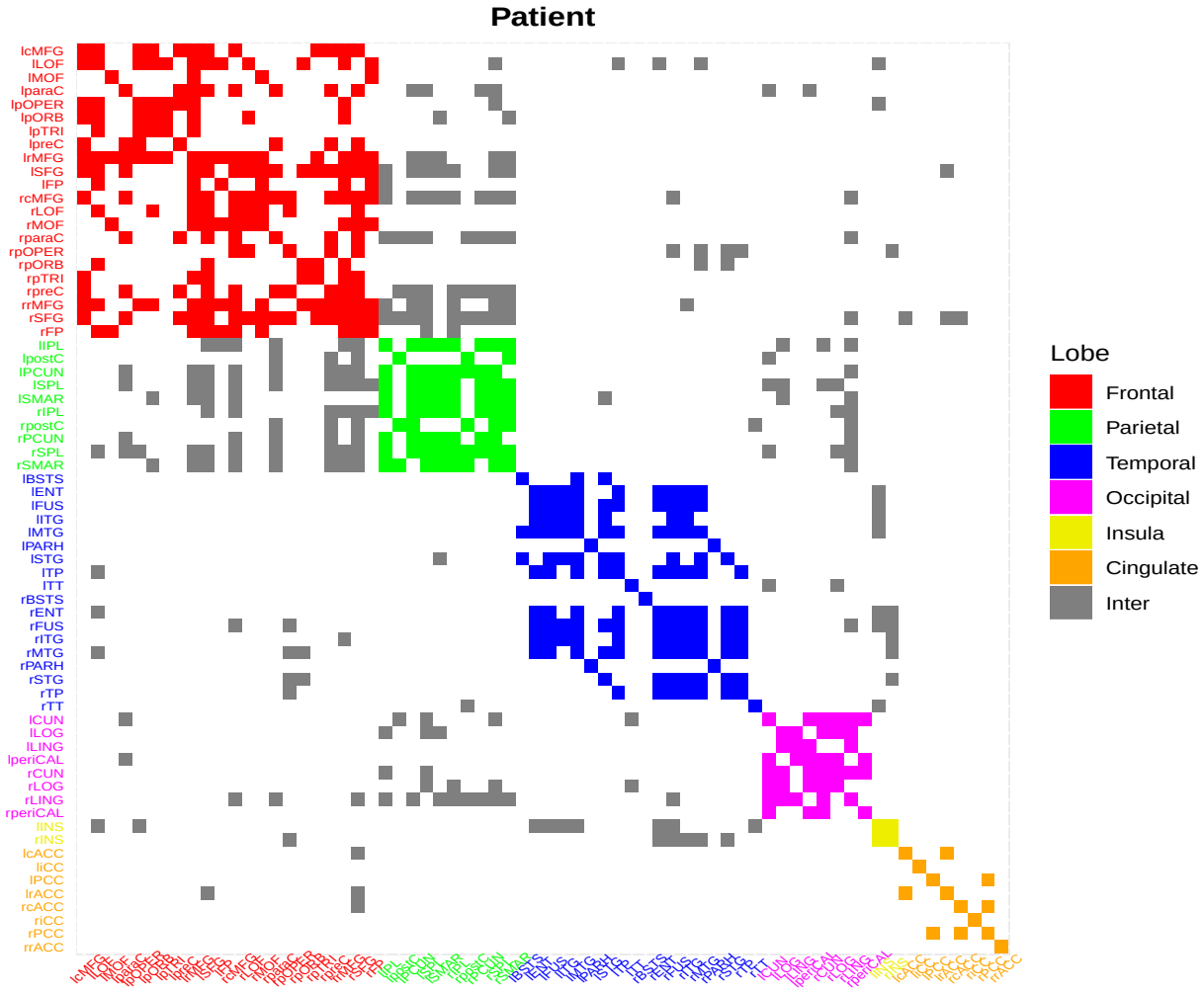

Figure 15.8: Adjacency matrix, group 2.

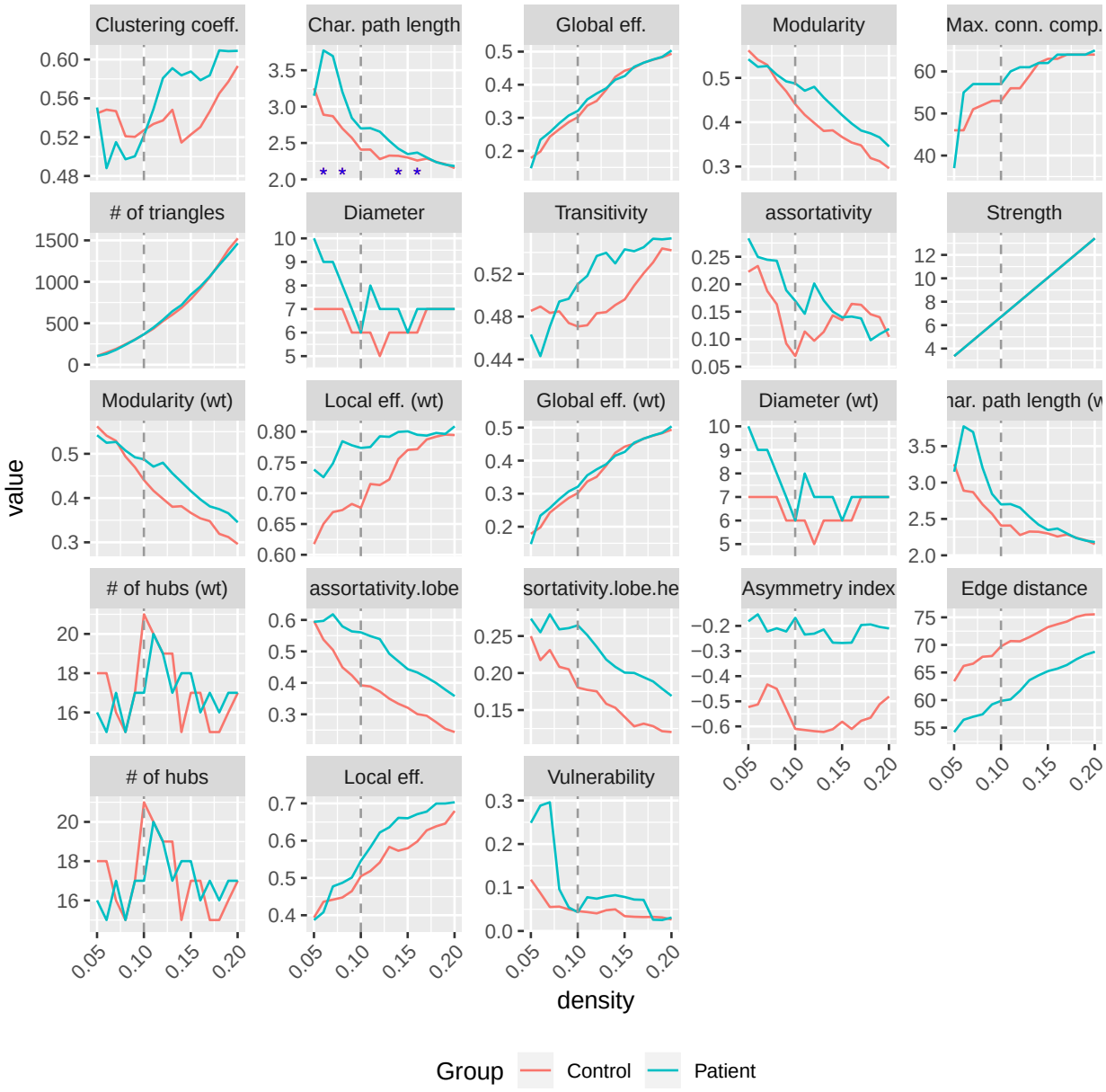

Figure 15.9: Global graph measures vs. density

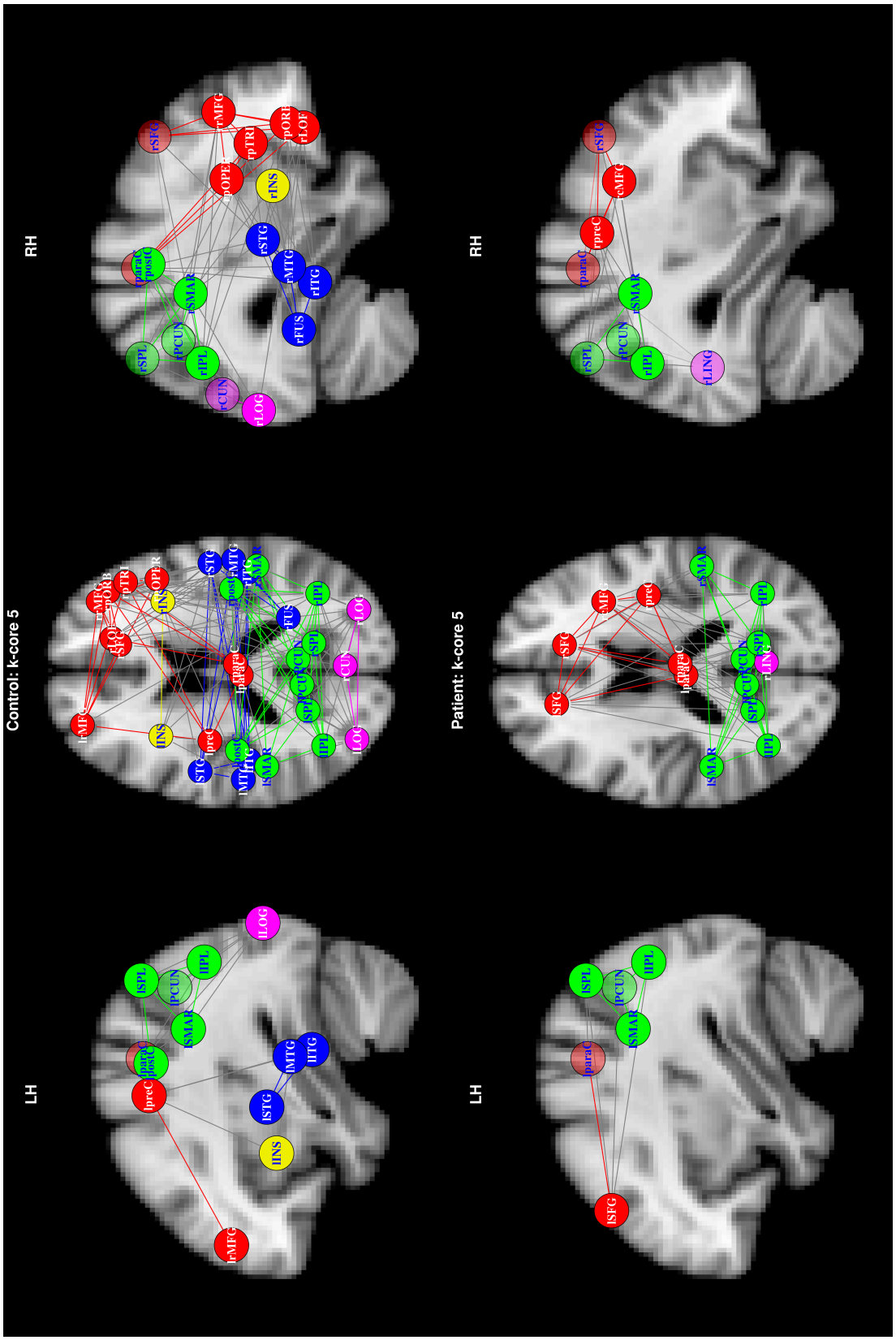

Figure 15.10: Vertices for which "coreness" is greater than 5.

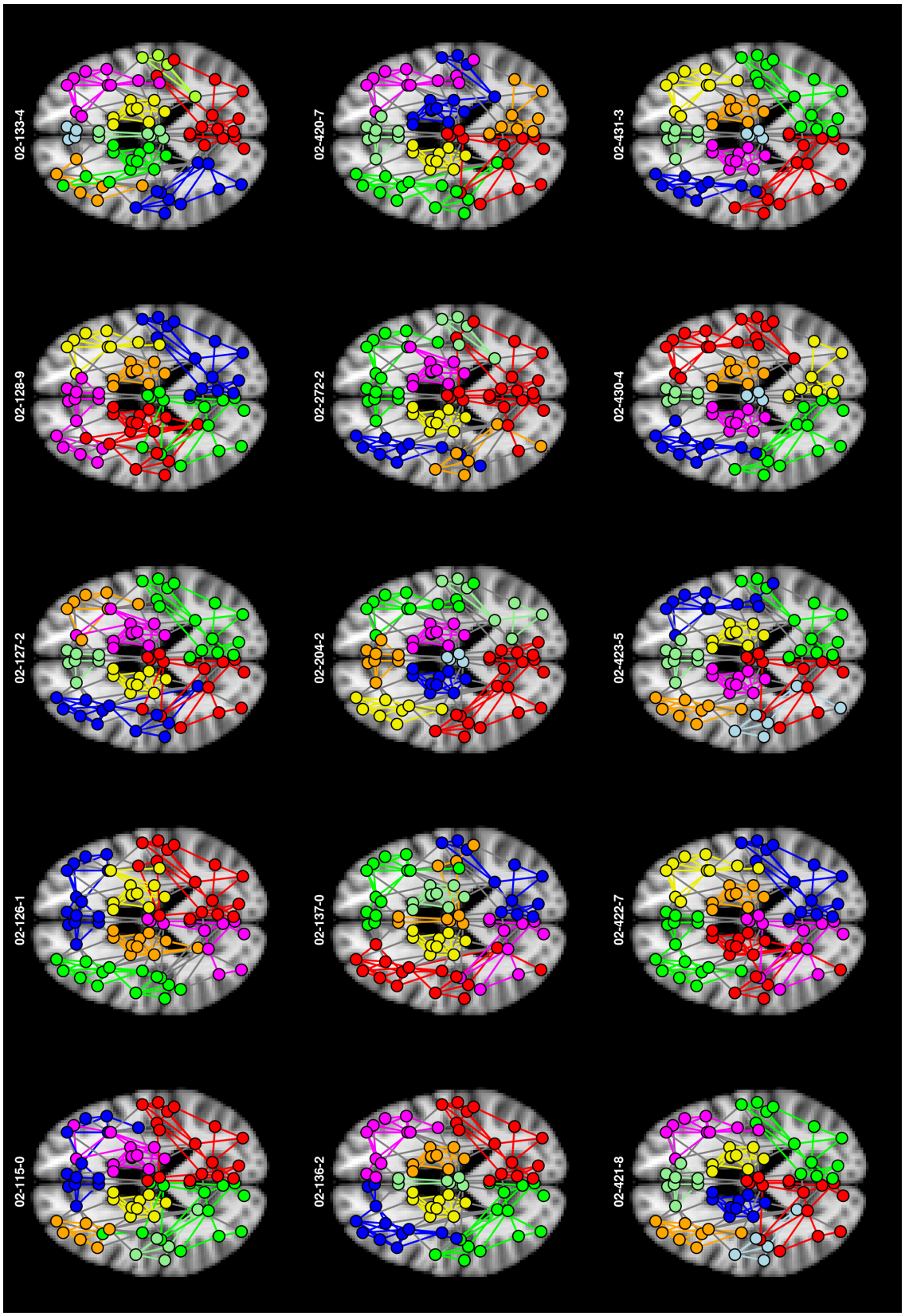

Figure 15.11: Axial views for 15 control subjects using `slicer`.

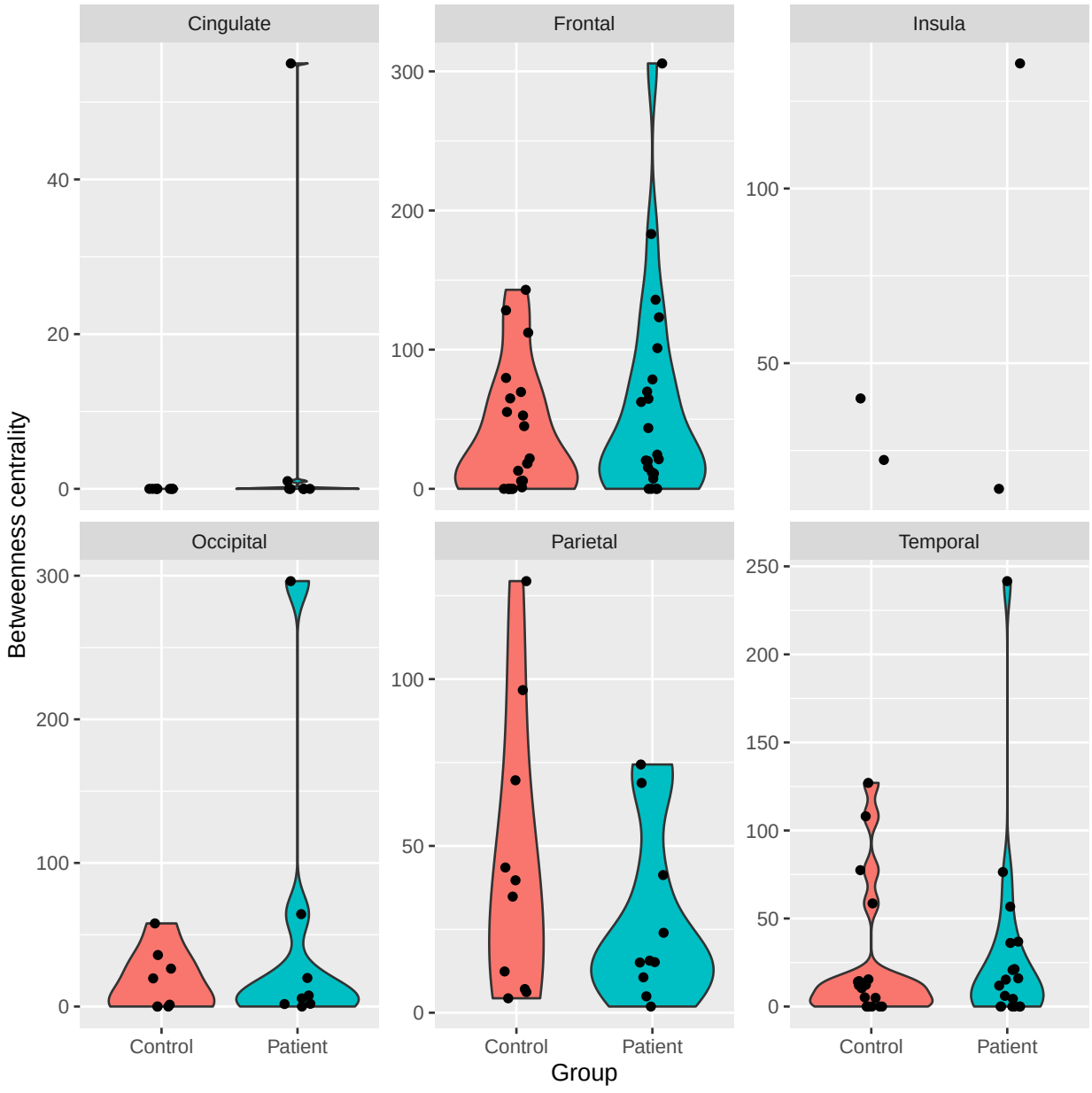

Figure 15.12: Vertex-level graph measures by lobe.

A

Attributes created by `set_brainGraph_attr`

A.1 Graph-level

A.1.1 Housekeeping

Some attributes are included only for "housekeeping" purposes. Most are optional, but I suggest supplying them.

version	The version of `brainGraph` that was used to create the graph
atlas	The atlas (e.g., `dk` for *Desikan-Killiany*)
clust.method	The clustering (community detection) algorithm used
clust.method.wt	Same as above, but using edge weights (if the graph is weighted). This is not necessarily different from the unweighted value.
Group	The subject/patient group name
name	The subject/study ID of the subject
modality	The imaging modality
weighting	The edge weighting. For example, `fa` (for FA-weighted matrices from tractography), or `partial` (for partial correlation coefficients).
threshold	The threshold applied to the raw connectivity matrices.
xfm.type	The method used to transform edge weights for distance-based metrics (if the graph is weighted)

A.1.2 Unweighted

Cp	The graph's *clustering coefficient* (using the `localaverage` option)
Lp	Characteristic path length
rich	A `data.table` of the rich-club coefficients and subgraph sizes
E.global	The *global efficiency*
E.local	The mean *local efficiency* across vertices
mod	*Modularity*, calculated using the algorithm specified by `clust.method`
density	

conn.comp A `data.frame` of the sizes of *connected components* present in the graph

max.comp The size of the largest connected component

num.tri The number of *triangles* in the graph

diameter

transitivity

assort *Degree assortativity*

assort.lobe Assortativity calculated using `lobe` membership

assort.lobe.hemi Assortativity calculated using `lobe` and `hemi` membership

assort.class Assortativity calculated using `class` membership (if the atlas is `destrieux`)

assort.network Assortativity calculated using `network` membership (if atlas is `dosenbach160`)

asymm Edge asymmetry

spatial.dist Mean of all (Euclidean) edge distances

vulnerability The maximum *vulnerability* across vertices

num.hubs The number of *hubs* (vertices with *hubness* score ≥ 2)

A.1.3 Weighted

strength Average vertex *strength*

rich.wt A `data.frame` of the weighted rich-club coefficients and subgraph sizes

mod.wt *Modularity*, calculated using the algorithm specified by `clust.method` and taking into account edge weights

E.local.wt The average of the vertices' weighted *local efficiency*

E.global.wt The weighted *global efficiency*

diameter.wt The graph's `diameter`, taking into account edge weights

Lp.wt The weighted *characteristic path length*, which is the mean of the vertices' average path lengths

num.hubs.wt The number of *hubs*, determined using edge weights to calculate path lengths, clustering coefficients, and vertex strength

A.2 Vertex-level

A.2.1 Housekeeping/Other

name The name of the brain region

lobe Lobe membership (e.g., `Frontal`)

hemi Hemisphere; either `L`, `R`, or `B` (for "both")

lobe.hemi Integer vector corresponding to a combination of the `lobe` and `hemi` attributes

class The "class", if the atlas is `destrieux`. Either `G` (gyral), `S` (sulcal), or `G_and_S` (both)

network The "network", if the atlas is `dosenbach160`

x,y,z The spatial coordinates

x.mni,y.mni,z.mni (same as above)

color.lobe The colors corresponding to `lobe` membership

color.class The colors corresponding to `class` membership (if the atlas is `destrieux`)

color.network The colors corresponding to `network` membership (if the atlas is `dosenbach160`)

color.comm Colors corresponding to community membership

color.comm.wt The colors corresponding to weighted community membership

color.comp Colors corresponding to component membership

A.2.2 Unweighted

degree

asymm Edge asymmetry

dist The mean (Euclidean) edge distance of each vertex's connections

dist.strength Vertices' `dist` multiplied by the `degree`

knn Average nearest neighbor degree

Lp The vertices' average shortest path lengths

btwn.cent Betweenness centrality

ev.cent Eigenvector centrality

lev.cent Leverage centrality

hubs "Hubness" scores (calculated by `hubness`)

k.core *k-core* membership

transitivity Transitivity

E.local *Local efficiency*

E.nodal *Nodal efficiency*

vulnerability

eccentricity The "longest shortest path length"; i.e., the maximum of shortest path lengths from a vertex to all other vertices

comm Integer vector of the vertices' community membership (from largest to smallest)

comp Integer vector of the vertices' *connected component* membership (from largest to smallest)

GC *Gateway coefficient*

PC *Participation coefficient*

z.score *Within-module degree z-score*

A.2.3 Weighted

strength

knn.wt	Average *nearest neighbor* strength
s.core	*s-core* membership
comm.wt	Integer vector of the vertices' (weighted) community membership (from largest to smallest)
GC.wt	The weighted version of the *gateway coefficient* (using `comm.wt`)
PC.wt	The weighted version of the *participation coefficient* (using `comm.wt`)
z.score.wt	The weighted version of the *within-module degree z-score* (using `comm.wt`)
transitivity.wt	Weighted transitivity
E.local.wt	Weighted *local efficiency*
E.nodal.wt	Weighted *nodal efficiency*
Lp.wt	The vertices' average shortest path lengths, taking into account edge weights
hubs.wt	"Hubness" scores using edge weights

A.3 Edge-level

weight	Edge weight
dist	The Euclidean distance of the edge
btwn	Edge betweenness
color.lobe	The colors corresponding to `lobe` membership
color.class	The colors corresponding to `class` membership (if the atlas is `destrieux`)
color.network	The colors corresponding to `network` membership (if the atlas is `dosenbach160`)
color.comm	The colors corresponding to *community* membership
color.comm.wt	The colors corresponding to *weighted* community membership
color.comp	The colors corresponding to *component* membership

B

GLM Statistics

New in v3.0.0

All of these functions/methods are new in `v3.0.0`.

B.1 Model fitting

There are several new model fitting functions that are much faster than other solutions (see Benchmarks). There are multiple default function arguments that greatly improve speed in various scenarios; this is most helpful when permuting and fitting the data.

Note

Only the first 2 of the following should be used directly.

fastLmBG Requires a design matrix X and numeric matrix of outcome variables Y. The outcome matrix should have 1 column for each outcome variable.

fastLmBG_3d Fits models when there are multiple design matrices, so X will be a 3-D array, and a single outcome variable, so Y will be a column matrix. The array X must have dimension names, and the input argument `runX` specifies the regions for which a model should be fit.

fastLmBG_3dY Fits models when there is both a different design and outcome variable for each region. This only occurs under permutation for the Freedman-Lane, ter Braak, and Still-White methods.

fastLmBG_3dY_1p Fits models when there is both a different design and outcome for each region, and also when X is a rank-1 matrix (i.e., has a single column). This only occurs under permutation with the Still-White method if there is a single regressor of interest.

B.2 Contrast-based statistics

There are 2 functions for calculating contrast-based statistics for F- and t-contrasts, respectively. Both require a *list* object that is returned from one of the model fitting functions (see previous section) as well as a list of contrast matrices (F-contrasts) or a single matrix (t-contrasts).

fastLmBG_t Returns a multidimensional array with the contrast of parameter estimates, standard error of the contrast, t-statistics, P-values, FDR-adjusted P-values, and confidence intervals

fastLmBG_f Returns a multidimensional array with the *extra sum of squares*, standard error (the sum of squared errors of the full model), F-statistic, P-values, and FDR-adjusted P-values

Details for the statistics calculated by these functions are as follows.

Gamma (if `con.type='t'`) The *contrast of parameter estimates*:

$$\hat{\gamma} = C^T \hat{\beta}$$

This is equivalent to the `cope?` images from FSL, the `con_????` images from SPM, and `gamma.mgh` from Freesurfer.

ESS (if `con.type='f'`) The *extra sum of squares* due to the full model, compared to the reduced model. This is specific to the contrast matrix, and is equivalent to the `ess_????` images from SPM.

Standard error (if `con.type='t'`) The standard error of the contrast, SE, is

$$SE = \sqrt{C \mathbf{\Sigma} C^T}$$

where $\mathbf{\Sigma}$, the *variance-covariance matrix*, is calculated as

$$\Sigma = s^2 \mathbf{X}^T \mathbf{X}$$

Standard error (if `con.type='f'`) The *sum of squared errors* (or *residual sum of sqares*) of the full model. This is, for residuals $\hat{e}$,

$$RSS = \hat{e}^T \hat{e}$$

stat The *t-statistic* for the contrast of interest is

$$T = \frac{\hat{\gamma}}{SE}$$

The *F-statistic* is

$$F = \frac{ESS/rank(C)}{RSS/df}$$

where $rank(C)$ is the *rank* of the contrast matrix and df equals the *residual degrees of freedom*

CI (if `con.type='t'`) The lower and upper confidence limits:

$$\hat{\gamma} \pm t_{\alpha/2,df} \times SE$$

B.3 GLM methods

There are now a few dozen methods for `bg_GLM` objects, many of which are meant to mimic methods of `lm` objects. All the methods for this class can be viewed with

```
methods(class='bg_GLM')
```

```
##  [1] [                   anova               case.names
##  [4] coef                confint             cooks.distance
##  [7] deviance            df.residual         dfbeta
## [10] dfbetas             extractAIC          fitted
## [13] formula             hatvalues           influence
## [16] labels              logLik              make_brainGraphList
## [19] nobs                nregions            plot
## [22] print               region.names        residuals
## [25] rstandard           rstudent            sigma
## [28] summary             terms               variable.names
## [31] vcov
## see '?methods' for accessing help and source code
```

The next few sections will list and describe these based on a few categories.

B.3.1 Basic information

Several of the methods just give basic information about the data/model. See `help('GLM basic info')` for details.

nobs The number of observations

terms Returns a list for each variable in the design matrix; it maps the variables to columns of the design matrix. Factor variables will be mapped to multiple columns.

labels The names of the `terms` list

formula The model formula (sometimes called "Wilkinson-Rogers notation")

case.names Character vector of the *case names*, which are typically subject IDs

variable.names Character vector of the design matrix column names

region.names Character vector of region names

nregions The number of regions

```
formula(anova2x3)

## [1] "E.nodal.wt ~ A * B"

terms(anova2x3)

## $Intercept
## [1] 1
##
## $A
## [1] 2
##
## $B
## [1] 3 4
##
## $`A:B`
## [1] 5 6

variable.names(anova2x3)

## [1] "Intercept" "A2"        "B2"        "B3"        "A2:B2"     "A2:B3"
```

B.3.2 Model fit statistics

These functions return basic model fit statistics. See `help('GLM statistics')` for details.
In the following list, n is the number of observations, p is the rank of the design matrix, and r is the number of regions.

coef A $p \times r$ matrix of model coefficients (parameter estimates)

confint A $p \times 2 \times r$ array of the confidence intervals of parameter estimates

fitted A $n \times r$ matrix of fitted values

residuals A $n \times r$ matrix of residuals. You can calculate "regular" or partial residuals.

deviance A vector of model deviance for each region. Also known as the *residual sum of squares*.

df.residual Residual *degrees of freedom*

sigma A vector of residual standard deviation for each region; sometimes called *root mean squared error (RMSE)*

vcov A $p \times p \times r$ array of variance-covariance matrices of model parameters

anova Performs "Type III" ANOVA tests, in addition to calculating several other statistics of interest. See the example in the following code block.

coeff_determ A vector of the *coefficient of determination*, or R^2 for each region

coeff_table Recreates the coefficients table that is calculated by `summary.lm`

```
coeff_table(diffs.perm)[, , 'lcACC']

##                Estimate Std. Error t value  Pr(>|t|)
## Intercept     0.0363158  0.0027050  13.425 1.527e-27
## Age.MRI       0.0003536  0.0001667   2.122 3.551e-02
## Sex          -0.0013172  0.0009532  -1.382 1.690e-01
## Scanner       0.0054788  0.0009674   5.664 7.269e-08
## GroupPatient -0.0007924  0.0005403  -1.467 1.445e-01

anova(diffs.perm)[[1]]

## Anova Table (Type III tests)
##
## Response: E.nodal.wt
##             Sum Sq Mean Sq  Df F value eta^2 Partial eta^2 omega^2
## Intercept 0.00619 0.00619   1  180.24 0.485         0.544   0.481
## Age.MRI   0.00015 0.00015   1    4.50 0.012         0.029   0.009
## Sex       0.00007 0.00007   1    1.91 0.005         0.012   0.002
## Scanner   0.00110 0.00110   1   32.08 0.086         0.175   0.083
## Group     0.00007 0.00007   1    2.15 0.006         0.014   0.003
## Residuals 0.00518 0.00003 151
##           Partial omega^2  Pr(>F)
## Intercept           0.535 < 2e-16 ***
## Age.MRI             0.022   0.036 *
## Sex                 0.006   0.169
## Scanner             0.166 7.3e-08 ***
## Group               0.007   0.145
## Residuals
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

B.3.3 Diagnostic/Influence measures

These functions return "leave-one-out" diagnostics, also called *influence* measures. These are important in determining outlier observations. See `?influence.bg_GLM` for more details.

Most of the following return a $n \times r$ matrix of values. `dfbeta` and `dfbetas` return a $n \times p \times r$ array.

influence	Calculates all influence measures detailed below, and identifies outliers based on pre-set values.
rstandard	Standard residuals
rstudent	Studentized residuals
hatvalues	The *leverage* (diagonal of the *hat/projection matrix*
cooks.distance	Cook's distance
dffits.bg_GLM	The change in fitted values when deleting observations
dfbeta	The change in parameter estimates when deleting observations
dfbetas	The *scaled* change in parameter estimates
covratio.bg_GLM	The covariance ratios; the change in the determinant of the covariance matrix of parameter estimates when deleting observations

```
str(influence(diffs.perm))

## List of 5
##  $ infmat: num [1:156, 1:9, 1:76] -0.0163 0.087 -0.1074 -0.0286 -0.0631 ...
##   ..- attr(*, "dimnames")=List of 3
##   .. ..$ : chr [1:156] "02-001-4" "02-003-2" "02-006-8" "02-008-7" ...
##   .. ..$ : chr [1:9] "dfb.Intercept" "dfb.Age.MRI" "dfb.Sex" "dfb.Scanner" ...
##   .. ..$ : chr [1:76] "lcACC" "lcMFG" "lCUN" "lENT" ...
##  $ is.inf: logi [1:156, 1:9, 1:76] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   ..- attr(*, "dimnames")=List of 3
##   .. ..$ : chr [1:156] "02-001-4" "02-003-2" "02-006-8" "02-008-7" ...
##   .. ..$ : chr [1:9] "dfb.Intercept" "dfb.Age.MRI" "dfb.Sex" "dfb.Scanner" ...
##   .. ..$ : chr [1:76] "lcACC" "lcMFG" "lCUN" "lENT" ...
##  $ f     : chr "E.nodal.wt ~ Age.MRI + Sex + Scanner + Group"
##  $ sigma : num [1:156, 1:76] 0.00588 0.00585 0.00584 0.00587 0.00584 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:156] "02-001-4" "02-003-2" "02-006-8" "02-008-7" ...
##   .. ..$ : chr [1:76] "lcACC" "lcMFG" "lCUN" "lENT" ...
##  $ wt.res: num [1:156, 1:76] 0.00142 -0.00654 0.008 0.00353 0.00795 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:156] "02-001-4" "02-003-2" "02-006-8" "02-008-7" ...
##   .. ..$ : chr [1:76] "lcACC" "lcMFG" "lCUN" "lENT" ...
##  - attr(*, "class")= chr [1:2] "infl.bg_GLM" "list"
```

B.3.4 Other statistics

These functions return other statistics.

vif.bg_GLM	Returns a $(p-1) \times 3 \times r$ array of *variance inflation factors*
logLik	Log-likelihood

extractAIC	Akaike's *An Information Criterion (AIC)*
AIC	Essentially the same as above
BIC	Bayesian Information Criterion

C

Functions for generic data

Although `brainGraph` has been developed specifically for brain MRI data, there are several functions that will work on generic (non-brain) data. These are functions that do not make use of graph attributes such as atlas, lobe, hemisphere, spatial coordinates, etc. Some functions require an attribute that, while not specific to brain MRI data, do need to be added by the user; these will be marked clearly in the following list.

- `fastLmBG` and the other model fitting functions (see GLM Statistics)
- Matrix utilities: `colMax`, `colMaxAbs`, `colMin`; `diag_sq` (simplified version of `diag`); `is_binary`; `symmetrize`; `inv` and `pinv` (methods for calculating matrix inverses; see `?inv`)
- Functions requiring a `brainGraphList` object will need at least a "dummy" `atlas` argument, and the `gnames` should match the *Study.ID* in the `covars` data table if applicable.
- `brainGraph_GLM` and its related functions (needs *name* graph attribute that matches the *Study.ID* column in the input `covars` data table)
- `brainGraph_GLM_design`
- `brainGraph_mediate` (same as GLM requirements)
- `mtpc` (same as GLM requirements)
- `NBS` (same as GLM requirements)
- `centr_lev`
- `centr_betw_comm`
- `communicability`
- `efficiency`
- `hubness`
- `make_ego_brainGraph`
- `make_intersection_brainGraph`
- `mean_distance_wt` Calculates weighted graph or vertex average shortest path lengths
- `partition` Partition a design matrix based on a contrast of interest (for randomization/permutation)
- `sim.rand.graph.clust`
- Rich-club functions (although `rich_club_norm` will require *Group* and possibly *Study.ID* graph attributes if you do not supply a list of random graphs)

- `robustness`
- `s_core`
- `small.world` (observed and random graphs all need Cp and Lp graph attributes)
- `gateway_coeff`, `part_coeff`, `within_module_deg_z_score`
- `vulnerability`

D

Benchmarks

Most functions will run quickly on just about any machine, since `igraph` is based on `C` routines. Nevertheless, here is some benchmarking information. The more CPU cores you have, the better.

The timing of the graph algorithms is extremely fast for "small" graphs (e.g., with any atlas-based brain parcellation), and are generally going to take either $O(n)$ or $O(m)$ time, where n is the # of vertices, and m is the # of edges. The longest processing times will be Random graph generation, Bootstrapping, Permutation Testing, and Random Failure Analysis.

All testing was done on the same system:

System Information

```
64-bit CentOS 7.7.1908
Intel Core i7-6500U (4 cores @ 2.50 GHz)
R version 3.6.0 (2019-04-26)
```

D.1 GLM-related benchmarks

D.1.1 Randomise

Benchmarks for `randomise` are shown in Table D.1. The first section is when `outcome=measure` (the default) and the second is for a different outcome (multiple design matrices). As expected, the latter case takes longer to run.

Outcome	Perm. method	Old	New	Speedup
Same	Freedman-Lane	9.057	0.868	10.43x
	Smith	3.886	1.006	3.86x
Different	Freedman-Lane	127.229	3.340	38.09x
	Smith	117.811	12.001	9.82x

Table D.1: **Runtimes (in seconds) for `randomise`.** These are analyses with 2 contrasts and using the `dk.scgm` atlas (82 vertices). The number of permutations for each run is 1,000. Where it says **Same**, it is for analyses in which `outcome=measure` .

When Y is very large—for example, in a graph with a very large number of vertices and/or edges—fitting a single model is still very fast. For example, it takes on the order of 0.15 seconds to fit a model where Y is a $101 \times 64,620$ matrix (which is equal to the total number of edges possible with a 360 vertex graph, like the HCP atlas). Furthermore, in these cases I recommend using one of the `smith` or `draperStoneman` permutation methods as they are more efficient.

D.1.2 NBS

Benchmarks for `NBS` are shown in Table D.2. This example used "raw" matrices with approximately 97% density (3,225 edges). As you can see, for $N = 5000$ permutations, what previously would have taken more than 10 minutes now takes 16 seconds.

# permutations	Old	New	Speedup
500	70.504	2.066	34.13x
1000	140.882	3.625	38.87x
2500	NA	7.645	NA
5000	740.503	16.472	44.96x

Table D.2: **Runtimes (in seconds) for `NBS`.** These analyses were in graphs with approx. 97% density (3,225 edges).

D.1.3 Mediation

Table D.3 lists runtime for `brainGraph_mediate` with the `dk.scgm` atlas (82 vertices).

Level	# of permutations	Old	New	Speedup
Vertex	1,000	47.816	7.246	6.599x
	5,000	159.831	35.959	4.445x
	10,000	268.213	63.060	4.253x
Graph	1,000	0.536	0.564	0.95x
	5,000	2.106	2.203	0.95x
	10,000	3.665	3.868	0.95x

Table D.3: **Runtimes for vertex- and graph-level `mediation` for graphs with 76 vertices.** These analyses were for 101 subjects in 3 groups. Runtimes are the median of 100 repetitions in seconds.

D.2 Random graph generation

For random graph generation, runtimes vary depending on whether or not the graph is *connected*, whether or not you control for *clustering*, on the graph *size* (i.e., number of edges), and on the number of subjects/groups (i.e., total number of graphs).

Note

These benchmarks were run on older `R` and `igraph` versions, so it's possible these will be faster on a similar system.

- The `dk` atlas (**68 vertices**) for cortical thickness covariance:

  Random graph generation (control for clustering) For 1 group and 1 density (20%), generating 1,000 random graphs, total processing time was **5 min. 45 sec.**. Compared to a runtime of 5 hr. 17 min. in the oldest version (which was on a machine with more and faster CPU cores), this is approximately a **55x** speed increase. (The processing time prior to `v2.5.0` was about 10 minutes).

- The `dkt` atlas (**62 vertices**) for cortical thickness covariance:

Random graph generation (control for clustering) For 2 groups and 44 densities (from 7-50%, steps of 1%), and with 100 random graphs generated (per density per group) (i.e., 8,800 graphs), total processing time was (estimated to be) **2 hr. 29 min.**. The # of iterations per graph was limited to 100 (the default value for the `max.iters` argument to `sim.rand.graph.clust`; see Random graph generation). Compared to a runtime of 18 hr. 33 min. in the older version (and on a machine with more and faster CPU cores), this is a **7.5x** speed increase.

- The `dk.scgm` atlas (**82 vertices**) for DTI tractography:

  Random graph generation For 3 groups, 30 thresholds, and 104 subjects total, generating 100 random graphs for all combinations (i.e., 312,000 graphs) took **6 hr. 6 min.**. Total disk space used was **4.0 GB**. The bulk of the increase in processing time was due to several of the thresholds containing *unconnected graphs*; for the connected graphs, generating 100 random graphs for 36 subjects took between **0 min. 42 sec. – 1 min. 32 sec.** per threshold.

  Random graph generation For 3 groups and 30 thresholds, generating 1,000 random graphs for all combinations (i.e., 90,000 graphs) took **2 hr. 29 min.**. Total disk space used was **1.2 GB**.

E

Computing environment

This document was typeset with LaTeX (through the `knitr` package in `R`), using the `book` document class. The main typeface is `Computer Modern`, designed by Donald Knuth for TeX. The bibliography was typeset with BibTeX using the `natbib` package.

The following versions of `R`, the operating system, and `R` packages used include:

- R version 3.6.0 (2019-04-26), `x86_64-redhat-linux-gnu`
- Running under: `CentOS Linux 7 (Core)`
- Random number generation:
- RNG: `Mersenne-Twister`
- Normal: `Inversion`
- Sample: `Rounding`
- Matrix products: default
- BLAS/LAPACK: `/usr/lib64/R/lib/libRblas.so`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, utils
- Other packages: abind 1.4-5, ade4 1.7-13, boot 1.3-23, brainGraph 3.0.0, cairoDevice 2.28, colorout 1.2-2, data.table 1.12.6, doMC 1.3.6, foreach 1.4.7, Formula 1.2-3, ggforce 0.3.1, ggplot2 3.2.1, ggrepel 0.8.1, gridExtra 2.3, Hmisc 4.2-0, igraph 1.2.4.1, iterators 1.0.12, knitr 1.25, lattice 0.20-38, MASS 7.3-51.4, Matrix 1.2-17, microbenchmark 1.4-7, nvimcom 0.9-83, oro.nifti 0.9.1, pacman 0.5.1, permute 0.9-5, plyr 1.8.4, RcppEigen 0.3.3.5.0, RGtk2 2.20.36, scales 1.0.0, setwidth 1.0-4, sig 0.0-5, survival 2.44-1.1
- Loaded via a namespace (and not attached): acepack 1.4.1, assertthat 0.2.1, backports 1.1.5, base64enc 0.1-3, bitops 1.0-6, checkmate 1.9.4, cluster 2.0.8, codetools 0.2-16, colorspace 1.4-1, compiler 3.6.0, crayon 1.3.4, digest 0.6.22, doParallel 1.0.15, dplyr 0.8.3, evaluate 0.14, farver 1.1.0, foreign 0.8-71, glue 1.3.1, grid 3.6.0, gtable 0.3.0, highr 0.8, htmlTable 1.13.2, htmltools 0.4.0, htmlwidgets 1.5.1, labeling 0.3, latticeExtra 0.6-28, lazyeval 0.2.2, lme4 1.1-21, lpSolve 5.6.13.3, magrittr 1.5, mediation 4.5.0, minqa 1.2.4, munsell 0.5.0, mvtnorm 1.0-11, nlme 3.1-139, nloptr 1.2.1, nnet 7.3-12, pillar 1.4.2, pkgconfig 2.0.3, polyclip 1.10-0, purrr 0.3.3, R6 2.4.0, RColorBrewer 1.1-2, Rcpp 1.0.4, rlang 0.4.1, RNifti 0.11.1, rpart 4.1-15, rstudioapi 0.10, sandwich 2.5-1, splines 3.6.0, stringi 1.4.3, stringr 1.4.0, tibble 2.1.3, tidyselect 0.2.5, tools 3.6.0, tweenr 1.0.1, withr 2.1.2, xfun 0.10, zoo 1.8-6

Bibliography

[1] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000. doi:10.1515/9781400841356.503.

[2] Marti Anderson and Cajo ter Braak. Permutation tests for multi-factorial analysis of variance. *Journal of Statistical Computation and Simulation*, 73(2):85–113, 2003. doi:10.1080/00949650215733. URL https://doi.org/10.1080/00949650215733.

[3] Corson N Areshenkoff, Joseph Y Nashed, R Matthew Hutchison, Melina Hutchison, Ron Levy, Douglas J Cook, Ravi S Menon, Stefan Everling, and Jason P Gallivan. Muting, not fragmentation, of functional brain networks under general anesthesia. *bioRxiv*, pages 1–24, 2020.

[4] Shweta Bansal, Shashank Khandelwal, and Lauren A Meyers. Exploring biological network structure with clustered random networks. *BMC Bioinformatics*, 10(1):405, 2009. doi:10.1186/1471-2105-10-405.

[5] Gaetano Barbagallo, Maria Eugenia Caligiuri, Gennarina Arabia, Andrea Cherubini, Angela Lupo, Rita Nisticò, Maria Salsone, Fabiana Novellino, Maurizio Morelli, Giuseppe Lucio Cascini, et al. Structural connectivity differences in motor network between tremor-dominant and nontremor Parkinson's disease. *Human Brain Mapping*, 38(9):4716–4729, 2017. doi:10.1002/hbm.23697.

[6] Reuben M Baron and David A Kenny. The moderator–mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of Personality and Social Psychology*, 51(6):1173, 1986. doi:10.1037/0022-3514.51.6.1173.

[7] Christian F Beckmann, Mark Jenkinson, and Stephen M Smith. General multilevel linear modeling for group analysis in FMRI. *NeuroImage*, 20(2):1052–1063, 2003.

[8] TEJ Behrens, MW Woolrich, M Jenkinson, H Johansen-Berg, RG Nunes, S Clare, PM Matthews, JM Brady, and SM Smith. Characterization and propagation of uncertainty in diffusion-weighted MR imaging. *Magnetic Resonance in Medicine*, 50(5):1077–1088, 2003. doi:10.1002/mrm.10609.

[9] TEJ Behrens, H Johansen Berg, Saad Jbabdi, MFS Rushworth, and MW Woolrich. Probabilistic diffusion tractography with multiple fibre orientations: What can we gain? *NeuroImage*, 34(1):144–155, 2007. doi:10.1016/j.neuroimage.2006.09.018.

[10] Boris C Bernhardt, Zhang Chen, Yong He, Alan C Evans, and Neda Bernasconi. Graph-theoretical analysis reveals disrupted small-world organization of cortical thickness correlation networks in temporal lobe epilepsy. *Cerebral Cortex*, 21(9):2147–2157, 2011. doi:10.1093/cercor/bhq291.

[11] Jeremy C Biesanz, Carl F Falk, and Victoria Savalei. Assessing mediational models: testing and interval estimation for indirect effects. *Multivariate Behavioral Research*, 45(4):661–701, 2010. doi:10.1080/00273171.2010.498292.

[12] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10): P10008, 2008. doi:10.1088/1742-5468/2008/10/P10008.

[13] Maria Eugenia Caligiuri, Gennarina Arabia, Gaetano Barbagallo, Angela Lupo, Maurizio Morelli, Rita Nisticò, Fabiana Novellino, Andrea Quattrone, Maria Salsone, Basilio Vescio, et al. Structural connectivity differences in essential tremor with and without resting tremor. *Journal of Neurology*, pages 1–10, 2017. doi:10.1007/s00415-017-8553-5.

[14] Qingjiu Cao, Ni Shu, Li An, Peng Wang, Li Sun, Ming-Rui Xia, Jin-Hui Wang, Gao-Lang Gong, Yu-Feng Zang, Yu-Feng Wang, et al. Probabilistic diffusion tractography and graph theory analysis reveal abnormal white matter structural connectivity networks in drug-naive boys with attention deficit/hyperactivity disorder. *Journal of Neuroscience*, 33(26):10676–10687, 2013. doi:10.1523/JNEUROSCI.4793-12.2013.

[15] Matteo Cinelli, Giovanna Ferraro, and Antonio Iovanella. Rich-club ordering and the dyadic effect: two interrelated phenomena. *Physica A: Statistical Mechanics and its Applications*, 490:808–818, 2018. doi:10.1016/j.physa.2017.08.122.

[16] Vittoria Colizza, Alessandro Flammini, M Angeles Serrano, and Alessandro Vespignani. Detecting rich-club ordering in complex networks. *Nature Physics*, 2(2):110–115, 2006. doi:10.1038/nphys209.

[17] Robert W Cox. AFNI: software for analysis and visualization of functional magnetic resonance neuroimages. *Computers and Biomedical Research*, 29(3):162–173, 1996. doi:10.1006/cbmr.1996.0014.

[18] R Cameron Craddock, G Andrew James, Paul E Holtzheimer, Xiaoping P Hu, and Helen S Mayberg. A whole brain fMRI atlas generated via spatially constrained spectral clustering. *Human Brain Mapping*, 33(8):1914–1928, 2012. doi:10.1002/hbm.21333.

[19] Jonathan J Crofts and Desmond J Higham. A weighted communicability measure applied to complex brain networks. *Journal of the Royal Society, Interface*, 6(33):411–414, 2009. doi:10.1098/rsif.2008.0484.

[20] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[21] Z Cui, S Zhong, P Xu, Y He, and G Gong. PANDA: a pipeline toolbox for analyzing brain diffusion images. *Frontiers in Human Neuroscience*, 7(7):42, 2013. doi:10.3389/fnhum.2013.00042.

[22] Dina R. Dajani, Catherine A. Burrows, Mary Beth Nebel, Stewart H. Mostofsky, Kathleen M. Gates, and Lucina Q. Uddin. Parsing heterogeneity in autism spectrum disorder and attention-deficit/hyperactivity disorder with individual connectome mapping. *Brain Connectivity*, 9(9):673–691, 2019. doi:10.1089/brain.2019.0669. URL `https://doi.org/10.1089/brain.2019.0669`. PMID: 31631690.

[23] Anthony Christopher Davison and David Victor Hinkley. *Bootstrap methods and their application*, volume 1. Cambridge University Press, 1997. doi:10.1017/CBO9780511802843.

[24] Marcel A de Reus and Martijn P van den Heuvel. Estimating false positives and negatives in brain networks. *NeuroImage*, 70:402–409, 2013. doi:10.1016/j.neuroimage.2012.12.066.

[25] Manuel Delgado-Baquerizo, Peter B Reich, Chanda Trivedi, David J Eldridge, Sebastián Abades, Fernando D Alfaro, Felipe Bastida, Asmeret A Berhe, Nick A Cutler, Antonio Gallardo, et al. Multiple elements of soil biodiversity drive ecosystem functions across biomes. *Nature Ecology & Evolution*, 4(2): 210–220, 2020.

[26] Daniel DellaPosta and Victor Nee. Emergence of diverse and specialized knowledge in a metropolitan tech cluster. *Social Science Research*, 86:102377, 2020.

[27] R. S. Desikan, F. Segonne, B. Fischl, B. T. Quinn, B. C. Dickerson, D. Blacker, R. L. Buckner, A. M. Dale, R. P. Maguire, B. T. Hyman, M. S. Albert, and R. J. Killiany. An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest. *NeuroImage*, 31(3):968–980, 2006. doi:10.1016/j.neuroimage.2006.01.021.

[28] Christophe Destrieux, Bruce Fischl, Anders Dale, and Eric Halgren. Automatic parcellation of human cortical gyri and sulci using standard anatomical nomenclature. *NeuroImage*, 53(1):1–15, 2010. doi:10.1016/j.neuroimage.2010.06.010.

[29] Nico UF Dosenbach, Binyam Nardos, Alexander L Cohen, Damien A Fair, Jonathan D Power, Jessica A Church, Steven M Nelson, Gagan S Wig, Alecia C Vogel, Christina N Lessov-Schlaggar, et al. Prediction of individual brain maturity using fMRI. *Science*, 329(5997):1358–1361, 2010. doi:10.1126/science.1194144.

[30] Mark Drakesmith, Karen Caeyenberghs, A Dutt, G Lewis, AS David, and Derek K Jones. Overcoming the effects of false positives and threshold bias in graph theoretical analyses of neuroimaging data. *NeuroImage*, 118:313–333, 2015. doi:10.1016/j.neuroimage.2015.05.011.

[31] Norman R Draper and David M Stoneman. Testing for the inclusion of variables in einear regression by a randomisation technique. *Technometrics*, 8(4):695–699, 1966.

[32] Stephen J Eglen, Ben Marwick, Yaroslav O Halchenko, Michael Hanke, Shoaib Sufi, Padraig Gleeson, R Angus Silver, Andrew P Davison, Linda Lanyon, Mathew Abrams, et al. Toward standard practices for sharing computer code and programs in neuroscience. *Nature Neuroscience*, 20(6):770–773, 2017. doi:10.1038/nn.4550.

[33] Marius Eidsaa and Eivind Almaas. S-core network decomposition: a generalization of k-core analysis to weighted networks. *Physical Review E*, 88(6):062819, 2013. doi:10.1103/PhysRevE.88.062819.

[34] Ernesto Estrada and Naomichi Hatano. Communicability in complex networks. *Physical Review E*, 77 (3):036111, 2008. doi:10.1103/PhysRevE.77.036111.

[35] Ernesto Estrada, Desmond J Higham, and Naomichi Hatano. Communicability betweenness in complex networks. *Physica A: Statistical Mechanics and its Applications*, 388(5):764–774, 2009. doi:10.1016/j.physa.2008.11.011.

[36] Carl F Falk and Jeremy C Biesanz. Two cross-platform programs for inferences and interval estimation about indirect effects in mediational models. *SAGE Open*, 6(1):2158244015625445, 2016. doi:10.1177/2158244015625445.

[37] Lingzhong Fan, Hai Li, Junjie Zhuo, Yu Zhang, Jiaojian Wang, Liangfu Chen, Zhengyi Yang, Congying Chu, Sangma Xie, Angela R Laird, et al. The human brainnetome atlas: a new brain atlas based on connectional architecture. *Cerebral Cortex*, 26(8):3508–3526, 2016. doi:10.1093/cercor/bhw157.

[38] Maria Feldmann, Ting Guo, Steven P Miller, Walter Knirsch, Raimund Kottke, Cornelia Hagmann, Beatrice Latal, and Andras Jakab. Delayed maturation of the structural brain connectome in neonates with congenital heart disease. *bioRxiv*, pages 1–17, 2020.

[39] Alex Fornito, Andrew Zalesky, and Edward Bullmore. *Fundamentals of Brain Network Analysis*. Academic Press, 2016. ISBN 9780124081185.

[40] David Freedman and David Lane. A nonstochastic interpretation of reported significance levels. *Journal of Business and Economic Statistics*, 1(4):292–298, 1983. doi:10.1080/07350015.1983.10509354. URL https://www.tandfonline.com/doi/abs/10.1080/07350015.1983.10509354.

[41] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Softw., Pract. Exper.*, 21(11):1129–1164, 1991. doi:10.1002/spe.4380211102.

[42] Matthew F Glasser, Timothy S Coalson, Emma C Robinson, Carl D Hacker, John Harwell, Essa Yacoub, Kamil Ugurbil, Jesper Andersson, Christian F Beckmann, Mark Jenkinson, et al. A multi-modal parcellation of human cerebral cortex. *Nature*, 536(7615):171–178, 2016.

[43] Gaolang Gong, Pedro Rosa-Neto, Felix Carbonell, Zhang J Chen, Yong He, and Alan C Evans. Age- and gender-related differences in the cortical anatomical network. *Journal of Neuroscience*, 29(50): 15684–15693, 2009. doi:10.1523/JNEUROSCI.2308-09.2009.

[44] Evan M Gordon, Timothy O Laumann, Babatunde Adeyemo, Jeremy F Huckins, William M Kelley, and Steven E Petersen. Generation and evaluation of a cortical area parcellation from resting-state correlations. *Cerebral Cortex*, 26(1):288–303, 2016. doi:10.1093/cercor/bhu239.

[45] Roger Guimera and Luis A Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, 2005. doi:10.1038/nature03288.

[46] Irwin Guttman. *Linear Models: an Introduction*. Krieger Pub Co, 1982.

[47] MohammadHossein Manuel Haqiqatkhah and Cees van Leeuwen. Adaptive rewiring in non-uniform coupled oscillators. *PsyArXiv*, pages 1–45, 2020.

[48] Yong He, Zhang Chen, and Alan Evans. Structural insights into aberrant topological patterns of large-scale cortical networks in alzheimer's disease. *Journal of Neuroscience*, 28(18):4756–4766, 2008.

[49] Markus Hirschberger, Yue Qi, and Ralph E Steuer. Randomly generating portfolio-selection covariance matrices with specified distributional characteristics. *European Journal of Operational Research*, 177 (3):1610–1625, 2007.

[50] Paul W Holland. Statistics and causal inference. *Journal of the American Statistical Association*, 81 (396):945–960, 1986.

[51] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical Review E*, 65(5):056109, 2002. doi:10.1103/PhysRevE.65.056109.

[52] Philipp Homan, Miklos Argyelan, Pamela DeRosse, Philip Szeszko, Juan A Gallego, Lauren Hanna, Delbert Robinson, John M Kane, Todd Lencz, and Anil K Malhotra. Structural similarity networks predict clinical outcome in early-phase psychosis. *Neuropsychopharmacology*, 0:1–33, 2019. doi:10.1038/s41386-019-0322-y. URL `https://doi.org/10.1038/s41386-019-0322-y`.

[53] SM Hadi Hosseini and Shelli R Kesler. Influence of choice of null network on small-world parameters of structural correlation networks. *PLoS One*, 8(6):e67354, 2013. doi:10.1371/journal.pone.0067354.

[54] SM Hadi Hosseini, Fumiko Hoeft, and Shelli R Kesler. GAT: a graph-theoretical analysis toolbox for analyzing between-group differences in large-scale structural and functional brain networks. *PloS One*, 7(7):e40709, 2012. doi:10.1371/journal.pone.0040709.

[55] Mark D Humphries and Kevin Gurney. Network 'small-world-ness': a quantitative method for determining canonical network equivalence. *PLoS One*, 3(4):e0002051, 2008. doi:10.1371/journal.pone.0002051.

[56] Kosuke Imai, Luke Keele, and Dustin Tingley. A general approach to causal mediation analysis. *Psychological Methods*, 15(4):309, 2010. doi:10.1037/a0020761.

[57] Kosuke Imai, Luke Keele, and Teppei Yamamoto. Identification, inference and sensitivity analysis for causal mediation effects. *Statistical Science*, 25(1):51–71, 2010. doi:10.1214/10-STS321.

[58] Kosuke Imai, Luke Keele, Dustin Tingley, and Teppei Yamamoto. Unpacking the black box of causality: learning about causal mechanisms from experimental and observational studies. *American Political Science Review*, 105(4):765–789, 2011. doi:10.1017/S0003055411000414.

[59] Amy C Janes, Maya Zegel, Kyoko Ohashi, Jennifer Betts, Elena Molokotos, David Olson, Lauren Moran, and Diego A Pizzagalli. Nicotine normalizes cortico-striatal connectivity in non-smoking individuals with major depressive disorder. *Neuropsychopharmacology*, page 1, 2018. doi:10.1038/s41386-018-0069-x.

[60] Saad Jbabdi, MW Woolrich, JLR Andersson, and TEJ Behrens. A bayesian framework for global tractography. *NeuroImage*, 37(1):116–129, 2007. doi:10.1016/j.neuroimage.2007.04.039.

[61] Mark Jenkinson, Christian F Beckmann, Timothy EJ Behrens, Mark W Woolrich, and Stephen M Smith. FSL. *NeuroImage*, 62(2):782–790, 2012. doi:10.1016/j.neuroimage.2011.09.015.

[62] Karen E Joyce, Paul J Laurienti, Jonathan H Burdette, and Satoru Hayasaka. A new measure of centrality for brain networks. *PLoS One*, 5(8):e12200, 2010. doi:10.1371/journal.pone.0012200.

[63] David Kaufman and Robert Sweet. Contrast coding in least squares regression analysis. *American Educational Research Journal*, 11(4):359–377, 1974. doi:10.2307/1162790.

[64] Luke Keele. Causal mediation analysis: warning! Assumptions ahead. *American Journal of Evaluation*, 36(4):500–513, 2015. doi:10.1177/1098214015594689.

[65] Daniel J King, Stefano Seri, Richard Beare, Cathy Catroppa, Vicki A Anderson, and Amanda G Wood. Developmental divergence of structural brain networks as an indicator of future cognitive impairments in childhood brain injury: executive functions. *Developmental Cognitive Neuroscience*, page 100762, 2020.

[66] Arno Klein and Jason Tourville. 101 labeled brain images and a consistent human cortical labeling protocol. *Frontiers in Neuroscience*, 6, 2012. doi:10.3389/fnins.2012.00171.

[67] Eric D. Kolaczyk. *Statistical Analysis of Network Data*. Springer Series in Statistics. Springer-Verlag New York, 2009. ISBN 978-0-387-88146-1. doi:10.1007/978-0-387-88146-1.

[68] Eric D Kolaczyk and Gábor Csárdi. *Statistical analysis of network data with R*, volume 65. Springer, 2014. doi:10.1007/978-0-387-88146-1.

[69] Marsh Königs, LW van Heurn, Roel Bakx, R Jeroen Vermeulen, J Carel Goslings, Bwee Tien Poll-The, Marleen van der Wees, Coriene E Catsman-Berrevoets, Jaap Oosterlaan, and Petra JW Pouwels. The structural connectome of children with traumatic brain injury. *Human Brain Mapping*, 38(7):3603–3614, 2017. doi:10.1002/hbm.23614.

[70] Michael H Kutner, Chris Nachtsheim, John Neter, and William Li. *Applied linear statistical models*. McGraw-Hill Irwin, 2005. doi:10.2307/1271154.

[71] Athen Ma and Raúl J Mondragón. Rich-cores in networks. *PloS One*, 10(3):e0119678, 2015. doi:10.1371/journal.pone.0119678.

[72] Jennifer K MacCormack, Andrea G Stein, Jian Kang, Kelly S Giovanello, Ajay B Satpute, and Kristen A Lindquist. Affect in the aging brain: a neuroimaging meta-analysis of older vs. younger adult affective experience and perception. *Affective Science*, pages 1–27, 2020.

[73] Patrick Mair. *Analysis of fMRI Data*. Springer International Publishing, Cham, 2018. ISBN 978-3-319-93177-7. doi:10.1007/978-3-319-93177-7_14. URL https://doi.org/10.1007/978-3-319-93177-7_14.

[74] Nikos Makris, Jill M Goldstein, David Kennedy, Steven M Hodge, Verne S Caviness, Stephen V Faraone, Ming T Tsuang, and Larry J Seidman. Decreased volume of left and total anterior insular lobule in schizophrenia. *Schizophrenia Research*, 83(2):155–171, 2006. doi:10.1016/j.schres.2005.11.020.

[75] Anvita Gupta Malhotra, Sudha Singh, Mohit Jha, and Khushhali Menaria Pandey. A parametric targetability evaluation approach for vitiligo proteome extracted through integration of gene ontologies and protein interaction topologies. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–14, 2018. ISSN 1545-5963. doi:10.1109/TCBB.2018.2835459.

[76] Irengbam Rocky Mangangcha, Md Zubbair Malik, Ömer Küçük, Shakir Ali, and RK Brojen Singh. Identification of key regulators in prostate cancer from gene expression datasets of patients. *bioRxiv*, page 643643, 2019.

[77] Bryan FJ Manly. Randomization and regression methods for testing for associations with geographical, environmental and biological distances between populations. *Population Ecology*, 28(2):201–218, 1986.

[78] R Milo, N Kashtan, S Itzkovitz, MEJ Newman, and U Alon. On the uniform generation of random graphs with prescribed degree sequences. *eprint arXiv:cond-mat/0312028*, 2003.

[79] Mark Newman. *Networks: An Introduction*. Oxford University Press, 2010. ISBN 0199206651. doi:10.1093/acprof:oso/9780199206650.001.0001.

[80] Thomas E Nichols and Andrew P Holmes. Nonparametric permutation tests for functional neuroimaging: a primer with examples. *Human Brain Mapping*, 15(1):1–25, 2002. doi:10.1002/hbm.1058.

[81] William Stafford Noble. A quick guide to organizing computational biology projects. *PLoS Computational Biology*, 5(7):e1000424, 2009. doi:10.1371/journal.pcbi.1000424.

[82] Kyoko Ohashi, Carl M. Anderson, Elizabeth A. Bolger, Alaptagin Khan, Cynthia E. McGreenery, and Martin H. Teicher. Susceptibility or resilience to maltreatment can be explained by specific differences in brain network architecture. *Biological Psychiatry*, 85(8):690–702, 2019. ISSN 0006-3223. doi:https://doi.org/10.1016/j.biopsych.2018.10.016. URL http://www.sciencedirect.com/science/article/pii/S0006322318319784.

[83] Agustín Ostachuk. What is it like to be a crab? a complex network analysis of eucaridan evolution. *Evolutionary Biology*, 46(2):179–206, 2019.

[84] John E Overall and Douglas K Spiegel. Concerning least squares analysis of experimental data. *Psychological Bulletin*, 72(5):311, 1969. doi:10.1037/h0028109.

[85] Dimitrios Pantazis, Anand Joshi, Jintao Jiang, David W Shattuck, Lynne E Bernstein, Hanna Damasio, and Richard M Leahy. Comparison of landmark-based and automatic methods for cortical surface registration. *NeuroImage*, 49(3):2479–2493, 2010. doi:10.1016/j.neuroimage.2009.09.027.

[86] Jonathan D Power, Alexander L Cohen, Steven M Nelson, Gagan S Wig, Kelly Anne Barnes, Jessica A Church, Alecia C Vogel, Timothy O Laumann, Fran M Miezin, Bradley L Schlaggar, et al. Functional network organization of the human brain. *Neuron*, 72(4):665–678, 2011. doi:10.1016/j.neuron.2011.09.006.

[87] Kristopher J Preacher. Advances in mediation analysis: a survey and synthesis of new developments. *Annual Review of Psychology*, 66:825–852, 2015. doi:10.1146/annurev-psych-010814-015258.

[88] Arnau Ramos-Prats, Julia Kölldorfer, Elena Paolo, Maximilian Zeidler, Gabriele Schmid, and Francesco Ferraguti. An appraisal of the influence of the metabotropic glutamate 5 (mGlu5) receptor on sociability and anxiety. *Frontiers in Molecular Neuroscience*, 12:30, 2019. ISSN 1662-5099. doi:10.3389/fnmol.2019.00030. URL https://www.frontiersin.org/article/10.3389/fnmol.2019.00030.

[89] Neda Rashidi-Ranjbar, Tarek K Rajji, Sanjeev Kumar, Nathan Herrmann, Linda Mah, Alastair J Flint, Corrine E Fischer, Meryl A Butters, Bruce G Pollock, Erin W Dickie, et al. Frontal-executive and corticolimbic structural brain circuitry in older people with remitted depression, mild cognitive impairment, Alzheimer's dementia, and normal cognition. *Neuropsychopharmacology*, pages 1–14, 2020.

[90] Jaideep Ray, Ali Pinar, and Comandur Seshadhri. Are we there yet? When to stop a Markov chain while generating random graphs. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 153–164. Springer, 2012. doi:10.1007/978-3-642-30541-2_12.

[91] Jonas Richiardi, Andre Altmann, Anna-Clare Milazzo, Catie Chang, M Mallar Chakravarty, Tobias Banaschewski, Gareth J Barker, Arun LW Bokde, Uli Bromberg, Christian Büchel, et al. Correlated gene expression supports synchronous activity in brain networks. *Science*, 348(6240):1241–1244, 2015. doi:10.1126/science.1255905.

[92] Sally Richmond, Richard Beare, Katherine A. Johnson, Nicholas B. Allen, Marc L. Seal, and Sarah Whittle. Structural covariance networks in children and their associations with maternal behaviors. *NeuroImage*, 202:115965, 2019. ISSN 1053-8119. doi:https://doi.org/10.1016/j.neuroimage.2019.06.043. URL http://www.sciencedirect.com/science/article/pii/S1053811919305403.

[93] Gerard Robert Ridgway. *Statistical analysis for longitudinal MR imaging of dementia*. PhD thesis, UCL (University College London), 2009.

[94] James A Roberts, Alistair Perry, Gloria Roberts, Philip B Mitchell, and Michael Breakspear. Consistency-based thresholding of the human connectome. *NeuroImage*, 145:118–129, 2017. doi:10.1016/j.neuroimage.2016.09.053.

[95] James M Robins and Sander Greenland. Identifiability and exchangeability for direct and indirect effects. *Epidemiology*, pages 143–155, 1992. doi:10.1097/00001648-199203000-00013.

[96] ET Rolls, Marc Joliot, and Nathalie Tzourio-Mazoyer. Implementation of a new parcellation of the orbitofrontal cortex in the automated anatomical labelling atlas. *NeuroImage*, 122:1–5, 2015. doi:10.1016/j.neuroimage.2015.07.075.

[97] Donald B Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5):688, 1974. doi:10.1037/h0037350.

[98] Donald B Rubin. Bayesian inference for causal effects: the role of randomization. *The Annals of Statistics*, pages 34–58, 1978. doi:10.1214/aos/1176344064.

[99] Valentina Saba, Enrico Premi, Viviana Cristillo, Stefano Gazzina, Fernando Palluzzi, Orazio Zanetti, Roberto Gasparotti, Alessandro Padovani, Barbara Borroni, and Mario Grassi. Brain connectivity and information-flow breakdown revealed by a minimum spanning tree-based analysis of MRI data in behavioral variant frontotemporal dementia. *Frontiers in Neuroscience*, 13:211, 2019. ISSN 1662-453X. doi:10.3389/fnins.2019.00211. URL https://www.frontiersin.org/article/10.3389/fnins.2019.00211.

[100] Manish Saggar, SM Hadi Hosseini, Jennifer L Bruno, Eve-Marie Quintin, Mira M Raman, Shelli R Kesler, and Allan L Reiss. Estimating individual contribution from group-based structural correlation networks. *NeuroImage*, 120:274–284, 2015. doi:10.1016/j.neuroimage.2015.07.006.

[101] Mohammed Saqr, Jalal Nouri, Henriikka Vartiainen, and Matti Tedre. Robustness and rich clubs in collaborative learning groups: a learning analytics study using network science. *Scientific Reports*, 10 (1):1–16, 2020.

[102] Alexander Schaefer, Ru Kong, Evan M Gordon, Timothy O Laumann, Xi-Nian Zuo, Avram J Holmes, Simon B Eickhoff, and BT Yeo. Local-global parcellation of the human cerebral cortex from intrinsic functional connectivity MRI. *Cerebral Cortex*, pages 1–20, 2017. doi:10.1093/cercor/bhx179.

[103] Lianne H Scholtens, Marcel A de Reus, Siemon C de Lange, Ruben Schmidt, and Martijn P van den Heuvel. An mri von economo–koskinas atlas. *NeuroImage*, 170:249–256, 2018. doi:10.1016/j.neuroimage.2016.12.069.

[104] Ronald C Serlin and Joel R Levin. Teaching how to derive directly interpretable coding schemes for multiple regression analysis. *Journal of Educational Statistics*, 10(3):223–238, 1985. doi:10.2307/1164794.

[105] David W Shattuck and Richard M Leahy. Brainsuite: an automated cortical surface identification tool. *Medical Image Analysis*, 6(2):129–142, 2002. doi:10.1016/S1361-8415(02)00054-3.

[106] David W Shattuck, Mubeena Mirza, Vitria Adisetiyo, Cornelius Hojatkashani, Georges Salamon, Katherine L Narr, Russell A Poldrack, Robert M Bilder, and Arthur W Toga. Construction of a 3d probabilistic atlas of human cortical structures. *NeuroImage*, 39(3):1064–1080, 2008. doi:10.1016/j.neuroimage.2007.09.031.

[107] Xilin Shen, F Tokoglu, Xenios Papademetris, and R Todd Constable. Groupwise whole-brain parcellation from resting-state fMRI data for network node identification. *NeuroImage*, 82:403–415, 2013. doi:10.1016/j.neuroimage.2013.05.081.

[108] Stephen Smith, Mark Jenkinson, Christian Beckmann, Karla Miller, and Mark Woolrich. Meaningful design and contrast estimability in FMRI. *NeuroImage*, 34(1):127–136, 2007. doi:10.1016/j.neuroimage.2006.09.019.

[109] AW Still and AP White. The approximate randomization test as an alternative to the F test in analysis of variance. *British Journal of Mathematical and Statistical Psychology*, 34(2):243–252, 1981.

[110] Toshiyuki Tanimizu, Justin W Kenney, Emiko Okano, Kazune Kadoma, Paul W Frankland, and Satoshi Kida. Functional connectivity of multiple brain regions required for the consolidation of social recognition memory. *Journal of Neuroscience*, 37(15):4103–4116, 2017. doi:10.1523/JNEUROSCI.3451-16.2017.

[111] Martin H Teicher, Kyoko Ohashi, and Alaptagin Khan. Additional insights into the relationship between brain network architecture and susceptibility and resilience to the psychiatric sequelae of childhood maltreatment. *Adversity and Resilience Science*, 1:49–64, 2020.

[112] Qawi K Telesford, Karen E Joyce, Satoru Hayasaka, Jonathan H Burdette, and Paul J Laurienti. The ubiquity of small-world networks. *Brain Connectivity*, 1(5):367–375, 2011. doi:10.1089/brain.2011.0038.

[113] Dustin Tingley, Teppei Yamamoto, Kentaro Hirose, Luke Keele, and Kosuke Imai. mediaton: R package for causal mediation analysis. *Journal of Statistical Software*, 59(5):1–38, 2014. doi:10.18637/jss.v059.i05.

[114] Basant K Tiwary. Computational medicine: quantitative modeling of complex diseases. *Briefings in Bioinformatics*, 0:1–12, 01 2019. ISSN 1477-4054. doi:10.1093/bib/bbz005. URL `https://doi.org/10.1093/bib/bbz005`.

[115] Nathalie Tzourio-Mazoyer, Brigitte Landeau, Dimitri Papathanassiou, Fabrice Crivello, Olivier Etard, Nicolas Delcroix, Bernard Mazoyer, and Marc Joliot. Automated anatomical labeling of activations in SPM using a macroscopic anatomical parcellation of the MNI MRI single-subject brain. *NeuroImage*, 15(1):273–289, 2002. doi:10.1006/nimg.2001.0978.

[116] Martijn van den Heuvel, Siemon de Lange, Andrew Zalesky, Caio Seguin, Thomas Yeo, and Ruben Schmidt. Proportional thresholding in resting-state fMRI functional connectivity networks and consequences for patient-control connectome studies: Issues and recommendations. *NeuroImage*, 2017. doi:10.1016/j.neuroimage.2017.02.005.

[117] Martijn P. van den Heuvel, René C. W. Mandl, Cornelis J. Stam, René S. Kahn, and Hilleke E. Hulshoff Pol. Aberrant frontal and temporal complex network structure in schizophrenia: a graph theoretical analysis. *Journal of Neuroscience*, 30(47):15915–15926, 2010. ISSN 0270-6474. doi:10.1523/JNEUROSCI.2874-10.2010. URL `http://www.jneurosci.org/content/30/47/15915`.

[118] Estefania Ruiz Vargas, Lindi M Wahl, et al. The gateway coefficient: a novel metric for identifying critical connections in modular networks. *The European Physical Journal B*, 87(7):1–10, 2014. doi:10.1140/epjb/e2014-40800-7.

[119] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. doi:10.1007/978-0-387-21706-2. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0.

[120] Umesh M Venkatesan and Frank G Hillary. Functional connectivity within lateral posterior parietal cortex in moderate to severe traumatic brain injury. *Neuropsychology*, 33(6):893, 2019.

[121] Fabien Viger and Matthieu Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. *Journal of Complex Networks*, 4(1):15–37, 2016. doi:10.1093/comnet/cnv013.

[122] Defeng Wang, Lin Shi, Shangping Liu, Steve CN Hui, Yongjun Wang, Jack CY Cheng, and Winnie CW Chu. Altered topological organization of cortical network in adolescent girls with idiopathic scoliosis. *PLoS One*, 8:83767, 2013. doi:10.1371/journal.pone.0083767.

[123] Ruopeng Wang, Thomas Benner, Alma Gregory Sorensen, and Van Jay Wedeen. Diffusion toolkit: a software package for diffusion imaging data processing and tractography. In *Proc Intl Soc Mag Reson Med*, volume 15. Berlin, 2007.

[124] Christopher G. Watson, Christian Stopp, Jane W. Newburger, and Michael J. Rivkin. Graph theory analysis of cortical thickness networks in adolescents with d-transposition of the great arteries. *Brain and Behavior*, 8(2):e00834, 2018. ISSN 2162-3279. doi:10.1002/brb3.834. URL http://dx.doi.org/10.1002/brb3.834.

[125] Christopher G. Watson, Dana DeMaster, and Linda Ewing-Cobbs. Graph theory analysis of DTI tractography in children with traumatic injury. *NeuroImage. Clinical*, 21:101673, 2019. ISSN 2213-1582. doi:10.1016/j.nicl.2019.101673. URL http://www.sciencedirect.com/science/article/pii/S2213158219300233.

[126] Duncan J Watts and Steven H Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393 (6684):440–442, 1998. doi:10.1515/9781400841356.301.

[127] Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10), 2014. doi:10.18637/jss.v059.i10.

[128] Ben James Winer, Donald R Brown, and Kenneth M Michels. *Statistical principles in experimental design*, volume 2. McGraw-Hill New York, 1971. doi:10.2307/3172747.

[129] Anderson M Winkler, Gerard R Ridgway, Matthew A Webster, Stephen M Smith, and Thomas E Nichols. Permutation inference for the general linear model. *NeuroImage*, 92:381–397, 2014. doi:10.1016/j.neuroimage.2014.01.060.

[130] Mingrui Xia, Jinhui Wang, Yong He, et al. Brainnet viewer: a network visualization tool for human brain connectomics. *PLoS One*, 8(7):e68910, 2013. doi:10.1371/journal.pone.0068910.

[131] Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. doi:10.1201/b15166. URL http://yihui.name/knitr/. ISBN 978-1498716963.

[132] Chao-Gan Yan and Yu-Feng Zang. DPARSF: a MATLAB toolbox for "pipeline" data analysis of resting-state fMRI. *Frontiers in Systems Neuroscience*, 4:13, 2010. doi:10.3389/fnsys.2010.00013.

[133] Andrew Zalesky, Alex Fornito, and Edward T Bullmore. Network-based statistic: identifying differences in brain networks. *NeuroImage*, 53(4):1197–1207, 2010. doi:10.1016/j.neuroimage.2010.06.041.

[134] Andrew Zalesky, Alex Fornito, and Ed Bullmore. On the use of correlation as a measure of network connectivity. *NeuroImage*, 60(4):2096–2106, 2012. doi:10.1016/j.neuroimage.2012.02.001.

[135] Shi Zhou and Raúl J Mondragón. The rich-club phenomenon in the internet topology. *IEEE Communications Letters*, 8(3):180–182, 2004. doi:10.4018/978-1-59140-993-9.ch066.