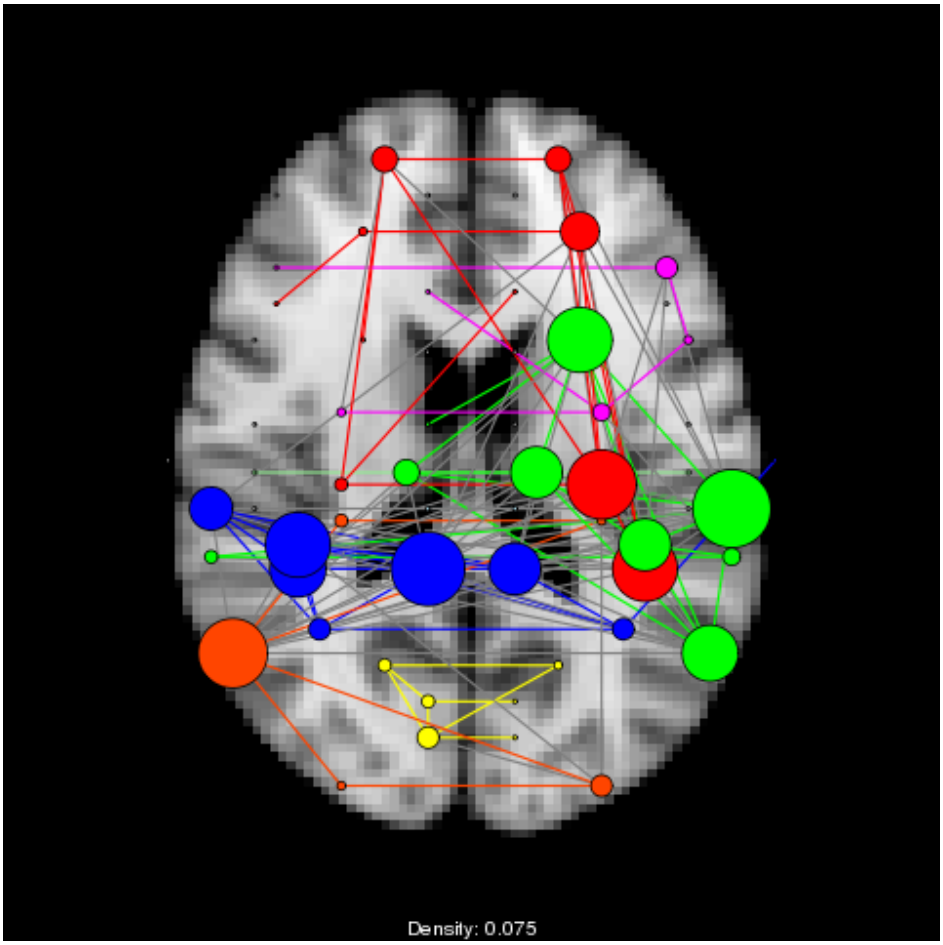
# brainGraph User Guide
Version 2.4.1

Christopher G. Watson, Ph.D.

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

August 18, 2018



# 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` (18)) and resting-state fMRI (e.g., `DPABI` (99) and `AFNI` (14)). 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.

## 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. (25) 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 Aug. 2018), 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` (98) 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]

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

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 LaTeXbecause 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 v2.x.x**

This box indicates a major change to `brainGraph` that was introduced in v2.0.0 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.

# 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 (17) 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. (92). You also can get some citation information with the following code:

```
citation('brainGraph')

##
## To cite package 'brainGraph' in publications use:
##
##   Christopher G. Watson (2018). brainGraph: Graph Theory Analysis
##   of Brain MRI Data. R package version 2.4.1.
##   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 = {2018},
##     note = {R package version 2.4.1},
##     url = {https://github.com/cwatson/brainGraph},
##   }
```

## Publications using `brainGraph`

- Caligiuri et al. (10)
- Barbagallo et al. (3)
- Tanimizu et al. (82)
- Cinelli et al. (12)
- Watson et al. (92)
- Janes et al. (45)
- Malhotra et al. (58)

# Part I

# Introductory Material

# 1

# Installation and Requirements

## 1.1 System Requirements

There aren't any, specifically. 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 was also done on 64-bit Windows 7.

**Packages** There are several packages that are required/recommended:

- `igraph` (current version is v1.2.2)
- `foreach` and `doMC`/`doSNOW` (multicore package for Linux/Windows)
- `oro.nifti` (a general *NIfTI* library)
- `data.table` (excellent general-purpose package)
- `RcppEigen` (for fast linear model calculations)
- `ggplot2`, `scales`, `ggrepel`, `gridExtra` (plotting-related)
- `Matrix`, `abind`, `expm` (functionality for matrices and multi-dimensional arrays)
- `boot` (for bootstrapping)
- `permute` (to generate random permutations)
- `Hmisc`, `MASS`, `ade4` (general-purpose packages for data analysis; `MASS` supports the Venables & Ripley textbook *Modern Applied Statistics with S*)(88)
- `mediation` (causal mediation analysis)
- `RGtk2` and `cairoDevice` (for the GUI)

**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 OS-specific instructions

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.

```
install.packages(devtools)
devtools::install_github('cwatson/brainGraph')
```

**New in v2.2.0**

As of `v2.2.0`, the graphical packages (`RGtk2` and `cairoDevice`) are no longer required. So, you will be able to install `brainGraph` on a headless server, or if you are having trouble installing either of those two (see the next few sections), you can still use all of the non-GUI functionality in the package. Thanks to `@michaelhallquist` for his contribution!

### 1.2.1 Linux

You may download the tarball and on the command line, type:

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

### 1.2.2 Mac

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

**macOS 10.13 High Sierra** see this GitHub Gist. Of note, look at the comments for more information.

**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.

As I mention above (under *macOS 10.13*), see the following link if you run into an issue with installing `RGtk2`: `https://gist.github.com/sebkopf/9405675#macos`. The comments section may contain helpful information as well.

### 1.2.3 Windows

You should first try to download directly from *CRAN*. If that causes any problems, I recommend using `devtools` and installing directly from *Github*. See also the URL listed under the *Mac* section for some instructions when it comes to installing `RGtk2`. You may end up needing to add/edit a couple *environment variables*.

| Atlas name | R object | Ref. |
|---|---|---|
| Desikan-Killiany | `dk` | (21) |
| Desikan-Killiany-Tourville | `dkt` | (51) |
| Destrieux | `destrieux` | (22) |
| FS atlases w/ scgm | `dk.scgm`, `dkt.scgm`, `destrieux.scgm` | |
| AAL | `aal90`, `aal116` | (85) |
| AAL2 | `aal2.94`, `aal2.120` | (71) |
| Dosenbach | `dosenbach160` | (23) |
| Craddock-200 | `craddock200` | (15) |
| Harvard-Oxford | `hoa112` | (57) |
| LONI probabilistic brain atlas | `lpba40` | (79) |
| Brainsuite | `brainsuite` | (64, 78) |

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

- Packages that *may* need to be installed manually are: `gWidgets`, `gWidgetsRGtk2`, `RGtk2Extras`
- 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')
```

## 1.3 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. (99))
- PANDA (for DTI analyses; see Ref. (18))
- TrackVis (for DTI analyses; see Ref. (91))

## 1.4 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. 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>
```

### 1.4.1 Using your own atlas: required format

If you have an atlas you would like to use, then just follow the data structure shown above, and it *should* work without issue. 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` is the only atlas to contain 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] See the structure of other atlases by typing e.g. `str(dosenbach160)`.

### 1.4.2 Atlases to be added

I intend on adding more atlases in the future. The following list is a work-in-progress.

- *Power-264* (65)
- *Gordon-333* (34)
- *Shen-268* (80)
- *Von Economo – Koskinas* (76)
- *Brainnetome* (30)
- *Willard-499* (68)
- *Schaefer-400* (75)

# 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 (60). 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 (52). To learn more about network statistics using `igraph` specifically, I recommend Kolaczyk & Csardi (2014) (part of the *Use R!* series) (53).

| 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 $\approx 0.03$ (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)
```

Beyond checking just network-related measures, I also (painfully) imported my cortical thickness data and covariates into `Matlab`, and used `glmfit` to get the residuals. The results were nearly identical; the only reason they weren't a perfect match is because `Matlab` returns the *Pearson* residuals, whereas I use the *studentized* residuals (and I couldn't be bothered to figure out how to calculate those in `Matlab`).

And, as a further sanity check, I asked a colleague who is a biostatistician to confirm my results. Even though he used `SAS`, and did partial correlations (as opposed to linear models), the results were nearly identical. The median difference in correlation thresholds across all densities tested was 0.003, and the maximum was 0.007.

# 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

Here, I assume there are 2 subject groups. The atlas being used in this section of the manual is the *Desikan-Killiany (DK)* atlas (21).

### 3.1.1 aparcstats2table

First, use `aparcstats2table` to get mean cortical thickness for each region and each subject into a single file. Replace the `SUBJECTS` variable definition with something that makes sense for your data. I have a script that wraps this step among others which makes it easy to change the parcellation or the measure (thickness, area, lgi, volume) on the command line.

**Loop through subjects and run `aparcstats2table`**

```
parc='aparc'    # Or 'aparc.DKTatlas40', or 'aparc.a2009s'
subjects=$(ls -d [[:lower:]]*[a-z0-9]*/ | grep -v fsaverage)
for h in 'lh rh'; do
    aparcstats2table --subjects ${subjects} --hemi ${h}
        --meas thickness -p ${parc}
        --delimiter=comma
        --tablefile ${parc}_${h}_thick.csv
done
```

### 3.1.2 sed

Next, use `sed` to abbreviate the brain regions' names, and to remove the final column (which, if present, is the mean thickness across the hemisphere; this is optional). This will need to be repeated for the RH thickness file, and these arguments should work for the `DK`, `DKT`, and `destrieux` atlases. It is probably a good

idea to copy the following into a text file and use with the `-e` option to `sed`. I have it all in a script that also can accept the `asegstats.csv` file, if you wish to include subcortical volumes.[1]

**Abbreviate region names**

```
SED_ARGS='
s/lh_/l/g
s/rh_/r/g
s/_thickness//g
s/_volume//g
s/_area//g
s/\///
s/bankssts/BSTS/g
s/caudalanteriorcingulate/cACC/g
s/caudalmiddlefrontal/cMFG/g
s/precuneus/PCUN/g
s/cuneus/CUN/g
s/entorhinal/ENT/g
s/fusiform/FUS/g
s/inferiorparietal/IPL/g
s/inferiortemporal/ITG/g
s/isthmuscingulate/iCC/g
s/lateraloccipital/LOG/g
s/lateralorbitofrontal/LOF/g
s/lingual/LING/g
s/medialorbitofrontal/MOF/g
s/middletemporal/MTG/g
s/parahippocampal/PARH/g
s/paracentral/paraC/g
s/parsopercularis/pOPER/g
s/parsorbitalis/pORB/g
s/parstriangularis/pTRI/g
s/pericalcarine/periCAL/g
s/postcentral/postC/g
s/posteriorcingulate/PCC/g
s/precentral/preC/g
s/rostralanteriorcingulate/rACC/g
s/rostralmiddlefrontal/rMFG/g
s/superiorfrontal/SFG/g
s/superiorparietal/SPL/g
s/superiortemporal/STG/g
s/supramarginal/SMAR/g
s/frontalpole/FP/g
s/temporalpole/TP/g
s/transversetemporal/TT/g
s/insula/INS/g
s/_div//g
s/L\./l/g
s/R\./r/g
s/Thalamus.Proper/THAL/g
s/Putamen/PUT/g
s/Pallidum/PALL/g
```

[1] Script available on request.

```
s/Caudate/CAUD/g
s/Hippocampus/HIPP/g
s/Amygdala/AMYG/g
s/Accumbens.area/ACCU/g
'

awk -F, 'NR == 1 {$1="Study.ID"}1' OFS=, aparc_lh_thick.csv >> tmp1.csv
sed -i -e "${SED_ARGS}" tmp1.csv

# Remove the last column (if the mean thickness column is still there):
cat tmp1.csv | cut -d',' -f1-35 >> lh.csv
rm tmp1.csv
```

The last few commands above are necessary if you are using a version of `Freesurfer` that outputs the mean cortical thickness. The `-f1-35` above is specific to the *DK* atlas; for *DKT* it should be 32, and for *Destrieux* it should be 75. These steps then need to be repeated for the second group (and if you have more groups). However, running the code a single time is sufficient if all subjects are in a single directory.

Now combine group files. If you have 2 groups: (repeat for the RH file as well)

```
tail -n +2 lhThick_group2.csv >> lh_dk_thickness.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.

### 3.2.1 Create/convert parcellated volume

First, the parcellation volume must be created if it doesn't exist, and converted to `NIfTI`.[2] 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:

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

**ROI label file**

```
1001 1001_lBSTS
1002 1002_lcACC
1003 1003_lcMFG
...
```

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`.

**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
        --ventricles -o ventricles.nii.gz
```

The final step of the `for` 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 (17) (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.

### 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 Import data and create connectivity matrices

The operations/functions for this step differ depending on the type of networks you have: for covariance networks, you import the region-wise brain metrics (e.g., cortical thickness; see Initialize variables) and create matrices from partial correlations, linear model residuals, etc. (see Correlation Matrix and Graph Creation). The same function that performs the correlations will also threshold the matrices. For single-subject networks (DTI tractography or resting state fMRI), you load the connectivity matrices from your neuroimaging software-of-choice (see Setting up); typically those software tools can generate the matrix for you (e.g., *fdt_network_matrix* from `FSL`'s *probtrackx2*). Next, you threshold these matrices based on criteria of your choosing (see Import, normalize, and filter matrices for all subjects).

### 4.1.3 Create graphs and calculate metrics

These operations are slightly different for covariance networks (Graph creation) and single-subject networks (Graph creation), as the nested *list* object will have a third "level" in the case of single-subject networks (group, threshold/density, and subject). In those same sections, I show how to calculate graph metrics of interest.

### 4.1.4 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 Visualize results

There is a GUI for quick manipulation and inspection of graphs, as well as several functions 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

**make_brainGraph** Creates a `brainGraph` graph object, given an `igraph` graph object. This assigns several attributes that are specific to brain MRI data (mostly related to the brain atlas in use).

**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 `make_empty_graph`.

**make_ego_brainGraph** Creates a graph of the union of multiple vertex neighborhoods. This is analogous to `make_ego_graph`.

**make_glm_brainGraph** Creates a graph specific to a GLM (or MTPC) analysis. See Vertex-wise group analysis (GLM) (or Multi-threshold permutation correction).

**make_nbs_brainGraph** Creates a graph specific to NBS analysis. See Network-based statistic (NBS).

**make_mediate_brainGraph** Creates a graph specific to mediation analysis. See Graph- and vertex-level mediation analysis.

### 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` (48)
- global, local, and nodal efficiency; `efficiency`
- *Vertex roles*; `gateway_coeff` (87), `part_coeff` and `within_module_degree_z_score` (35)
- Rich club calculations; `rich_club_coeff`, `rich_club_norm`, and `rich_core` (56)
- The *s-core* of vertices; `s_core` (26)
- Euclidean distances; `edge_spatial_dist` and `vertex_spatial_dist`
- Communicability; `communicability` (16, 27) and `centr_betw_comm` (28)
- Vertex vulnerability; `vulnerability` (1, 38)
- Edge counts; `count_homologous` (counts the number of edges between homologous brain regions) and `count_interlobar` (counts the number of edges from one lobe to all others)

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*.

| Class name | Creation function | Description |
|---|---|---|
| brainGraph | make_brainGraph | Any graph with certain attributes (see text) |
| brainGraph_GLM | make_glm_brainGraph | Graphs from GLM analysis |
| brainGraph_NBS | make_nbs_brainGraph | Graphs from NBS analysis |
| brainGraph_mtpc | make_glm_brainGraph | Graphs from MTPC analysis |
| brainGraph_mediate | make_mediate_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) |

Table 4.1: Class names and graph creation functions.

### 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)* (101): `NBS`. See Network-based statistic (NBS) for details.
- *Multi-threshold permutation correction (MTPC)* (24) 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. (74)
- Targeted attack and failure analyses: `robustness`. See the "Robustness" section in Further analysis for implementation and plotting.

### 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 (59). 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. (2), and is particularly important for partial correlation networks (e.g., cortical thickness correlations; see Refs. Hosseini and Kesler (39), Zalesky et al. (101)). 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.

## BOX 4.1 Classes and methods

### New in v2.0.0

I have introduced some simple *classes* and *methods* corresponding to different functions/analyses for `brainGraph`.

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 the objects with a special class 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*: you do not need to remember the full names of these classes; you can simply type the base method name 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 `set_brainGraph_attr`.

**bg_GLM** This class contains results from `brainGraph_GLM` (see Vertex-wise group analysis (GLM)).

**brainGraph_GLM** This class is the graph associated with `bg_GLM` objects. It has GLM-specific attributes (for plotting), created by the function `make_glm_brainGraph`.

**NBS** This class contains results from `NBS` (see Network-based statistic (NBS)).

**brainGraph_NBS** This class is the graph associated with `NBS` objects. It has NBS-specific attributes (for plotting), created by the function `make_nbs_brainGraph`.

**mtpc** This class contains results from `mtpc` (see Multi-threshold permutation correction).

**brainGraph_mtpc** This class is the graph associated with `mtpc` objects. It is also created by `make_glm_brainGraph`.

**bg_mediate** This class contains results from `brainGraph_mediate` (see Graph- and vertex-level mediation analysis).

**brainGraph_mediate** This class is the graph associated with `bg_mediate` objects. It has mediation-specific attributes (for plotting), created by the function `make_mediate_brainGraph`.

**brainGraph_boot** This class is specific to objects returned by `brainGraph_boot` (formerly `boot\_global`). See Bootstrapping for details.

**brainGraph_permute** This class is specific to objects returned by `brainGraph_permute` (formerly `permute.group`). See Permutation testing for details.

**brainGraph_resids** This class is specific to objects returned from `get.resid` for structural covariance networks (see Structural covariance networks). The new `plot` method replaces the old function `check.resid`.

# 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 (62) 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 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')
```

### 5.1.3 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 Control   F 14.17  Site 1
##   2:        2 Control   F 14.58  Site 1
##   3:        3 Control   F 16.42  Site 1
##   4:        4 Control   M 14.17  Site 1
##   5:        5 Control   F 16.33  Site 1
##  ---
## 137:      137 Patient   M 16.50  Site 1
## 138:      138 Patient   M 16.33  Site 1
## 139:      139 Patient   M 16.42  Site 1
## 140:      140 Patient   M 15.58  Site 2
## 141:      141 Patient   M 15.42  Site 1

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": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Sex     : Factor w/ 2 levels "F","M": 1 1 1 2 1 1 2 2 2 1 ...
##  $ Age     : num  14.2 14.6 16.4 14.2 16.3 ...
##  $ 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])
```

### 5.1.4 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 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** The package version used to create the 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 `modality='sld'`, it will print `Streamline density`)

**Density** The graph density (percent)

**Subject ID** The `Study.ID`, if one was supplied to `make_brainGraph` or `set_brainGraph_attrs`

**Group** The subject group, if one was supplied

```
summary(g.ex)

##
##  brainGraph version:  2.4.1
##  Brain atlas used:    Desikan-Killiany
##  Imaging modality:    Cortical thickness
##  Edge weighting:      Unweighted
##  Clustering method:   Louvain (multi-level modularity optimization)
##  Graph density:       15.72%
##  Subject ID:          N/A
##  Group:               Eg

## graph attributes-----------------------------

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

```
## vertex attributes----------------------------

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

## edge attributes----------------------------

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

### 5.2.2 Graph-level attributes

Graph-level attributes can be of any data type (e.g., a character string specifying the atlas used, or a numeric specifying the global efficiency, etc.). They are visible when you print the graph (by typing the object name), or with the following command:

```
graph_attr_names(g.ex)

##  [1] "Cp"                      "Lp"
##  [3] "rich"                    "E.global"
##  [5] "clust.method"            "mod"
##  [7] "density"                 "conn.comp"
##  [9] "max.comp"                "num.tri"
## [11] "diameter"                "transitivity"
## [13] "assortativity"           "atlas"
## [15] "version"                 "modality"
## [17] "Group"                   "assortativity.lobe"
## [19] "assortativity.lobe.hemi" "asymm"
## [21] "spatial.dist"            "num.hubs"
## [23] "E.local"                 "vulnerability"
```

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. (13), Zhou and Mondragón (102)). 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) $k = 1, 2, \cdots, 18$; $R$ is

[2] Calculated by the `igraph` function `components`, and set by `set_brainGraph_attr`

[3] See Rich-club Analysis for more details.

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 0.1572 68 358
##  2:   2 0.1572 68 358
##  3:   3 0.1572 68 358
##  4:   4 0.1572 68 358
##  5:   5 0.1654 65 344
##  6:   6 0.1700 63 332
##  7:   7 0.1794 59 307
##  8:   8 0.2041 49 240
##  9:   9 0.2348 39 174
## 10:  10 0.2557 33 135
## 11:  11 0.2821 27  99
## 12:  12 0.3083 16  37
## 13:  13 0.2727 11  15
## 14:  14 0.3214  8   9
## 15:  15 0.0000  4   0
## 16:  16    NaN  1   0
## 17:  17    NaN  1   0
## 18:  18    NaN  0   0
```

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 3d8b20d 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).

```
vertex_attr_names(g.ex)

##  [1] "degree"             "name"             "lobe"
##  [4] "lobe.hemi"          "hemi"             "x.mni"
##  [7] "x"                  "y.mni"            "y"
## [10] "z.mni"              "z"                "color.lobe"
## [13] "circle.layout"      "asymm"            "dist"
## [16] "dist.strength"      "knn"              "Lp"
## [19] "btwn.cent"          "hubs"             "ev.cent"
## [22] "lev.cent"           "k.core"           "transitivity"
```

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

```
## [25] "E.local"            "E.nodal"            "vulnerability"
## [28] "eccentricity"       "comm"               "color.comm"
## [31] "comp"               "color.comp"         "circle.layout.comm"
## [34] "GC"                 "PC"                 "z.score"

V(g.ex)$degree

##  [1]  8  6  8  9  6  9 11 12 11  5 12  7 12 10  9 12 12 15 10  5 11 11 16
## [24] 12 10 16 13  8 16  8  7 13  8  9 12 18 15  8 12 14  7  8 10  5 12 11
## [47] 14  9 15 12 13  9  9  8  9 10 12 13  9 13 15  7  9  8 11  8 10 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]

## + 5/358 edges from b52904a (vertex names):
## [1] lBSTS--lITG   lBSTS--lpostC lBSTS--lPCC   lBSTS--lrMFG  lBSTS--lSMAR

edge_attr_names(g.ex)

## [1] "color.lobe" "dist"       "btwn"       "color.comm" "color.comp"

head(E(g.ex)$btwn)

## [1] 12.586  9.031 10.056 10.640 19.033 13.051
```

## 5.3 Community detection

There are multiple community detection algorithms available in `igraph`. You can run community detection on your own or through `set_brainGraph_attr`. The benefit of the latter is that an appropriate method will be chosen depending on the type of input graph; for details, see the following text box.

**New in v2.4.0**

You can choose whichever method you like via the `clust.method` argument to `set_brainGraph_attr`. All clustering (i.e., community detection) 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 default is the *Louvain* algorithm (see Ref. Blondel et al. (9)), 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. If there are any negative edge weights, and you select anything other than `'walktrap'` or `'spinglass'`, then `'walktrap'` is selected. If `'edge_betweenness'` is selected, the edges are first transformed (using `xfm.weights`) 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 (32). 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))
```

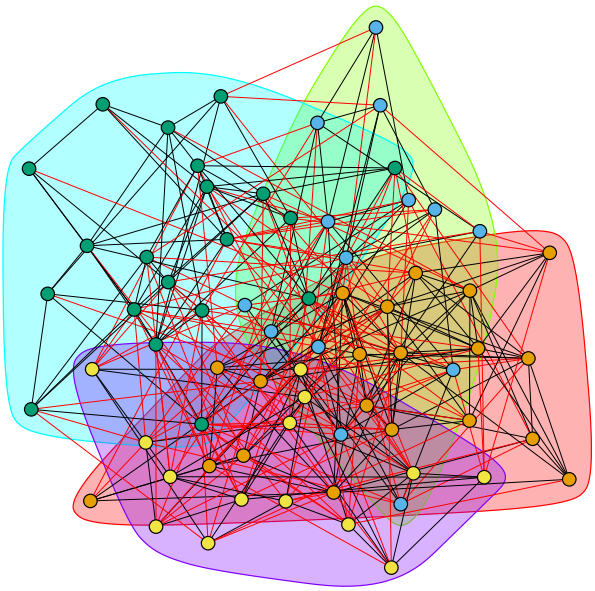


Figure 5.1: Communities plot, Fruchterman-Reingold layout.

## 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

[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).

can be chosen using the `hemi` argument. Additonally, you may choose to show a legend for vertex colors, using the `show.legend` argument.

**New in v2.0.0**

The `plot_brainGraph_mni` function has been removed; its functionality is now the default behavior in the `plot` method. You can specify `mni=FALSE` to omit showing the brain slice.

### 5.4.1 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')
```

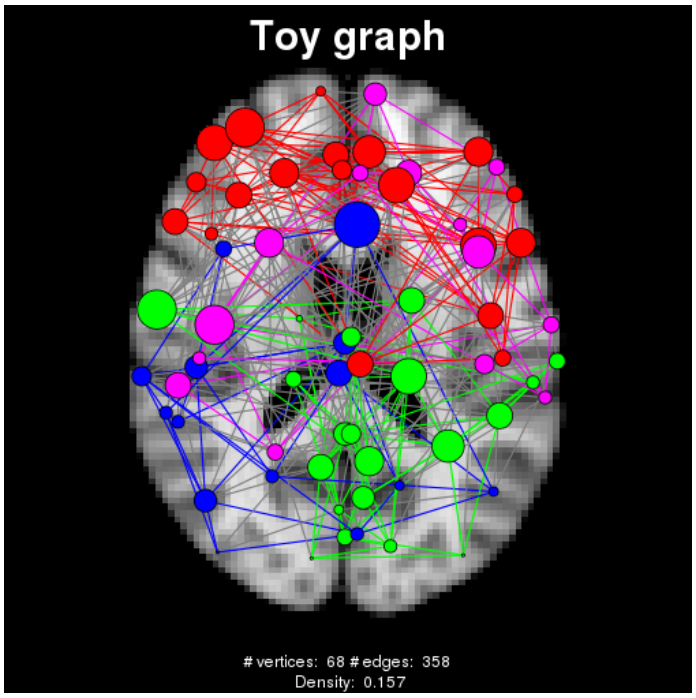



Figure 5.2: Axial view; vertex colors signify community membership.

### 5.4.2 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.

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

[7] Axial images are always in neurological orientation

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

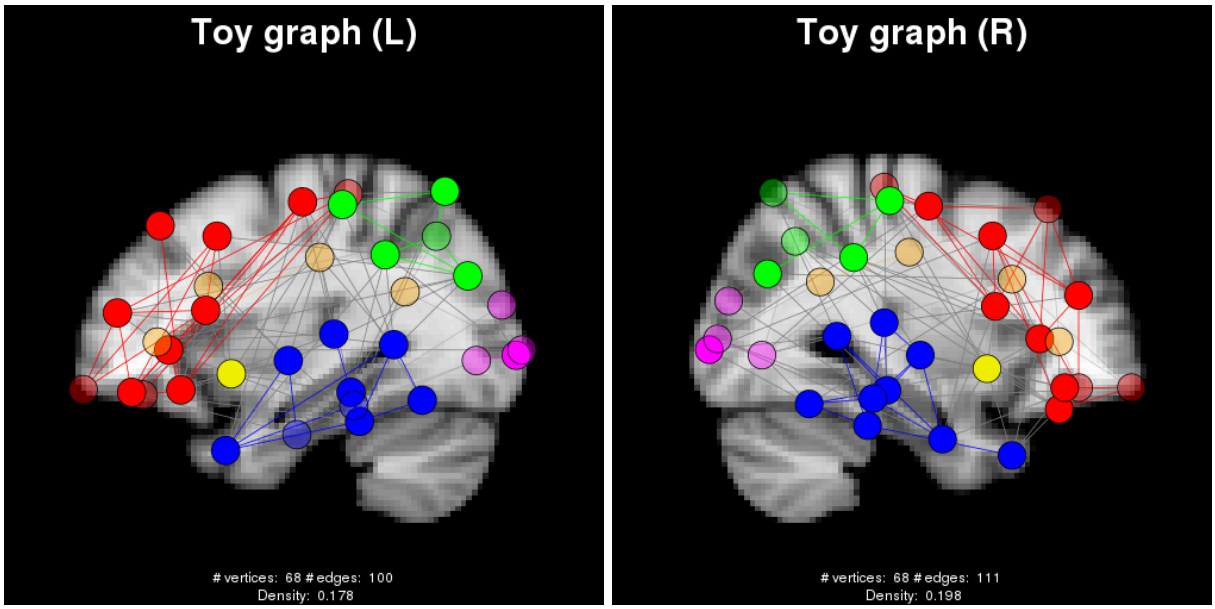



Figure 5.3: Sagittal view; vertex colors signify lobe membership.

### 5.4.3 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)
```

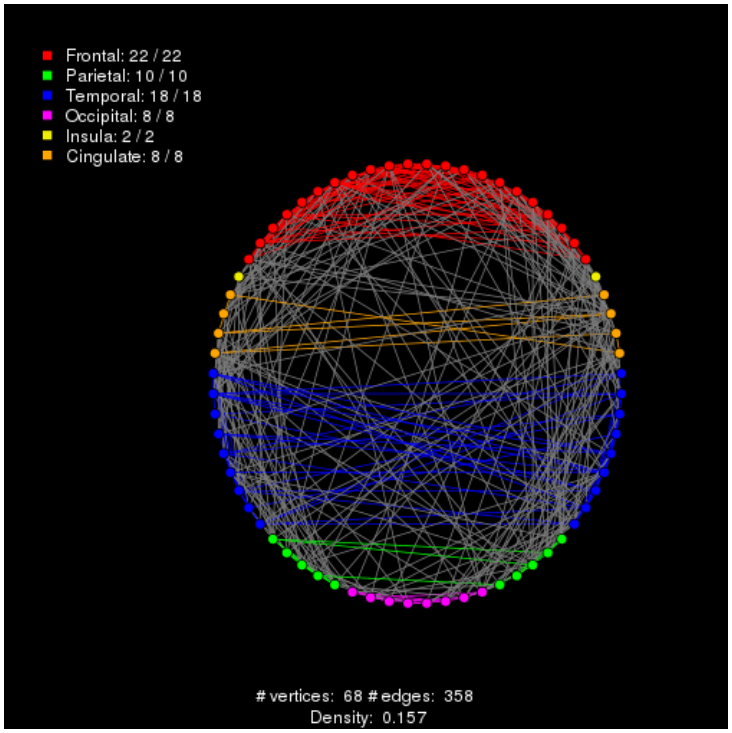



Figure 5.4: Circular view; vertex colors signify lobe membership.

# 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)*.

## 6.1 Overview

The following is the sequence of steps for completing the first steps of structural covariance network analysis. Each of these are 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 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.)

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')
init.vars <- brainGraph_init(atlas='dk', densities=seq(0.05, 0.20, 0.01),
                             datadir=datadir, modality='thickness',
                             exclude.subs=exclude.subs)
lapply(seq_along(init.vars), function(x)
       assign(names(init.vars)[x], eval(init.vars[[x]]), envir=.GlobalEnv))

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

# 3. Correlation matrices
```

```
corrs <- corr.matrix(all.dat.resids, densities=densities)

# 4. Create graphs and calculate graph metrics
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(groups))

# 5. data.table's of the metrics
dt.G <- rbindlist(lapply(g, graph_attr_dt))
dt.V <- rbindlist(lapply(g, function(x) rbindlist(lapply(x, vertex_attr_dt))))
```

## 6.2 Initialize variables

The following code block is what I put in my `01_load_project.R` script, and likely will be project-specific. There is a call to an "initialization function", `brainGraph_init`, that will create and return the variables needed for later steps.

You will need to have several files in the `datadir`:

- `covars.csv` Needs a field called `Study.ID`. The existence of this file on your system is required unless you supply a `data.table` of covariates yourself.
- `lh_${atlas}_${modality}.csv` See Structural data from Freesurfer for creating this file.
- `rh_${atlas}_${modality}.csv`
- `scgm.csv` Only required if you are including subcortical data; e.g., subcortical volumes to be analyzed along with cortical thickness. This should be in the same format as the cortical data.
- `covars.scgm.csv` Only required for SCGM data. This file may be different from `covars.csv` if you have additional covariates. A common usage would be to also include *Total Intracranial Volume* in the linear models for SCGM volumes, but not for cortical thickness.

```
datadir <- paste0('~/Dropbox/packages/brainGraph.other/data/Patient')

# 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)
init.vars <- brainGraph_init(atlas='dk', densities=seq(0.05, 0.20, 0.01),
                             datadir=datadir, modality='thickness',
                             exclude.subs=exclude.subs)

# Hack to place list elements in the global environment
lapply(seq_along(init.vars), function(x)
       assign(names(init.vars)[x], eval(init.vars[[x]]), envir=.GlobalEnv))

# In case one density in particular is of interest, here 10%
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`

## 6.3 Model residuals

**Note**

The functions `get.resid` and `corr.matrix` have changed quite a bit in `v2.1.0`. You can now perform linear models on a per-group basis, or across all groups at once, "natively" (i.e., without more code). See the next section for info on correlation matrices.

The next step is to get the model residuals. `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 `brainGraph_init`.

**covars** A `data.table` of covariates. It must have both a `Study.ID` and a `Group` column.

**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'`).

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

**exclude** 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:

**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**

**all.dat.tidy** The "tidied" data, which is a combination of `covars`, the structural data (in the `value` column) and the residuals.

**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).

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

### 6.3.3 Example

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

```
all.dat.resids <- get.resid(lhrh, covars=covars, exclude='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.

**New in v2.0.0**

The `plot` method replaces the old function `check.resid`. The `summary` method is new.

### 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 $\approx -3$; this is very likely to be an outlier. To check which subject this is, the following code block shows how to use the "tidy" data table; in this case, we see that it was subject #81. When I looked at the subject's brain MRI, it turns out he/she had a stroke in the right supramarginal/inferior parietal lobe.

```
all.dat.resids$all.dat.tidy[region == 'rSMAR', .SD[which.min(resids)]]

##    Study.ID   Group Sex Age Scanner region value resids
## 1:       81 Patient   M  16  Site 1  rSMAR  2.37 -3.271
```

This subject is also listed as an outlier in the `summary` method output for this object (in the next code block). If you do not include a value for the `regions` argument, then the function prints outlier information for all regions (not shown).

```
summary(all.dat.resids, region='rSMAR')

##
## Structural covariance residuals
## ---------------------------------------

## Number of outliers per region: (sorted in descending order)
## rSMAR
##     7
##
##
## Number of times each subject was an outlier: (sorted in descending order)
##  81   5  49  47  65 104  41
##   1   1   1   1   1   1   1
```

### 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(all.dat.resids, regions='rSMAR')[[1]]
plot(all.dat.resids, regions='rSMAR', cols=TRUE)[[1]]
```

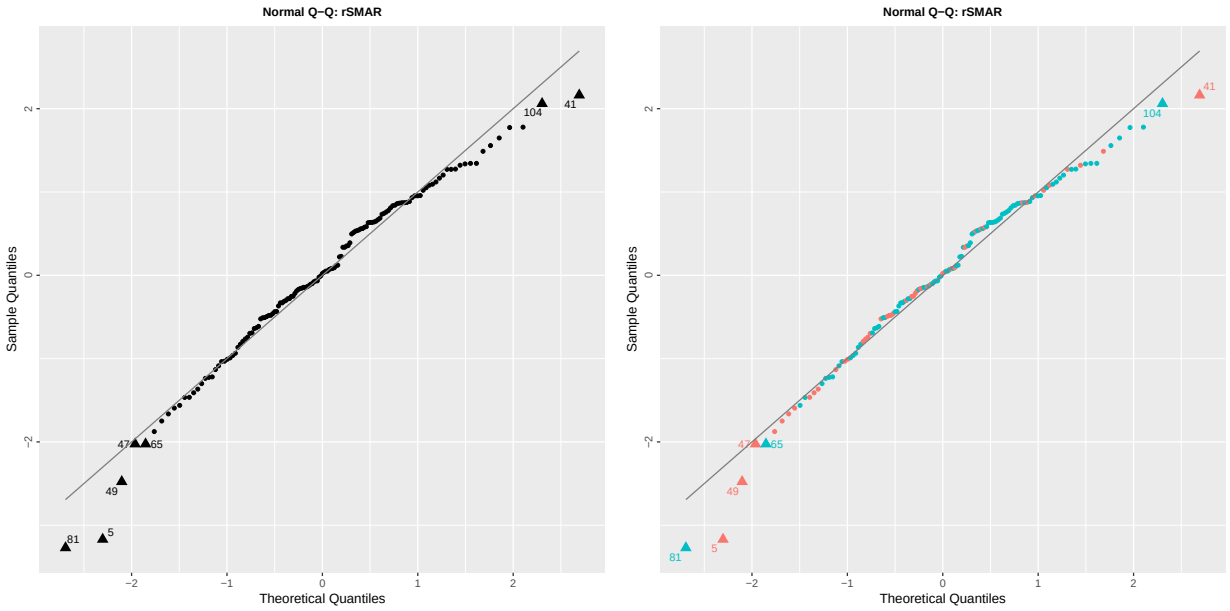



Figure 6.1: QQ plot of model residuals.

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

```
residPlots <- plot(all.dat.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(all.dat.resids, densities=densities)
t(sapply(corrs, with, thresholds))

##           [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
## Control 0.5428 0.5281 0.5156 0.5024 0.4892 0.4802 0.4700 0.4619 0.4541
```

```
## Patient 0.4876 0.4678 0.4554 0.4371 0.4236 0.4110 0.4024 0.3963 0.3874
##          [,10]  [,11]  [,12]  [,13]  [,14]  [,15]  [,16]
## Control 0.4475 0.4391 0.4318 0.4236 0.4149 0.4090 0.4047
## Patient 0.3784 0.3658 0.3574 0.3507 0.3405 0.3346 0.3291
```

Alternatively, you can skip the step above 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(all.dat.resids, densities=densities, exclude.reg=exclude.reg)
```

### 6.5.2 Graph creation

Then create the graphs:

```
# Create simple, undirected graphs for each group
g <- lapply(corrs, function(x)
            apply(x$r.thresh, 3,
                  graph_from_adjacency_matrix, mode='undirected', diag=F))
```

If you prefer that your graphs are weighted by the correlation coefficient, then a slight adjustment is needed:

```
g <- lapply(corrs, function(x)
    apply(x$r.thresh, 3, function(y)
          graph_from_adjacency_matrix(x$R * y, mode='undirected', diag=F, weighted=T)))
```

Finally, set the relevant graph-, vertex-, and edge-level attributes. This should only take a few seconds per graph, depending on the number of vertices.

```
g <- Map(function(x, y) llply(x, set_brainGraph_attr, atlas=atlas,
                              modality=modality, group=y, .progress='text'),
         g, as.list(groups))
```

*ASIDE*: An alternate method of getting a subgraph of e.g., the left hemisphere only:

```
g.lh <- lapply(g, lapply, function(x) induced.subgraph(x, V(x)$hemi == 'L'))
```

## 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, function(x) rbindlist(lapply(x, 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   rSPL     16
##  7: Control 0.08033  rPCUN     17
##  8: Patient 0.08033   lSFG     16
##  9: Control 0.09043   rMTG     18
## 10: Patient 0.09043   lSPL     18

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

## Empty data.table (0 rows) of 7 cols: Group,density,region,lobe,hemi,degree...

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

## Empty data.table (0 rows) of 2 cols: Group,N

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

## Empty data.table (0 rows) of 3 cols: Group,lobe,mean.deg
```

## 6.7 Tidying Data

To make some plotting functions easier, it's important to "tidy" the data (see Wickham (94)). This is an example of *reshaping* your data. Here is the required code:

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

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

##      density region     lobe hemi  modality atlas   Group variable   value
##   1: 0.05004  lBSTS Temporal    L thickness    dk Control   degree  2.0000
##   2: 0.05004  lBSTS Temporal    L thickness    dk Patient   degree  1.0000
##   3: 0.06014  lBSTS Temporal    L thickness    dk Control   degree  2.0000
```

```
##   4: 0.06014  lBSTS Temporal    L thickness    dk Patient   degree  1.0000
##   5: 0.07024  lBSTS Temporal    L thickness    dk Control   degree  2.0000
##  ---
## 604: 0.18042  lBSTS Temporal    L thickness    dk Patient  z.score -1.3014
## 605: 0.19008  lBSTS Temporal    L thickness    dk Control  z.score -1.5479
## 606: 0.19008  lBSTS Temporal    L thickness    dk Patient  z.score -1.3146
## 607: 0.20018  lBSTS Temporal    L thickness    dk Control  z.score -0.5774
## 608: 0.20018  lBSTS Temporal    L thickness    dk Patient  z.score -1.3038

# Example tidying global network measure data.table
# Number of rows = (number of densities) * (number of global measures)
(dt.G.tidy <- melt(dt.G, c('density', names(which(sapply(dt.G, is.character))))))

##      density atlas  modality   Group      variable   value
##   1: 0.05004    dk thickness Control            Cp 0.54718
##   2: 0.06014    dk thickness Control            Cp 0.56726
##   3: 0.07024    dk thickness Control            Cp 0.54520
##   4: 0.08033    dk thickness Control            Cp 0.54229
##   5: 0.09043    dk thickness Control            Cp 0.54096
##  ---
## 508: 0.16023    dk thickness Patient vulnerability 0.06811
## 509: 0.17032    dk thickness Patient vulnerability 0.06574
## 510: 0.18042    dk thickness Patient vulnerability 0.07347
## 511: 0.19008    dk thickness Patient vulnerability 0.07806
## 512: 0.20018    dk thickness Patient vulnerability 0.07464
```

# 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 (5, 6, 46, 47).

**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. For example, the directory structure might look like: (where `con01` is an example Study ID)

```
./Control/con01-fdt_network_matrix
./Control/con01-sizes.txt
./Control/con01-waytotal
etc.
./Patient/pat01-fdt_network_matrix
./Patient/pat01-sizes.txt
./Patient/pat01-waytotal
```

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 $76 \times 76$ 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
#============================================================================
groups <- c('Control', 'Patient')
atlas <- 'dkt.scgm'
modality <- 'dti'

# Get all relevant filenames
datadir <- '/home/cwatson/probtrackx_results/'
covars.all <- fread(paste0(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, Group, Study.ID)

# If your files aren't in separate directories, you don't need ``list.dirs''
matfiles <-
  list(A=list.files(list.dirs(datadir, recursive=F), 'fdt_network_matrix', full.names=T),
       way=list.files(list.dirs(datadir, recursive=F), 'waytotal', full.names=T),
       size=list.files(list.dirs(datadir, recursive=F), 'sizes.txt', full.names=T))
inds <- lapply(seq_along(groups), function(x)
               covars.dti[, which(Group == groups[x])])

# Output directory to save the data I generate
today <- format(Sys.Date(), '%Y-%m-%d')
savedir <- paste0('/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(list.dirs(datadir, recursive=T),
                         '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 (86)). Either way, the same function is used. In this section, I describe the inputs and outputs of `create_mats`.

### 7.2.1 Function arguments

**A.files** A character vector of the filenames containing the connectivity matrices.

**modality** A character string; either `dti` (default) or `fmri`. (new since v1.0.0)

**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 4 possible options. The 3rd and 4th options will enforce the same connections across all study subjects. (new since v1.1.0)

**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 (20).

**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. (86).

**mean** You may choose to specify a set of thresholds $\tau$ (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 (11, 33).

**consistency** Perform "consistency-based" thresholding (69). 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*).

**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_mats`. 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:

**Nvert** The # of vertices in the graph

**Nsub** The # of subjects (total, across all groups)

**Nthr** The # of thresholds applied

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

**Warning**

As of `v2.0.2`, the element `A.norm.sub` returns lists of 3D arrays in which the order along the 3rd dimension matches that of the input matrix files (i.e., the input argument `A.files` ), and therefore the raw matrices and normalized matrices in `A` and `A.norm`, respectively. This would not affect you unless you chose `threshold.by='consensus'` or `threshold.by='consistency'` . Furthermore, this would not affect you if your *Study ID's* (and therefore the matrix files) were already in the correct order, alphanumerically by `Study.ID` *and* by `Group`. Nevertheless, you should be aware of the change in behavior. Thanks to `@seantma` for pointing out the bug!

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

**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, e.g., *Nthr* arrays of size $Nvert \times Nvert \times Nsub$. Only valid if `threshold.by='consistency'` .

**A.bin.sums** A list of 2-d matrices, 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; each of those list elements is itself a list (if you have more than 1 group), e.g., 2. And finally each of those matrices is just $Nvert \times Nvert$. Only valid if `threshold.by='consistency'` .

**A.inds** A list of 2-d binary matrices, 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; the number of list elements equals the number of *thresholds*. Each of the list elements is of the same dimension as `A.norm` (e.g., $Nvert \times Nvert \times Nsub$); however, the connections which were deemed absent (e.g., 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` .

**A.norm.mean** A list of matrices; there is one for each group and each threshold. 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).

```
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. (54).

```
matfiles$W <- list.files(dirs$data, 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 based on the matrices from the previous section, and calculate the relevant attributes for these graphs. I use a similar "hack" as that in Random graph generation to generate the lists of graphs, save them to disk, and then later load them into one large list object (it seems to be faster, perhaps due to R's handling of memory). This actually gave me a speed increase of 1.85x, and seems to improve with increasing numbers of subjects and/or thresholds/densities.

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 3 "levels":

- `g[[X]]` The first index is for *subject group*
- `g[[X]][[Y]]` The second index is for *threshold*
- `g[[X]][[Y]][[Z]]` The third index is for *subject*

You may want to place the following code in its own script, e.g., `02_create_graphs_DTI.R`. The code for a *progress bar* is not necessary and simply lets you track how far along the processing is.

As of `v1.0.0`, I include the weighted adjacency matrix in the call to `set_brainGraph_attr`; this is not required but it does improve the speed (particularly when calculating weighted local efficiency; see Benchmarks for benchmarking). I also recommend using `foreach` for the subject-level graphs; on my machine this results in a speed-up of $\approx 3.4x$.

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

for (i in seq_along(groups)) {
  for (j in seq_along(thresholds)) {
    print(paste0('Threshold ', j, '/', length(thresholds), '; group ', i, '; ',
                 format(Sys.time(), '%H:%M:%S')))

    foreach (k=seq_along(inds[[i]])) %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=groups[i],
          use.parallel=FALSE, A=A.norm.sub[[j]][, , inds[[i]][k]])
      saveRDS(g.tmp, file=paste0(savedir,
                                 sprintf('g%i_thr%02i_subj%03i%s', i, j, k, '.rds')))
    }
  }

  # Group mean weighted graphs
  print(paste0('Group', i, '; ', format(Sys.time(), '%H:%M:%S')))
  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=groups[i],
                              A=A.norm.mean[[x]][[i]], use.parallel=FALSE),
          .parallel=TRUE)
}
```

And now we combine the list objects saved to disk in one list, for further processing. I perform a check to make sure the `name` graph attribute matches the `Study.ID` of the study subjects, and then remove all of the individual files that were created in the previous step.

```
for (i in seq_along(groups)) {
  g[[i]] <- fnames[[i]] <- vector('list', length=length(thresholds))
  for (j in seq_along(thresholds)) {
    fnames[[i]][[j]] <- list.files(savedir,
                                   sprintf('g%i_thr%02i.*', i, j), full.names=T)
    g[[i]][[j]] <- lapply(fnames[[i]][[j]], readRDS)
  }
  x <- all.equal(sapply(g[[i]][[1]], graph_attr, 'name'),
                 covars.dti[groups[i], Study.ID])
  if (isTRUE(x)) lapply(fnames[[i]], file.remove)
}
saveRDS(g, file=paste0(savedir, 'g.rds'))
saveRDS(g.group, file=paste0(savedir, 'g.group.rds'))
```

## 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. The line beginning `setcolorder` is optional; it changes the column ordering to whatever you specify.

```
# GROUP-LEVEL
#============================================================================
dt.V.group <- rbindlist(lapply(g.group, function(x)
                               rbindlist(lapply(x, vertex_attr_dt))))
dt.G.group <- rbindlist(Map(graph_attr_dt, g.group, as.list(groups)))

# SUBJECT-LEVEL
#============================================================================
dt.V <- vector('list', length=length(groups))
for (i in seq_along(groups)) {
  dt.V[[i]] <- lapply(g[[i]], llply, vertex_attr_dt, groups[i])
  dt.V[[i]] <- rbindlist(lapply(dt.V[[i]], rbindlist))
}
dt.V <- rbindlist(dt.V)

dt.G <- rbindlist(Map(function(x, y)
                      rbindlist(lapply(x, graph_attr_dt, y)),
                      g, as.list(groups)))

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)
setcolorder(dt.V.group, c('modality', 'atlas', 'weighting', 'sub.thresh',
                          'threshold', 'Group', names(dt.V.group)[1:33]))
setcolorder(dt.G.group, c('modality', 'atlas', 'weighting', 'sub.thresh',
                          'threshold', 'Group', names(dt.G.group)[c(1:4, 6:23)]))
setcolorder(dt.G, c('modality', 'atlas', 'weighting', 'sub.thresh', 'threshold',
                    'Group', 'Study.ID', names(dt.G)[c(1:4, 6:23)]))
setcolorder(dt.V, c('modality', 'atlas', 'weighting', 'sub.thresh', 'threshold',
                    'Group', 'Study.ID', names(dt.V)[1:33]))

dt.G.group.tidy <- melt(dt.G.group, names(dt.G)[1:6])
dt.G.tidy <- melt(dt.G, c(names(dt.G)[1:7], 'density'))
```

## 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
env.dti$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
env.dti$dt.V.group[lobe == 'Frontal',
                   .(p=t.test(E.nodal ~ Group)$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(env.dti$dt.G.tidy, xvar='threshold', exclude=exclude.vars)
```

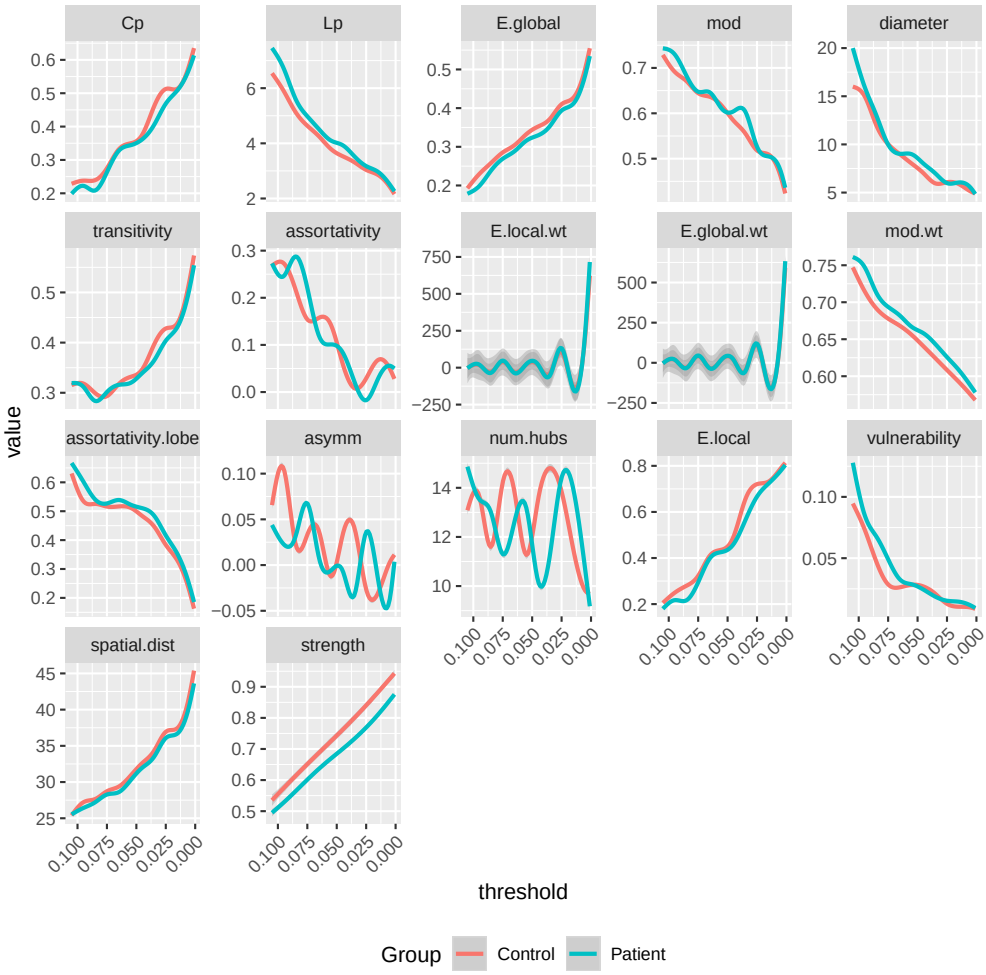



Figure 7.1: Global graph measures vs. density, DTI data.

# 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*).(31, 61, 96)

## 8.1 Function arguments

```
args(brainGraph_GLM)

## function (g.list, covars, measure, con.mat, con.type = c("t",
##     "f"), X = NULL, con.name = NULL, alternative = c("two.sided",
##     "less", "greater"), alpha = 0.05, level = c("vertex", "graph"),
##     permute = FALSE, N = 5000, perms = NULL, long = FALSE, ...)
## NULL
```

### 8.1.1 Mandatory

The following list details the *mandatory* function arguments:

**g.list** A list of `igraph` graph objects. You can include graphs from one or more subject groups. If you have multiple groups, you must *concatenate* the lists using the `c()` function (see Examples later).

**covars** A `data.table` of covariates. It should have as its first column `Study.ID`; 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.). This data object will be used to create the *design matrix*.

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

**con.mat** A numeric matrix specifying the contrast(s) of interest. It can be multiple rows, and you can specify row names.

**con.type** Either `t` or `f`, if you want to calculate t- or F-statistics. F-statistics are only "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 (`f`).

### 8.1.2 Optional

The following are *optional* function arguments:

**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.

**con.name** A character string of the name of the contrast; e.g., `'Control > Patient'`. If this argument is not provided, the function first checks if `con.mat` has row names (which you would create yourself); otherwise 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.

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`

**N** An *integer* indicating the number of permutations to create; only relevant if `permute=TRUE` ; 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

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`

**mean.center** A *logical* specifying whether or not to mean-center your non-factor variables; default: `FALSE` . Mean-centering will be done for all subjects, not within groups. Note that any factor variables that are binarized (see next bullet point) will also be mean-centered.

**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 will be 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.

**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. (55)).

So, to summarize, columns in the design matrix `X` will likely 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 *Group* column(s), and finally *interaction* columns (if applicable). You may wish to inspect the design matrix before specifying your contrast vector by typing, e.g., `head(brainGraph_GLM_design(covars, ...))` .

## 8.2 Return value

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. First, I describe the statistics that are calculated, and then the return object itself.

### 8.2.1 Statistics

For all analyses, the following statistics are calculated ($C$ is the contrast matrix):

**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

$$SE = \sqrt{C \boldsymbol{\Sigma} C^T}$$

where $\boldsymbol{\Sigma}$, 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* of the full model.

**stat** The *t-statistic* for the contrast of interest is

$$T = \frac{\hat{\gamma}}{SE}$$

The *F-statistic* is

$$F = \frac{ESS/rank(C)}{SSEF/df}$$

**CI** (if `con.type='t'` ) The lower and upper confidence limits:

$$\hat{\gamma} \pm t_{\alpha/2,df} \times SE$$

### 8.2.2 The `bg_GLM` object

The elements of the return object are:

**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* (the graph metric chosen by the `measure` function argument).

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

**con.mat** The *contrast matrix*.

**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** 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.

**N** The number of permutations (if applicable).

**perm** A *list* containing two `data.table` objects:

- **null.dist** The null distribution of the maximum statistic for each permutation.
- **thresh** The $(1-\alpha)$th percentile of the null distribution.

`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 (49, 63, 77, 95). For a complete treatment of this topic, see Kutner et al. (55); 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:

$$\begin{aligned}\hat{\beta}_1 &= \mu_1 \\ \hat{\beta}_2 &= \mu_2 \\ &\vdots \\ \hat{\beta}_k &= \mu_k\end{aligned}$$

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.

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

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, 10:11, 102:105), ]

##       Intercept GroupPatient
##  [1,]         1           -1
##  [2,]         1           -1
##  [3,]         1           -1
##  [4,]         1           -1
##  [5,]         1           -1
##  [6,]         1            1
##  [7,]         1            1
##  [8,]         1            1
##  [9,]         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', groups),
                    'Control > Patient', 'Patient > Control')
Cmat

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

# Some variables to make the LM fit faster across repetitions
XtX <- solve(crossprod(Xdes))
dfR <- nrow(Xdes) - ncol(Xdes)
CXtX <- apply(Cmat, 1, function(x) solve(t(x) %*% XtX %*% x))
fits <- vector('list', nrow(Cmat))
for (i in seq_along(fits)) {
  fits[[i]] <- brainGraph:::brainGraph_GLM_fit_t(Xdes, y, XtX, Cmat[i, , drop=FALSE])
  fits[[i]] <- as.data.table(fits[[i]])
}
fits <- rbindlist(fits)
cbind(rownames(Cmat), fits)

##                    V1     gamma        se
## 1:      Mean Control  0.045358 0.0009747
## 2:      Mean Patient  0.042102 0.0006109
## 3: Control > Patient  0.003256 0.0011504
## 4: Patient > Control -0.003256 0.0011504
```

## 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 *list* 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 $\alpha$ 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', con.mat=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, con.mat=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='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 GroupPatient:Age.MRI
## [1,]            0            1                0.000                1.236
## [2,]            0            1                0.000                1.613
## [3,]            0            1                0.000                1.635
## [4,]            0            1                0.000                3.078
## [5,]            1            0               -3.827                0.000
## [6,]            1            0               -1.812                0.000
## [7,]            1            0               -3.419                0.000
## [8,]            1            0               -4.249                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, con.mat=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
## [1,]         1            1    1                 1
## [2,]         1            1    1                 1
## [3,]         1            1    1                 1
## [4,]         1            1   -1                -1
## [5,]         1           -1    1                -1
## [6,]         1           -1    1                -1
## [7,]         1           -1   -1                 1
## [8,]         1           -1    1                -1
```

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

```
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, con.mat=con.mat, alt='greater')))

##
## brainGraph GLM results
## ----------------------------------

## Level:  vertex
## Graph metric of interest:  E.nodal.wt
##
## Contrast type:  T contrast
## Alternative hypothesis:  C > 0
## Contrast matrix:
##                          Intercept GroupPatient SexM GroupPatient:SexM
## Group x Sex interaction         0            0    0                 1

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

##     Region Estimate 95% CI low 95% CI high Std. error t value p-value
##  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
## 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
##     p-value (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
## [1,]                 0                 0                 0
## [2,]                 0                 0                 0
## [3,]                 0                 0                 0
## [4,]                 0                 0                 1
##      GroupPatient:SexM
## [1,]                 1
## [2,]                 1
## [3,]                 1
## [4,]                 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
## [1,]         1 -1 -1 -1     1     1
## [2,]         1  1  0  1     0     1
## [3,]         1 -1 -1 -1     1     1
## [4,]         1  1 -1 -1    -1    -1
## [5,]         1  1  0  1     0     1
## [6,]         1 -1 -1 -1     1     1

con.mat <- cbind(rep(0, 5), 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, con.mat=con.mat, alt='greater')
summary(anova2x3)

##
## brainGraph GLM results
## ----------------------------------

## Level:  vertex
## Graph metric of interest:  E.nodal.wt
##
## Contrast type:  T contrast
## Alternative hypothesis:  C > 0
## Contrast matrix:
##              Intercept A2 B2 B3 A2:B2 A2:B3
## Main A               0  1  0  0     0     0
## Main B (1st)         0  0  1  0     0     0
## Main B (2nd)         0  0  0  1     0     0
## A X B (1st)          0  0  0  0     1     0
```

```
## A X B (2nd)          0  0  0  0     0     1
##
## 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 p-value
## 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
##    p-value (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 p-value
##  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
##     p-value (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
## [1,]     1     0     0     0     0     0
## [2,]     0     0     0     0     0     1
## [3,]     1     0     0     0     0     0
## [4,]     0     0     0     1     0     0
## [5,]     0     0     0     0     0     1
## [6,]     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, con.mat=con.mat, alt='greater'))
```

### 8.4.6 Three-way between subjects ANOVA: 2x2x2

Since `v2.0.0`, three-way interactions are allowed. 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
## [1,]         1            1    1         1                 1
## [2,]         1            1    1         1                 1
## [3,]         1            1    1         1                 1
## [4,]         1            1   -1        -1                -1
## [5,]         1            1    1        -1                 1
## [6,]         1            1    1        -1                 1
##      GroupPatient:Scanner3T SexM:Scanner3T GroupPatient:SexM:Scanner3T
## [1,]                      1              1                            1
## [2,]                      1              1                            1
## [3,]                      1              1                            1
## [4,]                     -1              1                            1
## [5,]                     -1             -1                           -1
## [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).(31, 61, 96) The default matrix partitioning scheme is (like that of PALM) called `'beckmann'` .(81)

When `long=TRUE` , an additional element called `perm` is returned in the `bg_GLM` object. It is a list with 2 elements:

**null.dist** A `data.table` containing the null distribution of maximum statistic values for each contrast.

**thresh** A `data.table` of the $1 - \alpha$ %ile value of the null distribution of maximum statistics for each contrast.

### 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',
                             con.mat=con.mat, X=X, alt='greater',
                             permute=TRUE, N=1000, long=TRUE)
```

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

```
diffs.perm$perm
```

```
## $null.dist
##       contrast    V1
##    1:        1 2.059
##    2:        1 1.027
##    3:        1 2.858
##    4:        1 2.621
```

```
##    5:        1 1.997
##   ---
##  996:        1 2.024
##  997:        1 1.671
##  998:        1 3.348
##  999:        1 2.140
## 1000:        1 1.120
##
## $thresh
##    contrast    V1
## 1:        1 3.004
```

## 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]])
```

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)
```

## 8.7 Create a graph of the results

To create a graph with attributes specific to GLM, use `make_glm_brainGraph`. This will return a list (with length equal to the number of contrasts) of graphs with several additional attributes; see Table 8.1.

| 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_glm_brainGraph`.

```
g.anova2x3 <- make_glm_brainGraph(anova2x3, atlas='dkt.scgm')
# Length equals the # of contrasts
length(g.anova2x3)

## [1] 5
```

## 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)
```

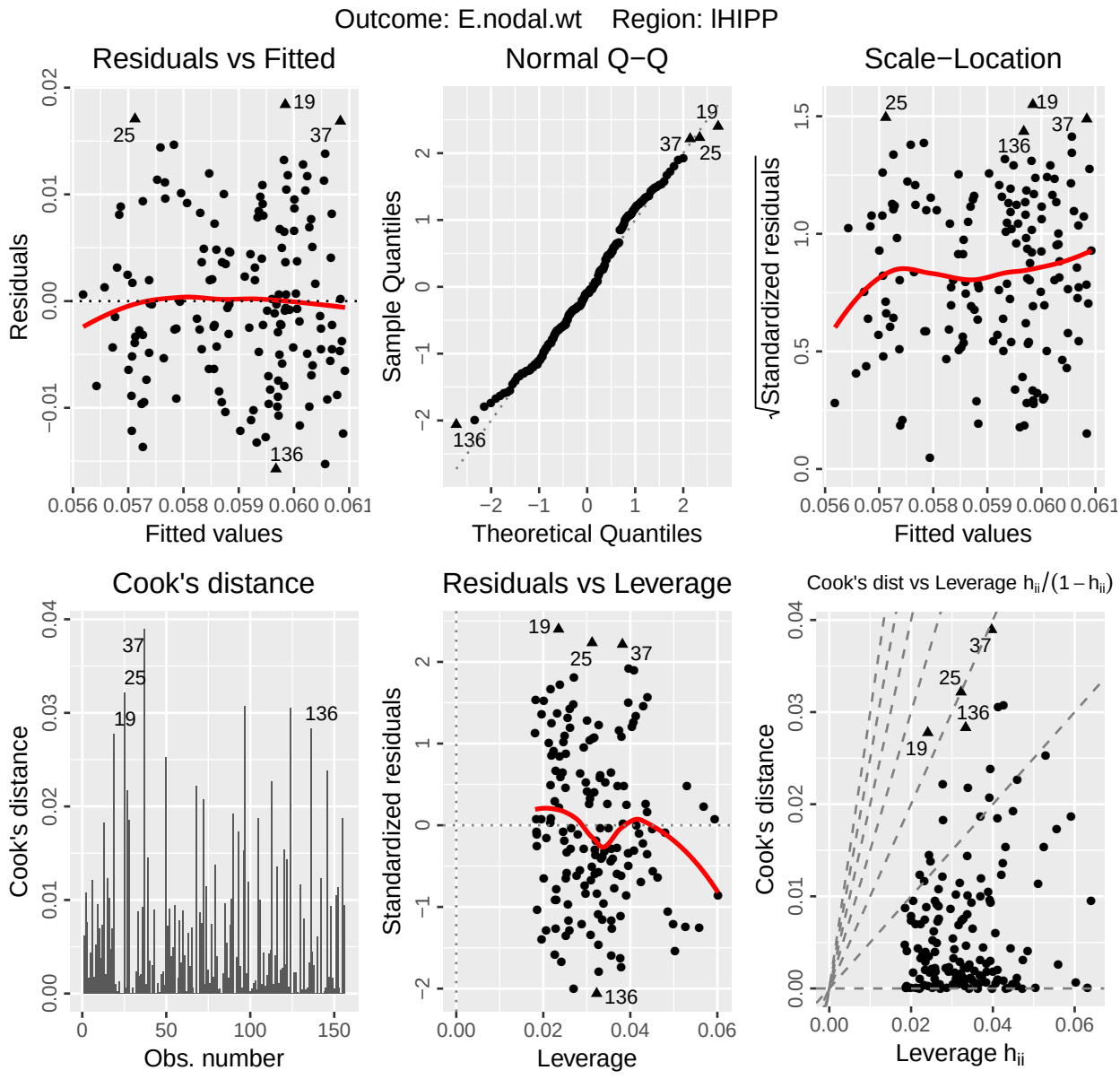



Figure 8.1: **GLM diagnostics.**

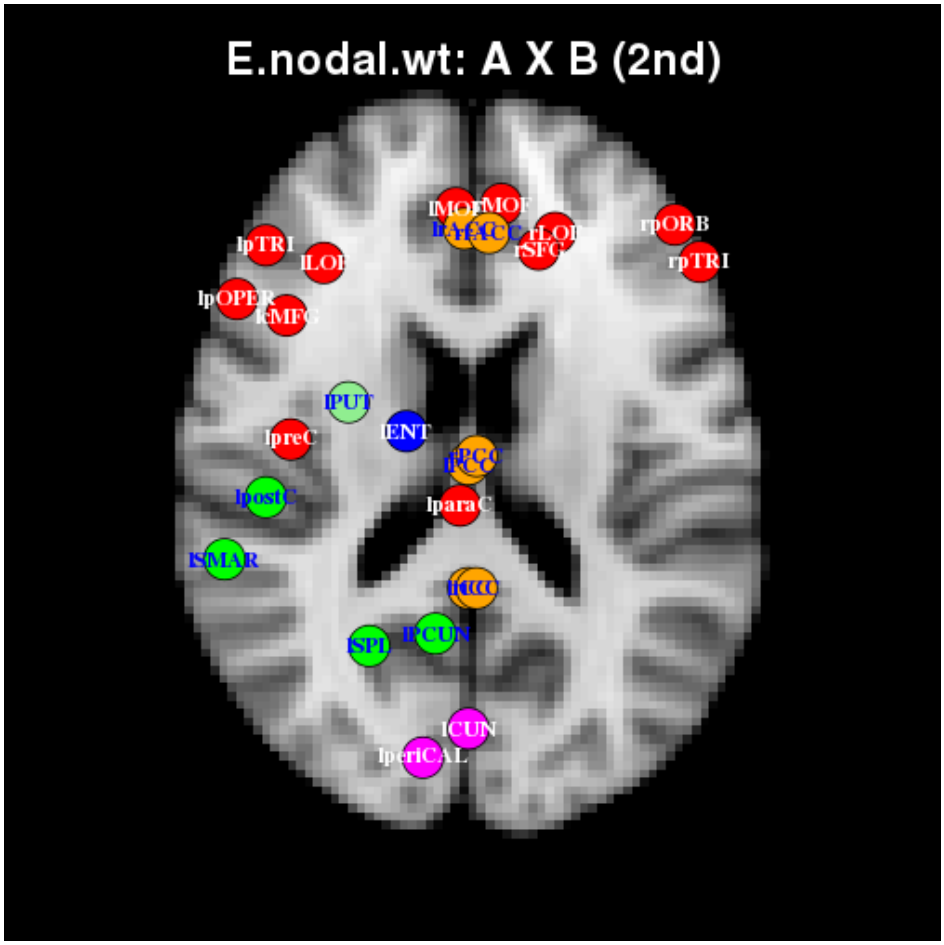



Figure 8.2: **GLM results.**

# 9

# Multi-threshold permutation correction



The *multi-threshold permutation correction (MTPC)* method for statistical inference was introduced by Drakesmith et al.(24) 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 $\tau$ 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 $\alpha$th 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, con.mat, con.type = c("t",
##     "f"), con.name = NULL, level = c("vertex", "graph"), clust.size = 3L,
##     N = 500L, perms = NULL, alpha = 0.05, res.glm = NULL, long = TRUE,
##     ...)
## NULL
```

### 9.2.1 Mandatory

**g.list** A *nested* list of `igraph` graph objects. If you followed the code in previous chapters, you can supply your entire `g` (or `g.norm`, or whatever you decide to call it) object. If you would like to isolate only one group (or a subset of all groups), you must "wrap" it in a list; e.g., `g.list=list(g[[2]])` will only look at group 2.

**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**

**con.mat**

### 9.2.2 Optional

**con.type** default: `'t'`

**con.name** default: `NULL`

**level** default: `'vertex'`

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

**N** default: `500`

**perms** default: `NULL`

**alpha** default: `0.05`

**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`

### 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: ($\tau$ refers to thresholds, and $N_\tau$ is the number of thresholds)

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

**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: $\tau_{mtpc}$, $S_{mtpc}$, $S_{crit}$, and $A_{crit}$.

**null.dist** A *list* containing a *numeric matrix* of the maximum statistic for each permutation and threshold. There will be $N_{rand}$ rows and $N_\tau$ columns. The number of elements in the list will equal the number of contrasts supplied.

**perm.order** The *numeric matrix* of permutation orderings.

**level**

**X**

**outcome**

**con.type**

**con.mat**

**con.name**

**alt**

**alpha**

**N**

## 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, con.mat=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, , groups[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
## ------------------------------------

## Level:  vertex
## Graph metric of interest:  E.nodal.wt
## # of permutations:  5,000
## # of thresholds:  30
## Cluster size (across thresholds):  3
##
## Contrast type:  T contrast
## Alternative hypothesis:  C != 0
## Contrast matrix:
##               Intercept age_mri_6w gender ScannerChange GroupPt1 GroupPt2
## Control > Pt1         0          0      0             0       -2       -1
## Control > Pt2         0          0      0             0       -1       -2
## Pt1 > Pt2             0          0      0             0        1       -1

##
## 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
## 1:        1     5555  99.86  3.255  13757
## 2:        2     5555 119.20  3.186  14520
## 3:        3     4920  84.92  3.260  13473
```

```
## Control > Pt1

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

## Control > Pt2

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

## Pt1 > Pt2

##        Outcome Region tau.mtpc S.mtpc S.crit A.mtpc A.crit
##  1: E.nodal.wt  lAMYG     5555  22.67   3.26  36411  13473
##  2: E.nodal.wt  rBSTS     6190  20.34   3.26  35767  13473
##  3: E.nodal.wt   lLOF     9365  16.81   3.26  27834  13473
##  4: E.nodal.wt   lENT     1222  16.31   3.26  18254  13473
##  5: E.nodal.wt    rFP     8730  13.24   3.26  20976  13473
## ---
## 12: E.nodal.wt   lCUN     8095   8.54   3.26  15358  13473
## 13: E.nodal.wt  rrACC      600   8.14   3.26  22893  13473
## 14: E.nodal.wt  lcACC     8095   7.03   3.26  17747  13473
## 15: E.nodal.wt  lrMFG     1533   6.63   3.26  14686  13473
## 16: E.nodal.wt   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. (24).

```
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.

```
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_glm_brainGraph`. 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_glm_brainGraph(res.mtpc, atlas='dk.scgm')
```

## 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.

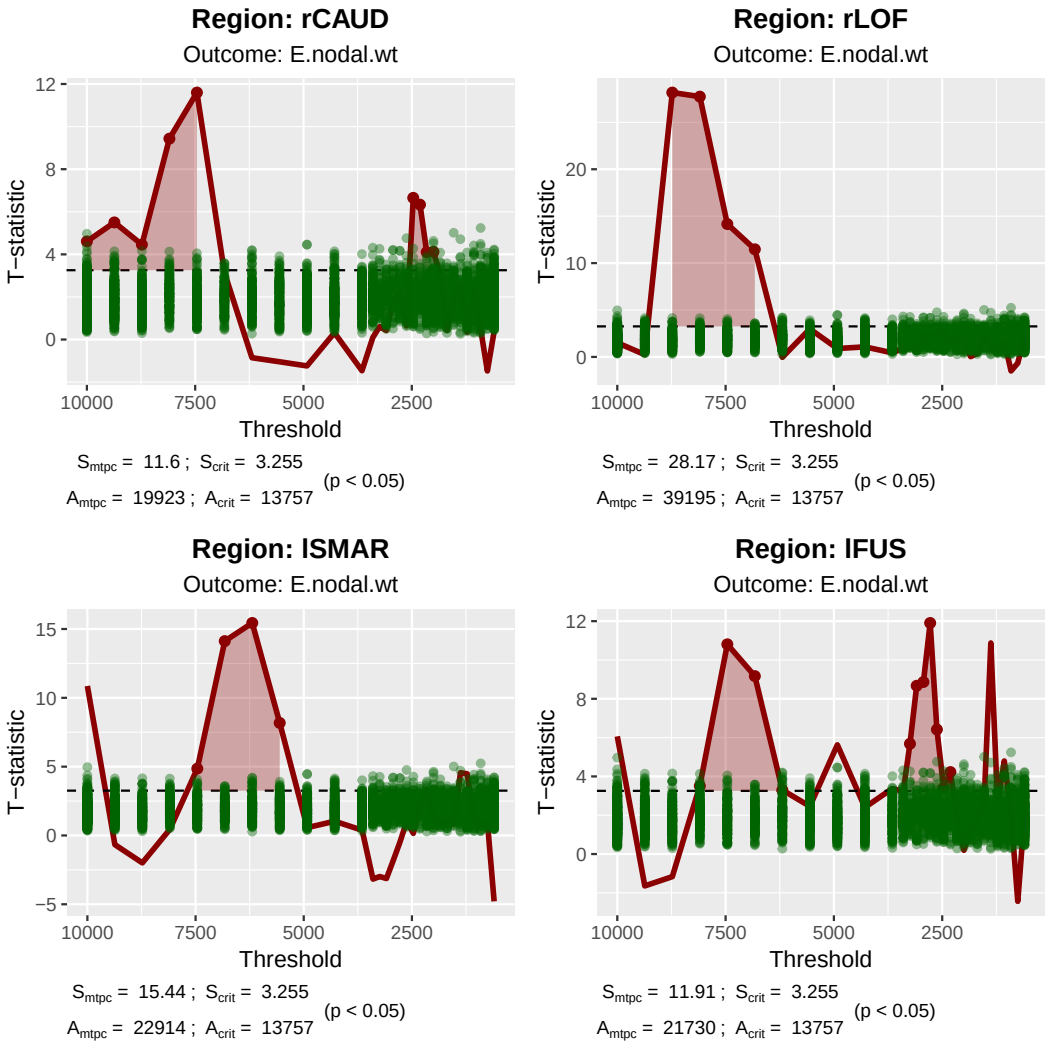



Figure 9.1: **MTPC statistics.** The red shaded areas are those that are above the critical null statistic (dashed line) for at least 3 consecutive thresholds. The green points are the maximum null statistics (per permutation).

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

| 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 value 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_glm_brainGraph`.

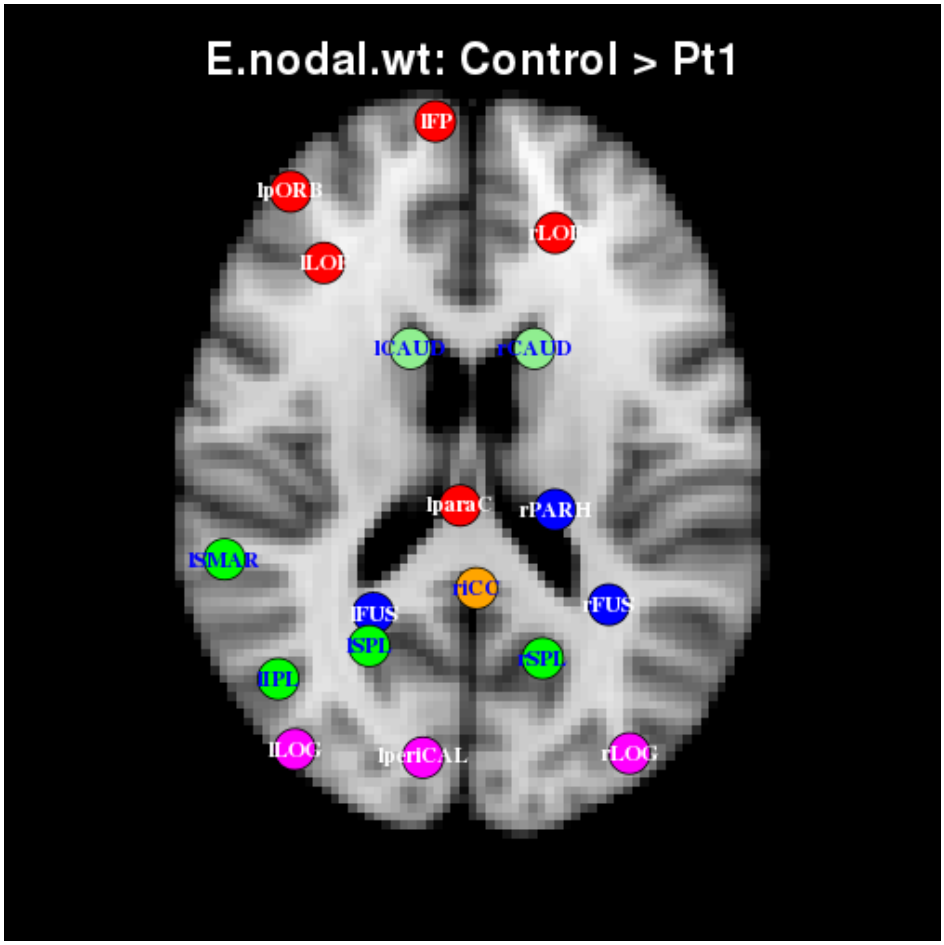



Figure 9.2: **MTPC results.**

# 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.(100) 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.(31)

## 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, con.mat, con.type = c("t", "f"), X = NULL,
##     con.name = NULL, p.init = 0.001, 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**

**con.mat**

**con.type**

**X**

**con.name**

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

**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).

**X**

**p.init** The initial p-value.

**con.type**

**con.mat**

**con.name**

**alt**

**N**

**removed**

**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.

**components** A list of two `data.tables`: the observed components and p-values, and 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$covars.dti, coding='effects',
                           binarize=c('Sex', 'Scanner'))
con.mat <- matrix(c(rep(0, 4), -2, rep(0, 4), 2), nrow=2, byrow=TRUE)
rownames(con.mat) <- c('Control > Patient', 'Patient > Control')
res.nbs <- NBS(A, env.glm$covars.dti, con.mat, X=X,
               p.init=0.001, N=1e3, alternative='greater')
```

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
## -----------------------------------

## Number of permutations:  1,000
## Initial p-value:  0.001
##
## Contrast type:  T contrast
## Alternative hypothesis:  C > 0
## Contrast matrix:
##                   Intercept Age.MRI Sex Scanner GroupPatient
## Control > Patient         0       0   0       0          -2
## Patient > Control         0       0   0       0           2

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

##      # vertices # edges p-value
## [1,]          6       5   0.001 ***
## [2,]          4       3   0.001 ***
## [3,]          3       2   0.004 **
## [4,]          2       2   0.198
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

## Patient > Control
```

| 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_nbs_brainGraph`.

```
##      # vertices # edges p-value
## [1,]          3       2   0.003 **
## [2,]          2       1   0.071 .
## ---
## 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_nbs_brainGraph`. 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_nbs_brainGraph(res.nbs, atlas=atlas)
```

## 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 $\alpha$. 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.

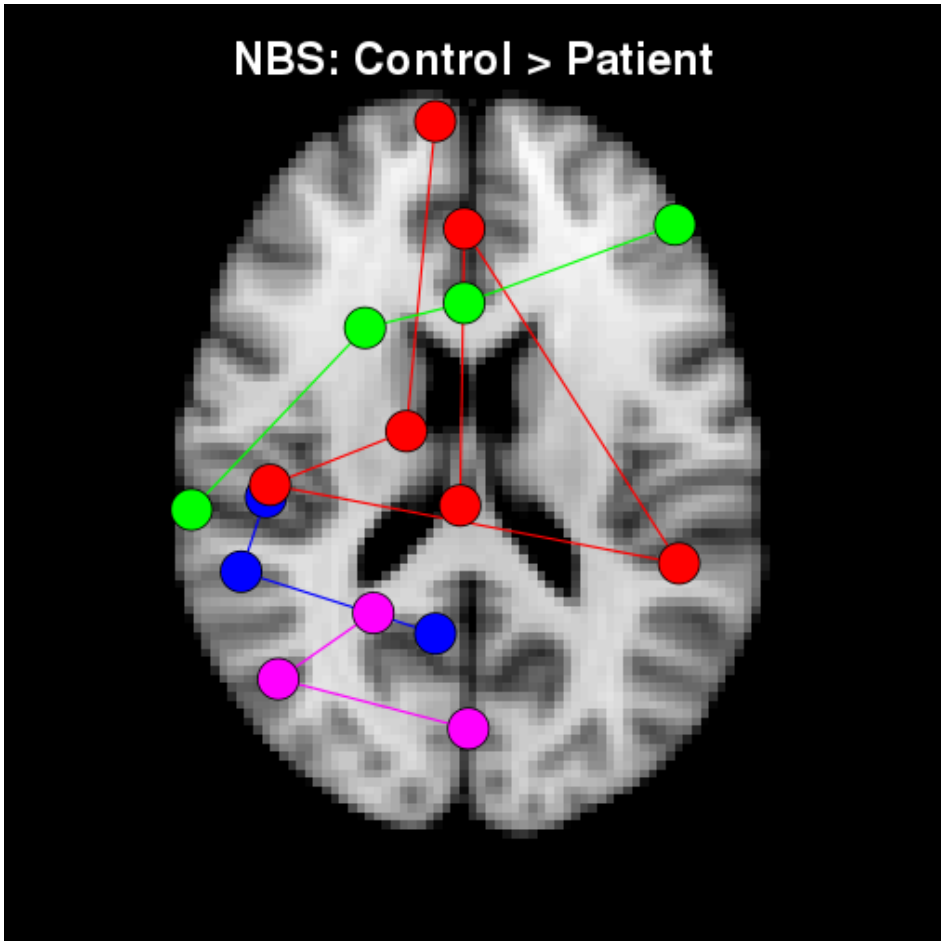



Figure 10.1: **Significant connected components.** Vertices of the same component are plotted in the same color.

# 11

# Graph- and vertex-level mediation analysis



**New in v2.0.0**

All of the functionality in this chapter is new to v2.0.0.

Performing mediation analysis in `brainGraph` follows the *potential outcomes* framework that is implemented in the mediation package (84), as opposed to the more well-known "Baron & Kenny approach" (4). A few advantages of this approach 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 (66), the *potential outcomes* framework for causal inference was re-introduced (or popularized) by Donald Rubin in the 1970's (72, 73).[1] In fact, Holland (37) 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 $X_i = 1$

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

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 (37) 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.,

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

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* $\delta$ 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 (66). 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 (72); Holland (37); Robins and Greenland (70). Furthermore, Imai and colleagues, who created the `mediation` package, have several papers that should be considered required reading (42, 43, 44, 84); and see also Keele (50).

## 11.2 Notation

As I mentioned, it is critical that you read Tingley et al. (84). 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` |

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

## 11.3 Function arguments

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

```
args(brainGraph_mediate)

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

### 11.3.1 Mandatory

**g.list** As with brainGraph_GLM, a "flat" list of graph objects for $\geq 1$ 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**

**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. (8) and Falk and Biesanz (29) 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`

**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.

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

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

**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.

```
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

```
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(do.call(Map, c(c, g))[[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')

##
## brainGraph mediation results
## -------------------------------------

## # of observations:  71
## Level:  vertex
## Mediator:  E.nodal.wt
## Treatment:  Group
##  Control value:  Control
##  Treatment value:  Patient
## Outcome:  brief_gec_raw_6m
## Pre-treatment covariates:
##
## 1 age
## 2 gender
## 3 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 Group brief_gec_raw_6m  2.14919       -5.788
##  2:    lAMYG E.nodal.wt Group brief_gec_raw_6m -0.04912      -13.119
##  3:    lBSTS E.nodal.wt Group brief_gec_raw_6m -0.29800       -5.046
##  4:    lCAUD E.nodal.wt Group brief_gec_raw_6m  0.61905       -4.795
##  5:     lCUN E.nodal.wt Group brief_gec_raw_6m -0.63081       -5.462
## ---
## 78: rperiCAL E.nodal.wt Group brief_gec_raw_6m  0.45775       -4.196
## 79:   rpostC E.nodal.wt Group brief_gec_raw_6m -2.17657       -7.697
## 80:    rpreC E.nodal.wt Group brief_gec_raw_6m -2.30992       -9.053
## 81:    rrACC E.nodal.wt Group brief_gec_raw_6m -0.77329       -5.903
## 82:    rrMFG E.nodal.wt Group brief_gec_raw_6m -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
##  3:   0.884
##  4:   0.784
##  5:   0.812
## ---
## 78:   0.834
## 79:   0.228
## 80:   0.312
## 81:   0.746
## 82:   0.970
```

| 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_mediate_brainGraph`.

## 11.6 Create a graph of the results

To create a graph with attributes specific to mediation analysis, use `make_mediate_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_mediate_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)
```

## 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.

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

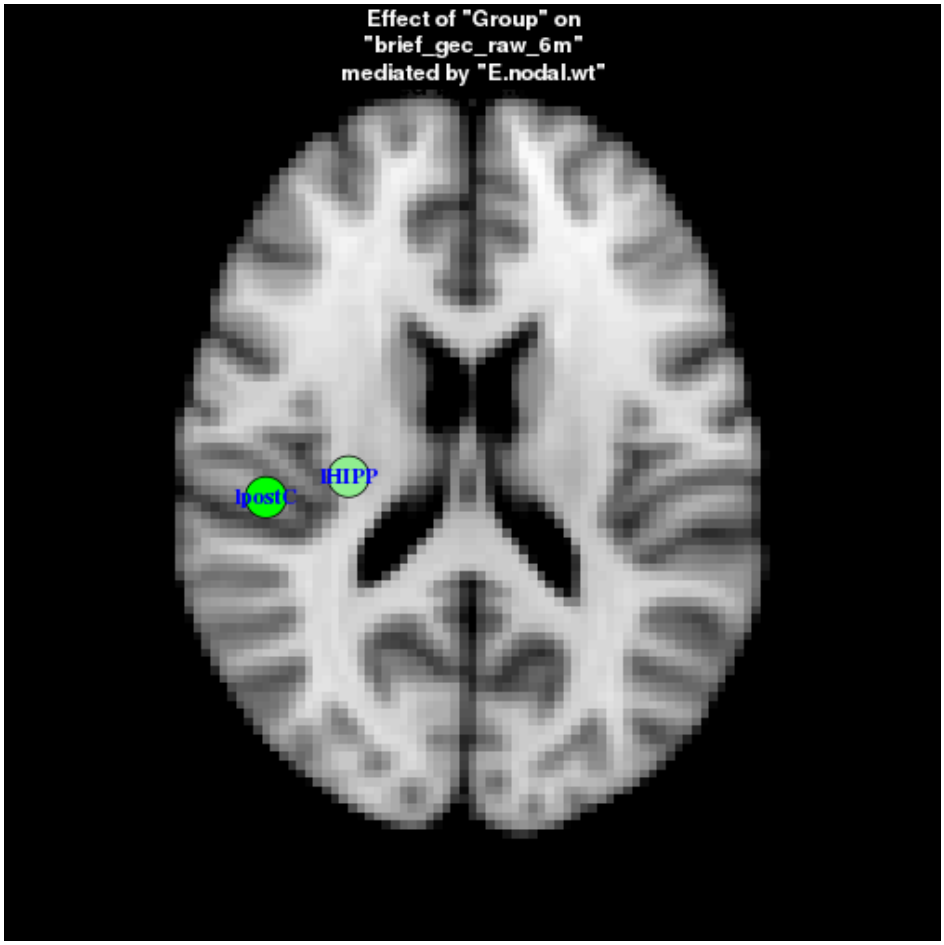



Figure 11.1: **Mediation results.**

# 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.(41, 93) 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.(39, 101) It is true for other data, as well, because random networks have low clustering in general. For more discussion, see Chapter 12 of Newman (60), and also Telesford et al. (83).

### 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 (89) for the method used; see also Milo et al. (59)). `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. (67)).

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 <- paste0(getwd(), '/../rand/')

rand_vars <- analysis_random_graphs(g, 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. (2). 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 $\omega$ (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
g.rand <- small.clust.dt <- vector('list', length=length(groups))
for (i in seq_along(groups)) {
  g.rand[[i]] <- vector('list', length=kNumDensities)
  for (j in seq_along(densities)) {
    g.rand[[i]][[j]] <- sim.rand.graph.par(g[[i]][[j]],
                                           kNumRandClust, clustering=T)
  }
  small.clust.dt[[i]] <- small.world(g[[i]], g.rand[[i]])
}
small.clust.dt <- rbindlist(small.clust.dt)
```

## 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 $\sigma$.(41) 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:    0.05 100 3.028 0.547   2.716   0.171   1.115   3.195 2.865 Control
## 2:    0.06 100 2.850 0.567   2.511   0.196   1.135   2.887 2.544 Control
## 3:    0.07 100 2.697 0.545   2.483   0.216   1.087   2.522 2.321 Control
## 4:    0.08 100 2.696 0.542   2.397   0.240   1.125   2.258 2.008 Control
## 5:    0.09 100 2.511 0.541   2.305   0.263   1.089   2.059 1.891 Control
## 6:    0.10 100 2.579 0.540   2.317   0.267   1.113   2.023 1.818 Control
```

To calculate $\omega$ (see Telesford et al. (83)), 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 $\sigma$ and $\omega$ 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 $\omega$ is:

```
small.clust.dt[, Group := rep(groups, each=kNumDensities)]
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.tidy <- 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 $\sigma$.

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')
```

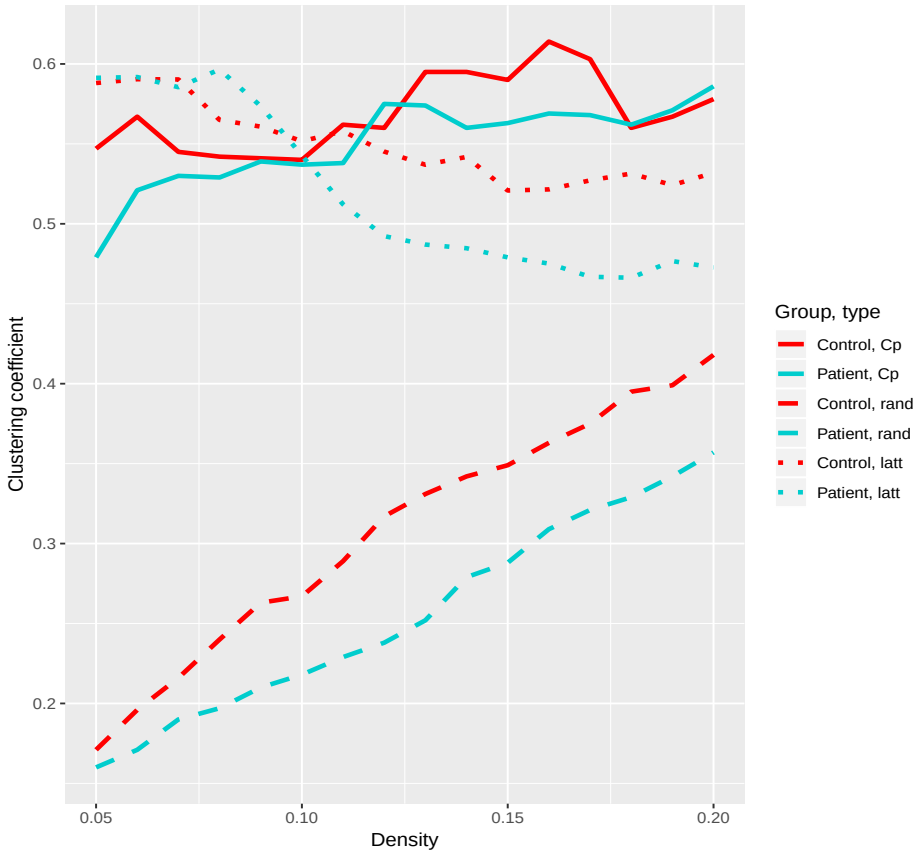



Figure 12.1: **Clustering coefficients for random networks.** The solid lines are clustering for the observed networks. The dotted line are values from networks generated by the Markov Chain approach, and the dashed lines are values from networks generated by the "simple" approach.

We can plot both $\sigma$ and $\omega$ in the same plot, shown in Figure 12.2. As stated in Telesford et al. (83), networks with $\omega$ closer to 0 indicate more balance between high clustering and low path length; networks with $\omega$ 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.tidy[variable == 'sigma', yint := 1]
small.tidy[variable == 'omega', yint := 0]
ggplot(small.tidy[variable %in% c('sigma', 'omega')],
       aes(x=density, y=value, col=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')
```

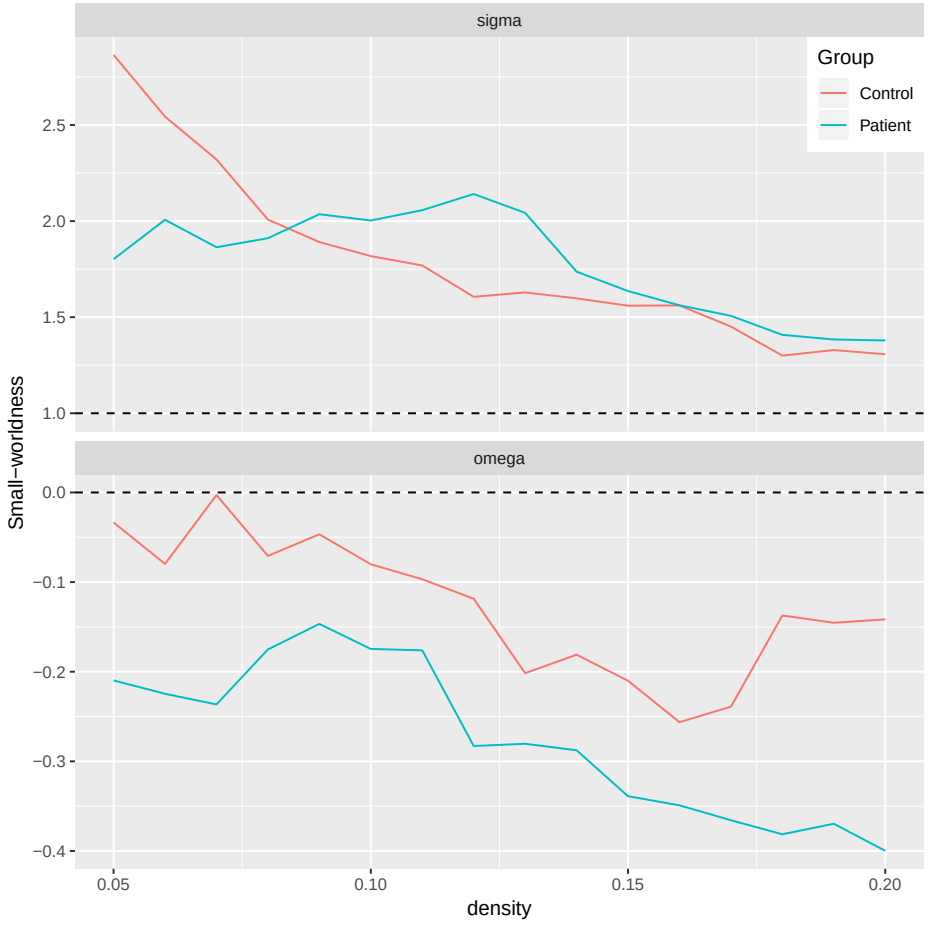



Figure 12.2: **Small-world indexes.** (top) The classic small-world index, $\sigma$; the dashed horizontal line at $y = 1$ is included to show the minimum value for a network to be considered "small-world" (93) (bottom) The small-world index $\omega$ introduced by Telesford et al. (83); the dashed horizontal line at $y = 0$ indicates the value at which the network displays a balance between clustering coefficient and characteristic path length.

## 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. (13), Zhou and Mondragón (102)).
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 (56). 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*). In the next code section, I show how to get this value, and then how to plot a shaded region that starts at the maximum cut-off value from 2 groups.

### 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=g)
```

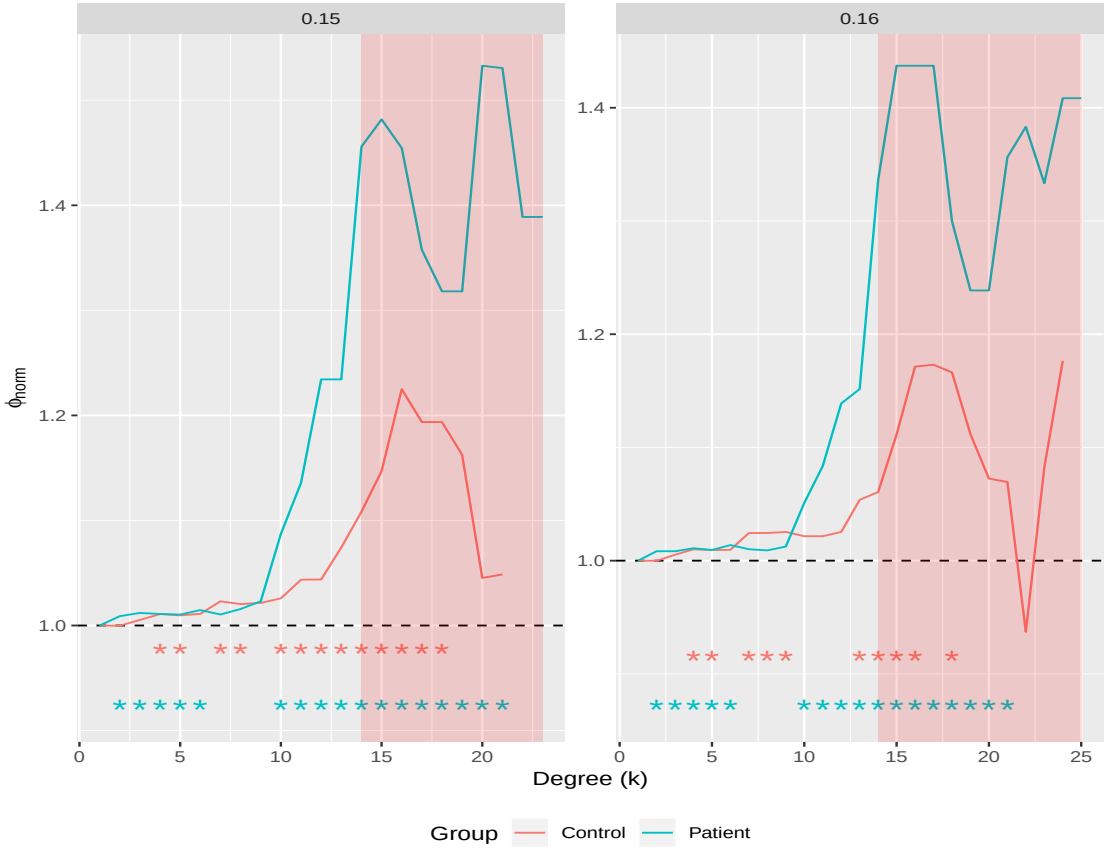



Figure 12.3: Normalized rich club coefficient vs. degree

### 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.

| 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`.

```
g[[1]][[N]] <- rich_club_attrs(g[[1]][[N]],
        c(rich_core(g[[1]][[N]])$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 == groups[1] & p.fdr < 0.05, range(k)]

plot(g[[1]][[N]], 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 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])
```

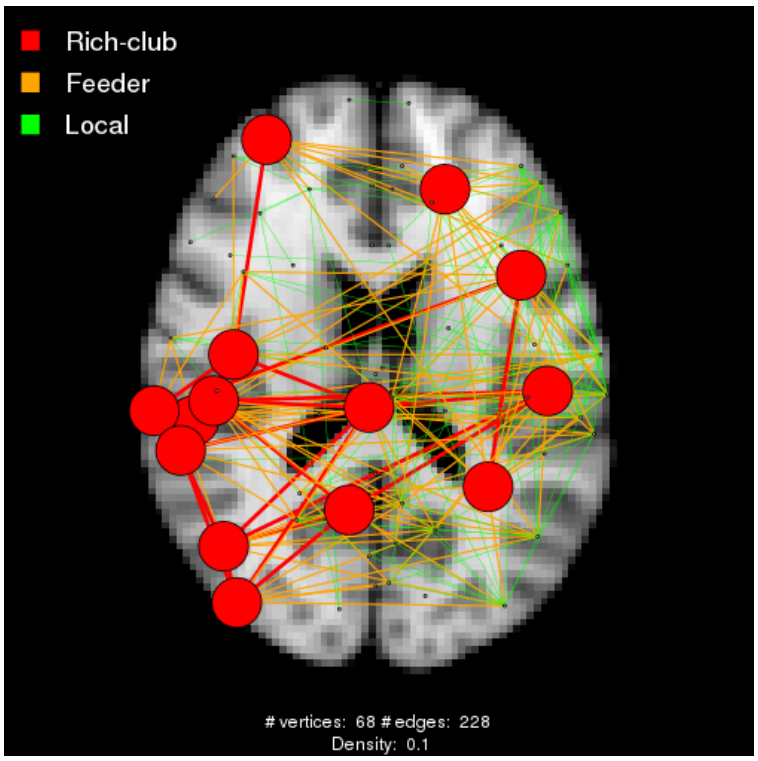



Figure 12.4: Rich-club attributes.

# 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.(19)

### 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 = TRUE, xfm.type = c("1/w", "-log(w)",
##         "1-w"))
## 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: `TRUE` .

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

**groups** A character vector of the subject groups

**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, all.dat.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   0.5514 0.05526 0.4998  0.7164
##  2: Control    0.06   0.5167 0.05259 0.4588  0.6649
##  3: Control    0.07   0.4902 0.05202 0.4285  0.6324
##  4: Control    0.08   0.4543 0.05142 0.3769  0.5785
##  5: Control    0.09   0.4272 0.05159 0.3407  0.5429
##  6: Control    0.10   0.4132 0.05071 0.3294  0.5281
##  7: Control    0.11   0.3995 0.05006 0.3180  0.5143
##  8: Control    0.12   0.3883 0.04893 0.3116  0.5034
##  9: Control    0.13   0.3794 0.04780 0.3083  0.4957
## 10: Control    0.14   0.3603 0.04696 0.2834  0.4675
## 11: Control    0.15   0.3469 0.04610 0.2697  0.4504
## 12: Control    0.16   0.3527 0.04537 0.2939  0.4717
## 13: Control    0.17   0.3376 0.04413 0.2762  0.4492
## 14: Control    0.18   0.3204 0.04258 0.2544  0.4213
## 15: Control    0.19   0.3113 0.04223 0.2456  0.4111
## 16: Control    0.20   0.3013 0.04104 0.2365  0.3974
## 17: Patient    0.05   0.5413 0.05242 0.4361  0.6415
## 18: Patient    0.06   0.5235 0.04920 0.4329  0.6258
## 19: Patient    0.07   0.4954 0.04653 0.4035  0.5859
```

```
## 20: Patient    0.08   0.5034 0.04309 0.4464  0.6153
## 21: Patient    0.09   0.4989 0.04108 0.4588  0.6198
## 22: Patient    0.10   0.4866 0.04024 0.4517  0.6095
## 23: Patient    0.11   0.4711 0.03871 0.4399  0.5916
## 24: Patient    0.12   0.4668 0.03778 0.4480  0.5960
## 25: Patient    0.13   0.4432 0.03647 0.4167  0.5597
## 26: Patient    0.14   0.4295 0.03535 0.4050  0.5436
## 27: Patient    0.15   0.4045 0.03470 0.3683  0.5043
## 28: Patient    0.16   0.3968 0.03359 0.3668  0.4985
## 29: Patient    0.17   0.3869 0.03261 0.3599  0.4877
## 30: Patient    0.18   0.3802 0.03185 0.3582  0.4831
## 31: Patient    0.19   0.3692 0.03112 0.3478  0.4698
## 32: Patient    0.20   0.3534 0.03063 0.3269  0.4469
##       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",
##         "degree", "E.nodal", "ev.cent", "knn", "transitivity",
##         "vulnerability"), atlas = NULL, .function = NULL)
## NULL
```

**densities**

**resids**

**N** The number of permutations (default: `5000` )

**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` )

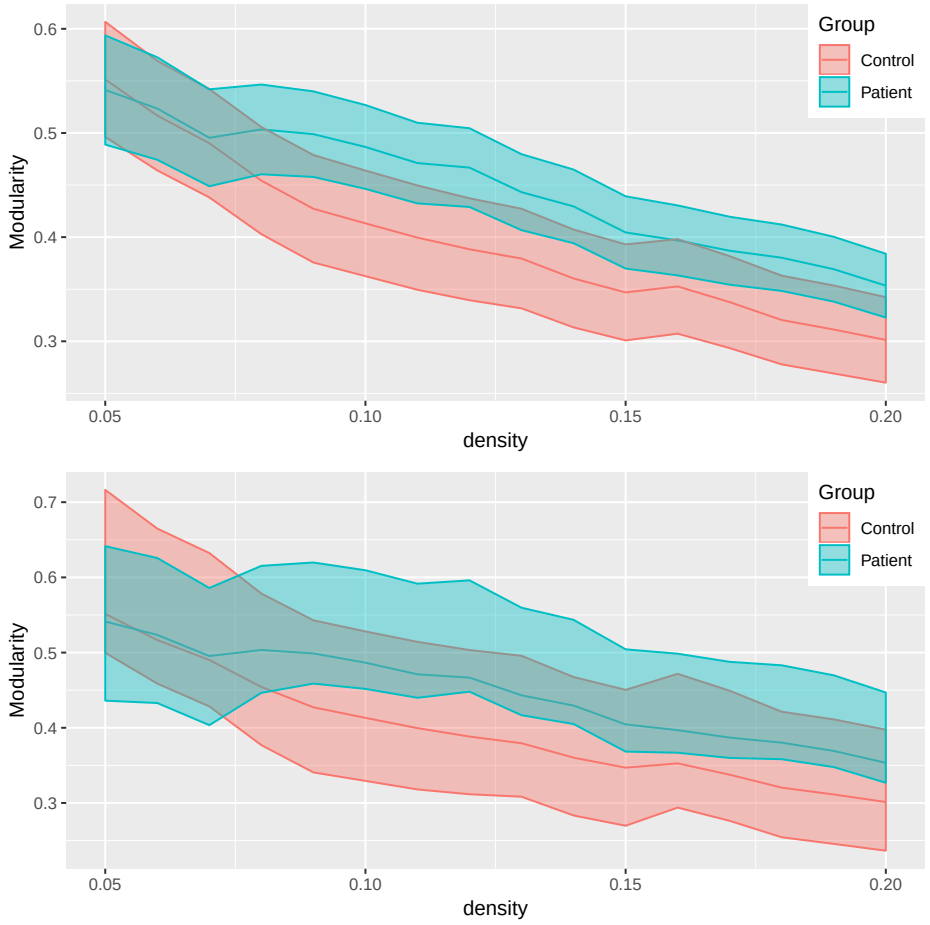



Figure 13.1: **Bootstrap analysis.** Modularity plotted across densities

**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

**atlas** Character string indicating which brain atlas the network data was generated from. Required if `level='graph'`

**.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.

**groups** Character string containing the 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=nrow(all.dat.resids$resids.all), nset=kNumPerms)
perms.all <- brainGraph_permute(densities, all.dat.resids, perms=myPerms,
                                level='graph', atlas=atlas)
```

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  ci.low ci.high
## 1:      0.05  graph      3.028      4.577  -1.5490 -1.4868   2.099
## 2:      0.06  graph      2.850      4.073  -1.2231 -1.1081   1.571
## 3:      0.07  graph      2.697      3.748  -1.0504 -0.7770   1.423
## 4:      0.08  graph      2.696      3.538  -0.8414 -0.5594   1.453
## 5:      0.09  graph      2.511      3.031  -0.5194 -0.4200   1.140
##    perm.diff        p  p.fdr
## 1:  -0.25029 0.036963 0.1183
## 2:  -0.25014 0.031968 0.1183
## 3:  -0.16690 0.014985 0.1183
## 4:  -0.10548 0.005994 0.0959
## 5:  -0.05375 0.025974 0.1183
```

### 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]])
```

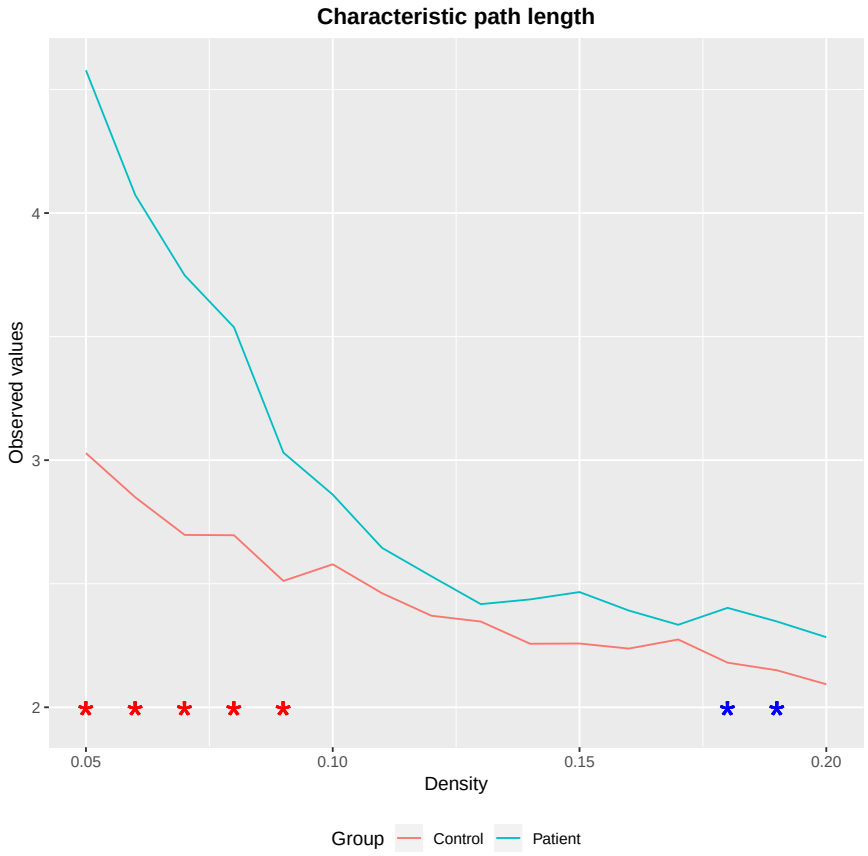



Figure 13.2: **Permutation testing, graph-level.** Observed avg. path length across densities

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]])
```

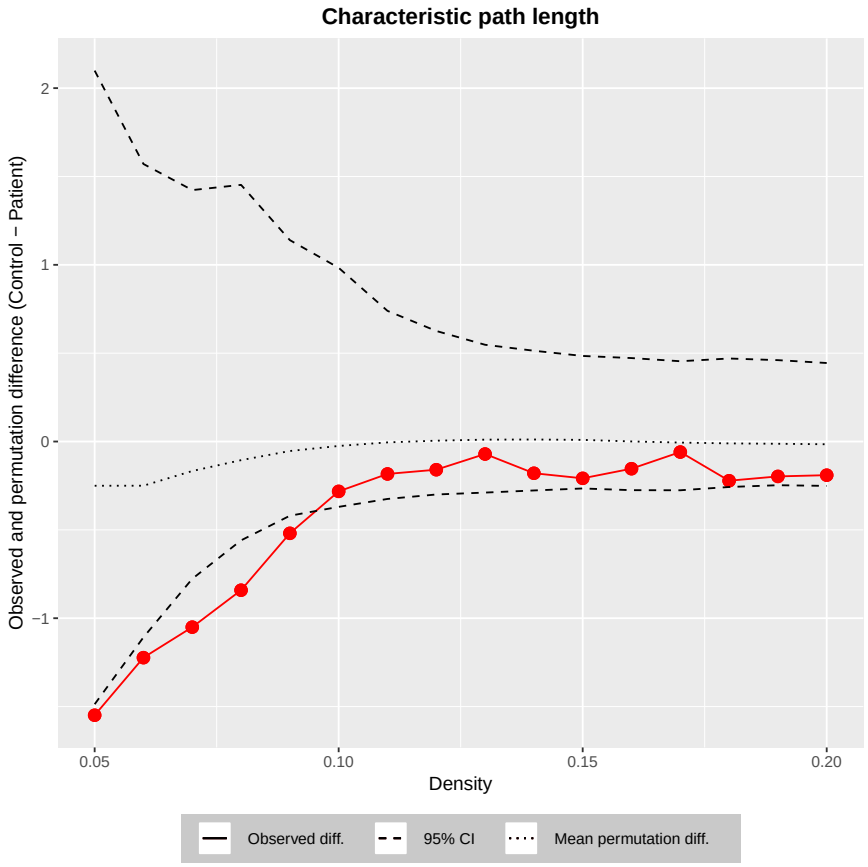



Figure 13.3: **Permutation testing, graph-level.** Observed and permuted differences in avg. path length across densities

### 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. (90) 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)], all.dat.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  ci.low
## 1:      0.11  rLING             8.414            343.30  -334.89 -259.83
## 2:      0.12   lCUN             0.000             86.51   -86.51  -72.07
## 3:      0.12  rLING             7.700            277.38  -269.68 -226.19
## 4:      0.13  rLING             6.393            236.89  -230.50 -216.01
## 5:      0.14  rLING             7.417            241.24  -233.83 -211.55
##    ci.high perm.diff        p  p.fdr
## 1:  243.29   -32.009 0.006993 0.1329
## 2:   98.58     4.342 0.045954 0.4366
## 3:  220.84   -33.829 0.022977 0.4366
## 4:  208.55   -30.514 0.037962 0.7213
## 5:  180.13   -27.407 0.020979 0.3986
```

**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. (36), Hosseini et al. (40)). The usage is identical to that of the previous sections, except you add `auc=TRUE` to the function call.

```
kNumPerms.auc <- 1e2
myPerms.auc <- permute::shuffleSet(n=nrow(all.dat.resids$resids.all), nset=kNumPerms.auc)
perms.all.auc <- brainGraph_permute(densities, all.dat.resids, perms=myPerms.auc,
                                    level='graph', atlas=atlas, 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, all.dat.resids, perms=myPerms.auc,
                                     level='vertex', auc=TRUE)
plot(perms.btwn.auc, alt='less')
```

### 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

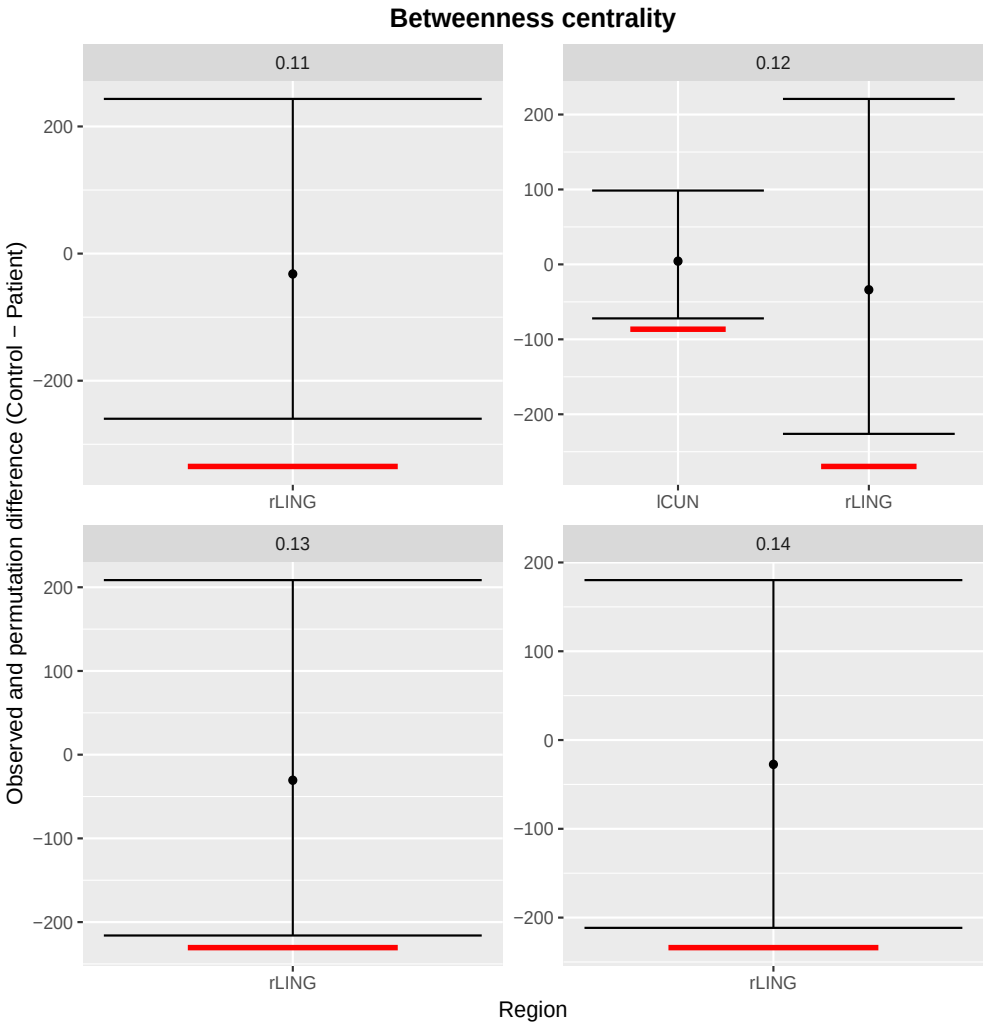



Figure 13.4: **Permutation testing, vertex-level.** Group differences in vertex betweenness centrality (observed difference in red)

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, all.dat.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":

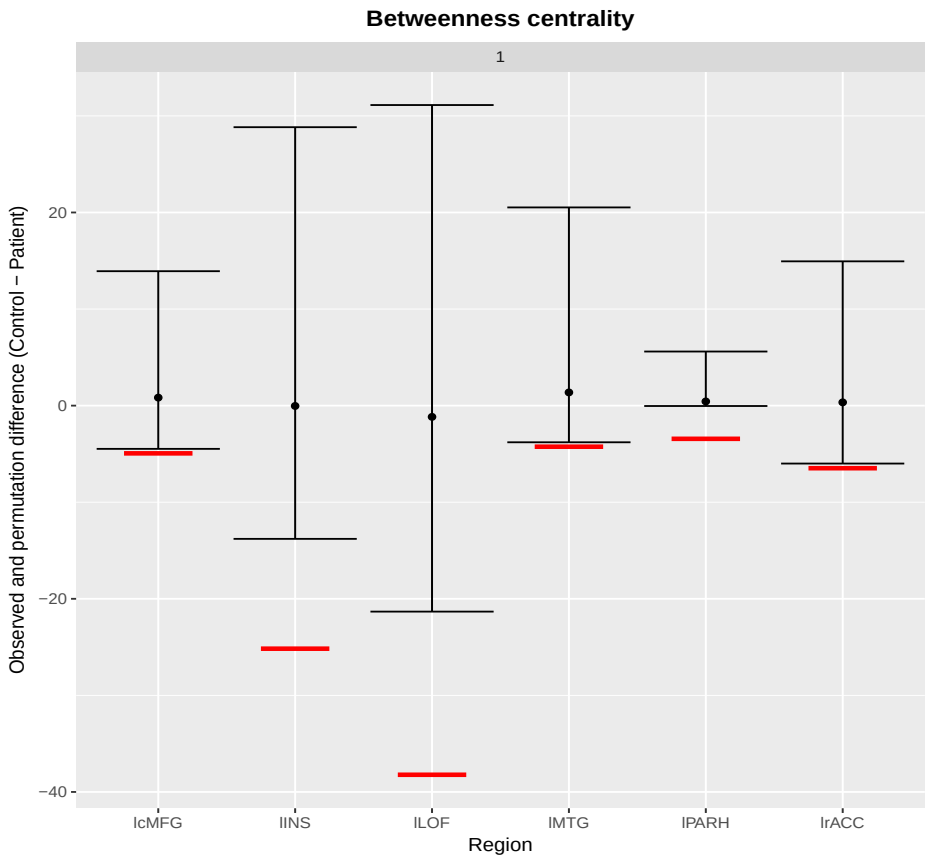



Figure 13.5: **Permutation testing, vertex-level.** Group differences in AUC of vertex betweenness centrality (observed difference in red)

```
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
##
##    densities region other.Control other.Patient obs.diff    ci.low ci.high
## 1:         1  lcACC             0             0 -0.01206 -0.009595 0.03676
##    perm.diff      p
## 1:  0.005481 0.0297
```

# 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. (7) 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.

```
v.removed <- seq(0, 1, length=(kNumVertices + 1))
e.removed <- seq(0, 1, length=(ecount(g[[1]][[N]]) + 1))
attack.vertex.degree <- c(sapply(g, function(x)
                                 robustness(x[[N]], type='vertex', measure='degree')))
attack.vertex.degree <- data.table(comp=attack.vertex.degree,
                                   removed=rep(v.removed, 2),
                                   Group=rep(groups, each=(kNumVertices + 1)))
attack.vertex.btwn <- c(sapply(g, function(x)
                               robustness(x[[N]], type='vertex', measure='btwn.cent')))
attack.vertex.btwn <- data.table(comp=attack.vertex.btwn,
                                 removed=rep(v.removed, 2),
                                 Group=rep(groups, each=(kNumVertices + 1)))
attack.edge <- c(sapply(g, function(x) robustness(x[[N]], type='edge')))
attack.edge <- data.table(comp=attack.edge,
                          removed=rep(e.removed, 2),
                          Group=rep(groups, each=(ecount(g[[1]][[N]]) + 1)))

attack.vertex <- rbind(attack.vertex.degree, attack.vertex.btwn)
attack.vertex$type <- rep(c('deg', 'btwn'), each=2*(kNumVertices+1))

mylabels <- gsub('\\.', ', ', attack.vertex[, levels(interaction(Group, type))])
attack <- ggplot(data=attack.vertex,
```

```
                aes(x=removed, y=comp, col=interaction(Group, type),
                    linetype=interaction(Group, type))) +
  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)
```

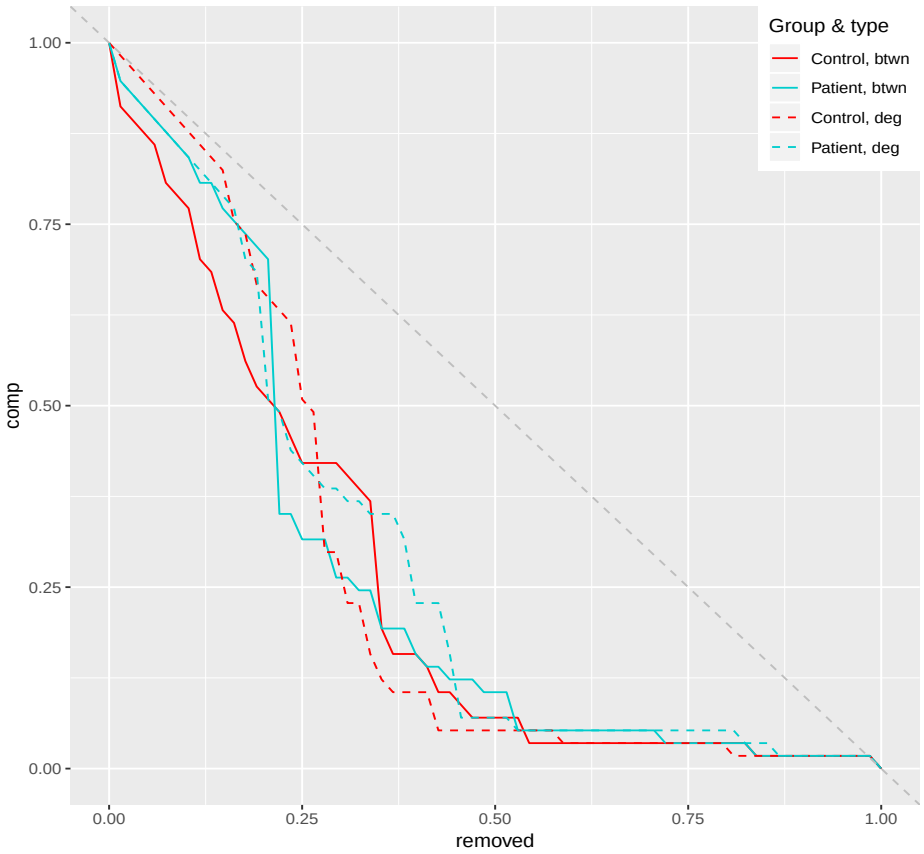



Figure 14.1: Maximal component size as a function of vertices removed.

If you are interested in *vulnerability*, see the function `vulnerability`; also, each graph is given both global- and vertex-level attributes called `vulnerability` after calling `set_brainGraph_attr`.

### 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. This is repeated for many iterations (here, I only use 100 to save time).

Here, I will perform this analysis for both vertices and edges. The 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 <- c(sapply(g, function(x)
                           robustness(x[[N]], measure='random', N=1e2)))
failure.edge <- c(sapply(g, function(x)
                         robustness(x[[N]], type='edge', measure='random', N=1e2)))

failure.vertex <- data.table(comp=failure.vertex,
                             removed=rep(seq(0, 1, length=(kNumVertices + 1)), 2),
                             Group=rep(groups, each=(kNumVertices + 1)))
failure.edge <- data.table(comp=failure.edge,
                           removed=rep(seq(0, 1, length=ecount(g[[1]][[N]]) + 1), 2),
                           Group=rep(groups, each=(ecount(g[[1]][[N]]) + 1)))

failure.edge[, type := 'Random edge removal']
failure.vertex[, type := 'Random vertex removal']
failure.dt <- rbind(failure.edge, failure.vertex)
attack.vertex.btwn[, type := 'Targeted vertex attack']
attack.edge[, type := 'Targeted edge attack']
robustness.dt <- rbind(failure.dt, attack.vertex.btwn, attack.edge)
```

```
ggplot(robustness.dt, aes(x=removed, y=comp, 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)) +
  xlab('% edges/vertices removed') +
  ylab('% of max. component remaining')
```

## 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 for more than 2 groups. Using the Kruskal-Wallis test in the code allows for an arbitrary number of subject groups, and if you only have 2, then it is equivalent to a Wilcoxon test anyway.

```
dists.dt <- data.table(density=rep(rep(densities, length(groups)),
                                   times=rep(sapply(g[[1]], ecount),
                                             length(groups))),
                       dists=do.call('c', sapply(g, sapply, function(x)
                                                 E(x)$dist)),
                       Group=rep(groups, times=rep(sum(sapply(g[[1]], ecount)),
                                                   length(groups))))
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))$p.val, by=density]
p.fdr <- p.adjust(unique(dists.dt$p), 'fdr')
```

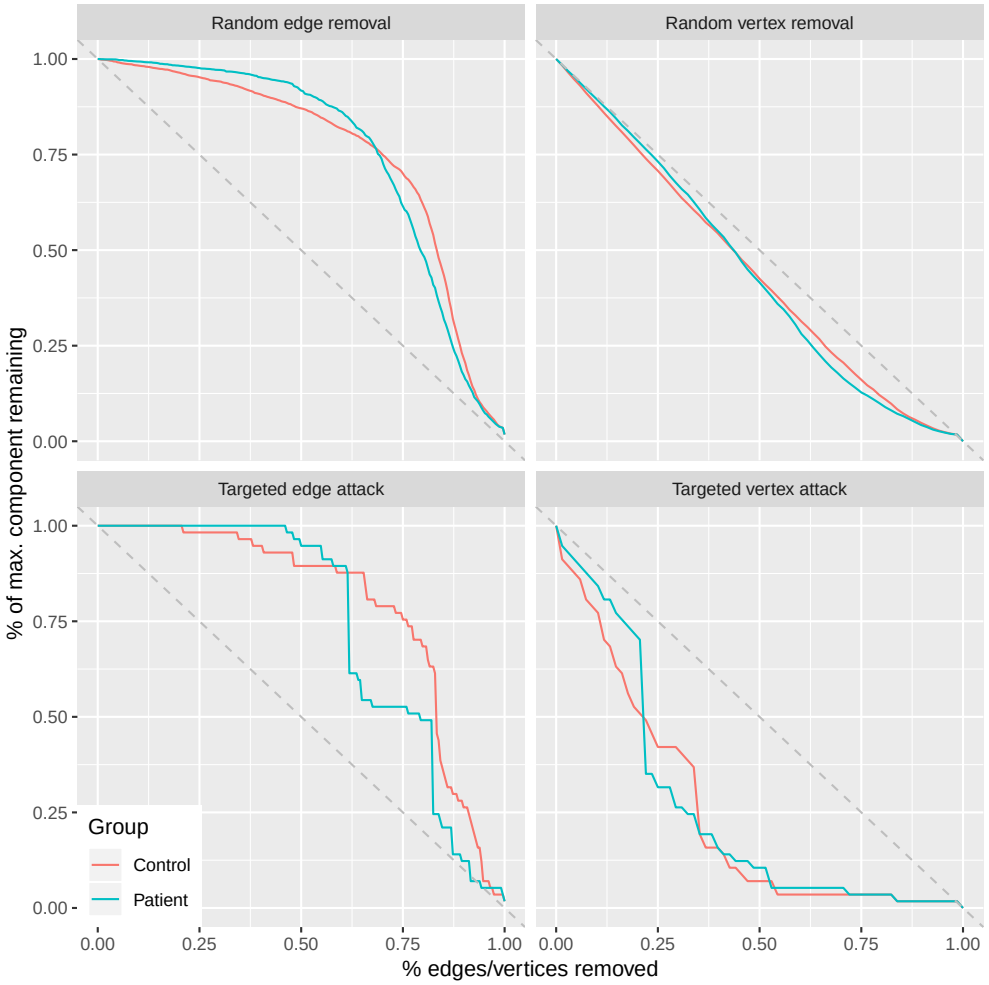



Figure 14.2: Maximal component size as a function of vertices (edges) removed.

```
dists.dt$p.fdr <- rep(rep(p.fdr, length(groups)),
                      times=rep(sapply(g[[1]], ecount), each=length(groups)))
dists.dt$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)]))
```

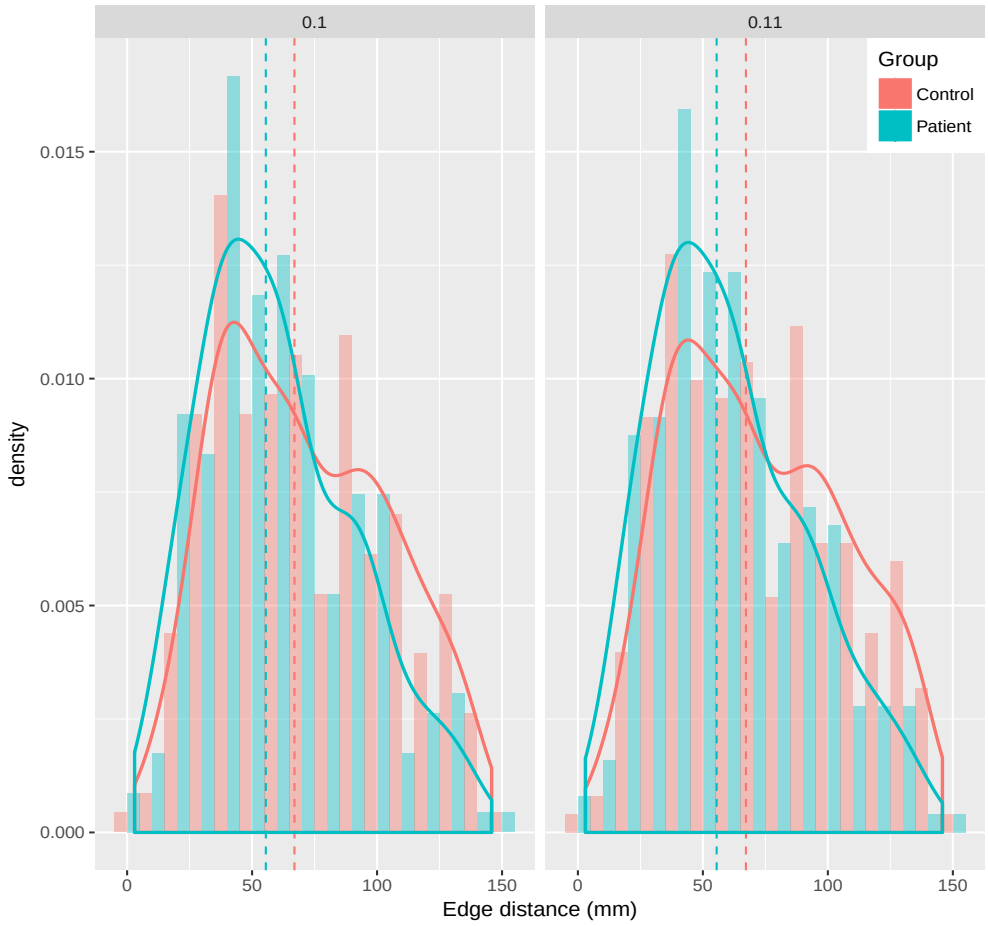



Figure 14.3: Edge distances (in mm)

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))
```

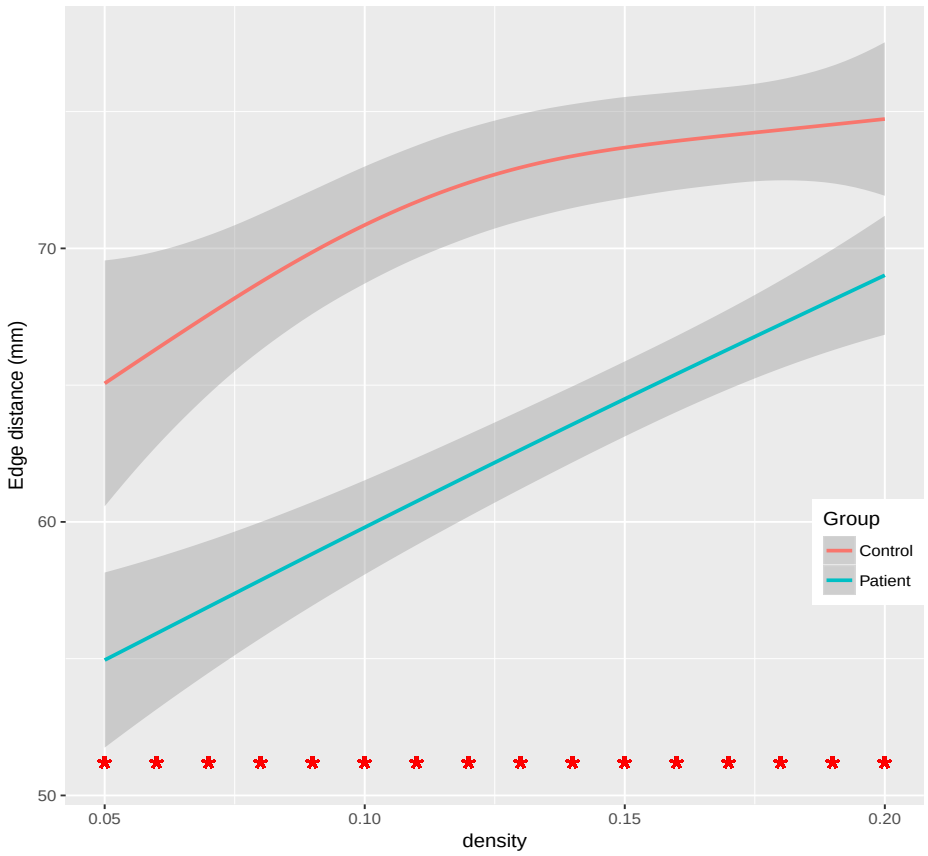



Figure 14.4: Average edge distances (in mm) with 99% confidence intervals

## 14.3 Individual contributions

Saggar et al. (2015) recently introduced two methods for estimating the individual contributions of subjects to a group graph (74). 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)*).

### 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(all.dat.resids, corrs[[1]]$R)
RC.aop <- aop(all.dat.resids, corrs[[1]]$R, level='regional')
```

We can also plot the regional contribution for the group (see Figure 14.5):

```
ggplot(RC.aop, aes(x=region, y=RC, col=Group, group=Group)) +
  geom_line(stat='summary', fun.y=mean) +
  theme(legend.position='none',
        axis.text.x=element_text(size=6, angle=45, vjust=0.5))
```

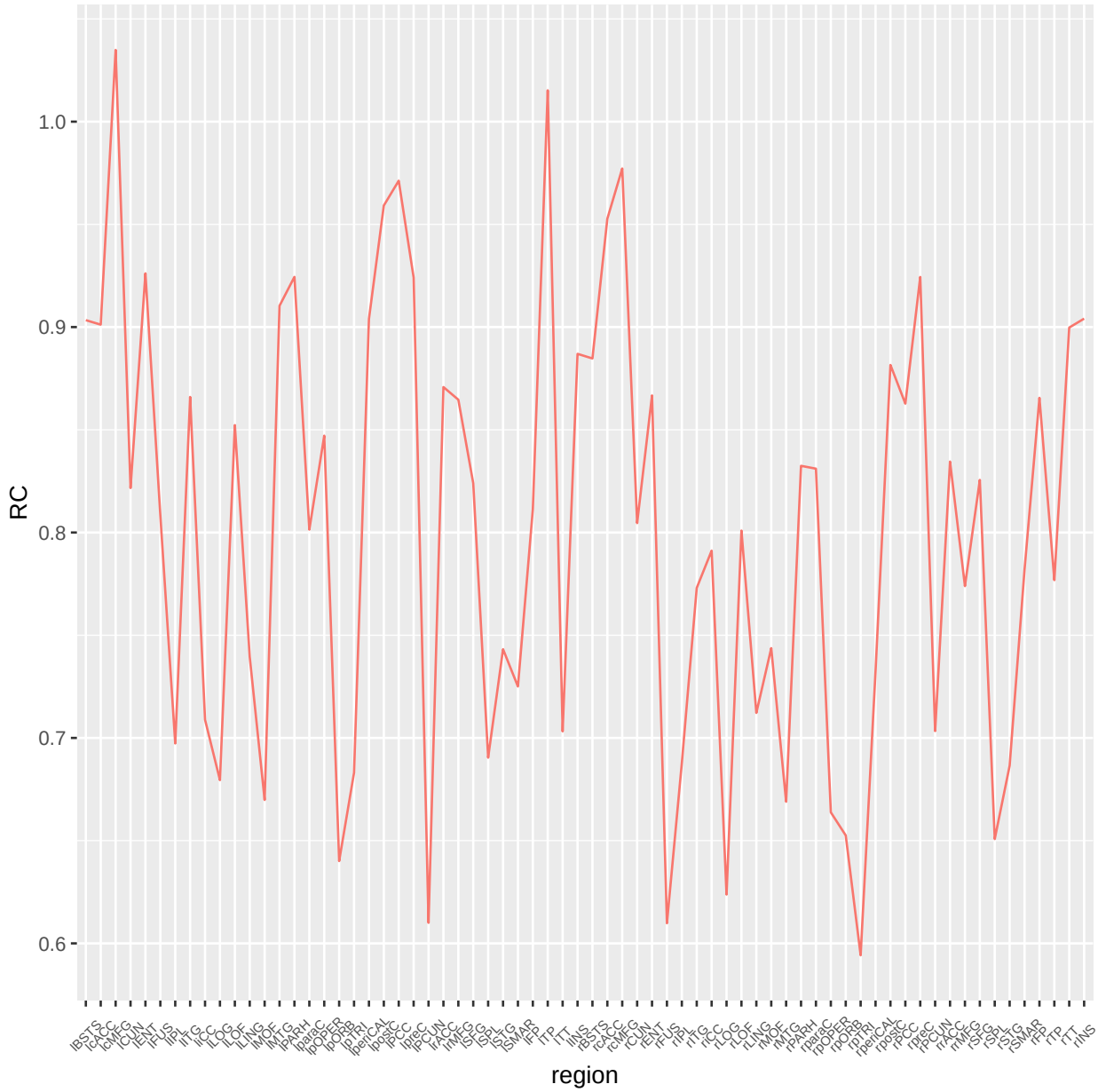



Figure 14.5: Regional contribution; add-one-patient

### 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(all.dat.resids, corrs)
RC.loo <- loo(all.dat.resids, corrs, 'regional')
```

We can also plot the regional contribution for the group (see Figure 14.6):

```
ggplot(RC.loo, aes(x=region, y=RC, col=Group, group=Group)) +
  geom_line(stat='summary', fun.y=mean) +
  theme(legend.position='bottom',
        axis.text.x=element_text(size=6, angle=45, vjust=0.5))
```


Figure 14.6: Regional contribution; leave-one-out

# Part VI

# Visualization

# 15

# GUI and other plotting functionality

## 15.1 The GUI

I created a GUI to quickly and flexibly compare graphs (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 two plotting sections for showing 2 groups (or subjects) side-by-side. Alternatively, if you have a single group, you can look at different densities or different orientations.

The # of vertices, # of edges, and the graph's density are printed at the bottom of each figure window. At the top of each figure window, the graph's `name` attribute is printed (in this case, group).

### 15.1.1 Orientation

You can choose one of: *axial*, *sagittal (L)*, *sagittal (R)*, or *circular*. The sagittal plots 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; you may also choose to show one hemisphere individually or only *inter*-hemispheric edges. There are options to show only edges between homologous regions, as well. Finally, you can select only inter-/intra-community edges, and inter-/intra-lobar edges.

### 15.1.3 Legend

If vertices are colored by *lobe*, *class*, or *network*, then you may choose to show a legend. The numbers represent the # of vertices present out of the total # of vertices for that lobe/class/network in that specific atlas.

### 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 color*: vertices will be colored according to either community membership (weighted or unweighted), lobe membership (as in Figure 15.1), component membership (useful if your graph is disconnected), and for the *Destrieux* atlas, I included a `Class` attribute, where `Class` is either *G*

*(gyral)*, *S (sulcal)*, or *G & S (gyral and sulcal)*. For the *Dosenbach160* atlas, there is also an option to color vertices by `network` (e.g., *Default-mode*).

- *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 Lobe

At the bottom-left of the GUI, you can choose to see vertices of all lobes, or *any* combination of other lobes (e.g., if you want to see Frontal & Temporal & Occipital only). To select multiple lobes, you can press `ctrl` and click, or hold `shift` and use the arrow keys.

### 15.1.7 Neighborhoods

In the dropdown `Plot` menu at the top of the window, there is an option to plot vertex neighborhoods. 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. To select multiple vertices, you can press `ctrl` and click, or hold `shift` and use the arrow keys.

### 15.1.8 Communities

From the same dropdown menu, you can plot individual (or any combination of multiple) communities. Figure 15.3 shows an example. The numbering here is in order from largest to smallest. Communities with 2 or fewer members are not included. Here I show the 2 largest (*red* is for the largest, *green* for the second-largest). To select multiple communities, you can press `ctrl` and click, or hold `shift` and use the arrow keys.

### 15.1.9 Keyboard shortcuts

There are a number of keyboard shortcuts available.

**alt+f** Opens the *File* dropdown menu

**alt+p** Opens the *Plot* dropdown menu

**ctrl+e** Plot Entire graphs

**ctrl+n** Plot Neighborhoods

**ctrl+c** Plot Communities

**alt+o** Same as clicking *Ok*

**alt+n** Same as clicking *Picking new graphs*

**alt+l** Toggle the vertex label checkbox

**alt+d** Toggle the *Show diameter* checkbox

**ctrl+1** Save figure 1 (left)

**ctrl+2** Save figure 2 (right)

**ctrl+q** Quit

## 15.2 Other plotting

### 15.2.1 Adjacency matrix plots

It is possible to plot the adjacency matrices. I haven't found these to be particularly useful, except when viewing the vertices in a specific order (e.g., by community membership). This is the default in the function `plot_corr_mat`. Figures 15.4 and 15.5 show them clustered into communities, and Figures 15.6 and 15.7 show them clustered into lobes (zoom in to read the axis labels).

```
matplot.comm1 <- plot_corr_mat(corrs[[1]]$r.thresh[, , 10], type='comm',
                               g=g[[1]][[10]], group=groups[1])
matplot.comm2 <- plot_corr_mat(corrs[[2]]$r.thresh[, , 10], type='comm',
                               g=g[[2]][[10]], group=groups[2])
matplot.lobe1 <- plot_corr_mat(corrs[[1]]$r.thresh[, , 10], type='lobe',
                               g=g[[1]][[10]], group=groups[1])
matplot.lobe2 <- plot_corr_mat(corrs[[2]]$r.thresh[, , 10], type='lobe',
                               g=g[[2]][[10]], group=groups[2])

# To plot both in the same device, uncomment the following line
#grid.arrange(matplot.comm1, matplot.comm2, nrow=1, ncol=2)
print(matplot.comm1)
```

```
print(matplot.comm2)
```

```
print(matplot.lobe1)
```

```
print(matplot.lobe2)
```

### 15.2.2 Plot global graph measures

Assuming you have already "tidied" your data (see Tidying Data), this can be done very easily with one function, `plot_global`, and is shown in Figure 15.8. There is an option to include a dashed vertical line at a density 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 `data.table` 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.`).

This isn't the cleanest solution, but all of the ugly code is hidden in the function. I wrote this function to prepare a figure easily for a manuscript (because I have a separate script to produce all figures for a given manuscript; this reduces the code I have to write for those). This function *should* be straightforward and flexible.

```
# Re-name the facet labels to whatever you want
# Match to the order of `dt.G.tidy[, levels(variable)]`
level.names <- c('Clustering coefficient', 'Char. path length',
                 'Global efficiency', 'Modularity', 'Max. conn. comp.',
                 '# of triangles', 'Diameter', 'Transitivity',
                 'Degree assortativity', 'Lobe assortativity',
                 'Lobe & hemi. assort.', 'Asymmetry Index', 'Spatial distance',
                 '# of hubs', 'Local efficiency', 'Vulnerability')

# mod, Cp, Lp, assortativity, E.global, assortativity.lobe, asymm
alt <- c('less', 'two.sided', 'less', 'less', 'two.sided', 'less', 'less')
print(plot_global(dt.G.tidy, vline=densities[N], level.names=level.names,
                  exclude='assortativity.lobe.hemi', perms=perms.all,
                  g=g, 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. It currently only works for group-level graphs (i.e., list objects with 2 levels). 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 groups, or multiple conditions (equaling in length to the number of groups). If you would like to have a single group and multiple subsets in the same plot, then repeat the group number in the `groups` argument. The following code block shows an example command, and its output is in Figure 15.9. To increase the size of the panel titles, supply the argument `cex.main` (with a value greater than 2.5).

```
plot_brainGraph_multi(g, groups=1:2, N=5,
  subgraph='coreness > 5', filename='kcore5.png', main='k-core 5',
  vertex.color='color.lobe', edge.color='color.lobe', vertex.label.cex=2)
```

### 15.2.4 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 "tidied" dataset (see Tidying Data). An example is shown in Figure 15.10.

```
plot_vertex_measures(dt.V.tidy[density == densities[N]], facet.by='lobe',
                     'btwn.cent', show.points=TRUE,
                     ylabel='Betweenness centrality')
```

### 15.2.5 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. An example is shown in Figure 15.11. 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.

```
plot_volumetric(all.dat.resids$all.dat.tidy, c('lSMAR', 'rSMAR'), 'histogram')
```

An example of looking at 9 regions at once is shown in Figure 15.12.

```
plot_volumetric(all.dat.resids$all.dat.tidy, 1:9, 'violin')
```

### 15.2.6 Save a list of graph plots

The function `plot_brainGraph_list` will save a series of *PNG* files of axial graph plots. The list may contain graphs for a single subject group at each density/threshold, or the list may represent single subject graphs; in the latter case, this is a quick way to "scroll" through all of the graphs 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_brainGraph_list(g[[1]], '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_brainGraph_list(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 *.png animation.gif

# Each frame will be displayed for 0.5 seconds
ffmpeg -framerate 2 -i group1_dk_%03d.png -c:v libx264 -r 30 \
    -pix_fmt yuv420p out.mp4
```

### 15.2.7 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* (97). 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]], node.color='community', node.size='degree',
               edge.wt='t.stat', file.prefix='group1')
```

[1] I plan on including some sort of outlier detection in the future.

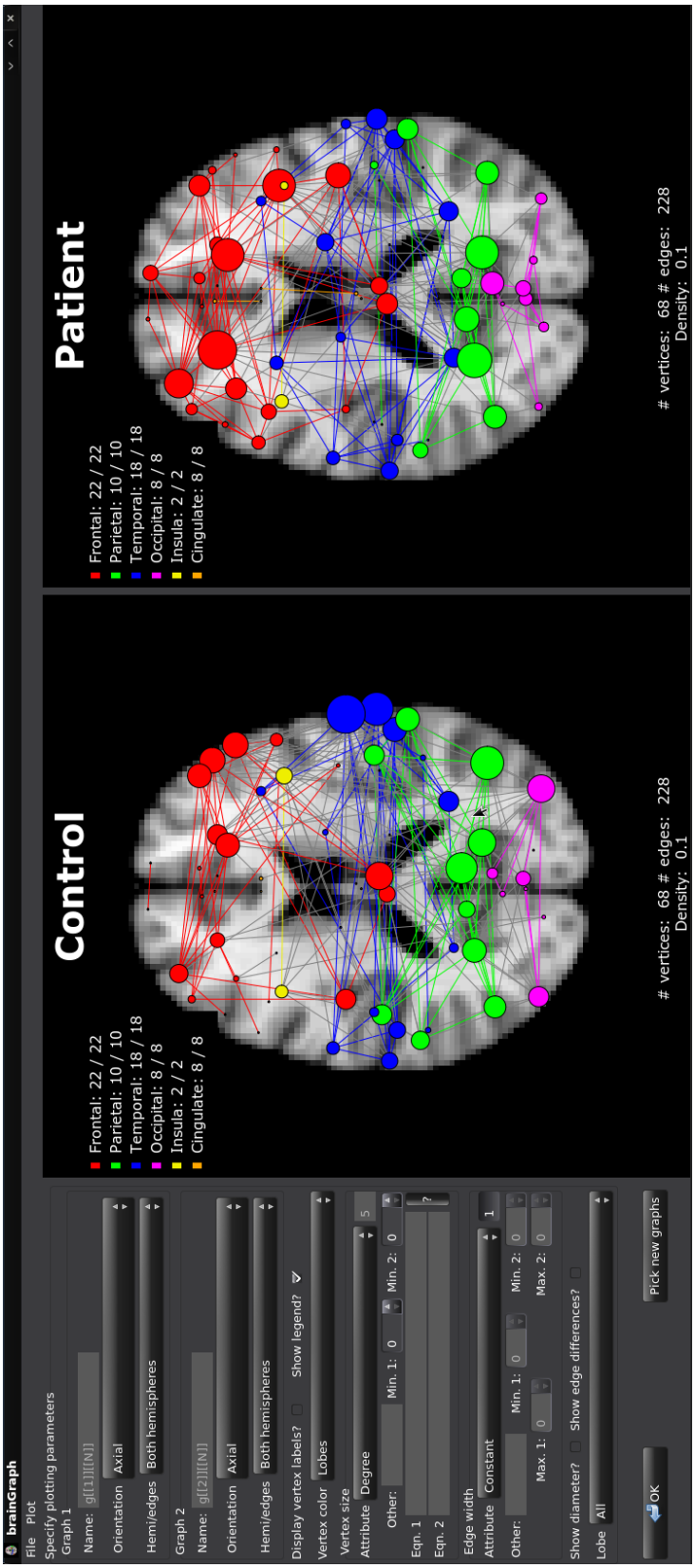



Figure 15.1: The `brainGraph` plotting GUI.

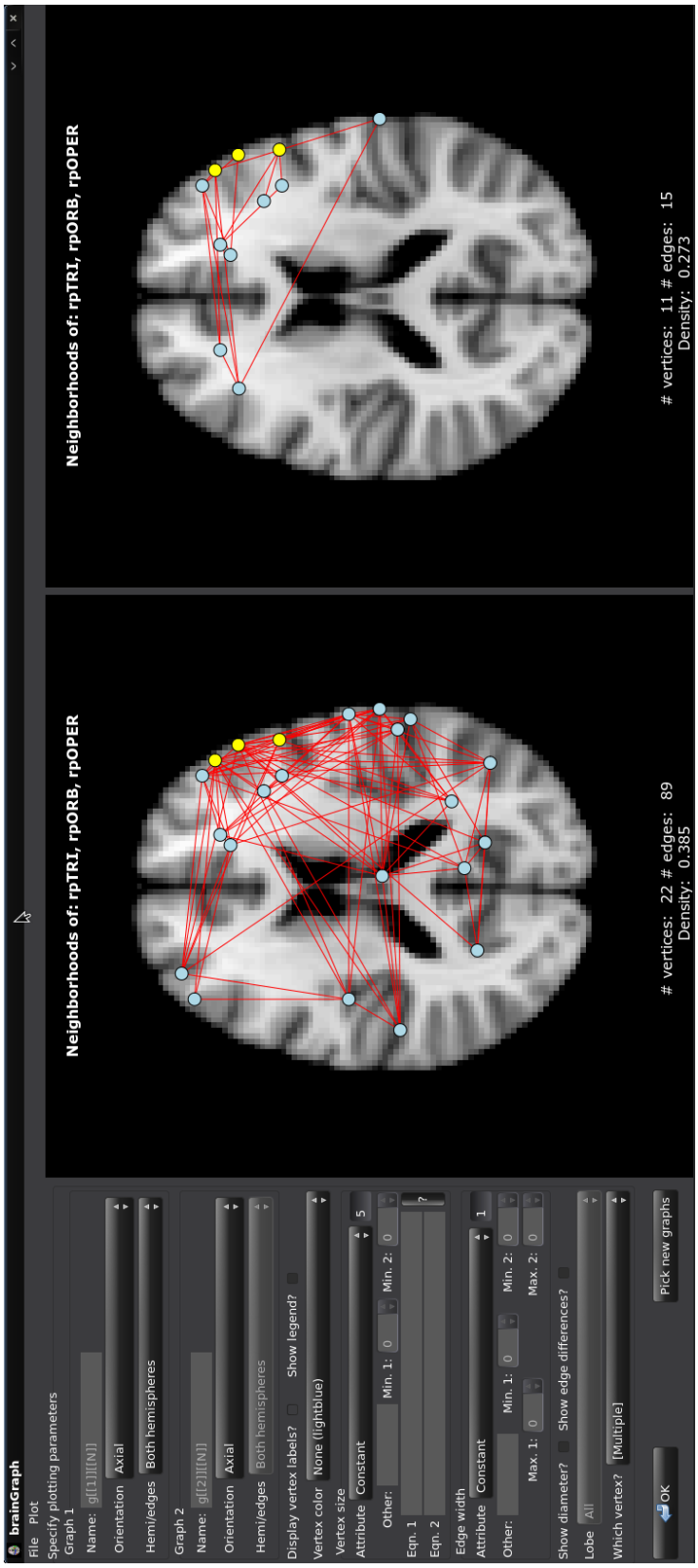



Figure 15.2: The `brainGraph` plotting GUI; neighborhoods.

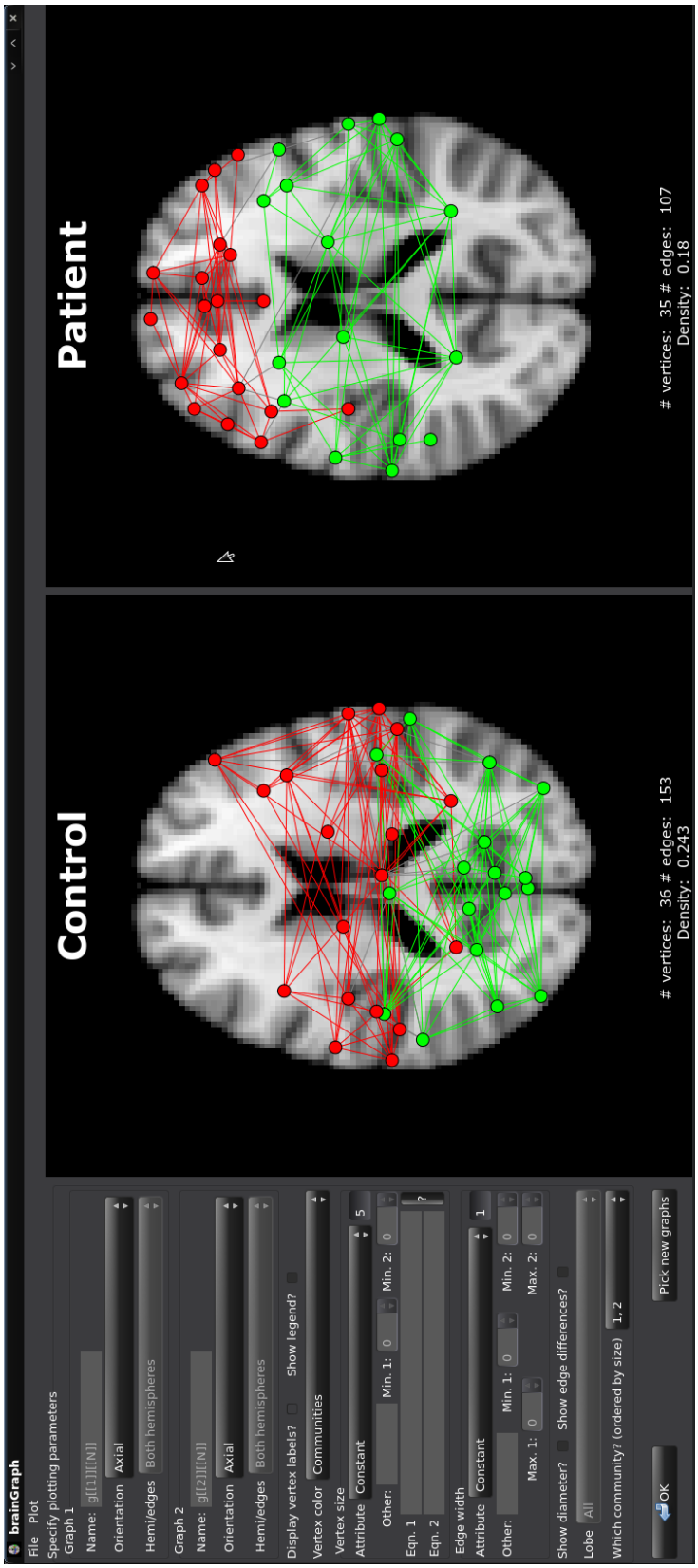



Figure 15.3: The `brainGraph` plotting GUI; communities.

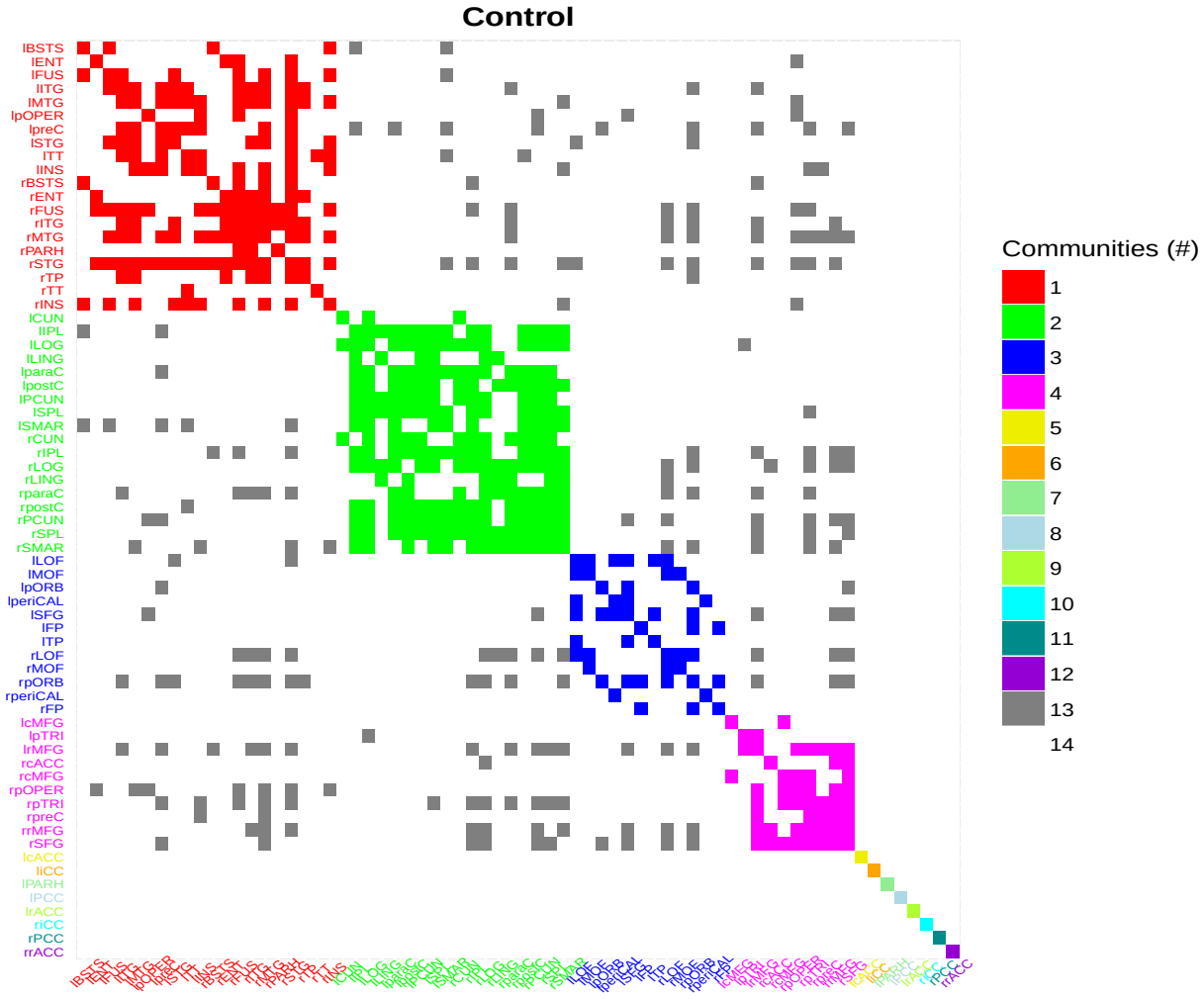



Figure 15.4: Adjacency matrix, group 1.

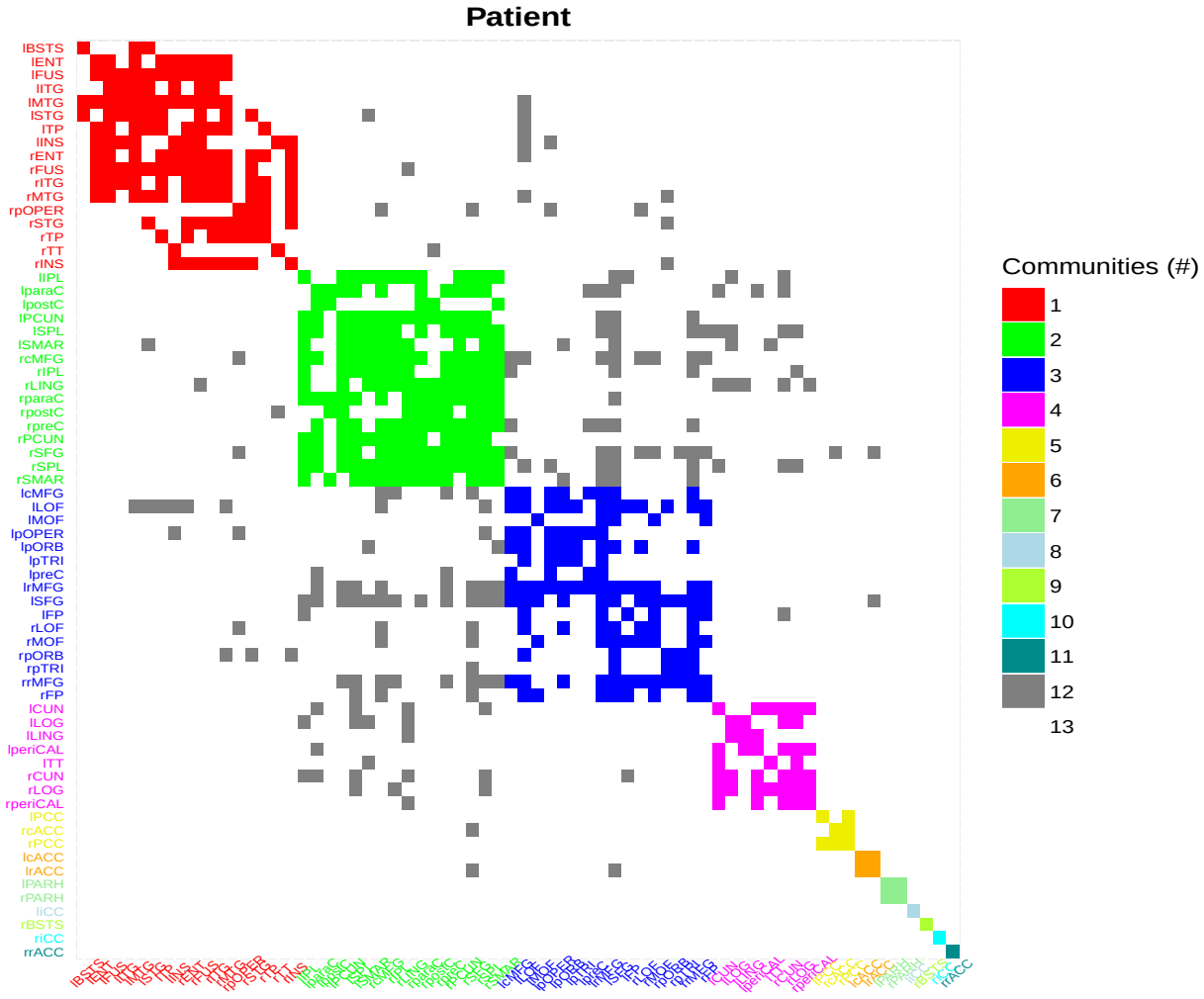



Figure 15.5: Adjacency matrix, group 2.

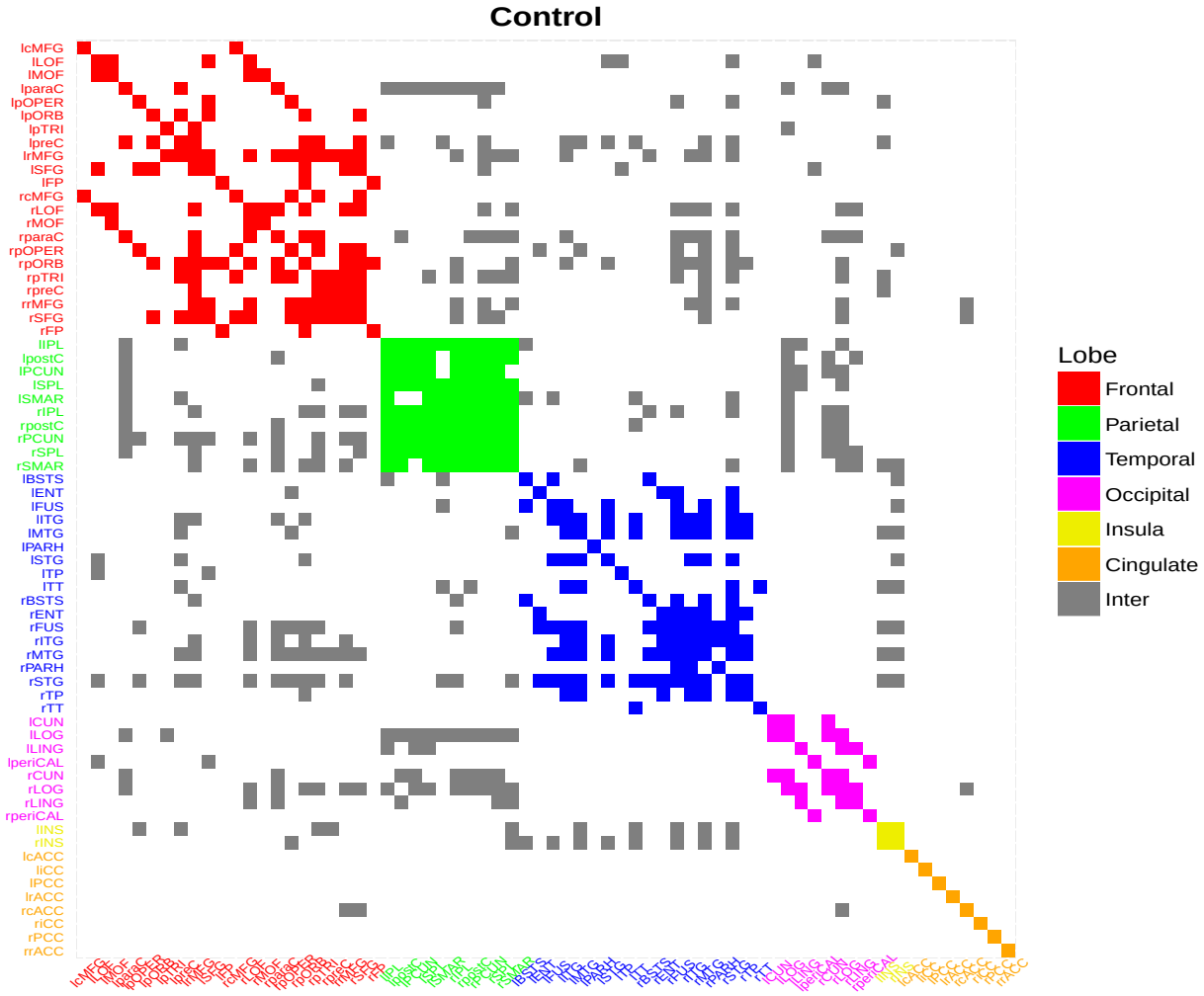



Figure 15.6: Adjacency matrix, group 1.

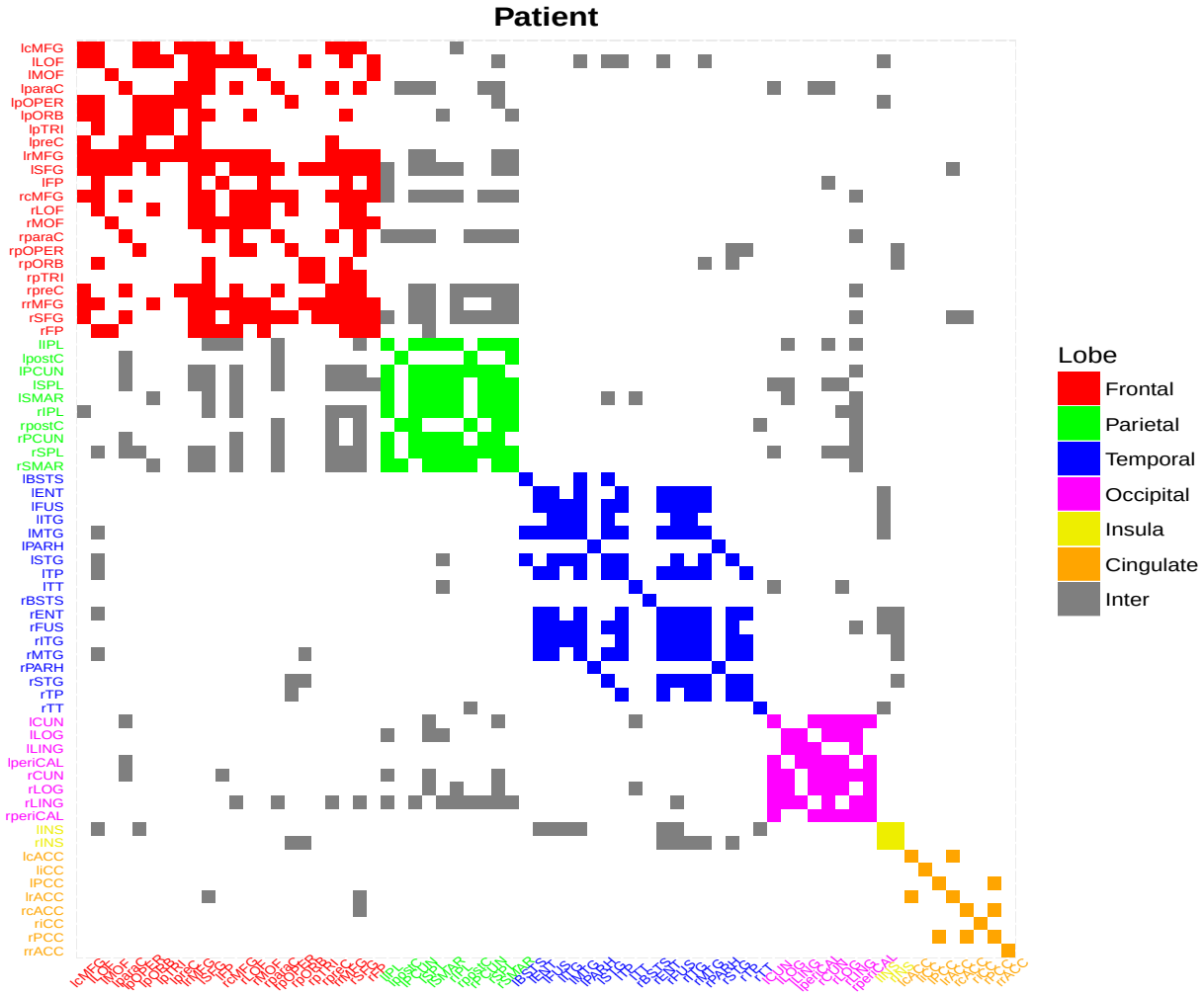



Figure 15.7: Adjacency matrix, group 2.

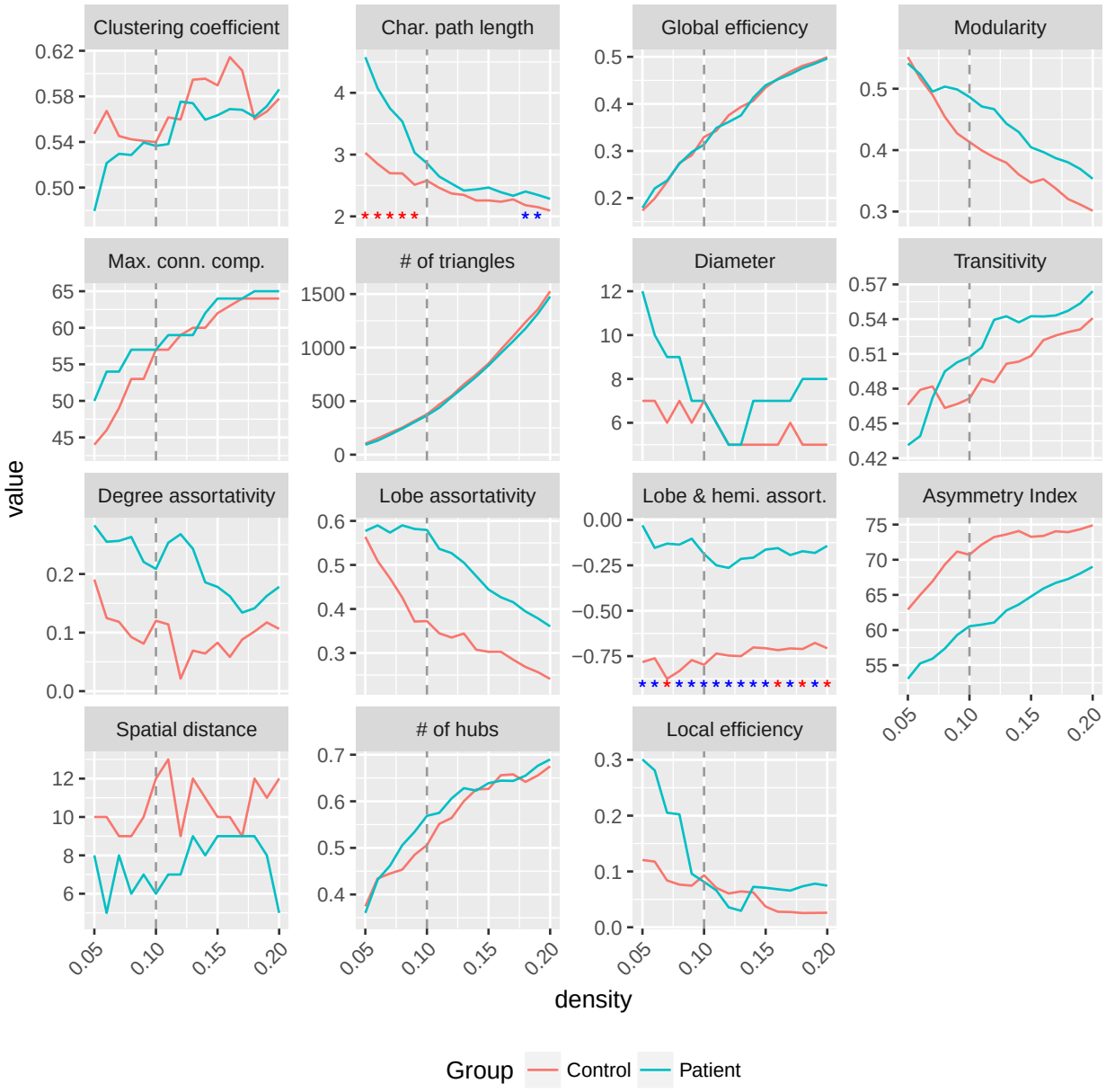



Figure 15.8: Global graph measures vs. density

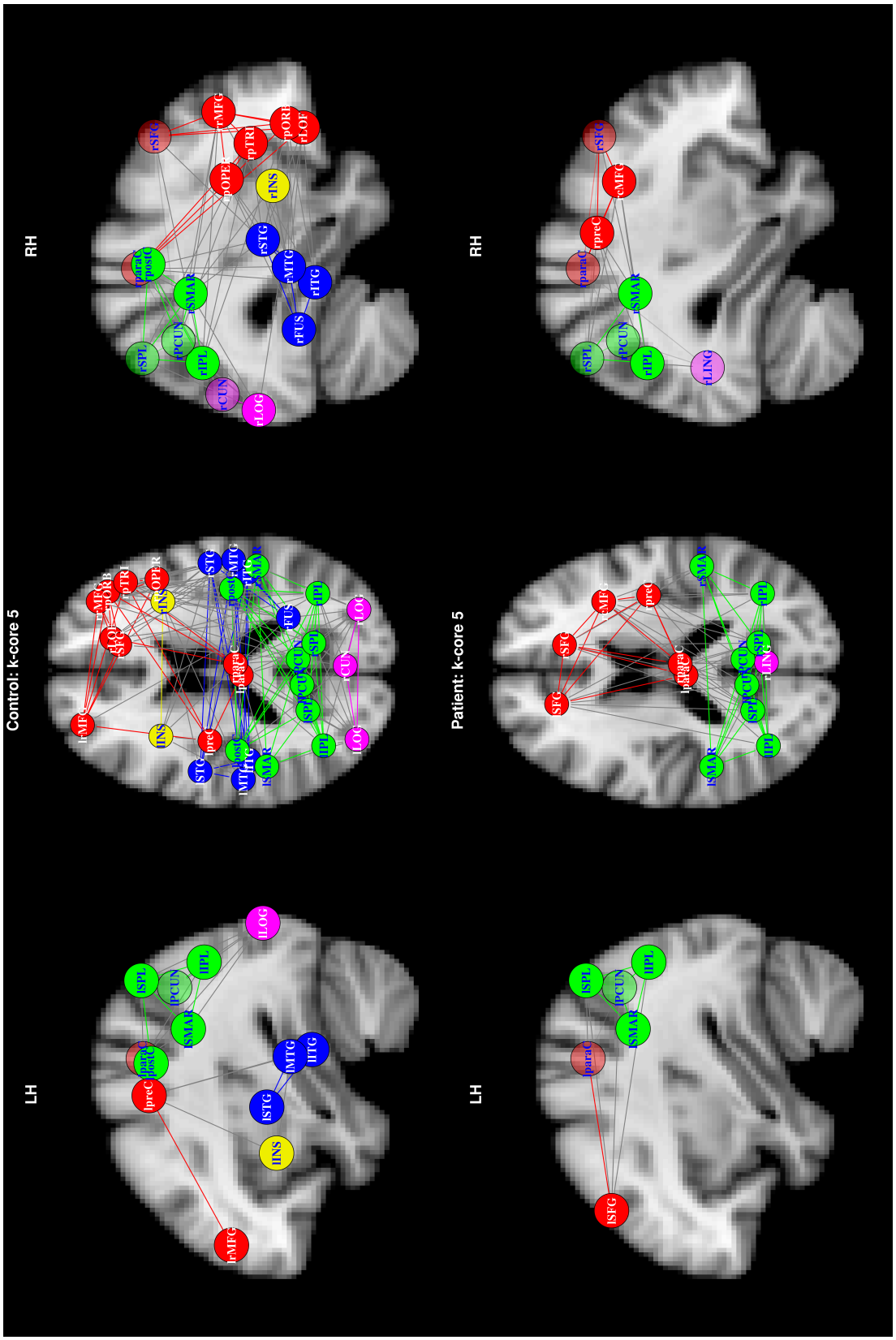



Figure 15.9: Vertices for which "coreness" is greater than 5.

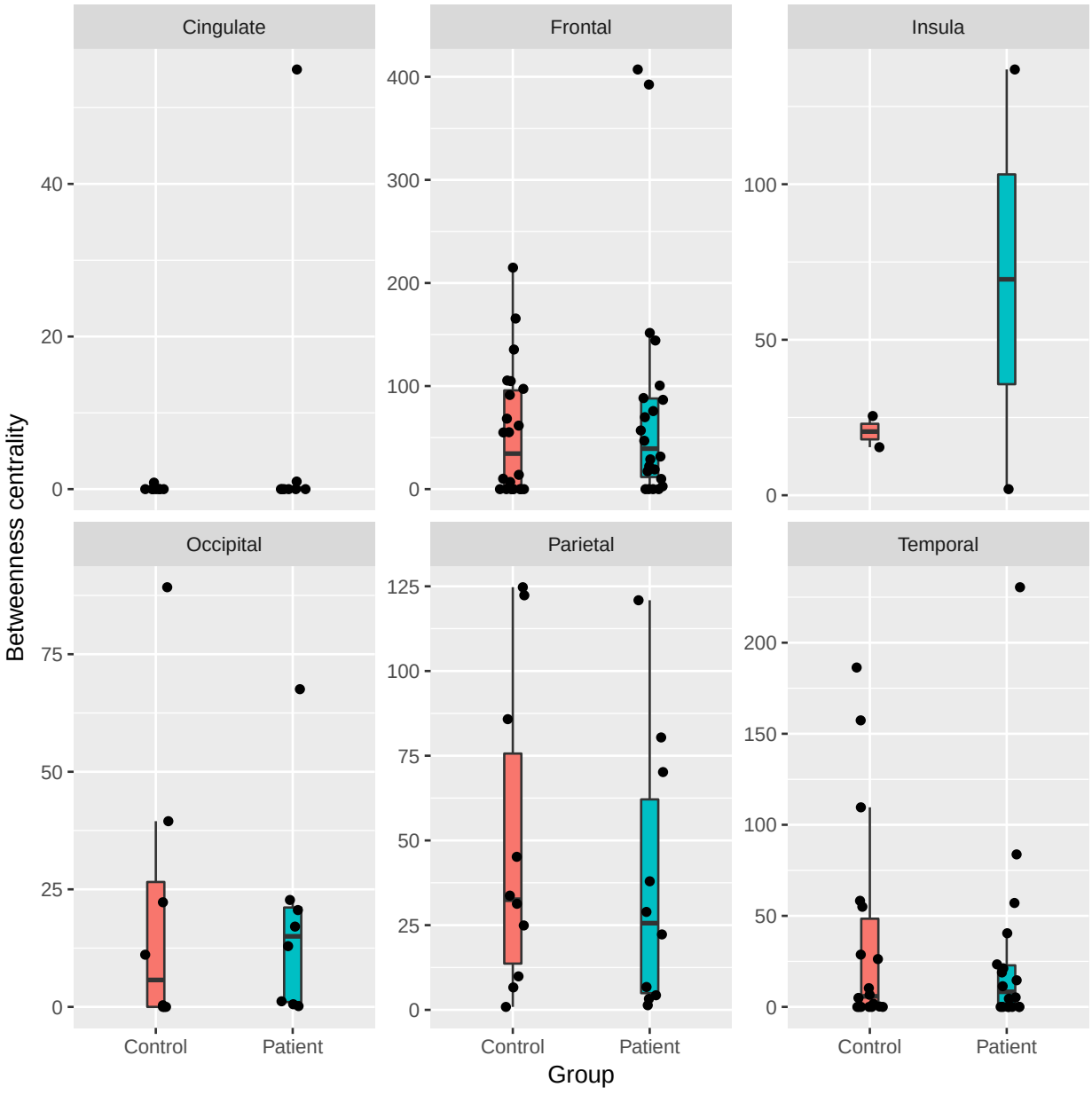



Figure 15.10: Vertex-level graph measures by lobe.

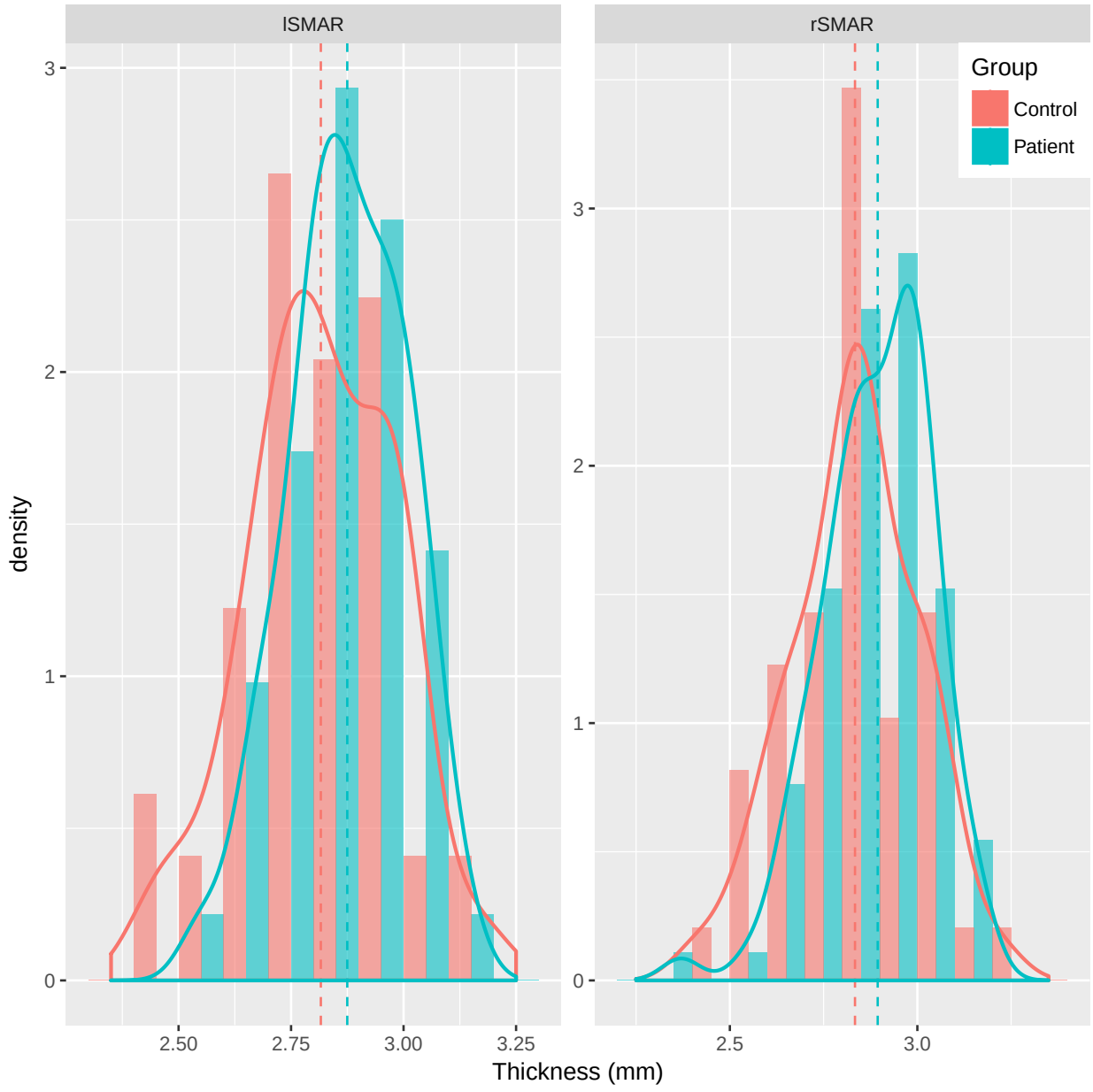



Figure 15.11: Cortical thickness.

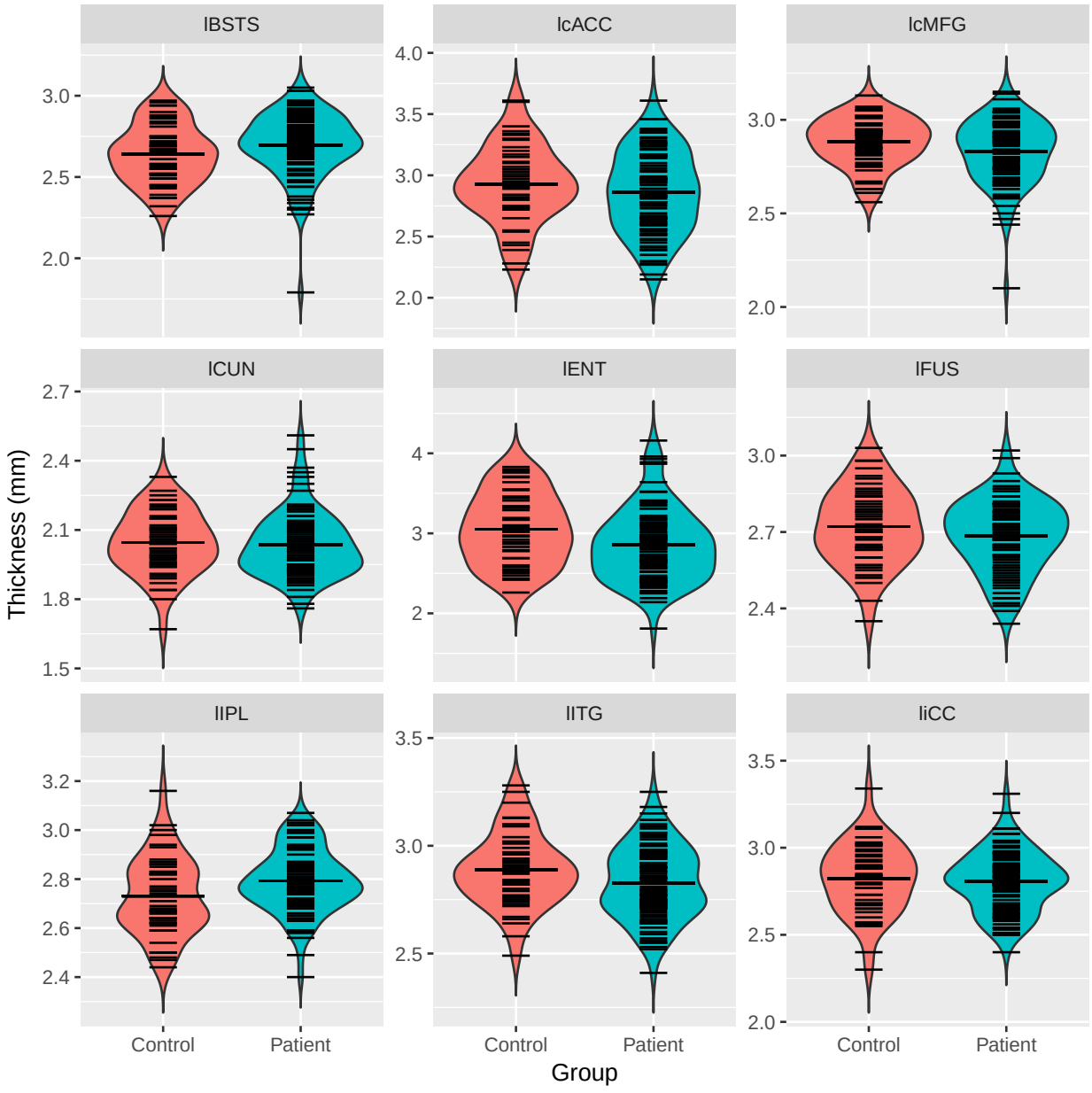



Figure 15.12: Cortical thickness.

# A

# Attributes created by `set_brainGraph_attr`

## A.1 Graph-level

### A.1.1 Bookkeeping

Some attributes are included only for "bookkeeping" 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** | |
| **assortativity** | *Degree assortativity* |
| **assortativity.lobe** | Assortativity calculated using `lobe` membership |
| **assortativity.lobe.hemi** | Assortativity calculated using `lobe` and `hemi` membership |
| **assortativity.class** | Assortativity calculated using `class` membership (if the atlas is `destrieux`) |
| **assortativity.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 $\geq 2$) |

### 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 Bookkeeping/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`) |
| **coreness** | *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

# 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.

- `brainGraph_GLM` and its related functions (needs *name* graph attribute that matches the *Study.ID* column in the input *covars* data table)
- `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`
- `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`

# C

# 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.

## C.1 Since v1.0.0

I updated several functions for `v1.0.0` which now run significantly faster. I list the speed increases for several functions and a few different graph sizes and densities in Table C.1 (cortical thickness covariance; 68 vertices), Table C.2 (DTI tractography; 76 vertices), and Table C.3 (resting-state fMRI; 160 vertices). For a given graph and vertex measure, the improvements tend to be less than 1 second, but when repeated for a large number of subjects and densities/thresholds, can amount to a lot of saved time. Furthermore, speed improvements are larger for larger graphs. Finally, these improvements can add up considerably when doing, for example, permutation testing with thousands of repetitions.

All testing was done on the same system:

**System Information**

```
64-bit CentOS 7.3.1611
Intel Core i7-6500U (4 cores @ 2.50 GHz)
R version 3.3.3 (2017-03-06)
```

For each function, I started a new instance of R and calculated runtimes using `microbenchmark`.

The speed improvements are most significant for `within_module_deg_z_score`, `edge_asymmetry`, `part_coeff`, and `centr_lev`, and increase with increasing vertex and edge counts. Random graph selection (controlling for clustering), with `sim.rand.graph.par`, shows the largest improvements in *absolute* time: generating 100 random graphs at a single density is **15 to 52** minutes faster. Since this is often repeated across several densities, the savings can reach several hours.

In the last section of Table C.2 and Table C.3, I show the timings for creating graphs and assigning attributes (via `set_brainGraph_attr`) for a group of 44 and 22 subjects, respectively, at 5 different densities. I used the new function argument `A`, supplying the weighted adjacency matrix (which we already create; see Graph creation). The speed increases range from about 40 to 60 seconds per threshold/density; this should scale linearly with an increase in subject numbers, and possibly supra-linearly with increase in graph size (vertex and edge counts).

| Function | m ($\rho$) | # reps | Old | New | Speedup |
|---|---|---|---|---|---|
| **edge_asymmetry (vertex)** | 114 (5.0%) | 1000 | 0.1049 | 0.0021 | 50.5x |
| | 616 (27.0%) | 1000 | 0.1724 | 0.0026 | 67.0x |
| | 1139 (50.0%) | 1000 | 0.2316 | 0.0032 | 71.6x |
| **edge_asymmetry (hemi)** | 114 (5.0%) | 1000 | 0.0075 | 0.0032 | 2.34x |
| | 616 (27.0%) | 1000 | 0.0114 | 0.0046 | 2.49x |
| | 1139 (50.0%) | 1000 | 0.0176 | 0.0068 | 2.58x |
| **vertex_spatial_dist** | 114 (5.0%) | 1000 | 0.0619 | 0.0193 | 3.21x |
| | 616 (27.0%) | 1000 | 0.1041 | 0.0235 | 4.43x |
| | 1139 (50.0%) | 1000 | 0.1451 | 0.0272 | 5.33x |
| **part_coeff** | 114 (5.0%) | 1000 | 0.2050 | 0.0014 | 142x |
| | 616 (27.0%) | 1000 | 0.1255 | 0.0009 | 137x |
| | 1139 (50.0%) | 1000 | 0.1195 | 0.0008 | 141x |
| **within-module degree z-score** | 114 (5.0%) | 1000 | 0.0452 | 0.0013 | 34x |
| | 616 (27.0%) | 1000 | 0.0550 | 0.0007 | 76x |
| | 1139 (50.0%) | 1000 | 0.0576 | 0.0007 | 87x |
| **centr_lev** | 114 (5.0%) | 1000 | 0.0656 | 0.0009 | 71x |
| | 616 (27.0%) | 1000 | 0.0765 | 0.0012 | 64x |
| | 1139 (50.0%) | 1000 | 0.0788 | 0.0012 | 63x |
| **local efficiency** | 114 (5.0%) | 1000 | 0.0851 | 0.0531 | 1.60x |
| | 616 (27.0%) | 1000 | 0.1255 | 0.0766 | 1.64x |
| | 1139 (50.0%) | 1000 | 0.1379 | 0.0873 | 1.58x |
| **Random graphs (w/ clustering) 100 iter's** | 114 (5.0%) | 100 | 953.609 | 56.977 | 16.74x |
| | 616 (27.0%) | 100 | 531.973 | 60.204 | 8.84x |
| | 1139 (50.0%) | 100 | 974.534 | 93.995 | 10.37x |
| **Random graphs (w/ clustering) 250 iter's** | 114 (5.0%) | 100 | 924.368 | 58.391 | 15.83x |
| | 616 (27.0%) | 100 | 1290.226 | 126.051 | 10.24x |
| | 1139 (50.0%) | 100 | 3359.081 | 253.283 | 13.26x |

Table C.1: **Speed increases for a graph with 68 vertices.** All times are the median (in seconds) based on the number of repetitions.
$m$: # of edges; $\rho$: graph density

### C.1.1 GLM-related benchmarks

Table C.4, Table C.5, and Table C.6 list some runtimes for `brainGraph_GLM` and `mtpc`, respectively. The system is the same as above. For the GLM table, there are 2 groups and 156 subjects total. The graphs have 76 vertices (the `dkt.scgm` atlas). Where only 80 (or 103) subjects are mentioned, the atlas used was `dk.scgm` which has 82 vertices. Finally, Table C.7 lists runtime for `brainGraph_mediate` for a couple *outcome–mediator* combinations, in an analysis with 2 subject groups (71 subjects total) and the `dk.scgm` atlas (82 vertices).

### C.1.2 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).

- The `dk` atlas (**68 vertices**) for cortical thickness covariance:

  **Random graph generation (control for clustering)** For 1 group and 1 density (22%), generating 1,000 random graphs, total processing time was **9 min. 58 sec.**. Compared to a runtime of 5 hr.

17 min. in the older version (which was on a machine with more and faster CPU cores), this is a more than **32x** speed increase.

- 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**.

  **NBS** For 2 groups, with 80 subjects total; 5,000 permutations took from **2 min. 56 sec. – 9 min. 37 sec.** (depending on the overall connectivity density, which in this study ranged from 5.6% – 31.1%).

| Function | m ($\rho$) | # reps | Old | New | Speedup |
|---|---|---|---|---|---|
| **edge_asymmetry (vertex)** | 146 (5.1%) | 1000 | 0.2103 | 0.0028 | 74x |
| | 342 (12.0%) | 1000 | 0.2672 | 0.0035 | 77x |
| | 703 (24.7%) | 1000 | 0.3374 | 0.0042 | 81x |
| **edge_asymmetry (hemi)** | 146 (5.1%) | 1000 | 0.0096 | 0.0041 | 2.34x |
| | 342 (12.0%) | 1000 | 0.0151 | 0.0062 | 2.43x |
| | 703 (24.7%) | 1000 | 0.0246 | 0.0099 | 2.48x |
| **vertex_spatial_dist** | 146 (5.1%) | 1000 | 0.0634 | 0.0276 | 2.3x |
| | 342 (12.0%) | 1000 | 0.0731 | 0.0303 | 2.4x |
| | 703 (24.7%) | 1000 | 0.0847 | 0.0339 | 2.5x |
| **part_coeff** | 146 (5.1%) | 1000 | 0.1671 | 0.0011 | 151x |
| | 342 (12.0%) | 1000 | 0.1350 | 0.0010 | 134x |
| | 703 (24.7%) | 1000 | 0.1327 | 0.0010 | 134x |
| **within-module degree z-score** | 146 (5.1%) | 1000 | 0.0893 | 0.0009 | 96x |
| | 342 (12.0%) | 1000 | 0.0935 | 0.0008 | 117x |
| | 703 (24.7%) | 1000 | 0.0974 | 0.0007 | 131x |
| **centr_lev** | 146 (5.1%) | 1000 | 0.1201 | 0.0014 | 87x |
| | 342 (12.0%) | 1000 | 0.1220 | 0.0014 | 86x |
| | 703 (24.7%) | 1000 | 0.1228 | 0.0015 | 82x |
| **local efficiency (weighted)** | 146 (5.1%) | 1000 | 0.1717 | 0.1330 | 1.29x |
| | 342 (12.0%) | 1000 | 0.1813 | 0.1374 | 1.32x |
| | 703 (24.7%) | 1000 | 0.1912 | 0.1466 | 1.30x |
| **local efficiency (unweighted)** | 146 (5.1%) | 1000 | 0.1612 | 0.1275 | 1.26x |
| | 342 (12.0%) | 1000 | 0.1688 | 0.1311 | 1.29x |
| | 703 (24.7%) | 1000 | 0.1716 | 0.1365 | 1.26x |
| **set_brainGraph_attr (44 subjects)** | 146 (5.1%) | 1 | 61.617 | 25.148 | 2.45x |
| | 240 (8.4%) | 1 | 61.317 | 26.197 | 2.34x |
| | 342 (12.0%) | 1 | 62.001 | 26.825 | 2.31x |
| | 516 (18.1%) | 1 | 65.889 | 30.571 | 2.16x |
| | 703 (24.7%) | 1 | 67.703 | 32.979 | 2.05x |

Table C.2: **Speed increases for a graph with 76 vertices.** All times are the median (in seconds) based on the number of repetitions.
$m$: # of edges; $\rho$: graph density

| Function | m ($\rho$) | # reps | Old | New | Speedup |
|---|---|---|---|---|---|
| **edge_asymmetry (vertex)** | 549 (4.3%) | 1000 | 0.3362 | 0.0025 | 132x |
| | 1623 (12.8%) | 1000 | 0.4526 | 0.0025 | 178x |
| | 3955 (31.1%) | 1000 | 0.7378 | 0.0031 | 236x |
| **edge_asymmetry (hemi)** | 549 (4.3%) | 1000 | 0.0085 | 0.0034 | 2.47x |
| | 1623 (12.8%) | 1000 | 0.0125 | 0.0051 | 2.44x |
| | 3955 (31.1%) | 1000 | 0.0207 | 0.0085 | 2.43x |
| **vertex_spatial_dist** | 549 (4.3%) | 1000 | 0.2073 | 0.0408 | 5.1x |
| | 1623 (12.8%) | 1000 | 0.3445 | 0.0468 | 7.4x |
| | 3955 (31.1%) | 1000 | 0.6145 | 0.0553 | 11.0x |
| **part_coeff** | 549 (4.3%) | 1000 | 0.4561 | 0.0022 | 211x |
| | 1623 (12.8%) | 1000 | 0.3961 | 0.0015 | 264x |
| | 3955 (31.1%) | 1000 | 0.4653 | 0.0014 | 343x |
| **within-module degree z-score** | 549 (4.3%) | 1000 | 0.1229 | 0.0013 | 96x |
| | 1623 (12.8%) | 1000 | 0.1461 | 0.0010 | 135x |
| | 3955 (31.1%) | 1000 | 0.1884 | 0.0010 | 178x |
| **centr_lev** | 549 (4.3%) | 1000 | 0.1801 | 0.0029 | 62x |
| | 1623 (12.8%) | 1000 | 0.1892 | 0.0032 | 60x |
| | 3955 (31.1%) | 1000 | 0.1857 | 0.0033 | 56x |
| **local efficiency (weighted)** | 549 (4.3%) | 1000 | 0.3132 | 0.1824 | 1.37x |
| | 1623 (12.8%) | 1000 | 0.3592 | 0.2114 | 1.70x |
| | 3955 (31.1%) | 1000 | 0.4999 | 0.3151 | 1.59x |
| **local efficiency (unweighted)** | 549 (4.3%) | 1000 | 0.2773 | 0.1571 | 1.77x |
| | 1623 (12.8%) | 1000 | 0.3357 | 0.1949 | 1.72x |
| | 3955 (31.1%) | 1000 | 0.3353 | 0.2142 | 1.57x |
| **set_brainGraph_attr (22 subjects)** | 549 (4.3%) | 1 | 62.012 | 23.034 | 2.69x |
| | 963 (7.6%) | 1 | 75.234 | 25.453 | 2.96x |
| | 1623 (12.8%) | 1 | 72.126 | 28.202 | 2.56x |
| | 2583 (20.3%) | 1 | 82.864 | 31.963 | 2.59x |
| | 3955 (31.1%) | 1 | 97.681 | 39.048 | 2.50x |

Table C.3: **Speed increases for a graph with 160 vertices.** All times are the median (in seconds) based on the number of repetitions.
$m$: # of edges; $\rho$: graph density

| Level | # reps | Runtime (s) |
|---|---|---|
| **graph** | 10,000 | 8.703 |
| | 10,000 | 11.219 |
| | 10,000 | 11.479 |
| **vertex** | 5,000 | 35.235 |
| | 5,000 | 35.098 |
| | 5,000 | 34.759 |

Table C.4: **Runtimes for `brainGraph_GLM` for graphs with 76 vertices.** These analyses were for 156 subjects in 2 groups.

| Level | N | Runtime (s) | |
|---|---|---|---|
| | | 80 sub. | 156 sub. |
| **Graph** | **5,000** | 73.618 | 70.631 |
| | | 73.373 | 88.962 |
| | | 74.366 | 93.684 |
| **Vertex** | **1,000** | 86.427 | 89.560 |
| | | 87.291 | 90.917 |
| | **2,000** | 178.248 | 202.044 |
| | | 179.312 | 205.846 |
| | **5,000** | 448.524 | 502.414 |
| | | 440.946 | 507.429 |

Table C.5: **Runtimes for `mtpc` for graphs with 76 or 82 vertices and 15 thresholds.**
*N*: # of permutations

| Level | Runtime (s) | | |
|---|---|---|---|
| | 1 con. | 2 con. | 3 con. |
| **Graph** | 153.623 | 281.572 | 457.312 |
| | 152.193 | 281.152 | 435.029 |
| | 154.747 | 285.569 | 437.076 |
| | 159.749 | 280.483 | 456.382 |
| | 157.722 | 283.426 | 428.564 |
| **Vertex** | 711.027 | 757.598 | 1947.120 |
| | 688.052 | 760.485 | 1971.640 |
| | 704.171 | 784.705 | 1978.283 |
| | 683.840 | 785.550 | |
| | 714.713 | 792.197 | |

Table C.6: **Runtimes for `mtpc` for graphs with 82 vertices and 30 thresholds.** There were 10,000 and 5,000 permutations for graph- and vertex-level analyses, respectively. The analyses for 1 and 3 contrasts contained data for 103 subjects; for 2 contrasts there were only 44, indicating an effect of # of subjects in addition to # of contrasts, # of thresholds, and # of permutations..
*con*: contrasts

| # outcomes, mediators | Runtime (s) | |
|---|---|---|
| | *N=1,000* | *N=5,000* |
| **1, 1** | 32.503 | 123.318 |
| | 33.458 | 129.577 |
| | 32.694 | 127.223 |
| | *N=100* | *N=1,000* |
| **3, 6** | 222.840 | 669.151 |
| | 223.920 | 649.174 |
| | 224.796 | 647.234 |

Table C.7: **Runtimes for vertex-level `mediation` for graphs with 82 vertices.** These analyses were for 71 subjects in 2 groups. In the case that the # of outcomes and # of mediators is more than 1, the runtime is for the entire loop across the combination.
*N*: # of bootstrap samples

# D

# 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 3.4.3 (2017-11-30), `x86_64-redhat-linux-gnu`
- Running under: `CentOS Linux 7 (Core)`
- Matrix products: default
- BLAS/LAPACK: `/usr/lib64/R/lib/libRblas.so`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, utils
- Other packages: brainGraph 2.4.1, colorout 1.1-2, data.table 1.11.4, doMC 1.3.5, foreach 1.4.4, ggplot2 3.0.0.9000, gridExtra 2.3, igraph 1.2.2, iterators 1.0.9, knitr 1.20, nvimcom 0.9-62, pacman 0.4.6, permute 0.9-4, plyr 1.8.4, setwidth 1.0-4
- Loaded via a namespace (and not attached): abind 1.4-5, acepack 1.4.1, ade4 1.7-11, assertthat 0.2.0, backports 1.1.2, base64enc 0.1-3, bindr 0.1, bindrcpp 0.2, bitops 1.0-6, boot 1.3-20, checkmate 1.8.2, cluster 2.0.6, codetools 0.2-15, colorspace 1.3-2, compiler 3.4.3, curl 3.2, devtools 1.13.5, digest 0.6.15, dplyr 0.7.4, evaluate 0.10, expm 0.999-2, foreign 0.8-69, Formula 1.2-1, ggrepel 0.8.0, git2r 0.21.0, glue 1.2.0.9000, grid 3.4.3, gtable 0.2.0, highr 0.6, Hmisc 4.1-1, htmlTable 1.12, htmltools 0.3.6, htmlwidgets 0.9, httr 1.3.1, labeling 0.3, lattice 0.20-35, latticeExtra 0.6-28, lazyeval 0.2.1, lme4 1.1-17, lpSolve 5.6.13, magrittr 1.5, MASS 7.3-50, Matrix 1.2-14, mediation 4.4.6, memoise 1.1.0, minqa 1.2.4, munsell 0.5.0, mvtnorm 1.0-6, nlme 3.1-131, nloptr 1.0.4, nnet 7.3-12, oro.nifti 0.9.1, pillar 1.2.2, pkgconfig 2.0.2, R6 2.2.2, RColorBrewer 1.1-2, Rcpp 0.12.18, RcppEigen 0.3.3.4.0, rlang 0.2.1.9000, RNifti 0.7.1, rpart 4.1-11, rstudioapi 0.7, sandwich 2.4-0, scales 1.0.0.9000, splines 3.4.3, stringi 1.2.2, stringr 1.3.1, survival 2.41-3, tibble 1.4.2, tools 3.4.3, withr 2.1.2, zoo 1.8-0

# Bibliography


[1] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.

[2] Shweta Bansal, Shashank Khandelwal, and Lauren A Meyers. Exploring biological network structure with clustered random networks. *BMC Bioinformatics*, 10(1):405, 2009.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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. *The Journal of Neuroscience*, 33(26):10676–10687, 2013.

[12] 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.

[13] Vittoria Colizza, Alessandro Flammini, M Angeles Serrano, and Alessandro Vespignani. Detecting rich-club ordering in complex networks. *Nature Physics*, 2(2):110–115, 2006.

[14] Robert W Cox. AFNI: software for analysis and visualization of functional magnetic resonance neuroimages. *Computers and Biomedical Research*, 29(3):162–173, 1996.

[15] 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.

[16] 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.

[17] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[18] 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.

[19] Anthony Christopher Davison and David Victor Hinkley. *Bootstrap methods and their application*, volume 1. Cambridge University Press, 1997.

[20] Marcel A de Reus and Martijn P van den Heuvel. Estimating false positives and negatives in brain networks. *NeuroImage*, 70:402–409, 2013.

[21] RS Desikan, F Segonne, B Fischl, BT Quinn, BC Dickerson, D Blacker, RL Buckner, AM Dale, RP Maguire, BT Hyman, MS Albert, and RJ 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] Ernesto Estrada and Naomichi Hatano. Communicability in complex networks. *Physical Review E*, 77 (3):036111, 2008.

[28] 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.

[29] 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.

[30] 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.

[31] David Freedman and David Lane. A nonstochastic interpretation of reported significance levels. *Journal of Business & Economic Statistics*, 1(4):292–298, 1983.

[32] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Softw., Pract. Exper.*, 21(11):1129–1164, 1991.

[33] 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. *The Journal of Neuroscience*, 29(50): 15684–15693, 2009.

[34] 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.

[35] Roger Guimera and Luis A Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, 2005.

[36] Yong He, Zhang Chen, and Alan Evans. Structural insights into aberrant topological patterns of large-scale cortical networks in Alzheimer's disease. *The Journal of Neuroscience*, 28(18):4756–4766, 2008.

[37] Paul W Holland. Statistics and causal inference. *Journal of the American Statistical Association*, 81 (396):945–960, 1986.

[38] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical Review E*, 65(5):056109, 2002.

[39] 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.

[40] 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.

[41] Mark D Humphries and Kevin Gurney. Network 'small-world-ness': a quantitative method for determining canonical network equivalence. *PLoS One*, 3(4):e0002051, 2008.

[42] Kosuke Imai, Luke Keele, and Dustin Tingley. A general approach to causal mediation analysis. *Psychological Methods*, 15(4):309, 2010.

[43] Kosuke Imai, Luke Keele, and Teppei Yamamoto. Identification, inference and sensitivity analysis for causal mediation effects. *Statistical Science*, 25(1):51–71, 2010.

[44] 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.

[45] 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.

[46] Saad Jbabdi, MW Woolrich, JLR Andersson, and TEJ Behrens. A bayesian framework for global tractography. *NeuroImage*, 37(1):116–129, 2007.

[47] Mark Jenkinson, Christian F Beckmann, Timothy EJ Behrens, Mark W Woolrich, and Stephen M Smith. FSL. *NeuroImage*, 62(2):782–790, 2012.

[48] 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.

[49] David Kaufman and Robert Sweet. Contrast coding in least squares regression analysis. *American Educational Research Journal*, 11(4):359–377, 1974.

[50] Luke Keele. Causal mediation analysis: warning! Assumptions ahead. *American Journal of Evaluation*, 36(4):500–513, 2015.

[51] Arno Klein and Jason Tourville. 101 labeled brain images and a consistent human cortical labeling protocol. *Frontiers in Neuroscience*, 6, 2012.

[52] ED Kolaczyk. *Statistical Analysis of Network Data, volume 69 of Springer Series in Statistics*. Springer New York, 2009.

[53] Eric D Kolaczyk and Gábor Csárdi. *Statistical analysis of network data with R*, volume 65. Springer, 2014.

[54] 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*, 2017.

[55] Michael H Kutner, Chris Nachtsheim, John Neter, and William Li. *Applied linear statistical models*. McGraw-Hill Irwin, 2005.

[56] Athen Ma and Raúl J Mondragón. Rich-cores in networks. *PloS One*, 10(3):e0119678, 2015.

[57] 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.

[58] 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.

[59] 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.

[60] Mark EJ Newman. *Networks: an introduction*. Oxford University Press, Oxford, 2010.

[61] 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.

[62] William Stafford Noble. A quick guide to organizing computational biology projects. *PLoS Computational Biology*, 5(7):e1000424, 2009.

[63] John E Overall and Douglas K Spiegel. Concerning least squares analysis of experimental data. *Psychological Bulletin*, 72(5):311, 1969.

[64] 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.

[65] 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.

[66] Kristopher J Preacher. Advances in mediation analysis: a survey and synthesis of new developments. *Annual Review of Psychology*, 66:825–852, 2015.

[67] 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.

[68] 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.

[69] 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.

[70] James M Robins and Sander Greenland. Identifiability and exchangeability for direct and indirect effects. *Epidemiology*, pages 143–155, 1992.

[71] 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.

[72] Donald B Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5):688, 1974.

[73] Donald B Rubin. Bayesian inference for causal effects: the role of randomization. *The Annals of Statistics*, pages 34–58, 1978.

[74] 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.

[75] 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.

[76] 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.

[77] 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.

[78] David W Shattuck and Richard M Leahy. Brainsuite: an automated cortical surface identification tool. *Medical Image Analysis*, 6(2):129–142, 2002.

[79] 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.

[80] 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.

[81] Stephen Smith, Mark Jenkinson, Christian Beckmann, Karla Miller, and Mark Woolrich. Meaningful design and contrast estimability in FMRI. *NeuroImage*, 34(1):127–136, 2007.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. ISBN 0-387-95457-0.

[89] Fabien Viger and Matthieu Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. *Journal of Complex Networks*, page cnv013, 2015.

[90] 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.

[91] 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.

[92] 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–n/a, 2018. ISSN 2162-3279. doi:10.1002/brb3.834. URL http://dx.doi.org/10.1002/brb3.834. e00834.

[93] Duncan J Watts and Steven H Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393 (6684):440–442, 1998.

[94] Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10), 2014.

[95] Ben James Winer, Donald R Brown, and Kenneth M Michels. *Statistical principles in experimental design*, volume 2. McGraw-Hill New York, 1971.

[96] 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.

[97] Mingrui Xia, Jinhui Wang, Yong He, et al. Brainnet viewer: a network visualization tool for human brain connectomics. *PloS One*, 8(7):e68910, 2013.

[98] Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.name/knitr/. ISBN 978-1498716963.

[99] 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.

[100] Andrew Zalesky, Alex Fornito, and Edward T Bullmore. Network-based statistic: identifying differences in brain networks. *NeuroImage*, 53(4):1197–1207, 2010.

[101] Andrew Zalesky, Alex Fornito, and Ed Bullmore. On the use of correlation as a measure of network connectivity. *NeuroImage*, 60(4):2096–2106, 2012.

[102] Shi Zhou and Raúl J Mondragón. The rich-club phenomenon in the internet topology. *Communications Letters, IEEE*, 8(3):180–182, 2004.