Package ‘brainGraph’

April 4, 2024

Type Package
Version 3.1.0
Date 2024-04-03
Title Graph Theory Analysis of Brain MRI Data
Description A set of tools for performing graph theory analysis of brain MRI data. It works with data from a Freesurfer analysis (cortical thickness, volumes, local gyrification index, surface area), diffusion tensor tractography data (e.g., from FSL) and resting-state fMRI data (e.g., from DPABI). It contains a graphical user interface for graph visualization and data exploration, along with several functions for generating useful figures.

URL https://github.com/cwatson/brainGraph
BugReports https://groups.google.com/forum/?hl=en#!forum/brainGraph-help
LazyData true
Depends R (>= 3.5.0), igraph (>= 1.2.4),
Imports abind, data.table (>= 1.12.4), doParallel,foreach,grid,

      lattice, MASS, Matrix, methods, permute, parallel
Suggests Hmisc, ade4, boot, car, expm, ggplot2, ggrepel, gridExtra,

      mediation, oro.nifti, scales
License GPL-3
RoxygenNote 6.1.1
Collate 'glm_stats.R' 'brainGraph,GLM,R' 'glm_methods.R' 'NBS.R'

      'analysis_random_graphs.R' 'atlas.R' 'auc.R' 'boot_global.R'

      'brainGraph_mediate.R' 'centr_leb.R' 'communicability.R'

      'contract_brainGraph.R' 'corr_matrix.R' 'count_edges.R'

      'create_graphs.R' 'create_mats.R' 'data.R' 'data_tables.R'
      'distances.R' 'edge_asymmetry.R' 'get_resid.R' 'glm_design.R'

      'glm_fit.R' 'glm_randomise.R' 'graph_efficiency.R' 'hubs.R'

      'import.R' 'individ_contrib.R' 'list.R' 'method_helpers.R'

      'mtpc.R' 'methods.R' 'permute_group.R' 'plot_brainGraph.R'

      'plot_brainGraph_multi.R' 'plot_global.R' 'plot_group_means.R'
R topics documented:

'plot_rich_norm.R' 'plot_vertex_measures.R' 'random_graphs.R'
'rich_club.R' 'robustness.R' 's_core.R'
'set_brainGraph_attributes.R' 'small_world.R' 'spatial_dist.R'
'utils.R' 'utils_matrix.R' 'vertex_roles.R' 'vulnerability.R'
'write_brainnet.R' 'zzz.R'

NeedsCompilation no

Author  Christopher G. Watson [aut, cre]
        (<https://orcid.org/0000-0002-7082-7631>)

Maintainer  Christopher G. Watson <cgwatson@bu.edu>

Repository CRAN

Date/Publication  2024-04-04 05:03:07 UTC

R topics documented:

apply_thresholds ............................................. 3
Atlas Helpers ................................................. 4
Attributes ..................................................... 6
Bootstrapping .................................................. 8
Brain Atlases .................................................. 10
brainGraph ..................................................... 13
brainGraph-methods .......................................... 13
brainGraphList ................................................ 14
brainGraph_permute .......................................... 18
centr_betw_comm ............................................. 20
centr_lev ...................................................... 21
check_sID ..................................................... 22
coeff_var ...................................................... 23
communicability .............................................. 23
contract_brainGraph ......................................... 24
cor.diff.test .................................................. 25
corr.matrix .................................................... 26
Count Edges .................................................... 29
create_mats .................................................... 30
Creating_Graphs .............................................. 32
Creating_Graphs_GLM ......................................... 35
edge_asymmetry ............................................... 37
efficiency ...................................................... 38
GLM .......................................................... 39
GLM basic info ............................................... 44
GLM design ..................................................... 45
GLM fits ......................................................... 47
GLM influence measures ...................................... 50
GLM model selection .......................................... 52
GLM statistics ................................................ 53
Graph Data Tables ............................................ 55
Graph Distances ............................................... 56
**apply_thresholds**

Threshold additional set of matrices

---

**Description**

apply_thresholds thresholds an additional set of matrices (e.g., FA-weighted matrices for DTI tractography) based on the matrices that have been returned from `create_mats`. This ensures that the same connections are present in both sets of matrices.

**Usage**

```r
apply_thresholds(sub.mats, group.mats, W.files, inds)
```
## Atlas Helpers

### Description

**guess_atlas** tries to determine which atlas is being used based on the data; i.e., the number of vertices/regions.

**as_atlas** and **create_atlas** converts/coerces an object to a data.table, or creates one, that is compatible with brainGraph.

### Arguments

- **sub.mats**: List (length equal to number of thresholds) of numeric arrays (3-dim) for all subjects
- **group.mats**: List (length equal to number of thresholds) of numeric arrays (3-dim) for group-level data
- **W.files**: Character vector of the filenames of the files with connectivity matrices
- **inds**: List (length equal to number of groups) of integers; each list element should be a vector of length equal to the group sizes

### Details

The argument `W.files` accepts the same formats as `A.files`; see **create_mats** for details.

### Value

List containing:

- **W**: A 3-d array of the raw connection matrices
- **W.norm.sub**: List of 3-d arrays of the normalized connection matrices for all given thresholds
- **W.norm.mean**: List of 3-d arrays of the normalized connection matrices averaged for each group

### Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

### Examples

```r
## Not run:
W.mats <- apply_thresholds(A.norm.sub, A.norm.mean, f.W, inds)
## End(Not run)
```

---

**Atlas Helpers** | **Atlas helper functions**

---

### Description

**guess_atlas** tries to determine which atlas is being used based on the data; i.e., the number of vertices/regions.

**as_atlas** and **create_atlas** converts/coerces an object to a data.table, or creates one, that is compatible with brainGraph.
Usage

\texttt{guess_atlas(x)}

\texttt{as_atlas(object)}

\texttt{create_atlas(regions, coords, lobes, hemis, other = NULL)}

Arguments

- **x, object**: An object to test or convert to an atlas data.table
- **regions**: Character vector of region names
- **coords**: Numeric matrix of spatial coordinates; must have 3 columns
- **lobes**: Character or factor vector of lobe membership
- **hemis**: Character or factor vector of hemisphere membership. There should probably not be more than 3 unique elements (for left, right, and bi-hemispheric regions)
- **other**: A named list of vectors with other data. The names of the list will become column names in the return object.

Value

- **guess_atlas**: Character string; either the matched atlas or NA
- **as_atlas** and **create_atlas** return a data.table that conforms to other atlases in the package, or exits with an error.

Guessing the atlas from an object

There are several valid inputs to **guess_atlas**:

- **data.table**: The atlas will be guessed based on the number of columns (subtracting by 1 if a “Study ID” column is present). This is the same behavior as for data.frame objects, as well.
- **igraph**: The vertex count
- **brainGraph**: If there is a atlas graph-level attribute, it will return that. Otherwise, the vertex count.
- **matrix, array**: The number of rows, which should equal the number of columns if the input is a connectivity matrix.

Note that this will only work properly for atlases that are currently in the package. If you are using a custom atlas and you receive errors, please open an issue on GitHub.

Coercing to an atlas

There are several things **as_atlas** tries to do to make it work without error:

- Coerce the object to data.table
- Add a column of integers named index
- Change columns named ‘x’, ‘y’, or ‘z’ to have .mni at the end
- Convert the lobe and hemi columns to be factors
Attributes

Examples

```r
my_atlas <- data.frame(name=paste('Region', 1:10), x.mni=rnorm(10),
                      y.mni=rnorm(10), z.mni=rnorm(10),
                      lobe=rep(c('Frontal', 'Parietal', 'Temporal', 'Occipital', 'Limbic'), 2),
                      hemi=c(rep('L', 5), rep('R', 5)))
my_atlas2 <- as_atlas(my_atlas)
str(my_atlas)
str(my_atlas2)
regions <- paste('Region', 1:10)
xyz <- matrix(rnorm(30), nrow=10, ncol=3)
lobe <- rep(c('Frontal', 'Parietal', 'Temporal', 'Occipital', 'Limbic'), 2)
other <- list(network=rep(c('Default mode', 'Task positive'), 5))
my_atlas <- create_atlas(regions, xyz, lobe, hemi, other)
str(my_atlas)
```

Description

`set_brainGraph_attr` is a convenience function that sets a number of graph, vertex, and edge attributes for a given graph object. Specifically, it calculates measures that are common in MRI analyses of brain networks.

Usage

```r
set_brainGraph_attr(g, type = c("observed", "random"),
                    use.parallel = TRUE, A = NULL, xfm.type = c("\1/w", \"-log(w)\",
                                                         \"1-w\", \"-log10(w/max(w))\", \"-log10(w/max(w)+1)\"),
                    clust.method = \"louvain\")
xfm.weights(g, xfm.type = c("\1/w", \"-log(w)\", \"1-w\", \"-log10(w/max(w))\",
                                  \"-log10(w/max(w)+1)\"), invert = FALSE)
```

Arguments

- **g**: A graph object
- **type**: Character string indicating the type of graphs. Default: observed
- **use.parallel**: Logical indicating whether to use `foreach`. Default: TRUE
- **A**: Numeric matrix; the (weighted) adjacency matrix, which can be used for faster calculation of local efficiency. Default: NULL
- **xfm.type**: Character string specifying how to transform the weights. Default: \1/w
- **clust.method**: Character string indicating which method to use for community detection. Default: 'louvain'
- **invert**: Logical indicating whether or not to invert the transformation. Default: FALSE
Attributes

Details

Including type='random' in the function call will reduce the number of attributes calculated. It will only add graph-level attributes for: clustering coefficient, characteristic path length, rich club coefficient, global efficiency, and modularity.

Value

A graph object with the following attributes:

Graph-level Density, connected component sizes, diameter, # of triangles, transitivity, average path length, assortativity, global & local efficiency, modularity, vulnerability, hub score, rich-club coefficient, # of hubs, edge asymmetry

Vertex-level Degree, strength; betweenness, eigenvector, and leverage centralities; hubs; transitivity (local); k-core, s-core; local & nodal efficiency; color (community, lobe, component); membership (community, lobe, component); gateway and participation coefficients, within-module degree z-score; vulnerability; and coordinates (x, y, and z)

Edge-level Color (community, lobe, component), edge betweenness, Euclidean distance (in mm), weight (if weighted)

xfm.weights returns the same graph object, with transformed edge weights plus a graph attribute (xfm.type) recording the method of transformation

Negative edge weights

If there are any negative edge weights in the graph, several of the distance-based metrics will not be calculated, because they can throw errors which is undesirable when processing a large dataset. The metrics are: local and nodal efficiency, diameter, characteristic path length, and hubness.

Transforming edge weights

For distance-based measures, it is important to transform the edge weights so that the strongest connections are re-mapped to having the lowest weights. Then you may calculate e.g., the shortest path length which will include the strongest connections.

xfm.type allows you to choose from 5 options for transforming edge weights when calculating distance-based metrics (e.g., shortest paths). There is no “best-practice” for choosing one over the other, but the reciprocal is probably most common.

1/w reciprocal (default)

-\log(w) the negative (natural) logarithm

1-w subtract weights from 1

-\log10(w/\max(w)) negative (base-10) log of normalized weights

-\log10(w/\max(w)+1) same as above, but add 1 before taking the log

To transform the weights back to original values, specify invert=TRUE.
Community detection

clust.method allows you to choose from any of the clustering (community detection) functions available in igraph. These functions begin with cluster:: the function argument should not include this leading character string. There are a few possibilities, depending on the value and the type of input graph:

1. By default, louvain is used, calling `cluster_louvain`
2. Uses spinglass if there are any negative edges and/or the selected method is spinglass
3. Uses walktrap if there are any negative edge weights and any other method (besides spinglass) is selected
4. Automatically transforms the edge weights if `edge_betweenness` is selected and the graph is weighted, because the algorithm considers edges as distances

Author(s)

Christopher G. Watson, <cgwaston@bu.edu>

See Also

components, diameter, centr_betw, betweenness, centr_eigen, transitivity, distances, assortativity, coreness, communities, knn

---

**Bootstrapping**

**Bootstrapping for global graph measures**

**Description**

Perform bootstrapping to obtain groupwise standard error estimates of a global graph measure.

The `plot` method returns two ggplot objects: one with shaded regions based on the standard error, and the other based on confidence intervals (calculated using the normal approximation).

**Usage**

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

```r
## S3 method for class 'brainGraph_boot'
summary(object, ...)
```

```r
## S3 method for class 'brainGraph_boot'
plot(x, ..., alpha = 0.4)
```
Arguments

densities Numeric vector of graph densities to loop through
resids An object of class brainGraph_resids (the output from get.resid)
R Integer; the number of bootstrap replicates. Default: 1e3
measure Character string of the measure to test. Default: mod
cnf Numeric; the level for calculating confidence intervals. Default: 0.95
.progress Logical indicating whether or not to show a progress bar. Default: getOption('bg.progress')
.xfm.type Character string specifying how to transform the weights. Default: 1/w
.object, x A brainGraph_boot object
... Unused
.alpha A numeric indicating the opacity for the confidence bands

Details

The confidence intervals are calculated using the *normal approximation* at the 100 \( \times \) conf\% level (by default, 95\%).

For getting estimates of *weighted global efficiency*, a method for transforming edge weights must be provided. The default is to invert them. See xfm.weights.

Value

brainGraph_boot – an object of class brainGraph_boot containing some input variables, in addition to a list of boot objects (one for each group).
plot – list with the following elements:
- se A ggplot object with ribbon representing standard error
- ci A ggplot object with ribbon representing confidence intervals

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

boot, boot.ci

Other Group analysis functions: GLM, Mediation, NBS, brainGraph_permute, mtpc

Other Structural covariance network functions: IndividualContributions, Residuals, brainGraph_permute, corr.matrix, import_scn, plot_volumetric

Examples

```r
## Not run:
boot.E.global <- brainGraph_boot(densities, resids.all, 1e3, 'E.global')
## End(Not run)
```
Brain Atlases

Description

Datasets containing spatial coordinates for: the original AAL atlases, the newer AAL2 atlases, Freesurfer atlases, Brainsuite, Craddock200, Dosenbach160, Harvard-Oxford, and LONI probabilistic brain atlas. In addition to coordinates, there are indices for the major lobes and hemispheres of the brain, the class variable (for Destrieux atlases), functional networks (for Dosenbach, Power, and Gordon atlases; plus the Yeo network labels for the Brainnetome atlas).

Usage

- aal116
- aal90
- aal2.120
- aal2.94
- destrieux
- destrieux.scgm
- dk
- dk.scgm
- dkt
- dkt.scgm
- brainsuite
- craddock200
- dosenbach160
- hoa112
- lpba40
- hcp_mmp1.0
- power264
Brain Atlases

brainnetome
gordon333

Format

A data frame with 90 or 116 (for the original AAL atlases), 94 or 120 (for the newer AAL2 atlases), 148 or 162 (for Destrieux), 68 or 82 (for DK), 62 or 76 (for DKT), 74 (Brainsuite), 200 (Craddock), 160 (Dosenbach), 112 (Harvard-Oxford), 40 (LONI), 246 (Brainnetome), 360 (HCP), 264 (Power), or 333 (Gordon) observations on (some of) the following 19 variables:

name a character vector of region names
x.mni a numeric vector of x-coordinates (in MNI space)
y.mni a numeric vector of y-coordinates (in MNI space)
z.mni a numeric vector of z-coordinates (in MNI space)
lobe a factor with some of levels Frontal Parietal Temporal Occipital Insula Limbic Cingulate SCGM Cerebellum (for aal116 and aal2.120) and Brainstem (for craddock200)
hemi a factor with levels L R and B (for dosenbach160)
index a numeric vector
name.full a character vector of full region names, for the DK and DKT atlases
class a factor with levels G G_and_S S, for the Destrieux atlases
network (dosenbach160) a factor with levels default fronto-parietal cingulo-opercular sensorimotor cerebellum occipital
 gyrus (brainnetome) Abbreviated names of gyri/regions (including subcortical), with 24 unique values
gyrus.full (brainnetome) Full names of gyrus
subregion (brainnetome) Abbreviated names of subregions (including subdivisions of subcortical gray matter)
subregion.full (brainnetome) Full names of subregion
Yeo_7network (brainnetome) Factor with 8 levels consisting of SCGM plus the 7 networks from Yeo et al.
Yeo_17network (brainnetome) Factor with 18 levels consisting of SCGM plus the 17 networks from Yeo et al.
area (HCP) a factor with 23 cortical areas
Anatomy (power264) Full region/gyrus names for the Power atlas; contains 53 unique regions
Brodmann (power264) Integer values for Brodmann areas

Note

Use of the HCP parcellation is subject to the terms at https://balsa.wustl.edu/WN56. In particular: "I will acknowledge the use of WU-Minn HCP data and data derived from WU-Minn HCP data when publicly presenting any results or algorithms that benefitted from their use."

Region names in the gordon333 atlas were chosen to match those of the hcp_mmp1.0 atlas. Many were determined from the coordinates (using FSL's atlasquery), while the rest were entered manually by me. The lobe values were matched to the HCP atlas, as well.


**Brain Atlases**

**Source**


**References**


brainGraph

Default options for brainGraph

Description

brainGraph is a package for performing graph theory analysis of brain MRI data.

Package options

brainGraph uses the following options to configure behavior:

- bg.subject_id: character string specifying the name your project/study uses as a subject identifier. All imported data (e.g., covariates tables) MUST have a column matching this. One possible alternative is 'participant_id', recommended by BIDS. Default: 'Study.ID'
- bg.group: character string specifying the name your project/study uses as a group identifier. All imported data (e.g., covariates tables) MUST have a column matching this. One possible alternative is 'group', recommended by BIDS. Default: 'Group'
- bg.session: character string specifying the name your project/study uses as a “time” or session identifier, in the case of longitudinal studies. All imported data (e.g., covariates tables) MUST have a column matching this. One possible alternative is 'session_id', recommended by BIDS. Default: 'Time'
- bg.progress: logical indicating whether to show progress bars for functions that provide the option. Default: TRUE
- bg.ncpus: integer indicating the number of cores to use for parallel operations. Only used if you have not already registered a parallel backend (see Chapter 5 of the User Guide or https://github.com/cwatson/brainGraph/blob/master/README.md for examples). Default: 2L

brainGraph-methods

brainGraph generic methods

Description

These functions are S3 generics for various brainGraph-defined objects.

groups returns the “Group” graph attribute for each graph or observation in the object.

region.names is a generic method for extracting region names from various brainGraph objects. These are generally convenience functions.

nregions is a generic method for extracting the number of regions from various brainGraph objects.
Usage

```r
## S3 method for class 'brainGraphList'
groups(x)
```

```r
## S3 method for class 'corr_mats'
groups(x)
```

```r
region.names(object)
```

```r
## S3 method for class 'data.table'
region.names(object)
```

```r
nregions(object)
```

Arguments

- **x** object: An object

Details

For a `data.table`, `region.names` assumes that it contains a *factor* column named `region`.

brainGraphList

Create a list of `brainGraph` graphs

Description

`make_brainGraphList` creates a `brainGraphList` object, a list containing a set of graphs for all subjects (or group-average graphs) in a study at a specific threshold (or density), in addition to some graph-level attributes common to those graphs.

The `[` method will let you subset/slice the graphs for individual subjects and/or groups.

`as_brainGraphList` coerces a list of graphs to a `brainGraphList` object. It is assumed that certain metadata attributes – threshold, package version, atlas, imaging modality, edge weighting, and whether they are random graphs – are identical for all graphs in the list.

Usage

```r
make_brainGraphList(x, atlas, type = c("observed", "random"),
  level = c("subject", "group", "contrast"), set.attrs = TRUE,
  modality = NULL, weighting = NULL, threshold = NULL,
  gnames = NULL, ...)
```

```r
## S3 method for class 'array'
make_brainGraphList(x, atlas, type = c("observed", "random"),
  level = c("subject", "group", "contrast"),
  set.attrs = TRUE, modality = NULL, weighting = NULL,
  ...)
```

```r
## S3 method for class 'brainGraphList'
make_brainGraphList(x, atlas, type = c("observed", "random"),
  level = c("subject", "group", "contrast"),
  set.attrs = TRUE, modality = NULL, weighting = NULL,
  ...)
```
Arguments

- **x**: 3-D numeric array of all subjects’ connectivity matrices (for a single threshold) or a corr_mats object
- **atlas**: Character string specifying the brain atlas
- **type**: Character string indicating the type of graphs. Default: observed
- **level**: Character string indicating whether the graphs are subject-, group-, or contrast-specific. Default: 'subject'
- **set.attrs**: Logical indicating whether to assign all graph-, vertex-, and edge-level attributes (via set_brainGraph_attr). Default: TRUE
- **modality**: Character string indicating imaging modality (e.g., 'dti'). Default: NULL
- **weighting**: Character string indicating how the edges are weighted (e.g., 'fa', 'pearson', etc.). Default: NULL
- **threshold**: Integer or number indicating the threshold used when “sparsifying” the connectivity matrix (if any). Default: NULL
- **gnames**: Character vector of graph names (e.g., study IDs if level='subject'). Default: NULL
- **...**: Other arguments passed to set_brainGraph_attr
- **grpNames**: Character (or factor) vector of group names. If level == 'group', then you do not need to include this argument (the group names will be the same as gnames). Default: NULL
subnet
Integer or character vector indicating the vertices to keep, if you are interested in working with a subset of an atlas. By default, all vertices are used.

mode
Character string defining how the matrix should be interpreted. Default: 'undirected'

weighted
Logical specifying whether to create a weighted network.

diag
Logical indicating whether to include the diagonal of the connectivity matrix. Default: FALSE

.progress
Logical indicating whether to print a progress bar. Default: getOption('bg.progress')

i
Integer, character, or logical vector for subsetting by subject, or by group (if x$level='group')

g
Integer, character, or logical vector for subsetting by group (if x$level='subject')

drop
If TRUE (the default), then return only the list of graphs; otherwise, subset the graphs and return the entire object

object
A brainGraphList object

g.list
List of graph objects

Details
In addition to creating the initial igraph graphs from the connectivity matrices, then attributes will be calculated and assigned for each graph via set_brainGraph_attr if set.attrs=TRUE. Other arguments can be passed to that function. You may display a progress bar by setting .progress=TRUE.

This object can be considered comparable to a 4-D NIfTI file, particularly that returned by FSL’s TBSS “prestats” step since that file contains the FA volumes for all study subjects.

To convert an object with 3 “levels” (i.e., subject-level lists from an older brainGraph version), see the code in the Examples below.

Value
make_brainGraphList returns an object of class brainGraphList with elements:

threshold
The specified threshold/density

version
The versions of R, igraph, and brainGraph used when creating the graphs

atlas
The atlas common to all the graphs

modality
The imaging modality (if supplied)

weighting
A string indicating what edge weights represent (if applicable)

graphs
A named list of brainGraph graphs; the names correspond to the individual graphs’ Study IDs

[ – A brainGraphList object (if drop=FALSE) or a list of graphs

Subsetting/extracting
The first index is for subsetting the individual graphs. The second index is for subsetting by group membership and requires that the graphs have a Group graph attribute. When both are included, the first index cannot have length or numeric value greater than the number of remaining subjects after subsetting by group.

If the indexing vector(s) is (are) character, the vector(s) must contain one (or more) of the subject or group names. If logical, its length must equal the number of subjects or groups.
brainGraphList

Note

If the input is a corr_mats object, and the extent of the 3-D array is greater than 1, then only the first will be converted to a graph.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

Other Graph creation functions: Creating_Graphs_GLM, Creating_Graphs, make_ego_brainGraph

Examples

```r
## Not run:
# Create a list, one for each threshold
g <- vector('list', length(thresholds))
for (i in seq_along(thresholds)) {
  g[[i]] <- make_brainGraphList(A.norm.sub[[i]], thresholds[i], atlas,
                               covars.dti$Study.ID, covars.dti$Group, modality='dti', weighting='fa')
}

## End(Not run)

## Not run:
# Subset the first 10 subjects, irrespective of group
my.bgl[1:10]

# Return object for only 'Control' subjects
my.bgl[, 'Control']

# Return object with graphs from groups 1 and 3
my.bgl[g=c(1, 3), drop=FALSE]

# Subset the first 10 subjects of group 2
my.bgl[1:10, 2]

## End(Not run)

## Not run:
## Convert old version single-subject graph lists
## g[[1]] is group 1, g[[1]][[1]] is threshold 1, g[[1]][[1]][[1]] is subj. 1
kNumThresholds <- length(g[[1]])
g.l <- vector('list', kNumThresholds)
for (i in seq_len(kNumThresholds)) {
  g.l[[i]] <- as_brainGraphList(do.call(Map, c(c, g))[[i]])
}

## End(Not run)
```
brainGraph_permute  

Permutation test for group difference of graph measures

Description

brainGraph_permute draws permutations from linear model residuals to determine the significance of between-group differences of a global or vertex-wise graph measure. It is intended for structural covariance networks (in which there is only one graph per group), but can be extended to other types of data.

Usage

brainGraph_permute(densities, resids, N = 5000, perms = NULL, 
auc = FALSE, level = c("graph", "vertex", "other"), 
"vulnerability"), .function = NULL)

## S3 method for class 'brainGraph_permute'
summary(object, measure = object$measure, 
alternative = c("two.sided", "less", "greater"), alpha = 0.05, 
p.sig = c("p", "p.fdr"), ...)

## S3 method for class 'brainGraph_permute'
plot(x, measure = x$measure, 
alternative = c("two.sided", "less", "greater"), alpha = 0.05, 
p.sig = c("p", "p.fdr"), ptitle = NULL, ...)

Arguments

densities  Numeric vector of graph densities
resids  An object of class brainGraph_resids (the output from get.resid)
N  Integer; the number of permutations (default: 5e3)
perms  Numeric matrix of permutations, if you would like to provide your own (default: NULL)
auc  Logical indicating whether or not to calculate differences in the area-under-the-curve of metrics (default: FALSE)
level  A character string for the attribute “level” to calculate differences (default: graph)
measure  A character string specifying the vertex-level metric to calculate, only used if level='vertex' (default: btwn.cent). For the summary method, this is to focus on a single graph-level measure (since multiple are calculated at once).
.function  A custom function you can pass if level='other'
object, x  A brainGraph_permute object (output by brainGraph_permute).
alternative Character string, whether to do a two- or one-sided test. Default: 'two.sided'
alpha Numeric; the significance level. Default: 0.05
p.sig Character string specifying which p-value to use for displaying significant results (default: p)
... Unused
ptitle Character string specifying a title for the plot (default: NULL)

Details

If you would like to calculate differences in the area-under-the-curve (AUC) across densities, then specify auc=TRUE.

There are three possible “levels”:

1. graph Calculate modularity (Louvain algorithm), clustering coefficient, characteristic path length, degree assortativity, and global efficiency.
2. vertex Choose one of: centrality metrics (betweenness, closeness, communicability, eigenvector, leverage, pagerank, subgraph); k-core; degree; eccentricity; nodal or local efficiency; k-nearest neighbor degree; shortest path length; transitivity; or vulnerability.
3. other Supply your own function. This is useful if you want to calculate something that I haven’t hard-coded. It must take as its own arguments: g (a list of lists of igraph graph objects); and densities (numeric vector).

Value

An object of class brainGraph_permute with input arguments in addition to:

DT A data table with permutation statistics
obs.diff A data table of the observed group differences
Group Group names

The plot method returns a list of ggplot objects

Author(s)

Christopher G. Watson, cgwatson@bu.edu

See Also

Other Group analysis functions: Bootstrapping, GLM, Mediation, NBS, mtpc
Other Structural covariance network functions: Bootstrapping, IndividualContributions, Residuals, corr.matrix, import_scn, plot_volumetric
Examples

```r
## Not run:
myResids <- get.resid(lhrh, covars)
myPerms <- shuffleSet(n=nrow(myResids$resids.all), nset=1e3)
out <- brainGraph_permute(densities, m, perms=myPerms)
out <- brainGraph_permute(densities, m, perms=myPerms, level='vertex')
out <- brainGraph_permute(densities, m, perms=myPerms, level='other', .function=myFun)
## End(Not run)
```

**centr_betw_comm**

*Calculate communicability betweenness centrality*

**Description**

`centr_betw_comm` calculates the *communicability betweenness* of the vertices of a graph. The centrality for vertex \( r \) is

\[
\omega_r = \frac{1}{C} \sum_p \sum_q \left( (e^A)_{pq} - (e^A + E(r))_{pq} \right) / (e^A)_{pq}
\]

where \( C = (n - 1)^2 - (n - 1) \) is a normalization factor.

**Usage**

```r
centr_betw_comm(g, A = NULL)
```

**Arguments**

- `g` An igraph graph object
- `A` Numeric matrix; the adjacency matrix of the input graph. Default: NULL

**Value**

A numeric vector of the centrality for each vertex

**Author(s)**

Christopher G. Watson, <cgwatson@bu.edu>

**References**


**See Also**

Other Centrality functions: `centr_lev`
centr_lev

Calculate a vertex's leverage centrality

Description

Calculates the leverage centrality of each vertex in a graph.

Usage

centr_lev(g, A = NULL)

Arguments

- `g`: An igraph graph object
- `A`: Numeric matrix; the adjacency matrix of the input graph. Default: NULL

Details

The leverage centrality relates a vertex’s degree with the degree of its neighbors. The equation is:

\[ l_i = \frac{1}{k_i} \sum_{j \in N_i} \frac{k_i - k_j}{k_i + k_j} \]

where \( k_i \) is the degree of the \( i^{th} \) vertex and \( N_i \) is the set of neighbors of \( i \). This function replaces NaN with NA (for functions that have the argument `na.rm`).

Value

A vector of the leverage centrality for all vertices.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References


See Also

Other Centrality functions: `centr_betw_comm`
check_sID  

Test if an object is a character vector of numbers

Description

check_sID is a convenience function to test if a vector (typically the subject ID column in a data.table) is a character vector of numbers, a factor vector of numbers, or a numeric vector. If so, it will zero-pad the variable to have equal width.

pad_zeros pads a vector with zeros to avoid issues with ordering a column of integers or integers converted to character.

Usage

check_sID(x)

pad_zeros(x)

Arguments

x  
pad_zeros accepts either a vector (numeric or character) or a single integer.
check_sID accepts a character, numeric, or factor vector

Details

This function is meant to avoid issues that arise when sorting a vector of numbers that have been converted to character. For example, import_scn automatically reads in the first column (with FreeSurfer outputs this is the column of subject IDs) as a character variable. If the subject IDs had been all numbers/integers, then sorting (i.e., setting the key in a data.table) would be incorrect: e.g., it might be '1', '10', '2', ....

If "x" is a numeric vector, then the resultant string width will be determined by max(x) or x itself if the input is a single integer. For example, if x=10, it will return '01', '02', ..., '10'. If "x" is a character vector, then the output's string width will be max(nchar(x)). For example, if x includes both '1' and '1000', it will return '0001', etc.

Value

check_sID returns either the input vector or a character vector padded with 0
A character vector with zero-padded values

Examples

pad_zeros(10)  # '01' '02' ... '10'
x <- c(1, 10, 100)

pad_zeros(x)  # '001' '010' '100'
x <- as.character(x)

pad_zeros(x)  # '001' '010' '100'
**coeff_var**

*Calculate coefficient of variation*

**Description**

`coeff_var` is a S3 generic that calculates the *coefficient of variation*, defined as

\[
CV(x) = \frac{sd(x)}{mean(x)}
\]

**Usage**

`coeff_var(x, na.rm = FALSE, ...)`

## Default S3 method:
`coeff_var(x, na.rm = FALSE, ...)`

**Arguments**

- **x**
  - Numeric vector, matrix, or array

- **na.rm**
  - Logical indicating whether NA values should be stripped when calculating sums.
    - Default: FALSE

- **...**
  - Unused

**Details**

If `x` is a matrix, it will calculate the CV for each *column*. If `x` is a 3D array, it will calculate the coefficient of variation for each *row-column* combination. If the input dimensions are \( n \times n \times r \), a matrix with size \( n \times n \) will be returned.

**Value**

A numeric vector or matrix

---

**communicability**

*Calculate communicability*

**Description**

`communicability` calculates the communicability of a network, a measure which takes into account all possible paths (including non-shortest paths) between vertex pairs.

**Usage**

`communicability(g, weights = NULL)`
Arguments

- **g**: An igraph graph object
- **weights**: Numeric vector of edge weights; if NULL (the default), and if the graph has edge attribute weight, then that will be used. To avoid using weights, this should be NA.

Details

The communicability $G_{pq}$ is a weighted sum of the number of walks from vertex $p$ to $q$ and is calculated by taking the exponential of the adjacency matrix $A$:

$$G_{pq} = \sum_{k=0}^{\infty} \frac{(A^k)_{pq}}{k!} = (e^A)_{pq}$$

where $k$ is walk length.

For weighted graphs with $D = diag(d_i)$ a diagonal matrix of vertex strength,

$$G_{pq} = (e^{D^{-1/2} A D^{-1/2}})_{pq}$$

Value

A numeric matrix of the communicability

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References


contract_brainGraph

Create a new graph after merging vertices within specified groups. By default, groups are brain lobe and hemisphere membership.

Usage

```r
contract_brainGraph(g, vgroup = "lobe.hemi")
```
**cor.diff.test**

**Arguments**

- `g` A `brainGraph` graph object
- `vgroup` Character string; the name of the vertex attribute to use when contracting the graph. Default: `'lobe.hemi'`

**Details**

The size vertex-level attribute of the resultant graph is equal to the number of vertices in each group. The x-, y-, and z-coordinates of the new graph are equal to the mean coordinates of the vertices per group. The new edge weights are equal to the number of inter-group connections of the original graph.

**Value**

A new `brainGraph` graph object with vertex-level attributes representing the mean spatial coordinates, and vertex- and edge-level attributes of color names

**Author(s)**

Christopher G. Watson, <cgwatson@bu.edu>

**See Also**

`contract`

---

**cor.diff.test**

*Calculate the p-value for differences in correlation coefficients*

**Description**

Given two sets of correlation coefficients and sample sizes, this function calculates and returns the *z-scores* and *p-values* associated with the difference between correlation coefficients.

**Usage**

```r
cor.diff.test(r1, r2, n, alternative = c("two.sided", "less", "greater"))
```

**Arguments**

- `r1, r2` Numeric (vector or matrix) of correlation coefficients for both groups
- `n` Integer vector; number of observations for both groups
- `alternative` Character string, whether to do a two- or one-sided test. Default: `'two.sided'`

**Value**

A list with elements `p` and `z`, the p-values and z-scores for the difference in correlations.
Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

Examples

```r
## Not run:
kNumSubjs <- summary(covars$Group)
corr.diffs <- cor.diff.test(corrs$R[, , 1], corrs$R[, , 2], kNumSubjs)
edge.diffs <- t(sapply(which(corr.diffs$p < .05), function(x)
    mapply('[',
        dimnames(corr.diffs$p),
        arrayInd(x, dim(corr.diffs$p)))))

## End(Not run)
```

## S3 method for class 'corr_mats'
x[i, g = NULL]

## S3 method for class 'corr_mats'
plot(x, mat.type = c("thresholded", "raw"),
    thresh.num = 1L, ordered = TRUE, order.by = "lobe",
    graphs = NULL, grp.names = NULL, legend.title = NULL, ...)

## S3 method for class 'corr_mats'
region.names(object)

## S3 method for class 'corr_mats'
nregions(object)

Description

corr.matrix calculates the correlation between all column pairs of a given data frame, and thresholds the resultant correlation matrix based on a given density (e.g., 0.1 if you want to keep only the 10% strongest correlations). If you want to threshold by a specific correlation coefficient (via the thresholds argument), then the densities argument is ignored.

The plot method will plot “heat maps” of the correlation matrices.

Usage

corr.matrix(resids, densities, thresholds = NULL, what = c("resids",
    "raw"), exclude.reg = NULL, type = c("pearson", "spearman"),
    rand = FALSE)

## S3 method for class 'corr_mats'
x[i, g = NULL]

## S3 method for class 'corr_mats'
plot(x, mat.type = c("thresholded", "raw"),
    thresh.num = 1L, ordered = TRUE, order.by = "lobe",
    graphs = NULL, grp.names = NULL, legend.title = NULL, ...)

## S3 method for class 'corr_mats'
region.names(object)

## S3 method for class 'corr_mats'
nregions(object)
Arguments

resids  An object of class brainGraph_resids (the output from get.resid)
densities  Numeric vector indicating the resultant network densities; keeps the top X% of correlations
thresholds  Numeric; absolute correlation value to threshold by (default: NULL)
what  Character string indicating whether to correlate the residuals or the raw structural MRI values (default: ‘resids’)
exclude.reg  Character vector of regions to exclude (default: NULL)
type  Character string indicating which type of correlation coefficient to calculate (default: ‘pearson’)
rand  Logical indicating whether the function is being called for permutation testing; not intended for general use (default: FALSE)
x, object  A corr_mats object
i  Integer for subsetting by density/threshold
g  Integer, character, or logical for subsetting by group
mat.type  Character string indicating whether to plot raw or thresholded (binarized) matrices. Default: ‘raw’
thresh.num  Integer specifying which threshold to plot (if mat.type=’thresholded’). Default: 1L
ordered  Logical indicating whether to order the vertices by some grouping. Default: TRUE
order.by  Character string indicating how to group vertices. Default: ‘lobe’
graphs  A brainGraphList object containing graphs with the vertex-level attribute of interest. Default: NULL
grp.names  Character vector specifying the names of each group of vertices. Default: NULL
legend.title  Character string for the legend title. Default is to leave blank
...  Unused

Details

If you wish to exclude regions from your analysis, you can give the indices of their columns with the exclude.reg argument.

By default, the Pearson correlation coefficients are calculated, but you can return Spearman by changing the type argument.

Value

A corr_mats object containing the following components:

R, P  Numeric arrays of correlation coefficients and P-values. The length of the 3rd dimension equals the number of groups
r.thresh  A list of 3-d binary arrays indicating correlations that are above a certain threshold. The length of the list equals the number of groups, and the length of the 3rd dimension equals the number of thresholds/densities.
correlation matrices

There are several ways to control the plot appearance. First, you may plot the “raw” correlations, or only those of the thresholded (binarized) matrices. Second, you may order the vertices by a given vertex attribute; by default, they will be ordered by lobe, but you may also choose to order by, e.g., network (for the dosenbach160 atlas) or by community membership. In the latter case, you need to pass a brainGraphList object to the graphs argument; each graph in the object must have a vertex attribute specified in order.by. Finally, you can control the legend text with grp.names.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

rcorr

Other Structural covariance network functions: Bootstrapping, IndividualContributions, Residuals, brainGraph_permute, import_scn, plot_volumetric

Examples

```r
## Not run:
myResids <- get.resid(lhrh, covars)
corrs <- corr.matrix(myResids, densities=densities))

## End(Not run)
## Not run:
corrs <- corr.matrix(myResids, densities)
plot(corrs, order.by='comm', graphs=g.list, grp.names='Community')

## End(Not run)
```
Count Edges

Count number of edges of a brain graph

Description

count_homologous counts the number of edges between homologous regions in a brain graph (e.g. between L and R superior frontal).

count_inter counts the number of edges between and within all vertices in one group (e.g. lobe, hemi, network, etc.).

Usage

count_homologous(g)

count_inter(g, group = c("lobe", "hemi", "network", "class", "gyrus", "Yeo_7network", "Yeo_17network", "area", "Brodmann"))

Arguments

g A brainGraph graph object

group Character string specifying which grouping to calculate edge counts for. Default: ‘lobe’

Value

count_homologous - a named vector of the edge ID’s connecting homologous regions

count_inter - a data.table of total, intra-, and inter-group edge counts

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

Examples

## Not run:
g1.lobecounts <- count_inter(g[[1]][[N]], 'lobe')

## End(Not run)
create_mats

Description

create_mats creates arrays from connection matrices (e.g., fdt_network_matrix from FSL or ROICorrelation.txt from DPABI). You may choose to normalize these matrices by the waytotal or region size (tractography), or not at all.

Usage

create_mats(A.files, modality = c("dti", "fmri"), divisor = c("none", "waytotal", "size", "rowSums"), div.files = NULL, threshold.by = c("consensus", "density", "mean", "consistency", "raw"), mat.thresh = 0, sub.thresh = 0.5, inds = list(seq_along(A.files)), algo = c("probabilistic", "deterministic"), P = 5000, ...)

Arguments

A.files Character vector of the filenames with connection matrices
modality Character string indicating data modality (default: dti)
divisor Character string indicating how to normalize the connection matrices; either 'none' (default), 'waytotal', 'size', or 'rowSums' (ignored if modality equals fmri)
div.files Character vector of the filenames with the data to normalize by (e.g. a list of waytotal files) (default: NULL)
threshold.by Character string indicating how to threshold the data; choose density, mean, or consistency if you want all resulting matrices to have the same densities (default: consensus)
mat.thresh Numeric (vector) for thresholding connection matrices (default: 0)
sub.thresh Numeric (between 0 and 1) for thresholding by subject numbers (default: 0.5)
inds List (length equal to number of groups) of integers; each list element should be a vector of length equal to the group sizes
algo Character string of the tractography algorithm used (default: 'probabilistic'). Ignored if modality is fmri.
P Integer; number of samples per seed voxel (default: 5000)
... Arguments passed to symmetrize

Value

A list containing:

A A 3-d array of the raw connection matrices
A.norm A 3-d array of the normalized connection matrices
create_mats

A.bin A list of 3-d arrays of binarized connection matrices, one array for each threshold
A.bin.sums A list of 3-d arrays of connection matrices, with each entry signifying the number of subjects with a connection present; the number of list elements equals the length of mat.thresh, and the extent of the arrays equals the number of groups
A.inds A list of arrays of binarized connection matrices, containing 1 if that entry is to be included
A.norm.sub List of 3-d arrays of the normalized connection matrices for all given thresholds
A.norm.mean List of 3-d arrays of connection matrices averaged for each group

Connection matrix files

The A.files argument is mandatory and may be specified in a few ways:

1. A character vector of the filenames (preferably with full path).
2. A single character string specifying the directory in which all connectivity matrices are located. This will load all files in the directory.
3. A named list in which the names match the arguments to list.files. This will load all files in path that match the pattern argument, if present, and will load all files in child directories if recursive=TRUE. See examples below.

The same options apply to div.files as well.

Thresholding methods

The argument threshold.by has 5 options:

1. consensus Threshold based on the raw (normalized, if selected) values in the matrices. If this is selected, it uses the sub.thresh value to perform “consensus” thresholding.
2. density Threshold the matrices to yield a specific graph density (given by the mat.thresh argument).
3. mean Keep only connections for which the cross-subject mean is at least 2 standard deviations higher than the threshold (specified by mat.thresh).
4. consistency Threshold based on the coefficient of variation to yield a graph with a specific density (given by mat.thresh). The edge weights will still represent those of the input matrices. See Roberts et al. (2017) for more on “consistency-based” thresholding.
5. raw Threshold each subject’s matrix individually, irrespective of group membership. Ignores sub.thresh.

The argument mat.thresh allows you to choose a numeric threshold, below which the connections will be replaced with 0; this argument will also accept a numeric vector. The argument sub.thresh will keep only those connections for which at least X% of subjects have a positive entry (the default is 0.5, or 50%).

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>
Creating Graphs

References


Examples

```r
## Not run:
thresholds <- seq(from=0.001, to=0.01, by=0.001)
fmri.mats <- create_mats(f.A, modality='fmri', threshold.by='consensus',
    mat.thresh=thresholds, sub.thresh=0.5, inds=inds)
dti.mats <- create_mats(f.A, divisor='waytotal', div.files=f.way,
    mat.thresh=thresholds, sub.thresh=0.5, inds=inds)

# Specify a directory and filename pattern
c Conn_files <- list(path='~/data', pattern='.*fdt_network_matrix')
dti.mats <- create_mats(conn_files, ...)
## End(Not run)
```

Creating_graphs

Create a brainGraph object

Description

make_brainGraph is the main creation function for creating a brainGraph graph object. This is simply an igraph graph object with additional attributes (at all levels). Several of the graph-level attributes serve the purpose of providing metadata on how the connectivity matrices/networks were created.

make_brainGraph.bg_mediate creates a graph only for vertex-level analyses.

make_empty_brainGraph creates an empty undirected brainGraph object with vertex count equal to the atlas specified; i.e., it creates a graph with 0 edges. Typically used to present results from an analysis in which edges don’t make sense (e.g., GLM comparing differences in a vertex-level attribute).

Usage

```r
make_brainGraph(x, atlas, type = c("observed", "random"),
    level = c("subject", "group", "contrast"), set.attrs = TRUE,
    modality = NULL, weighting = NULL, threshold = NULL, ...)
```

```r
## S3 method for class 'igraph'
make_brainGraph(x, atlas, type = c("observed", "random"),
    level = c("subject", "group", "contrast"), set.attrs = TRUE,
    modality = NULL, weighting = NULL, threshold = NULL, name = NULL, Group = NULL, subnet = NULL, ...)
```

```r
## S3 method for class 'matrix'
```
make_brainGraph(x, atlas = x$atlas,
    type = "observed", level = "subject", set.attrs = TRUE,
    modality = NULL, weighting = NULL,
    threshold = NULL, name = NULL, Group = NULL, subnet = NULL,
    mode = "undirected", weighted = NULL, diag = FALSE, ...)

## S3 method for class 'bg_mediate'
make_brainGraph(x, atlas = x$atlas,
    type = "observed", level = "contrast", set.attrs = FALSE,
    modality = NULL, weighting = NULL, threshold = NULL, ...)

is.brainGraph(x)

## S3 method for class 'brainGraph'
summary(object, print.attrs = c("all", "graph",
    "vertex", "edge", "none"), ...)

make_empty_brainGraph(atlas, type = c("observed", "random",
    level = c("subject", "group", "contrast"), modality = NULL,
    weighting = NULL, threshold = NULL, name = NULL, Group = NULL,
    ...)
Creating Graphs

diag Logical indicating whether to include the diagonal of the connectivity matrix. Default: FALSE

object A brainGraph object

print.attrs Character string indicating whether or not to list the object’s attributes (default: all)

Value

A brainGraph graph object with additional graph-, vertex-, and edge-level attributes (see below). The method for bg_mediate returns a brainGraph_mediate object, which has extra attributes:

Graph mediator, treat, outcome, nobs

Vertex b?.acme, p?.acme, b?.ade, p?.ade, b?.prop, p?.prop, b.tot, p.tot

make_empty_brainGraph – An empty brainGraph graph object

Graph-level attributes

Graph-level attributes added are:

version The R, brainGraph, and igraph package versions used to create the graph
date The creation date, from as.POSIXct
atlas Character string denoting the brain atlas used
type Character string specifying whether this is an observed or random graph
modality The imaging modality; you can choose anything you like, but the summary.brainGraph knows about dti, fmri, thickness, area, and volume
weighting What edge weights represent; you can choose anything you like, but summary.brainGraph knows about fa, sld (streamline density, tractography), pearson, spearman, kendall, and partial (partial correlation coefficient)
threshold Numeric indicating the threshold used to create the final connectivity matrix (if any)
name Character string specifying the study ID or group/contrast name, depending on the level argument
Group Character string specifying the experimental group that the given subject belongs to, or if it is a group-level graph
subnet Integer vector, if subnet was specified in the call

Vertex attributes

Vertex-level attributes added are:

name The names of the brain regions in the network
lobe The names of the major brain lobes for each vertex
hemi The names of the hemisphere for each vertex (either ’L’, ’R’, or ’B’)
lobe.hemi The lobe-hemisphere combination (represented as an integer vector)
class The tissue class (if applicable)
network  The network (if the atlas is dosenbach160)
x,y,z  The spatial coordinates of the (centers-of-mass) brain regions in MNI space
x.mni,y.mni,z.mni  Same as above
color.lobe,color.class,color.network  Colors for vertices of their respective membership
circle.layout  Integer vector indicating the order (going counter-clockwise from the top) for circular layouts

Edge attributes

Edge-level attributes added are:
color.lobe,color.class,color.network  Correspond to the vertex attribute of the same name. Inter-group edges will be colored gray

Specifying a subnetwork

You can create a graph for a subset of an atlas's regions with the subnet argument. This can either be a numeric or character vector. If the input object (either a matrix or an igraph graph) has fewer rows/columns or vertices, respectively, than the atlas then the subnet graph attribute will also be added to the return object. This may occur if, for example, you use make_auc_brainGraph on graphs that were initially created from subnetworks.

See Also

Other Graph creation functions: Creating_Graphs_GLM, brainGraphList, make_ego_brainGraph

Examples

```r
## Not run:
bg <- make_brainGraph(A, 'dkt', modality='dti', weighting='fa',
  mode='undirected', diag=FALSE, weighted=TRUE)

## End(Not run)
```

Creating_Graphs_GLM Create a graph list with GLM-specific attributes

Description

These methods create a brainGraphList with attributes specific to the results of brainGraph_GLM, mtpc, or NBS. The graphs element of the returned object will contain one graph for each contrast.
Usage

```r
## S3 method for class 'bg_GLM'
make_brainGraphList(x, atlas = x$atlas,
                   type = "observed", level = "contrast", set.attrs = FALSE,
                   modality = NULL, weighting = NULL, threshold = NULL,
                   gnames = x$con.name, ...)

## S3 method for class 'mtpc'
make_brainGraphList(x, atlas = x$atlas,
                   type = "observed", level = "contrast", set.attrs = FALSE,
                   modality = NULL, weighting = NULL, threshold = NULL,
                   gnames = x$con.name, ...)

## S3 method for class 'NBS'
make_brainGraphList(x, atlas, type = "observed",
                   level = "contrast", set.attrs = TRUE, modality = NULL,
                   weighting = NULL, threshold = NULL, gnames = x$con.name,
                   mode = "undirected", weighted = TRUE, diag = FALSE, ...)
```

Arguments

- `x` A bg_GLM, mtpc, or NBS object
- `atlas` Character string specifying the brain atlas to use
- `type` Character string indicating the type of graphs. Default: observed
- `level` Character string indicating whether the graphs are subject-, group-, or contrast-specific. Default: 'subject'
- `set.attrs` Logical indicating whether to assign all graph-, vertex-, and edge-level attributes (via `set_brainGraph_attr`). Default: TRUE
- `modality` Character string indicating imaging modality (e.g. 'dti'). Default: NULL
- `weighting` Character string indicating how the edges are weighted (e.g., 'fa', 'pearson', etc.). Default: NULL
- `threshold` Integer or number indicating the threshold used when “sparsifying” the connectivity matrix (if any). Default: NULL
- `gnames` Character vector of graph names (e.g., study IDs if level='subject'). Default: NULL
- `...` Other arguments passed to `set_brainGraph_attr`
- `mode` Character string defining how the matrix should be interpreted. Default: 'undirected'
- `weighted` Logical specifying whether to create a weighted network
- `diag` Logical indicating whether to include the diagonal of the connectivity matrix. Default: FALSE

Value

A brainGraphList object, with a graph object for each contrast with additional attributes:
edge_asymmetry

Graph name (contrast name), outcome (the outcome variable), alpha (the significance level); for MTPC: tau.mtpc, S.mtpc, S.crit, A.crit

Vertex size2 (t-statistic); size (the t-stat transformed for visualization purposes); p (equal to \(1 - p\)); p.fdr (equal to \(1 - p_{FDR}\), the FDR-adjusted p-value); effect.size (the contrast of parameter estimates for t-contrasts; the extra sum of squares for F-contrasts); se (the standard error of gamma); A.mtpc, sig (binary indicating whether A.mtpc > A.crit) (for MTPC)

make_brainGraphList.NBS returns graphs with additional attributes:

Vertex comp (integer vector indicating connected component membership), p.nbs (P-value for each component)

Edge stat (the test statistic for each connection), p (the P-value)

Note
Only valid for vertex-level and NBS analyses.

See Also
brainGraph_GLM, mtpc, NBS

Other Graph creation functions: Creating_Graphs, brainGraphList, make_ego_brainGraph

edge_asymmetry Calculate an asymmetry index based on edge counts

Description
Calculate an asymmetry index, a ratio of intra-hemispheric edges in the left to right hemisphere of a graph for brain MRI data.

Usage
edge_asymmetry(g, level = c("hemi", "vertex"), A = NULL)

Arguments

- **g**: An igraph graph object
- **level**: Character string indicating whether to calculate asymmetry for each region, or the hemisphere as a whole (default: ‘hemi’)
- **A**: Numeric matrix; the adjacency matrix of the input graph. Default: NULL
The equation is:

\[ A = \frac{E_{lh} - E_{rh}}{0.5 \times (E_{lh} + E_{rh})} \]

where \( lh \) and \( rh \) are left and right hemispheres, respectively. The range of this measure is \([-2, 2]\) (although the limits will only be reached if all edges are in one hemisphere), with negative numbers indicating more edges in the right hemisphere, and a value of 0 indicating equal number of edges in each hemisphere.

The level argument specifies whether to calculate asymmetry for each vertex, or for the whole hemisphere.

A data table with edge counts for both hemispheres and the asymmetry index; if level is vertex, the data table will have \( vcount(g) \) rows.

Christopher G. Watson, <cgwatson@bu.edu>

This function calculates the global efficiency of a graph or the local or nodal efficiency of each vertex of a graph.

**Usage**

```r
efficiency(g, type = c("local", "nodal", "global"), weights = NULL,
          xfm = FALSE, xfm.type = NULL, use.parallel = TRUE, A = NULL,
          D = NULL)
```

**Arguments**

- `g` An igraph graph object
- `type` Character string; either local, nodal, or global. Default: local
- `weights` Numeric vector of edge weights; if NULL (the default), and if the graph has edge attribute weight, then that will be used. To avoid using weights, this should be NA.
- `xfm` Logical indicating whether to transform the edge weights. Default: FALSE
- `xfm.type` Character string specifying how to transform the weights. Default: 1/w
- `use.parallel` Logical indicating whether or not to use foreach. Default: TRUE
- `A` Numeric matrix; the adjacency matrix of the input graph. Default: NULL
- `D` Numeric matrix; the graph’s “distance matrix”
Details

Local efficiency for vertex $i$ is:

$$E_{local}(i) = \frac{1}{N} \sum_{i \in G} E_{global}(G_i)$$

where $G_i$ is the subgraph of neighbors of $i$, and $N$ is the number of vertices in that subgraph.

Nodal efficiency for vertex $i$ is:

$$E_{nodal}(i) = \frac{1}{N-1} \sum_{j \in G} 1$$

Global efficiency for graph $G$ with $N$ vertices is:

$$E_{global}(G) = \frac{1}{N(N-1)} \sum_{i \neq j \in G} 1$$

where $d_{ij}$ is the shortest path length between vertices $i$ and $j$. Alternatively, global efficiency is equal to the mean of all nodal efficiencies.

Value

A numeric vector of the efficiencies for each vertex of the graph (if type is local|nodal) or a single number (if type is global).

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References


---

**GLM**

*Fit General Linear Models at each vertex of a graph*

**Description**

brainGraph_GLM specifies and fits a General Linear Model (GLM) at each vertex for a given vertex measure (e.g., degree) or at the graph-level (e.g., global efficiency). Given a contrast matrix or list of contrast(s), and contrast type (for t- or F-contrast(s), respectively) it will calculate the associated statistic(s) for the given contrast(s).
The summary method prints the results, only for which \( p < \alpha \), where \( \alpha \) comes from the bg\_GLM object. “Simple” P-values are used by default, but you may change this to the FDR-adjusted or permutation P-values via the function argument p\_sig. You may also choose to subset by contrast.

The plot method plots the GLM diagnostics (similar to that of \texttt{plot.lm}). There are a total of 6 possible plots, specified by the which argument; the behavior is the same as in \texttt{plot.lm}. Please see the help for that function.

The \( \_[] \) method allows you to select observations (i.e., rows of \( X \) and \( y \)) and independent variables (i.e., columns of \( X \)) from a bg\_GLM object.

### Usage

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

## S3 method for class 'bg_GLM'
print(x, ...)

## S3 method for class 'bg_GLM'
summary(object, p.sig = c("p", "p.fdr", "p.perm"), contrast = NULL, alpha = object$alpha, digits = max(3L, getOption("digits") - 2L), print.head = TRUE, ...)

## S3 method for class 'bg_GLM'
plot(x, region = NULL, which = c(1L:3L, 5L), ids = TRUE, ...)

## S3 method for class 'bg_GLM'
x[i, j]
\end{verbatim}

### Arguments

\begin{itemize}
  \item \textbf{g.list} A \texttt{brainGraphList} object
  \item \textbf{covars} A \texttt{data.table} of covariates
  \item \textbf{measure} Character string of the graph measure of interest
  \item \textbf{contrasts} Numeric matrix (for T statistics) or list of matrices (for F statistics) specifying the contrast(s) of interest; if only one contrast is desired, you can supply a vector (for T statistics)
  \item \textbf{con.type} Character string; either 't' or 'f' (for t or F-statistics). Default: 't'
  \item \textbf{outcome} Character string specifying the name of the outcome variable, if it differs from the graph metric (measure)
  \item \textbf{X} Numeric matrix, if you wish to supply your own design matrix. Ignored if outcome != measure.
\end{itemize}
### Parameters

- **con.name**: Character vector of the contrast name(s); if contrasts has row/list names, those will be used for reporting results.
- **alternative**: Character string, whether to do a two- or one-sided test. Default: `"two.sided"`.
- **alpha**: Numeric; the significance level. Default: 0.05.
- **level**: Character string; either `vertex` (default) or `graph`.
- **permute**: Logical indicating whether or not to permute group labels. Default: `FALSE`.
- **perm.method**: Character string indicating the permutation method. Default: `"freedmanLane"`.
- **part.method**: Character string; the method of partitioning the design matrix into covariates of interest and nuisance. Default: `"beckmann"`.
- **N**: Integer; number of permutations to create. Default: 5e3.
- **perms**: Matrix of permutations, if you would like to provide your own. Default: `NULL`.
- **long**: Logical indicating whether or not to return all permutation results. Default: `FALSE`.
- **...**: Arguments passed to `brainGraph_GLM_design`.
- **object, x**: A `bg_GLM` object.
- **p.sig**: Character string specifying which P-value to use for displaying significant results (default: `p`).
- **contrast**: Integer specifying the contrast to plot/summarize; defaults to showing results for all contrasts.
- **digits**: Integer specifying the number of digits to display for P-values.
- **print.head**: Logical indicating whether or not to print only the first and last 5 rows of the statistics tables (default: `TRUE`).
- **region**: Character string specifying which region’s results to plot; only relevant if `level` = `"vertex"`. Default: `NULL`.
- **which**: Integer vector indicating which of the 6 plots to print to the plot device. Default: `c(1:3, 5)`.
- **ids**: Logical indicating whether to plot subject ID’s for outliers. Otherwise plots the integer index.
- **i, j**: Integer/character vector; the observation number(s) or row names to select or remove.

### Details

The `measure` argument will be the graph- or vertex-level measure of interest. Often, this will serve as the model’s `outcome` (or dependent, or response) variable; i.e., the variable typically denoted by `y` in GLMs. In other cases, you may wish to choose some other variable as the outcome; e.g., IQ, age, etc. Then you could test for a direct association between the network measure and outcome of interest, or test for another association while adjusting for the network metric. For these applications, you must provide the variable name via the `outcome` argument. This is analogous to `~evperdat` in FSL’s PALM and to `--pvr` in FreeSurfer.
Value

An object of class bg_GLM containing some input-specific variables (level, outcome, measure, con.type, contrasts, con.name, alt, alpha, permute, perm.method, part.method, N) in addition to:

DT.Xy
A data table from which the design matrices are created and the outcome variable, for all regions.

X
A named numeric matrix or a 3D array of the design matrix. Rownames are subject IDs, column names are predictor variables, and dimnames along the 3rd dimension are region names (if applicable). This is a 3D array only if outcome != measure and level == 'vertex'.

y
A named numeric matrix of the outcome variable. Rownames are Study IDs and column names are regions. There will be multiple columns only if outcome == measure and level == 'vertex'.

DT
A data table with an entry for each vertex (region) containing statistics of interest

removed.subs
A named integer vector in which the names are subject ID's of those removed due to incomplete data (if any). The integers correspond to the row number in the input covars table.

runX
If outcome != measure and level == 'vertex', this will be a character vector of the regions for which the design matrix is invertible. Otherwise, it is NULL.

runY
Character vector of the regions for which the outcome variable has 0 variability. For example, if level='vertex' and measure='degree', some regions may be disconnected or have the same degree for all subjects.

atlas
Character string of the atlas used (guessed based on the vertex count).

perm
A list containing: null.dist (the null distribution of maximum statistics), thresh (the statistic value corresponding to the 100 \times (1 - \alpha)\% percentile of the null distribution)

The plot method returns a list of ggplot objects (if installed) or writes the plots to a PDF in the current directory named bg_GLM_diagnostics.pdf

A bg_GLM object with the specified row(s) selected or removed from both X and y, and column(s) selected/removed from X

Design matrix

The GLM’s design matrix will often be identical to the model matrix associated with lm objects (if “dummy” coding, the default, is used) and is created from the input data.table and arguments passed to brainGraph_GLM_design. The first column should have the name of getOption('bg.subject_id') and its values must match the name graph-level attribute of the input graphs. The covariates table must be supplied even if you provide your own design matrix X. If level='vertex' and outcome == measure, there will be a single design for all regions but a separate model for each region (since the graph measure varies by region). If level='vertex' and outcome != measure, there will be a separate design (and, therefore, a separate model) for each region even though the outcome is the same in all models.
Contrasts and statistics

Either t- or F-contrasts can be calculated (specified by con.type). Multiple t-contrasts can be specified by passing a multi-row matrix to contrasts. Multiple F-contrasts can be specified by passing a list of matrices; all matrices must have the same number of columns. All F-contrasts are necessarily two-sided; t-contrasts can be any direction, but only one can be chosen per function call. If you choose con.type="f", the calculated effect size is represented by the ESS ("extra sum of squares"), the additional variance explained for by the model parameters of interest (as determined by the contrast matrix). The standard error for F-contrasts is the sum of squared errors of the full model.

Non-parametric permutation tests

You can calculate permutations of the data to build a null distribution of the maximum statistic which corrects for multiple testing. To account for complex designs, the design matrix must be partitioned into covariates of interest and nuisance; the default method is the Beckmann method. The default permutation strategy is that of Freedman & Lane (1983), and is the same as that in FSL's randomise. See randomise.

Note

The [] method is used when calculating studentized residuals and other “leave-one-out” diagnostics, and typically should not be called directly by the user.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

plot.lm
Other GLM functions: GLM design, GLM fits, mtpc
Other Group analysis functions: Bootstrapping, Mediation, NBS, brainGraph_permute, mtpc

Examples

## Not run:
conmat <- matrix(c(0, 0, 0, 1), nrow=1)
rownames(conmat) <- 'Control > Patient'
res.lm <- brainGraph_GLM(g[[6]], covars=covars.all[tract == 1],
measure='strength', contrasts=conmat, alt='greater', permute=TRUE, long=TRUE)
## End(Not run)

## Not run:
## Save objects and then to multipage PDF
lmPlots <- plot(x)
ggsave('lmPlots.pdf', lmPlots)

## Save all the GLM sub-objects from MTPC analysis
res.mtpc <- mtpc(...)

## Not run:
glmPlots <- lapply(res.mtpc$res.glm, plot, which=1:6)
ml <- marrangeGrob(glmPlots, nrow=1, ncol=1)
ggsave('glmPlots.pdf', ml, width=8.5, height=11)

## End(Not run)

---

GLM basic info  
**Extract basic information from a bg_GLM object**

**Description**

These functions return the terms, *term labels*, *model formula*, “case names”, “variable names”, *region names*, and number of observations for a bg_GLM object. The term labels are used for ANOVA tables.

**Usage**

```r
## S3 method for class 'bg_GLM'
noobs(object, ...)

## S3 method for class 'bg_GLM'
terms(x, ...)

## S3 method for class 'bg_GLM'
formula(x, ...)

## S3 method for class 'bg_GLM'
labels(object, ...)

## S3 method for class 'bg_GLM'
case.names(object, ...)

## S3 method for class 'bg_GLM'
variable.names(object, ...)

## S3 method for class 'bg_GLM'
region.names(object)

## S3 method for class 'bg_GLM'
nregions(object)
```

**Arguments**

- `...`: Unused
- `x, object`: A bg_GLM object
Value

terms returns a named integer list in which the names are the term labels and the list elements are the column(s) of the design matrix for each term. nobs returns an integer. The other functions return character vectors.

Note

formula returns a character string, not a formula object.

---

**GLM design**

Create a design matrix for linear model analysis

**Description**

brainGraph_GLM_design takes a data.table of covariates and returns a design matrix to be used in linear model analysis.

**Usage**

brainGraph_GLM_design(covars, coding = c("dummy", "effects", "cell.means"), factorize = TRUE, binarize = NULL, int = NULL, mean.center = FALSE, center.how = c("all", "within-groups"), center.by = getOption("bg.group"))

**Arguments**

covars A data.table of covariates
coding Character string indicating how factor variables will be coded. Default: 'dummy'
factorize Logical indicating whether to convert character columns into factor. Default: TRUE
binarize Character vector specifying the column name(s) of the covariate(s) to be converted from type factor to numeric. Default: NULL
int Character vector specifying the column name(s) of the covariate(s) to test for an interaction. Default: NULL
mean.center Logical indicating whether to mean center non-factor variables. Default: FALSE
center.how Character string indicating whether to use the grand mean or groupwise means. Default: 'all'
center.by Character string indicating which grouping variable to use for calculating means (if applicable). Default: 'Group'

**Details**

There are three different ways to code factors: dummy, effects, or cell-means (chosen by the argument coding). Effects coding is sometimes referred to as deviation coding. Dummy coding is the default when calling lm. To understand the difference between these, see Chapter 8 of the User Guide.
Value

A numeric matrix. Rownames are subject ID’s and column names are the variable names. There will be additional attributes recording the coding, factorize, and mean.center function arguments. There will also be attributes for binarize and int if they are not NULL, and center.how and center.by if mean.center=TRUE.

Character variables

The default behavior is to convert all character columns (excluding the Study ID column and any that you list in the binarize argument) to factor variables. To change this, set factorize=FALSE. So, if your covariates include multiple character columns, but you want to convert Scanner to binary instead of a factor, you may still specify binarize='Scanner' and get the expected result. binarize will convert the given factor variable(s) into numeric variable(s), which is performed before centering (if applicable).

Centering

The argument mean.center will mean-center (i.e., subtract the mean of from each variable) any non-factor variables (including any dummy/indicator covariates). This is done after “factorizing” and “binarizing”. If center.how='all', then the “grand mean” will be used; otherwise, the group-wise means will be used. The grouping variable is determined by center.by and is by default 'Group'.

Interactions

int specifies which variables should interact with one another. This argument accepts both numeric/continuous (e.g., Age) and factor variables (e.g., Sex). All interaction combinations will be generated: if you supply 3 variables, all two-way and the single three-way interaction will be generated. This variable must have at least two elements; it is otherwise ignored. It is generally recommended that centering be performed when including interaction terms.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

Other GLM functions: GLM fits, GLM, mtpc

Examples

```r
## Not run:
# Recreate design matrix when "outcome == measure"
DT <- res.glm$DT.Xy[region == levels(region)[1L],
  !c('region', res.glm$outcome),
  with=FALSE]
X <- do.call(brainGraph_GLM_design, c(list(covars=DT),
  attributes(res.glm$X)[-c(1L, 2L)]))
all.equal(X, res.glm$X)
```
GLM fits

Fit design matrices to one or multiple outcomes

Description

These are the “base” model-fitting functions that solve the least squares problem to estimate model coefficients, residuals, etc. for brain network data.

fastLmBG_t and fastLmBG_f calculate contrast-based statistics for T or F contrasts, respectively. It accepts any number of contrasts (i.e., a multi-row contrast matrix).

Usage

fastLmBG(X, Y, QR = qr.default(X), Q = qr.Q2(QR, n = n, p = p),
  R = qr.R2(QR, p), n = dim(X)[1L], p = QR$rank, ny = dim(Y)[2L],
  dfR = n - p, XtXinv = inv(QR))

fastLmBG_3d(X, Y, runX, QR = qr(X[, , runX, drop = FALSE]),
  Q = lapply(QR, qr.Q2, n = n, p = p), R = lapply(QR, qr.R2, p),
  n = dim(X)[1L], p = QR[[1L]]$rank, ny = length(runX), dfR = n - p,
  XtXinv = inv(QR))

fastLmBG_3dY(X, Y, runX, QR = qr(X[, , runX, drop = FALSE]),
  Q = lapply(QR, qr.Q2, n = n, p = p), R = lapply(QR, qr.R2, p),
  n = dim(X)[1L], p = QR[[1L]]$rank, ny = length(runX), dfR = n - p,
  XtXinv = inv(QR))

fastLmBG_3dY_1p(X, Y, runX, QR = qr(X[, , runX, drop = FALSE]),
  Q = lapply(QR, qr.Q2, diag(1L, n, 1L), n, 1L), R = lapply(QR,
  function(r) r$qr[[1L]]), n = dim(X)[1L], p = 1L, ny = length(runX),
  dfR = n - 1L, XtXinv = inv(QR))

fastLmBG_t(fits, contrasts, alternative = c("two.sided", "less",
  "greater"), alpha = NULL)

fastLmBG_f(fits, contrasts, rkC = NULL, nC = length(contrasts))

Arguments

X  Design matrix or 3D array of design matrices

Y  Numeric matrix; there should be 1 column for each outcome variable (so that in
  a graph-level analysis, this is a column matrix)

QR, Q, R  The QR decomposition(s) and Q and R matrix(es) of the design matrix(es). If X
  is a 3D array, these should be lists
GLM fits

n, p, ny, dfR Integers; the number of observations, model rank, number of regions/outcome variables, and residual degrees of freedom

XtXinv Numeric matrix or array; the inverse of the cross-product of the design matrix(es)

runX Character vector of the regions for which the design matrix is not singular

fits List object output by one of the model fitting functions (e.g., fastLmBG)

contrasts Numeric matrix (for T statistics) or list of matrices (for F statistics) specifying the contrast(s) of interest; if only one contrast is desired, you can supply a vector (for T statistics)

alternative Character string, whether to do a two- or one-sided test. Default: 'two.sided'

alpha Numeric; the significance level. Default: 0.05

rkC, nC Integers; the rank of the contrast matrix and number of contrasts, respectively (for F contrasts)

Value

A list with elements

- coefficients Parameter estimates
- rank Model rank
- df.residual Residual degrees of freedom
- residuals Model residuals
- sigma The residual standard deviation, or root mean square error (RMSE)
- fitted.values Model fitted values
- qr The design matrix QR decomposition(s)
- cov.unscaled The “unscaled covariance matrix”

fastLmBG_t – A multidimensional array with the third dimension equaling the number of contrasts; each matrix contains the contrast of parameter estimates, standard error of the contrast, T-statistics, P-values, FDR-adjusted P-values, and confidence intervals (if alpha is given)

fastLmBG_f – A numeric matrix with columns for the effect size, standard error, F statistic, P-values, and FDR-adjusted P-values

Parameter estimation

These functions use the QR decomposition to calculate the least squares solution which is the same as the base lm function. If we substitute \( X = QR \) in the standard normal equations, the equation to be solved reduces to

\[ X^T X \hat{\beta} = X^T y \Rightarrow R \hat{\beta} = Q^T y \]

Since \( R \) is an upper-triangular matrix, we can use the backsolve function which is a bit faster than solve. In some cases, the fastLmBG* functions are about as fast or faster (particularly when \( X \) is not permuted) as one in which the normal equations are solved directly; additionally, using the QR method affords greater numerical stability.
Different scenarios

There are a few different scenarios for fitting models of the data, with a separate function for each:

- **fastLmBG**: The main function for when there is a single design matrix \( X \) and any number of outcome variables \( Y \).
- **fastLmBG_3d**: Fits models when there is a different design matrix \( X \) for each region and a single outcome variable \( Y \), which in this case will be a column matrix.
- **fastLmBG_3dY**: Fits models when there is both a different design matrix \( X \) and outcome variable \( Y \) for each region. Occurs under permutation for the Freedman-Lane, ter Braak, and Still-White methods.
- **fastLmBG_3dY_1p**: Fits models when there is both a different design and outcome variable for each region, and also when \( X \) is a rank-1 matrix (i.e., it has 1 column). Only occurs under permutation with the Still-White method if there is a single regressor of interest.

In the last case above, model coefficients are calculated by simple (i.e., non-matrix) algebra.

Improving speed/efficiency

Speed/efficiency gains will be vast for analyses in which there is a single design matrix \( X \) for all regions, there are multiple outcome variables (i.e., vertex-level analysis), and the permutation method chosen does not permute \( X \). Specifically, these are Freedman-Lane, ter Braak, and Manly methods. Therefore, the QR decomposition, the \( Q \) and \( R \) matrices, and the “unscaled covariance matrix” (which is \((X^TX)^{-1}\)) only need to be calculated once for the entire analysis. Other functions (e.g., `lm.fit`) would recalculate these for each permutation.

Furthermore, this (and the other model fitting functions in the package) will likely only work in models with full rank. I sacrifice proper error checking in favor of speed, but hopefully any issues with the model will be identified prior to the permutation step. Finally, the number of observations, model rank, number of outcome variables, and degrees of freedom will not change and therefore do not need to be recalculated (although these probably amount to a negligible speed boost).

In case there are multiple design matrices, or the permutation method permutes the design, then the QR decomposition will need to be calculated each time anyway. For these cases, I use more simplified functions `qr.Q2` and `qr.R2` to calculate the \( Q \) and \( R \) matrices, and then the fitted values, residuals, and residual standard deviation are calculated at the same time (whereas `lm.fit` and others would calculate these each time).

Contrast-based statistics

The contrast of parameter estimates, \( \gamma \), for T contrasts is

\[
\gamma = C\hat{\beta}
\]

where \( C \) is the contrast matrix with size \( k \times p \) (where \( k \) is the number of contrasts) and \( \hat{\beta} \) is the matrix of parameter estimates with size \( p \times r \) (where \( r \) is the number of regions). For F contrasts, the effect size is the extra sum of squares and is calculated as

\[
\gamma(C(X^TX)^{-1}C^T)^{-1}\gamma^T
\]

The standard error of a T contrast is

\[
\sqrt{\hat{\sigma}(X^TX)^{-1}}
\]
where $\hat{\sigma}$ is the residual standard deviation of the model and the second term is the unscaled covariance matrix. The standard error for F contrasts is simply the residual sum of squares. P-values and FDR-adjusted P-values (across regions) are also calculated. Finally, if $\alpha$ is provided for T contrasts, confidence limits are calculated.

Author(s)
Christopher G. Watson, <cgwatson@bu.edu>

See Also
randomise
Other GLM functions: GLM design, GLM, mtpc

GLM influence measures
Influence measures for a bg_GLM object

Description
These functions compute common (leave-one-out) diagnostics for the models in a bg_GLM object.

Usage
```r
## S3 method for class 'bg_GLM'
rstandard(model, type = c("sd.1", "predictive"), ...)

## S3 method for class 'bg_GLM'
rstudent(model, ...)

## S3 method for class 'bg_GLM'
hatvalues(model, ...)

## S3 method for class 'bg_GLM'
cooks.distance(model, ...)

dffits.bg_GLM(model)

## S3 method for class 'bg_GLM'
dfbeta(model, ...)

## S3 method for class 'bg_GLM'
dfbetas(model, ...)

covratio.bg_GLM(model)

## S3 method for class 'bg_GLM'
influence(model, do.coef = TRUE, region = NULL, ...)
```
 Arguments

- **model**: A `bg_GLM` object
- **type**: The type of standardized residuals. Default: 'sd.1'
- **do.coef**: Unused
- **region**: Character string of the region(s) to return results for. Default is to calculate for all regions

 Details

The `influence` method calculates all diagnostics present in `lm.influence` and `influence.measures`, consisting of the following functions:

- **rstandard**: Standardized residuals. Choosing `type='predictive'` returns leave-one-out cross validation residuals. The "PRESS" statistic can be calculated as `colSums(resids.p^2)`
- **rstudent**: Studentized residuals
- **hatvalues**: The leverage, or the diagonal of the `hat/projection matrix`
- **cooks.distance**: Cook's distance
- **dffits.bg_GLM**: The change in fitted values when deleting observations
- **dfbeta**: The change in parameter estimates (coefficients) when deleting observations
- **dfbetas**: The scaled change in parameter estimates
- **covratio.bg_GLM**: The covariance ratios, or the change in the determinant of the covariance matrix of parameter estimates when deleting observations

 Value

Most influence functions return a numeric matrix in which rownames are Study ID's and column names are regions. `dfbeta` and `dfbetas` return a numeric array in which each column is a parameter estimate and the 3rd dimension is for each region. `influence` returns a list with class `infl.bg_GLM` and elements:

- **infmat**: Numeric array (like `dfbeta`) with DFBETAs, DFFITs, covratios, Cook's distance, and hat values
- **is.inf**: Logical array of the same data as `infmat`; values of `TRUE` indicate the subject-variable-region combination is an outlier value
- **f**: The model `formula`
- **sigma**: The leave-one-out residual standard deviation
- **wt.res**: Model residuals

 Outlier values

Each variable has a different criterion for determining outliers. In the following: `x` is the influence variable (for `dfbeta`, the criterion applies to all `dfbetas`); `k` is the number of columns of the design matrix; `dfR` is the residual degrees of freedom; and `n` is the number of observations.
GLM model selection

DFBETAs  If $|x| > 1$

DFFITs  If $|x| > 3\sqrt{k/dfR}$

covratio  If $|1 - x| > (3k/dfR)$

cook  If $F_{k,dfR}(x) > 0.5$

hat  If $x > 3k/n$

The return object of influence has a print method which will list the subjects/variables/regions for which an outlier was detected.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

GLM

GLM model selection  Model selection for bg_GLM objects

Description

These functions compute the log-likelihood and Akaike’s An Information Criterion (AIC) of a bg_GLM object. See logLik.lm and extractAIC for details.

Usage

## S3 method for class 'bg_GLM'
logLik(object, REML = FALSE, ...)

## S3 method for class 'bg_GLM'
extractAIC(fit, scale = 0, k = 2, ...)

Arguments

object, fit  A bg_GLM object

REML  Logical indicating whether to return the restricted log-likelihood. Default: FALSE

...  Unused

scale  Should be left at its default

k  Numeric; the weight of the equivalent degrees of freedom

Details

The functions AIC and BIC will also work for bg_GLM objects because they each call logLik.
GLM statistics

Value

logLik returns an object of class logLik with several attributes. extractAIC returns a numeric vector in which the first element is the equivalent degrees of freedom and the remaining are the AIC’s for each region.

GLM statistics Extract model fit statistics from a bg_GLM object

Description

These functions extract or calculate model fit statistics of a bg_GLM object. These can be found in the output from summary.lm.

Usage

## S3 method for class 'bg_GLM'
coef(object, ...)

## S3 method for class 'bg_GLM'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'bg_GLM'
fitted(object, ...)

## S3 method for class 'bg_GLM'
residuals(object, type = c("response", "partial"), ...)

## S3 method for class 'bg_GLM'
deviance(object, ...)

coeff_determ(object, adjusted = FALSE)

## S3 method for class 'bg_GLM'
df.residual(object, ...)

## S3 method for class 'bg_GLM'
sigma(object, ...)

## S3 method for class 'bg_GLM'
vcov(object, ...)

coeff_table(object, CI = FALSE, level = 0.95)

## S3 method for class 'bg_GLM'
anova(object, region = NULL, ...)

Arguments

object  A bg_GLM object
...  Unused
parm  Vector of parameters to calculate confidence intervals for. Default is to use all parameters
level  The confidence level. Default: 0.95
type  Character string specifying the type of residuals to return. Default: 'response'
adjusted  Logical indicating whether to calculate the adjusted R-squared. Default: FALSE
CI  Logical indicating whether to include confidence intervals of parameter estimates in the coefficient summary table. Default: FALSE
region  Character vector indicating the region(s) to calculate ANOVA statistics for. Default: NULL (use all regions)

Details

These mimic the same functions that operate on lm objects, and include:

* coef  Regression coefficients (parameter estimates)
* confint  Confidence intervals (by default, 95%) for parameter estimates
* fitted  Fitted (mean) values; i.e., the design matrix multiplied by the parameter estimates, $X\hat{\beta}$
* residuals  Model residuals; i.e., the response/outcome variable minus the fitted values. Partial residuals can also be calculated
* deviance  Model deviance, or the residual sum of squares
* coeff_determ  Calculate the coefficient of determination (or $R^2$), adjusted or unadjusted
* df.residual  Residual degrees of freedom
* sigma  Residual standard deviation, sometimes called the root mean squared error (RMSE)
* vcov  Variance-covariance matrix of the model parameters

coeff_table returns model coefficients, standard errors, T-statistics, and P-values for all model terms and regions in a bg_GLM object. This is the same as running summary(x)$coefficients for a lm object.

Value

A named numeric vector, matrix, or array, depending on the function:

* coef  Matrix in which rownames are parameter names and column names are regions
* fitted,residuals  Matrix in which rownames are Study ID’s and column names are regions. If type='partial', an array is returned in which columns are terms and the 3rd dimension are regions
* deviance,coeff_determ,sigma  Numeric vector with elements for each region
* df.residual  Single integer; the degrees of freedom
ANOVA tables

The `anova` method calculates the so-called Type III test statistics for a `bg_GLM` object. These standard ANOVA statistics include: sum of squares, mean squares, degrees of freedom, F statistics, and P-values. Additional statistics calculated are: $\eta^2$, partial $\eta^2$, $\omega^2$, and partial $\omega^2$ as measures of effect size.

Note

`sigma` – The denominator is not the number of observations, but rather the model’s residual degrees of freedom.

When calculating partial residuals, the parameter estimates are not re-calculated after removing one of the model terms.

Author(s)

Christopher G. Watson, `<cgwatson@bu.edu>`

See Also

`GLM`, `Anova`

---

**Description**

`graph_attr_dt` is a helper function that takes a `brainGraphList` or a list of graphs and creates a `data.table` of global measures for each graph. Each row will be for a different graph.

`vertex_attr_dt` is a helper function that creates a `data.table` in which each row is a vertex and each column is a different network measure (degree, centrality, etc.).

**Usage**

`graph_attr_dt(bg.list)`

`vertex_attr_dt(bg.list)`

**Arguments**

`bg.list` A `brainGraphList` object, or a list of graph objects
Graph Distances

Value
A data.table

See Also

graph_attr, graph_attr_names
vertex_attr, vertex_attr_names, graph_from_data_frame

Graph Distances

Calculate Euclidean distance of edges and vertices

Description

data.spatial_dist calculates the Euclidean distance of an igraph graph object's edges. The distances are in mm and based on MNI space. These distances are NOT along the cortical surface, so can only be considered approximations, particularly concerning inter-hemispheric connections. The input graph must have atlas as a graph-level attribute.

data.spatial_dist calculates, for each vertex of a graph, the average Euclidean distance across all of that vertex's connections.

Usage

data.spatial_dist(g)

data.spatial_dist(g)

Arguments

g An igraph graph object

Value

data.spatial_dist - a numeric vector with length equal to the edge count of the input graph, consisting of the Euclidean distance (in mm) of each edge

data.spatial_dist - a named numeric vector with length equal to the number of vertices, consisting of the average distance (in mm) for each vertex

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References

hubness

Calculate vertex hubness

Description

hubness calculates the “hubness” (see reference) of the vertices in a graph. These are vertices which meet at least two of the following four criteria:

1. Have high degree/strength
2. Have high betweenness centrality
3. Have low clustering coefficient
4. Have low average path length

For each criterion, “high” or “low” means “in the top 20%” across all vertices. Vertices meeting any of the criteria get a value of 1 for that metric; these are summed to yield the hubness score which ranges from 0-4. As in the reference article, vertices with a score of 2 or higher are to be considered hubs, although that determination isn’t made in this function.

Usage

hubness(g, xfm.type = g$xfm.type, weights = NULL, prop.keep = 0.2)

Arguments

g An igraph graph object
xfm.type Character string specifying how to transform the weights. Default: 1/w
weights Numeric vector of edge weights; if NULL (the default), and if the graph has edge attribute weight, then that will be used. To avoid using weights, this should be NA.
prop.keep Numeric (between 0 and 1) indicating the proportion of vertices to consider as having a high score. Default: 0.2 (20%)

Value

A numeric vector with the vertices’ hubness score

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References

import_scn

Import data for structural connectivity analysis

Description

Given a directory, atlas name, and imaging modality/structural metric, this function imports data for structural connectivity analysis. It expects files containing a table of region-wise structural MRI measures (e.g., mean cortical thickness), with one file for each hemisphere. The first column of all files should contain the subject ID; the column name will be changed to the value of getOption('bg.subject_id').

Usage

import_scn(datadir, atlas, modality = "thickness", exclude.subs = NULL, custom.atlas = NULL)

Arguments

datadir  The path name of the directory containing the data files
atlas    Character string specifying the atlas in use. For a custom atlas, please specify 'custom', and provide the name to the custom.atlas argument
modality The structural imaging measure (default: 'thickness')
exclude.subs  Vector indicating the subjects to exclude, if any (default: NULL)
custom.atlas  Character string specifying the name of the R object for the atlas in use, if atlas='custom' was also supplied (default: NULL)

Details

The files should have specific names; the second in the following list is only required for atlases/parcellations that include subcortical gray matter (e.g., dk.scgm).

- \${parcellation}\_${hemi}\_${modality}.csv for cortical volume, thickness, surface area, or local gyrification index (LGI). Here, \${parcellation} can be aparc, aparc.DKTatlas40, or aparc.a2009s. For example, for cortical thickness with the Desikan-Killiany atlas, the filename should be aparc_lh_thickness.csv. If you are using a custom atlas, see the Note below. The \${hemi} variable is either lh or rh. Finally, \${modality} should be either volume, thickness, area, or lgi.
- asegstats.csv for SCGM volume

Value

A list containing:

atlas    Character string
modality Character string
lhrh     A data.frame of structural MRI measures for both hemispheres
**IndividualContributions**

`aseg`  
A data.table of structural MRI measures for subcortical gray matter, if applicable

`subs.excluded`  
Vector of subject ID’s that were excluded

`subs.missing`  
Vector of subject ID’s that are not present in both the cortical and subcortical tables (if applicable)

**Note**

When using a custom atlas, the name of the atlas’s data.table should match the `${parcellation}` portion of the filename (specification shown above). Furthermore, it must conform to the output of Freesurfer’s `aparcstats2table` (and `asegstats2table`, if applicable). Otherwise, please contact me for inclusion of a different data type.

The subject ID column will be zero-padded (to the left) to avoid issues when the variable is numeric; this ensures that all ID’s will have the same number of characters and sorting will be done properly.

**Author(s)**

Christopher G. Watson, `<cgwatson@bu.edu>`

**See Also**

Other Structural covariance network functions: `Bootstrapping`, `IndividualContributions`, `Residuals`, `brainGraph_permute`, `corr.matrix`, `plot_volumetric`

**Examples**

```r
## Not run:
raw_data <- import_scn('/home/cwatson/data', atlas='dkt',
                        exclude.subs=c('con07', 'con23', 'pat15'))

## End(Not run)
```

---

**IndividualContributions**

*Approaches to estimate individual network contribution*

**Description**

`loo` calculates the individual contribution to group network data for each subject in each group using a “leave-one-out” approach. The residuals of a single subject are excluded, and a correlation matrix is created. This is compared to the original correlation matrix using the Mantel test.

`aop` calculates the individual contribution using an “add-one-patient” approach. The residuals of a single patient are added to those of a control group, and a correlation matrix is created. This is repeated for all individual patients and each patient group.

The `summary` method prints the group/region-wise means and standard deviations.

The `plot` method is only valid for regional contribution estimates, and plots the average regional contribution for each vertex/region.
Usage

```r
loo(resids, corrs, level = c("global", "regional"))
```

```r
aop(resids, corrs, level = c("global", "regional"), control.value = 1L)
```

```r
## S3 method for class 'IC'
summary(object, region = NULL, digits = max(3L,
    getOption("digits") - 2L), ...)
```

```r
## S3 method for class 'IC'
plot(x, plot.type = c("mean", "smooth", "boxplot"),
    region = NULL, ids = TRUE, ...)
```

Arguments

- `resids` An object of class `brainGraph_resids` (the output from `get.resid`)
- `corrs` List of lists of correlation matrices (as output by `corr.matrix`).
- `level` Character string; the level at which you want to calculate contributions (either `global` or `regional`)
- `control.value` Integer or character string specifying the control group (default: 1L)
- `object, x` A `IC` object
- `region` Character vector specifying which regions' IC's to print. Only relevant if `method='Leave one out'
- `digits` Integer specifying the number of digits to display for P-values
- `...` Unused
- `plot.type` Character string indicating the type of plot; the default is to plot the mean (along with standard errors)
- `ids` Logical indicating whether to plot Study ID's for outliers. Otherwise plots the integer index

Value

A `data.table` with columns for

- `Study.ID` Subject identifier
- `Group` Group membership
- `region` If `level='regional'
- `IC,RC` The value of the individual/regional contributions

Note

For `aop`, it is assumed by default that the control group is the first group.

Author(s)

Christopher G. Watson, `<cgwatson@bu.edu>`
References


See Also

Other Structural covariance network functions: `Bootstrapping.Residuals`, `brainGraph_permute`, `corr.matrix`, `import_scn`, `plot_volumetric`

Examples

```r
## Not run:
IC <- loo(resids.all, corrs)
RC <- loo(resids.all, corrs, level='regional')

## End(Not run)
## Not run:
IC <- aop(resids.all, corrs)
RC <- aop(resids.all, corrs, level='regional')

## End(Not run)
```

Inverse

*Calculate the inverse of the cross product of a design matrix*

Description

`inv` is a S3 generic that calculates the inverse of the cross product of a design matrix, also referred to as the “unscaled covariance matrix”.

`pinv` calculates $M^+ = (M^T M)^{-1} M^T$ for full (column) rank matrices. However, it does not verify the matrix's rank.

Usage

```r
deprecated()
inv(x, ...)

## S3 method for class 'matrix'
inv(x, y = NULL, transpose = FALSE, ...)

## S3 method for class 'array'
inv(x, y = NULL, transpose = FALSE, ...)

## S3 method for class 'qr'
inv(x, p = x$rank, ...)

## S3 method for class 'list'
```
\texttt{inv(x, p = x[[1L]]$rank, r = length(x),
  vnames = dimnames(x[[1L]]$qr)[[2L]], nms = names(x), ...)

\texttt{pinv(x)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} A numeric matrix or array, a \texttt{qr} object, or a list of \texttt{qr} objects
  \item \texttt{...} \hspace{1cm} Unused
  \item \texttt{y} \hspace{1cm} A numeric matrix or vector (for the \texttt{matrix} and \texttt{array} methods). If supplied, this will be multiplied by \texttt{x} before the inverse is calculated. Default: \texttt{NULL}
  \item \texttt{transpose} \hspace{1cm} Logical. If \texttt{FALSE} (the default), take the cross product of the arguments. If \texttt{TRUE}, use \texttt{tcrossprod}
  \item \texttt{p} \hspace{1cm} The rank of the original matrix
  \item \texttt{r} \hspace{1cm} The number of design matrices; i.e., the length of the input list
  \item \texttt{vnames} \hspace{1cm} Character vector of the design matrix’s variable names
  \item \texttt{nms} \hspace{1cm} The region names; i.e., the names of the input list
\end{itemize}

\textbf{Details}

If \texttt{x} is a matrix, the Cholesky decomposition of the cross product is calculated (or using \texttt{tcrossprod} if \texttt{transpose=TRUE}), and the inverse is calculated from that result. That is,

\[
\text{inv}(X) = (X^T X)^{-1}
\]

\[
\text{inv}(X, \text{transpose} = \text{TRUE}) = (XX^T)^{-1}
\]

\[
\text{inv}(X, y) = (X^T y)^{-1}
\]

If \texttt{x} is a 3-dimensional array, then the inverse will be calculated for each matrix along the 3rd dimension, with the same input arguments for each.

Finally, there is a method for objects with class \texttt{qr}, and lists of QR decomposition objects.

\textbf{Value}

A numeric matrix or array

\texttt{pinv} returns the input matrix’s pseudoinverse

\textbf{Note}

These methods should only be used on full-rank matrices, as there is no error checking being performed.
**make_auc_brainGraph**  
*Calculate the AUC across densities of given attributes*

**Description**

Given a list of `brainGraphList` objects, this function will calculate the area under the curve (AUC) across all thresholds/densities for each subject or group.

**Usage**

```r
make_auc_brainGraph(g.list, g.attr = NULL, v.attr = NULL, norm = FALSE)
```

**Arguments**

- `g.list`: A list of `brainGraphList` objects
- `g.attr`: A character vector of graph attribute name(s). Default: `NULL`
- `v.attr`: A character vector of vertex attribute name(s). Default: `NULL`
- `norm`: Logical indicating whether to normalize threshold values to be between 0 and 1 (inclusive). Default: `FALSE`

**Details**

If the elements of the input list do not have a `threshold` element (or if it is `NULL`) then the AUC will be calculated on the interval \([0, 1]\) (inclusive). This has the same effect as specifying `norm=TRUE` in the function call.

**Value**

A `brainGraphList` object with one graph for each subject

**Examples**

```r
## Not run:
g.auc <- make_auc_brainGraph(g.fa, g.attr='E.global.wt')
## End(Not run)
```
make_ego_brainGraph  Create a graph of the union of multiple vertex neighborhoods

Description

This function accepts multiple vertices, creates graphs of their neighborhoods (of order 1), and returns the union of those graphs.

Usage

make_ego_brainGraph(g, vs)

Arguments

g An igraph graph object
vs Either a character or integer vector (vertex names or indices, respectively) for the vertices of interest

Value

An igraph graph object containing the union of all edges and vertices in the neighborhoods of the input vertices; only the vertex attribute name will be present

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

ego

Other Graph creation functions: Creating_Graphs_GLM, Creating_Graphs, brainGraphList

Examples

## Not run:
subg <- make_ego_brainGraph(g1[[N]], c(24, 58))
subg <- make_ego_brainGraph(g1[[N]], c('lPCUN', 'rPCUN'))
## End(Not run)
make_intersection_brainGraph

Create the intersection of graphs based on a logical condition

Description

Returns a graph object with vertices that meet certain criteria. By default, only vertices that meet these criteria for all input graphs will be retained.

Usage

make_intersection_brainGraph(..., subgraph, keep.all.vertices = FALSE)

Arguments

... Graph objects or lists of graph objects
subgraph Character string specifying an equation (logical condition) for the vertices to subset
keep.all.vertices Logical indicating whether to keep all vertices that meet the criteria in at least 1 input graph. Default: FALSE

Details

If no vertices meet criteria for all input graphs, then an igraph graph object with 0 vertices is returned. If keep.all.vertices=TRUE, this is essentially performing a union of vertex sets that meet the criteria. In any case, the return graph will have 0 edges.

Value

An igraph graph object

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

Examples

## Not run:
res.mtpc <- mtpc(g, covars, ...)
g.mtpc <- make_glm_brainGraph(res.mtpc, atlas)

## All vertices with a significant MTPC result for all contrasts:
g.mtpc.int <- make_intersection_brainGraph(g.mtpc, subgraph='sig == 1')

## Return graphs with vertices with degree > 0 for each group separately
tapply(g.list, groups(g.list), make_intersection_brainGraph, subgraph='degree > 0')
Matrix utilities

Matrix/array utility functions

Description

These functions are utility/helper functions when working with matrices or arrays.

**diag_sq** is a pared-down version of **diag** for square matrices. It does not return any dimnames, does not check if x is a square matrix, and it cannot be used to create a matrix with a given value along the diagonal. Meant to be used in code that is called repeatedly (thousands of times).

**get_thresholds** calculates the threshold values that would result in a specific graph density. These depend, necessarily on the values in the matrix themselves.

**qr.array** will calculate the QR decomposition for each matrix in a 3D array.

**qr.Q2** and **qr.R2** are simplified versions of **qr.Q** and **qr.R**.

**symm_mean** returns a symmetric matrix in which the off-diagonal elements \(A[i,j]\) and \(A[j,i]\) are set to the mean of the values in the input matrix.

**symmetrize** will symmetrize a numeric matrix (or each matrix in an array) by assigning to the off-diagonal elements either the max (default), min, or average of \(\{A(i,j), A(j,i)\}\).

Usage

```r
colMax(x, n = dim(x)[1L])
colMaxAbs(x, n = dim(x)[1L])
colMin(x, n = dim(x)[1L])
diag_sq(x, n = dim(x)[1L], inds = 1L + 0L:(n - 1L) * (n + 1L))
get_thresholds(x, densities, emax = dim(x)[1L] * (dim(x)[1L] - 1L)/2, ...)
is_binary(x)
```

## S3 method for class 'array'

```r
qr(x, ...)
```

```r
qr.Q2(QR, y = diag(1, n, p), n = dim(QR$qr)[1L], p = QR$rank)
```

```r
qr.R2(QR, p = QR$rank)
```

```r
symm_mean(x)
```
symmetrize(x, ...)

## S3 method for class 'matrix'
symmetrize(x, symm.by = c("max", "min", "avg"), ...)

## S3 method for class 'array'
symmetrize(x, symm.by = c("max", "min", "avg"), ...)

### Arguments

- **x**
  - Numeric matrix or array (the latter, for qr.array and symmetrize.array)
- **n, p**
  - Integer; the number of rows or rank (respectively) of the input matrix or QR decomposition
- **inds**
  - Vector-based indices of the diagonal
- **densities**
  - Numeric vector of densities
- **emax**
  - Integer; the maximum number of edges
- **...**
  - Arguments passed to either sort (for get_thresholds) or qr.default (for qr.array). For the former, this will typically only be decreasing=TRUE, if that is the desired behavior
- **QR**
  - A qr object
- **y**
  - A numeric matrix with 1 along the diagonal, of the same size as the input matrix (i.e., QR$qr
- **symm.by**
  - Character string; how to create symmetric off-diagonal elements. Default: max

### Details

Given a vector of densities, get_thresholds returns the numeric values that will result in graphs of the given densities after thresholding by those values. In the Examples section, the thresholds should result in graphs with densities of 5, 15, ..., 55 percent.

### Value

diag_sq returns an unnamed numeric vector with the values along the diagonal of the input matrix
get_thresholds returns a numeric vector of the thresholds
is_binary returns a logical of length 1
qr.array returns a list in which each element is the QR decomposition of each matrix along x’s 3rd dimension

### Examples

```r
x <- matrix(runif(25 * 25), 25, 25)
x <- symmetrize(x)
diag(x) <- 0
densities <- seq(0.05, 0.55, by=0.1)
thresholds <- get_thresholds(x, densities)
## Verify that the densities are correct
```
graphs <- lapply(threshes, function(th) {
  graph_from_adjacency_matrix(x * (x > th), mode='undirected',
    diag=FALSE, weighted=TRUE)
})
sapply(graphs, graph.density)

---

**mean_distance_wt**

Calculate weighted shortest path lengths

**Description**

Calculate graph or vertex average shortest path lengths. For vertices, this is just the row means of the distance matrix. For the graph-level, it is the overall mean of the distance matrix.

**Usage**

```r
mean_distance_wt(g, level = c("graph", "vertex"), weights = NULL,
  xfm = FALSE, xfm.type = NULL, D = NULL)
```

**Arguments**

- `g`: An igraph graph object
- `level`: Character string indicating whether to calculate vertex- or graph-level shortest path length. Default: 'graph'
- `weights`: Numeric vector of edge weights; if NULL (the default), and if the graph has edge attribute weight, then that will be used. To avoid using weights, this should be NA.
- `xfm`: Logical indicating whether to transform the edge weights. Default: FALSE
- `xfm.type`: Character string specifying how to transform the weights. Default: 1/w
- `D`: Numeric matrix; the graph’s “distance matrix”

**Details**

By default, edge weights are not transformed (e.g., inverted). However, if set to TRUE, then the input graph must have a graph-level attribute called 'xfm.type' or you must supply a value in the function call. If you supply a distance matrix (the D argument), it is not necessary to transform edge weights, as it is assumed the the distance matrix was calculated from a graph with transformed edge weights already.

**Value**

Numeric vector (if level='vertex') of each vertex’s shortest path length, or a single number for the graph-level average
Mediation analysis with brain graph measures as mediator variables

Description

brainGraph_mediate performs simple mediation analyses in which a given graph- or vertex-level measure (e.g., weighted global efficiency) is the mediator $M$. The outcome (or dependent/response) variable $Y$ can be a neuropsychological measure (e.g., IQ) or can be a disease-specific metric (e.g., recovery time).

bg_to_mediate converts the results into an object of class mediate. In brainGraph, it is only used for the summary.mediate method, but you can similarly use its output for the plot.mediate method.

Usage

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

## S3 method for class 'bg_mediate'
summary(object, mediate = FALSE, region = NULL, digits = max(3L, getOption("digits") - 2L), ...)

bg_to_mediate(x, region = NULL)

Arguments

g.list A brainGraphList object
covars A data table containing covariates of interest. It must include columns for getOption("bg.subject_id"), treat, outcome, and covar.names.
mediator Character string; the name of the graph measure acting as the mediating variable
treat Character string; the treatment variable (e.g., Group)
outcome Character string; the name of the outcome variable of interest
covar.names Character vector of the column name(s) in covars to include in the models as pre-treatment covariate(s).
level Character string; either vertex (default) or graph
control.value Value of treat to be used as the control condition. Default: 0
treat.value Value of treat to be used as the treatment condition. Default: 1
int Logical indicating whether or not to include an interaction of the mediator and treatment. Default: FALSE
boot Logical indicating whether or not to perform bootstrapping. This should always be done. Default: TRUE
boot.ci.type: Character string; which type of CI's to calculate. Default: perc

N: Integer; the number of bootstrap samples to run. Default: 1e3

conf.level: Numeric between 0 and 1; the level of the CI's to calculate. Default: 0.95 for the 2.5 and 97.5 percentiles)

long: Logical indicating whether or not to return all bootstrap samples. Default: FALSE

...: Other arguments passed to brainGraph_GLM_design (e.g., binarize) (unused in the summary method)

object: A bg_mediate object

mediate: Logical indicating whether or not to use the summary method from mediate (default: FALSE). If TRUE, only a single region can be printed.

region: Character string specifying which region's results to summarize; only relevant if level='vertex' (default: NULL)

digits: Integer specifying the number of digits to display for P-values

x: Object output from brainGraph_mediate

Details

This code was adapted closely from mediate in the mediation package, and the procedure is exactly the same as theirs (see the references listed below). If you use this function, please cite their work.

Value

An object of class bg_mediate with elements:

level: Either graph or vertex.

removed.subs: A character vector of Study.ID's removed due to incomplete data

X.m, X.y: Design matrix and numeric array for the model with the mediator as the outcome variable (X.m) and for the model with the mediator as an additional predictor (X.y), respectively

y.m, y.y: Outcome variables for the associated design matrices above. y.m will be a matrix of size # subj. X # regions

res.obs: A data.table of the observed values of the point estimates.

res.ci: A data.table of the confidence intervals for the effect estimates.

res.p: A data.table of the two-sided p-values for the effect estimates

boot: Logical, the boot argument.

boot.ci.type: Character string indicating which type of bootstrap confidence intervals were calculated.

res.boot: A data.table with N rows of the bootstrap results for all effects.

treat: Character string of the treatment variable.

mediator: Character string of the mediator variable.

outcome: Character string of the outcome variable.
covariates Returns NULL; not used in this package.

INT Logical indicating whether the models included an interaction between treatment and mediator.

conf.level The confidence level.

control.value The value of the treatment variable used as the control condition.

treat.value The value of the treatment variable used as the treatment condition.

nobs Integer; the number of observations in the models.

sims Integer; the number of bootstrap replications.

covar.names The pre-treatment covariate names.

bg_to_mediate returns an object of class mediate

Note

As of brainGraph v2.0.0, this function has been tested only for a treatment (independent) variable \( X \) being a factor (e.g., disease group, old vs. young, etc.). If your treatment variable has more than 2 levels, then you must explicitly specify the levels you would like to compare; otherwise, the baseline and first levels are taken to be the control and treatment values, respectively. Be aware that these are 0 indexed; that is, if you have 3 groups and you would like the treatment group to be the 3rd, you should specify as either the group’s character string or as treat.value=2.

Allowing for treatment-mediator interaction (setting int=TRUE) currently will only work properly if the mediator is a continuous variable; since the mediator is always a graph metric, this should always be the case.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References


See Also

mediate

Other Group analysis functions: Bootstrapping, GLM, NBS, brainGraph_permute, mtpc

Examples

```r
## Not run:
med.EglobWt.FSIQ <- brainGraph_mediate(g[[5]], covars.med, 'E.global.wt',
  'Group', 'FSIQ', covar.names=c('age', 'gender'), N=1e4)
med.strength.FSIQ <- brainGraph_mediate(g[[5]], covars.med, 'strength',
  'Group', 'FSIQ', covar.names=c('age', 'gender'), level='vertex')

## End(Not run)
```

mtpc

Multi-threshold permutation correction

Description

Applies the multi-threshold permutation correction (MTPC) method to perform inference in graph theory analyses of brain MRI data.

Plot the statistics from an MTPC analysis, along with the maximum permuted statistics. The output is similar to Figure 11 in Drakesmith et al. (2015).

Usage

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

```
## S3 method for class 'mtpc'
summary(object, contrast = NULL, digits = max(3L,
 getOption("digits") - 2L), print.head = TRUE, ...)
```

```
## S3 method for class 'mtpc'
plot(x, contrast = 1L, region = NULL,
  only.sig.regions = TRUE, show.null = TRUE, caption.stats = FALSE, ...
)
```

```
## S3 method for class 'mtpc'
nobs(object, ...)
```

```
## S3 method for class 'mtpc'
```
### Arguments

- `g.list`: A list of `brainGraphList` objects for all thresholds
- `thresholds`: Numeric vector of the thresholds applied to the raw connectivity matrices.
- `covars`: A `data.table` of covariates
- `measure`: Character string of the graph measure of interest
- `contrasts`: Numeric matrix (for T statistics) or list of matrices (for F statistics) specifying the contrast(s) of interest; if only one contrast is desired, you can supply a vector (for T statistics)
- `con.type`: Character string; either 't' or 'f' (for t or F-statistics). Default: 't'
- `outcome`: Character string specifying the name of the outcome variable, if it differs from the graph metric (`measure`)
- `con.name`: Character vector of the contrast name(s); if `contrasts` has row/list names, those will be used for reporting results
- `level`: Character string; either vertex (default) or graph
- `clust.size`: Integer indicating the size of “clusters” (i.e., consecutive thresholds for which the observed statistic exceeds the null) (default: 3L)
- `perm.method`: Character string indicating the permutation method. Default: 'freedmanLane'
- `part.method`: Character string; the method of partitioning the design matrix into covariates of interest and nuisance. Default: 'beckmann'
- `N`: Integer; number of permutations to create. Default: 5e3
- `perms`: Matrix of permutations, if you would like to provide your own. Default: NULL
alpha Numeric; the significance level. Default: 0.05

res.glm A list of bg_GLM objects, as output by a previous run of mtpc. Useful if you want to change the cluster size without re-running all of the GLM’s and permutations (default: NULL)

long Logical indicating whether or not to return all permutation results. Default: FALSE

... Other arguments passed to brainGraph_GLM and/or brainGraph_GLM_design

object, x A mtpc object

contrast Integer specifying the contrast to plot/summarize; defaults to showing results for all contrasts

digits Integer specifying the number of digits to display for P-values

print.head Logical indicating whether or not to print only the first and last 5 rows of the statistics tables (default: TRUE)

region Character string specifying which region’s results to plot; only relevant if level=’vertex’. Default: NULL

only.sig.regions Logical indicating whether to plot only significant regions (default: TRUE)

show.null Logical indicating whether to plot points of the maximum null statistics (per permutation)

caption.stats Logical indicating whether to print the MTPC statistics in the caption of the plot. Default: FALSE

Details

This is a multi-step procedure: (steps 3-4 are the time-consuming steps)

1. Apply thresholds $\tau$ to the networks, and compute network metrics for all networks and thresholds. (already done beforehand)

2. Compute test statistics $S_{obs}$ for each threshold. (done by brainGraph_GLM)

3. Permute group assignments and compute test statistics for each permutation and threshold. (done by brainGraph_GLM)

4. Build a null distribution of the maximum statistic across thresholds (and across brain regions) for each permutation. (done by brainGraph_GLM)

5. Determine the critical value, $S_{crit}$ from the null distribution of maximum statistics.

6. Identify clusters where $S_{obs} > S_{crit}$ and compute the AUC for these clusters (denoted $A_{MTPC}$).

7. Compute a critical AUC ($A_{crit}$) from the mean of the supra-critical AUC’s for the permuted tests.

8. Reject $H_0$ if $A_{MTPC} > A_{crit}$

Value

An object of class mtpc with some input arguments plus the following elements:
Design matrix, QR decomposition, and unscaled covariance matrix, if the design is the same across thresholds

The contrast matrix or list of matrices

Contrast names

Named integer vector of subjects with incomplete data

The atlas of the input graphs

The model rank and residual degrees of freedom

List with length equal to the number of thresholds; each list element is the output from brainGraph_GLM

A data.table for all thresholds, combined from the outputs of brainGraph_GLM

A data.table containing S.mtpc (the max. observed statistic), tau.mtpc (the threshold of the max. observed statistic), S.crit (the critical statistic value), and A.crit (the critical AUC)

Numeric array with N columns and number of rows equal to the number of thresholds. The 3rd dimension is for each contrast. Each element of the array is the maximum statistic for that permutation, threshold, and contrast combination.

Numeric matrix; the permutation set applied for all thresholds (each row is a separate permutation)

The plot method returns a trellis object or a list of ggplot objects

Christopher G. Watson, <cgwatson@bu.edu>


Other Group analysis functions: Bootstrapping, GLM, Mediation, NBS, brainGraph_permute

Other GLM functions: GLM design, GLM fits, GLM

## Not run:
diffs.mtpc <- mtpc(g.list=g.norm, thresholds=thresholds, N=N, covars=covars.dti, measure="E.nodal.wt", coding='effects', contrasts=c(0, 0, 0, 0, -2), alt='greater', binarize=c('Sex', 'Scanner'), con.name='Group 1 > Group 2')
sig.regions <- diffs.mtpc$DT[A.mtpc > A.crit]
Network-based statistic for brain MRI data

Description

Calculates the network-based statistic (NBS), which allows for family-wise error (FWE) control over network data, introduced for brain MRI data by Zalesky et al. Requires a three-dimensional array of all subjects' connectivity matrices and a data table of covariates, in addition to a contrast matrix or list. A null distribution of the largest connected component size is created by fitting a GLM to permuted data. For details, see GLM.

Usage

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

## S3 method for class 'NBS'
summary(object, contrast = NULL, digits = max(3L, getOption("digits") - 2L), ...)

## S3 method for class 'NBS'
nobs(object, ...)

## S3 method for class 'NBS'
terms(x, ...)

## S3 method for class 'NBS'
formula(x, ...)

## S3 method for class 'NBS'
labels(object, ...)

## S3 method for class 'NBS'
case.names(object, ...)
## S3 method for class 'NBS'
variable.names(object, ...)

## S3 method for class 'NBS'
df.residual(object, ...)

## S3 method for class 'NBS'
nregions(object)

### Arguments
- **A**: Three-dimensional array of all subjects' connectivity matrices
- **covars**: A data.frame of covariates
- **contrasts**: Numeric matrix (for T statistics) or list of matrices (for F statistics) specifying the contrast(s) of interest; if only one contrast is desired, you can supply a vector (for T statistics)
- **con.type**: Character string; either 't' or 'f' (for t or F-statistics). Default: 't'
- **X**: Numeric matrix, if you wish to supply your own design matrix. Ignored if outcome ! = measure.
- **con.name**: Character vector of the contrast name(s); if contrasts has row/list names, those will be used for reporting results
- **p.init**: Numeric; the initial p-value threshold (default: 0.001)
- **perm.method**: Character string indicating the permutation method. Default: 'freedmanLane'
- **part.method**: Character string; the method of partitioning the design matrix into covariates of interest and nuisance. Default: 'beckmann'
- **N**: Integer; number of permutations to create. Default: 5e3
- **perms**: Matrix of permutations, if you would like to provide your own. Default: NULL
- **symm.by**: Character string; how to create symmetric off-diagonal elements. Default: max
- **alternative**: Character string, whether to do a two- or one-sided test. Default: 'two.sided'
- **long**: Logical indicating whether or not to return all permutation results. Default: FALSE
- **...**: Arguments passed to brainGraph_GLM_design
- **object, x**: A NBS object
- **contrast**: Integer specifying the contrast to plot/summarize; defaults to showing results for all contrasts
- **digits**: Integer specifying the number of digits to display for P-values

### Details
When printing a summary, you can include arguments to printCoefmat.
Value

An object of class NBS with some input arguments in addition to:

- **X**: The design matrix
- **removed.subs**: Character vector of subject ID’s removed due to incomplete data (if any)
- **T.mat**: 3-d array of (symmetric) numeric matrices containing the statistics for each edge
- **p.mat**: 3-d array of (symmetric) numeric matrices containing the P-values
- **components**: List containing data tables of the observed and permuted connected component sizes and P-values
- **rank, df.residual, qr, cov.unscaled**: The rank, residual degrees of freedom, QR decomposition, and unscaled covariance matrix of the design matrix

Note

It is assumed that the order of the subjects in covars matches that of the input array A. You will need to ensure that this is the case. Prior to v3.0.0, the covars table was sorted by Study.ID before creating the design matrix.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References


See Also

Other Group analysis functions: Bootstrapping, GLM, Mediation, brainGraph_permute, mtpc

Examples

```r
## Not run:
max.comp.nbs <- NBS(A.norm.sub[[1]], covars.dti, N=5e3)

## End(Not run)
```
plot.brainGraph

Plot a brain graph with a specific spatial layout

Description

plot.brainGraph plots a graph in which the spatial layout of the nodes is important. The network itself is plotted over a brain MRI slice from the MNI152 template by default (when mni=TRUE).

Usage

## S3 method for class 'brainGraph'
plot(x, plane = c("axial", "sagittal", "circular"),
    hemi = c("both", "L", "R"), mni = TRUE, subgraph = NULL,
    main = NULL, subtitle = "default", label = NULL, side = 1,
    line = -2, adj = 0.025, cex = 2.5, col = "white", font = 2,
    show.legend = FALSE, rescale = FALSE, asp = 0, ...)

Arguments

- **x**: A brainGraph graph object
- **plane**: Character string indicating which orientation to plot. Default: 'axial'
- **hemi**: Character string indicating which hemisphere to plot. Default: 'both'
- **mni**: Logical indicating whether or not to plot over a slice of the brain. Default: TRUE
- **subgraph**: Character string specifying a logical condition for vertices to plot. Default: NULL
- **main**: Character string; the main title. Default: NULL
- **subtitle**: Character string; the subtitle. Default: 'default'
- **label**: Character string specifying text to display in one corner of the plot (e.g., 'A.'). Default: NULL
- **side**: Label placement. Default: 1 (bottom)
- **line**: Which margin line to place the text.
- **adj**: If side=1, a value closer to 0 places the text closer to the left margin. Default: 0.025
- **cex**: Amount of character expansion of the label text. Default: 2.5
- **col**: Label font color. Default: 'white'
- **font**: Integer specifying the font type. Default: 2 (bold face)
- **show.legend**: Logical indicating whether or not to show a legend. Default: FALSE
- **rescale**: Logical, whether to rescale the coordinates. Default: FALSE
- **asp**: Numeric constant; the aspect ratio. Default: 0
- **...**: Other parameters (passed to plot.igraph). See plot.common for details.
Selecting specific vertices to display

With the argument subgraph, you can supply a simple logical expression specifying which vertices to show. For example, 'degree > 10' will plot only vertices with a degree greater than 10. Combinations of AND (i.e., &amp;) and OR (i.e., |) are allowed. This requires that any vertex attribute in the expression must be present in the graph; e.g., V(g)$degree must exist.

Title, subtitle, and label

By default, a title (i.e., text displayed at the top of the figure) is not included. You can include one by passing a character string to main, and control the size with cex.main. A subtitle (i.e., text at the bottom), is included by default and displays the number of vertices and edges along with the graph density. To exclude this, specify subtitle=NULL. A “label” can be included in one corner of the figure (for publications). For example, you can choose label=’A.’ or label=’a’). Arguments controlling the location and appearance can be changed, but the default values are optimal for bottom-left placement. See mtext for more details. The label-specific arguments are:

- side The location. 1 is for bottom placement.
- line If side=1 (bottom), a negative number places the text above the bottom of the figure; a higher number could result in the bottom part of the text to be missing. This can differ if plane=’circular’, in which case you may want to specify a positive number.
- adj Seems to be the percentage away from the margin. So, for example, adj=0.1 would place the text closer to the center than the default value, and adj=0.5 places it in the center.
- cex The degree of “character expansion”. A value of 1 would not increase the text size.
- col The text color.
- font The font type. The default font=2 is bold face. See par for details.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

Other Plotting functions: Plotting GLM graphs, plot.brainGraphList, plot_brainGraph_multi

Examples

```r
## Not run:
plot(g[[1]], hemi='R')
plot(g[[1]], subgraph='degree > 10 &amp; btwn.cent > 50')

## Place label in upper-left
plot(g.ex, label='A', side=3, line=-2.5)

## End(Not run)
```
plot.brainGraphList

Plot a brainGraphList and write to PDF

Description

The `plot` method will write a PDF file containing plots for all graphs in the given object.

Usage

```r
## S3 method for class 'brainGraphList'
plot(x, plane, hemi, filename.base, 
    diffs = FALSE, ...)
```

Arguments

- `x`: A `brainGraphList` object
- `plane`: Character string indicating which orientation to plot. Default: 'axial'
- `hemi`: Character string indicating which hemisphere to plot. Default: 'both'
- `filename.base`: Character string specifying the base of the filename
- `diffs`: Logical, indicating whether edge differences should be highlighted. Default: FALSE
- `...`: Other parameters (passed to `plot.brainGraph`)

Details

You can choose to highlight edge differences between subsequent list elements; in this case, new/different edges are colored pink. This is useful mostly for a list of group-level graphs.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

Other Plotting functions: `Plotting GLM graphs, plot.brainGraph, plot_brainGraph_multi`
Plotting GLM graphs

Plot a graph with results from GLM-based analyses

Description

These methods are convenience functions for plotting a graph based on results from GLM-based analyses (i.e., `brainGraph_GLM`, `brainGraph_mediate`, `mtpc`, `NBS`). There are several default arguments which differ depending on the input object.

Usage

```r
## S3 method for class 'brainGraph_NBS'
plot(x, alpha = 0.05,
     subgraph = paste("p.nbs >", 1 - alpha), vertex.label = NA,
     vertex.color = "color.comp", edge.color = "color.comp",
     subtitle = NULL, main = paste0("NBS: ", x$name), cex.main = 2, ...)

## S3 method for class 'brainGraph_GLM'
plot(x, p.sig = c("p", "p.fdr", "p.perm"),
     subgraph = NULL, main = paste0(x$outcome, ": ", x$name),
     subtitle = NULL, cex.main = 2, ...)

## S3 method for class 'brainGraph_mtpc'
plot(x, subgraph = "sig == 1",
     main = paste0(x$outcome, ": ", x$name), subtitle = NULL,
     cex.main = 2, ...)

## S3 method for class 'brainGraph_mediate'
plot(x, subgraph = "p.acme > 0.95",
     main = sprintf("Effect of \"%s\" on \"%s\" mediated by \"%s\"",
                    x$treat, x$outcome, x$mediator), subtitle = NULL,
     cex.main = 1, ...)
```

Arguments

- **x**: A `brainGraph_GLM`, `brainGraph_mtpc`, `brainGraph_mediate`, or `brainGraph_NBS` object
- **alpha**: Numeric; the significance level. Default: 0.05
- **subgraph**: Character string specifying the condition for subsetting the graph.
- **vertex.label**: Character vector of the vertex labels to be displayed.
- **vertex.color**: Character string specifying the vertex attribute to color the vertices by.
- **edge.color**: Character string specifying the edge attribute to color the edges by.
- **subtitle**: Character string; the subtitle. Default: 'default'
- **main**: Character string; the main title. Default: NULL
- **cex.main**: Numeric; the scaling factor for text size; see `par`
Other arguments passed to `plot.brainGraph`

p.sig
Character string indicating which p-value to use for determining significance (default: p)

Details

The default arguments are specified so that the user only needs to type `plot(x)` at the console, if desired. For all methods, the plot’s subtitle will be omitted.

NBS

By default, a subgraph will be plotted consisting of only those vertices which are part of a significant connected component. Vertex/edge colors will correspond to connected component membership. Vertex names will be omitted. Finally, the plot title will contain the contrast name.

brainGraph_GLM

By default, a subgraph will be plotted consisting of only those vertices for which $p < \alpha$. It will also include a plot title with the outcome measure and contrast name.

mtpc

By default, a subgraph will be plotted consisting of only those vertices for which $A_{\text{mtpc}} > A_{\text{crit}}$. It will also include a plot title with the outcome measure and contrast name.

brainGraph_mediate

By default, a subgraph will be plotted consisting of only those vertices for which $P_{\text{acme}} < \alpha$. It will also include a plot title with the treatment, mediator, and outcome variable names.

See Also

Other Plotting functions: `plot.brainGraphList`, `plot.brainGraph`, `plot_brainGraph_multi`

plot_brainGraph_multi

Save PNG of one or three views for all graphs in a brainGraphList

Description

`plot_brainGraph_multi` writes a PNG file to disk containing three views (columns) of 1 or more `brainGraph` objects (from left-to-right): left sagittal, axial, and right sagittal. The number of rows in the figure will equal the number of graphs to plot.

`slicer` writes a PNG file to disk containing a single view (i.e., either sagittal, axial, or circular) of all `brainGraph` objects in the input list `brainGraphList`. 
plot_brainGraph_multi

Usage

plot_brainGraph_multi(g.list, filename = "orthoview.png",
subgraph = NULL, main = NULL, label = NULL, cex.main = 1, ...)

slicer(g.list, nrows, ncols, plane = "axial", hemi = "both",
filename = "all.png", main = NULL, cex.main = 1, ...)

Arguments

g.list
A brainGraphList or a list of brainGraph objects

filename
Character string of the filename of the PNG to be written.

subgraph
A vector or list of character strings to (optionally) subset the graph(s), possibly by multiple conditions

main
A vector or list of character strings to be placed in the main title of the center (axial) plot for each graph

label
A vector or list of character strings to be placed in one of the corners of the left plot (sagittal) in each row

cex.main
Numeric specifying the level of character expansion for the plot titles. Default: 1 (no expansion)

...
Other arguments passed to plot.brainGraph

nrows
Integer; the number of rows in the figure

ncols
Integer; the number of columns in the figure

plane
Character string indicating which orientation to plot. Default: 'axial'

hemi
Character string indicating which hemisphere to plot. Default: 'both'

Details

Whether the first input is a brainGraphList object or a list of brainGraph objects, all graphs in the object will be displayed in the figure. For plot_brainGraph_multi, this may be undesirable if you have more than 4 or 5 graphs in one object. You can choose fewer by using simple subsetting operations (see Examples below).

Using subgraphs, titles, and labels

There are three arguments that can differ for each graph to be displayed. Each follows the same “rules”. If you would like the same value applied to all graphs, you can specify a character string. If you would like a different value for each group, you must supply a vector or list with length equal to the number of graphs. If its length is less than the number of graphs, values will be recycled. To “skip” applying a value to one (or more) graph(s), you can use the NULL value only within a list (see the Examples below).

subgraph Can be used to apply one or more conditions for subsetting the graph(s).

main Controls the main plot title, which appears in the axial view along with each graph’s name attribute. Depending on the level of the brainGraphList, this will either be a Study ID, Group name, or contrast name.

label Can be used to print a text label in a corner for each group/graph. For example, you can print a letter if you will refer to, e.g., “Figure 1A”, “Figure 1B”, etc.
Note

All other arguments (passed to `plot.brainGraph`) will be applied to all graphs. For example, if you include `vertex.label=NA` in the function call, vertex labels will be omitted for all graphs.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

Other Plotting functions: `Plotting GLM graphs`, `plot.brainGraphList`, `plot.brainGraph`

Examples

```r
## Not run:
## "g.hubs" contains 2 groups; apply same subset to both
plot_brainGraph_multi(g.hubs, filename='Figure01_hubs.png',
  subgraph='N > 0', vertex.color='color.lobe', vertex.size=15,
  show.legend=TRUE, vertex.label.cex=1.5)

## Single group, different subgraphs for both plots
## "g" is a "brainGraphList" object
gg <- g[rep(1, 3), drop=FALSE]
plot_brainGraph_multi(gg, filename='group1_5-6-7core.png',
  vertex.color='color.lobe', edge.color='color.lobe', vertex.label=NA,
  subgraph=as.list(paste('coreness >', 5:7)),
  main=as.list(paste('k-core', 5:7)))

## Apply different subset for groups 1 & 3; no subset for group 2
plot_brainGraph_multi(g, groups=1:3, vertex.label=NA,
  subgraph=list('degree > 5', NULL, 'degree > 4'))
## End(Not run)
```

plot_global

Plot global graph measures across densities

Description

Create a faceted line plot of global graph measures across a range of graph densities, calculated from a list of `brainGraphList` objects. This requires that the variables of interest are graph-level attributes of the input graphs.

Usage

```r
plot_global(g.list, xvar = c("density", "threshold"), vline = NULL,
  level.names = "default", exclude = NULL, perms = NULL,
  alt = "two.sided")
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g.list</td>
<td>List of <code>brainGraphList</code> objects; the length of this list should equal the number of thresholds/densities in the study</td>
</tr>
<tr>
<td>xvar</td>
<td>A character string indicating whether the variable of interest is “density” or “threshold” (e.g. with DTI data)</td>
</tr>
<tr>
<td>vline</td>
<td>Numeric of length 1 specifying the x-intercept if you would like to plot a vertical dashed line (e.g., if there is a particular density of interest). Default: NULL</td>
</tr>
<tr>
<td>level.names</td>
<td>Character vector of variable names, which are displayed as facet labels. If you do not want to change them, specify NULL. By default, they are changed to preset values.</td>
</tr>
<tr>
<td>exclude</td>
<td>Character vector of variables to exclude. Default: NULL</td>
</tr>
<tr>
<td>perms</td>
<td>A <code>data.table</code> of permutation group differences</td>
</tr>
<tr>
<td>alt</td>
<td>Character vector of alternative hypotheses; required if <code>perms</code> is provided, but defaults to “two.sided” for all variables</td>
</tr>
</tbody>
</table>

Details

You can choose to insert a dashed vertical line at a specific density/threshold of interest, rename the variable levels (which become the facet titles), exclude variables, and include a `brainGraph_permute` object of permutation data to add asterisks indicating significant group differences.

Value

Either a `trellis` or `ggplot` object

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

---

**plot_rich_norm**  
*Plot normalized rich club coefficients against degree threshold*

Description

Returns a line plot of the normalized rich club coefficient. Optionally, can include a shaded region demarcating the `rich_core` cutoff (if you supply a list of graph objects to the `g` argument).

Usage

```r
plot_rich_norm(rich.dt, facet.by = c("density", "threshold"), densities, alpha = 0.05, fdr = TRUE, g.list = NULL, smooth = TRUE)
```
plot_vertex_measures

Arguments

- **rich.dt**: A `data.table` with rich-club coefficients
- **facet.by**: A character string indicating whether the variable of interest is “density” or “threshold” (e.g. with DTI data)
- **densities**: A numeric vector of the densities to plot
- **alpha**: The significance level. Default: 0.05
- **fdr**: A logical, indicating whether or not to use the FDR-adjusted p-value for determining significance. Default: TRUE
- **g.list**: A list `brainGraphList` objects; required if you want to plot a shaded region demarcating the `rich_core`
- **smooth**: Logical indicating whether or not to plot a smooth curve when data from multiple subjects (per group) are present. Default: TRUE. Ignored for group-level data.

Value

A trellis or `ggplot` object

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

Other Rich-club functions: `Rich Club`, `rich_club_attrs`

Examples

```r
## Not run:
plot_rich_norm(rich.dt, facet.by='density', densities[N:(N+1)], g=g)
## End(Not run)

plot_vertex_measures(g.list, measure, facet.by = NULL, group.by = getOption("bg.group"), type = c("violin", "boxplot"), show.points = FALSE, ylabel = measure, ...)
```

Description

Creates boxplots of a single vertex-level graph measure at a single density or threshold, grouped by the variable specified by `group.by` and optionally faceted by another variable (e.g., `lobe` or `network`).

Usage

```r
plot_vertex_measures(g.list, measure, facet.by = NULL, group.by = getOption("bg.group"), type = c("violin", "boxplot"), show.points = FALSE, ylabel = measure, ...)
```
plot_volumetric

Description

This function takes a “tidied” dataset of cortical volumetric measures (thickness, volume, LGI, etc.) and plots a histogram or violin plot for 1 or more groups, and of 1 or more brain regions.

Usage

```r
plot_volumetric(dat, regions, type = c("violin", "histogram"),
                all.vals = TRUE, modality = c("thickness", "volume", "lgi", "area"))
```
Perform an analysis with random graphs for brain MRI data

Arguments

dat A data table of volumetric data; needs columns for 'Group', 'region', and 'value'
regions A vector of character strings or integers of the brain region(s) to plot; if integer, the region(s) is/are chosen from the input data table based on the index
type A character string indicating the plot type; either 'histogram' or 'violin'
all.vals A logical indicating whether or not to plot horizontal lines for all observations (only valid for 'violin' plots) (default: TRUE)
modality A character string indicating the type of volumetric measure ('thickness', 'volume', 'lgi', or 'area')

Value

A trellis or ggplot object

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

Other Structural covariance network functions: Bootstrapping, IndividualContributions, Residuals, brainGraph_permute, corr.matrix, import_scn

Description

analysis_random_graphs performs the steps needed for doing typical graph theory analyses with brain MRI data if you need to generate equivalent random graphs. This includes calculating small world parameters and normalized rich club coefficients.

sim.rand.graph.par simulates N simple random graphs with the same clustering (optional) and degree sequence as the input. Essentially a wrapper for sample_degseq (or, if you want to match by clustering, sim.rand.graph.clust) and make_brainGraph. It uses foreach for parallel processing.

sim.rand.graph.clust simulates a random graph with a given degree sequence and clustering coefficient. Increasing the max.iters value will result in a closer match of clustering with the observed graph.

sim.rand.graph.hqs generates a number of random covariance matrices using the Hirschberger-Qi-Steuer (HQS) algorithm, and create graphs from those matrices.
Usage

analysis_random_graphs(g.list, level = g.list[[1L]]$level, N = 100L, savedir = ".", ...)  
sim.rand.graph.par(g, level = c("subject", "group"), N = 100L, clustering = FALSE, rewire.iters = max(10 * ecount(g), 10000L), cl = g$transitivity, max.iters = 100L, ...)  
sim.rand.graph.clust(g, rewire.iters = 10000, cl = g$transitivity, max.iters = 100)  
sim.rand.graph.hqs(resids, level = c("subject", "group"), N = 100L, weighted = TRUE, r.thresh = NULL, ...)

Arguments

g.list List of brainGraphList objects; the length of this list should equal the number of thresholds/densities in the study  
level Character string indicating whether the graphs are subject-, group-, or contrast-specific. Default: 'subject'  
N Integer; the number of random graphs to simulate. Default: 100  
savedir Character string specifying the directory in which to save the generated graphs. Default: current working directory  
... Other arguments passed to make_brainGraph  
g A graph object  
clustering Logical; whether or not to control for clustering. Default: FALSE  
rewire.iters Integer; number of rewiring iterations for the initial graph randomization. Default: 1e4  
cl The clustering measure. Default: transitivity  
max.iters The maximum number of iterations to perform; choosing a lower number may result in clustering that is further away from the observed graph's. Default: 100  
resids A brainGraph_resids object, a data.table of residuals, or a numeric matrix  
weighted Logical indicating whether to create weighted graphs. If true, a threshold must be provided.  
r.thresh Numeric value for the correlation threshold, if weighted=FALSE.

Details

analysis_random_graphs does the following:

1. Generate N random graphs for each graph and density/threshold  
2. Write graphs to disk in savedir. Read them back into R and combine into lists; then write these lists to disk. You can later delete the individual .rds files afterwards.  
3. Calculate small world parameters, along with values for a few global graph measures that may be of interest.

If you do not want to match by clustering, then simple rewiring of the input graph is performed (the number of rewires equaling the larger of 1e4 and 10 \times m, where m is the graph’s edge count).

`sim.rand.graph.hqs` - The first step is to create the observed covariance of residuals (or whatever matrix/data.table is provided). Then random covariance matrices are created with the same distributional properties as the observed matrix, they are converted to correlation matrices, and finally graphs from these matrices. By default, weighted graphs will be created in which the edge weights represent correlation values. If you want binary matrices, you must provide a correlation threshold.

**Value**

`analysis_random_graphs` returns a list containing:

- `rich` A data table containing normalized rich-club coefficients and p-values
- `small` A data table with small-world parameters
- `rand` A data table with some global graph measures for all random graphs generated

`sim.rand.graph.par` - A list of N random graphs with some additional vertex and graph attributes

`sim.rand.graph.clust` - A single igraph graph object

`sim.rand.graph.hqs` - A list of random graphs from the null covariance matrices

**Author(s)**

Christopher G. Watson, <cgwatson@bu.edu>

**References**


**See Also**

- `small.world`
- `rewire`, `sample_degseq`, `keeping_degseq`
- `transitivity`

Other Random graph functions: Rich Club

**Examples**

```r
## Not run:
rand_all <- analysis_random_graphs(g.norm, 1e2, 
   savedir='~/home/cwatson/dti/rand', clustering=F)

## End(Not run)
```
## Not run:
rand1 <- sim.rand.graph.par(g[[1]][[N]], N=1e3)
rand1.cl <- sim.rand.graph.par(g[[1]][[N]], N=1e2,
  clustering=T, max.iters=1e3)
## End(Not run)

randomise

**GLM non-parametric permutation testing**

### Description
randomise and randomise_3d perform non-parametric permutation testing for analyses in which there is a single or multiple design matrix per region, respectively. In the latter case, $X$ should be a 3D array.

partition partitions a full design matrix into separate matrices of covariates of interest and nuisance covariates based on a given contrast and partition method.

### Usage

```r
partition(M, contrast, part.method = c("beckmann", "guttman", "ridgway"))

randomise(perm.method, part.method, N, perms, X, y, contrasts, ctype, nC,
  skip = NULL, n = dim(X)[1L], p = qr.default(X)$rank,
  ny = dim(y)[2L], dfR = n - p)

randomise_3d(perm.method, part.method, N, perms, X, y, contrasts, ctype,
  nC, runX = dimnames(X)[[3L]], n = dim(X)[1L], p = qr.default(X[, ,
  1L])$rank, ny = length(runX), dfR = n - p)
```

### Arguments

- **M**: Numeric matrix or array of the full design matrix(es)
- **contrast**: For partition, a numeric matrix with 1 or more rows (for T and F contrasts, respectively) representing a single contrast.
- **part.method**: Character string; the method of partitioning the design matrix into covariates of interest and nuisance. Default: 'beckmann'
- **perm.method**: Character string indicating the permutation method. Default: 'freedmanLane'
- **N**: Integer; number of permutations to create. Default: 5e3
- **perms**: Matrix of permutations, if you would like to provide your own. Default: NULL
- **X**: Numeric matrix or 3D array of the design matrix(es)
- **y**: Numeric matrix of outcome variables, with 1 column per region, or a single column if there is a different design matrix per region
- **contrasts**: Numeric matrix (for T statistics) or list of matrices (for F statistics) specifying the contrast(s) of interest; if only one contrast is desired, you can supply a vector (for T statistics)
**ctype**  
The contrast type

**nC**  
Integer; the number of contrasts

**skip**  
Integer vector indicating which (if any) contrasts to skip. Only used by NBS.

**n, p, ny, dfR**  
Integers for the number of observations, design matrix columns (its rank), number of regions/outcome variables, and residual degrees of freedom, respectively

**runX**  
Character vector of regions with non-singular designs

**Value**

`partition` returns a list containing:

- **Mp**: Numeric array; the combined partitioned arrays
- **X**: Numeric array for the covariates of interest
- **Z**: Numeric array for the nuisance covariates
- **eCm**: The *effective contrast*, equivalent to the original, for the partitioned model \([X, Z]\) and considering all covariates
- **eCx**: Same as `eCm`, but considering only \(X\)

A numeric array with dimensions \(n_y \times N \times n_c\); the number of rows equals number of regions/outcome variables, number of columns equals \(N\), and the 3rd dimension is the number of contrasts

**Model partitioning**

Consider the matrix formulation of the *general linear model*:

\[
Y = M \psi + \varepsilon
\]

where \(Y\) is the vector of outcomes, \(M\) is the full design matrix (including nuisance covariates), \(\psi\) is the vector of parameter estimates, and \(\varepsilon\) is the vector of error terms. In a permutation framework, algorithms are applied differently depending on the presence/absence of nuisance covariates; thus the model is separated depending on the contrast of interest:

\[
Y = X \beta + Z \gamma + \varepsilon
\]

where \(X\) contains covariates of interest, \(Z\) contains nuisance covariates, and \(\beta\) and \(\gamma\) are the associated parameter estimates.

The manner of partitioning depends on the method. For example, for the guttmann method, \(X\) is formed from the columns of contrast that have non-zero entries.

**Permutation methods**

The permutation methods can be split into 2 groups, depending on which part of the model they permute. For full details, see Winkler et al., 2014.

**Permute Y**  
Freedman-Lane, Manly, and ter Braak

**Permute X**  
Smith, Draper-Stoneman, and Still-White
Depending on the size of the data, it may be faster to use a method that permutes $Y$ instead of $X$. For example, in NBS with dense matrices (more than 400-500 edges), it will be somewhat faster to use the “Smith” method compared to “Freedman-Lane”. If using brainGraph\_GLM, the number of vertices follows the same relationship.

Furthermore, all methods except Still-White include the $Z$ (nuisance covariate) matrix when calculating the permuted statistics.

References


---

Residuals

*Linear model residuals in structural covariance networks*

**Description**

`get.resid` runs linear models across brain regions listed in a `data.table` (e.g., cortical thickness), adjusting for variables in `covars` (e.g. age, sex, etc.), and calculates the externally Studentized (or leave-one-out) residuals.

The `[` method reorders or subsets residuals based on a given numeric vector. However, this is used in bootstrap and permutation analysis and should generally not be called directly by the user.
The summary method prints the number of outliers per region, and the number of times a given subject was an outlier (i.e., across regions).

The plot method lets you check the model residuals for each brain region in a structural covariance analysis. It shows a qqplot of the studentized residuals, as output from `get.resid`.

Usage

```r
get.resid(dt.vol, covars, method = c("comb.groups", "sep.groups"),
  use.mean = FALSE, exclude.cov = NULL, atlas = NULL, ...)
```

## S3 method for class 'brainGraph_resids'

```r
x[i, g = NULL]
```

## S3 method for class 'brainGraph_resids'

```r
summary(object, region = NULL,
  outlier.thresh = 2, ...)
```

## S3 method for class 'brainGraph_resids'

```r
plot(x, region = NULL, outlier.thresh = 2,
  cols = FALSE, ids = TRUE, ...)
```

## S3 method for class 'brainGraph_resids'

```r
nobs(object, ...)
```

## S3 method for class 'brainGraph_resids'

```r
case.names(object, ...)
```

## S3 method for class 'brainGraph_resids'

```r
groups(x)
```

## S3 method for class 'brainGraph_resids'

```r
region.names(object)
```

## S3 method for class 'brainGraph_resids'

```r
nregions(object)
```

Arguments

- `dt.vol` A data.table containing all the volumetric measure of interest (i.e., the object lhrh as output by `import_scn`)
- `covars` A data.table of the covariates of interest
- `method` Character string indicating whether to test models for subject groups separately or combined. Default: `comb.groups`
- `use.mean` Logical should we control for the mean hemispheric brain value (e.g. mean LH/RH cortical thickness). Default: `FALSE`
- `exclude.cov` Character vector of covariates to exclude. Default: `NULL`
- `atlas` Character string indicating the brain atlas
Arguments passed to `brainGraph_GLM_design` (optional)

- **x**, object: A `brainGraph_resids` object
- **i**: Numeric vector of the indices
- **g**: Character string indicating the group. Default: `NULL`
- **region**: Character vector of region(s) to focus on; default behavior is to show summary for all regions
- **outlier.thresh**: Number indicating how many standard deviations above/below the mean indicate an outlier. Default: 2
- **cols**: Logical indicating whether to color by group. Default: `FALSE`
- **ids**: Logical indicating whether to plot subject ID’s for outliers. Otherwise plots the integer index

**Details**

You can choose to run models for each of your subject groups separately or combined (the default) via the `method` argument. You may also choose whether to include the mean, per-hemisphere structural measure in the models. Finally, you can specify variables that are present in `covars` which you would like to exclude from the models. Optional arguments can be provided that get passed to `brainGraph_GLM_design`.

If you do not explicitly specify the atlas name, then it will be guessed from the size of your data. This could cause problems if you are using a custom atlas, with or without the same number of regions as a dataset in the package.

**Value**

- **get.resid** - an object of class `brainGraph_resids` with elements:
  - **data**: A data.table with the input volume/thickness/etc. data as well as the covariates used in creating the design matrix.
  - **X**: The design matrix, if using default arguments. If `use.mean=TRUE` then it will be a named list with a separate matrix for the left and right hemispheres. If `method='sep.groups'`, a nested named list for each group and hemisphere.
  - **method**: The input argument method
  - **use.mean**: The input argument `use.mean`
  - **resids.all**: The “wide” data.table of residuals
  - **Group**: Group names
  - **atlas**: The atlas name

`summary.brainGraph_resids` returns a list with two data tables, one of the residuals, and one of only the outlier regions.

The plot method returns a trellis object or a list of `ggplot` objects.
Note

It is assumed that dt.vol was created using import_scn. In older versions, there were issues when the Study ID was specified as an integer and was not “zero-padded”. This is done automatically by import_scn, so if you are using an external program, please be sure that the Study ID column is matched in both dt.vol and covars.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

influence.measures, qqnorm

Other Structural covariance network functions: Bootstrapping, IndividualContributions, brainGraph_permute, corr.matrix, import_scn, plot_volumetric

Examples

```r
## Not run:
myresids <- get.resids(lhrh, covars)
residPlots <- plot(myresids, cols=TRUE)

## Save as a multi-page PDF
ml <- marrangeGrob(residPlots, nrow=3, ncol=3)
ggsave('/quotesingle.Residuals.pdf', ml)

## End(Not run)
```

---

Rich Club  Rich club calculations

Description

rich_club_coeff calculates the rich club of a graph, returning the rich-club coefficient, \( \phi \), and the subgraph of rich club vertices.

rich_club_all is a wrapper for rich_club_coeff that calculates the rich-club coefficient for all degrees present in the graph. It returns a data.table with the coefficients and vertex and edge counts for each successive rich club.

rich_club_norm will (optionally) generate a number of random graphs, calculate their rich club coefficients (\( \phi \)), and return \( \phi_{\text{norm}} \) of the graph of interest, which is the observed rich-club coefficient divided by the mean across the random graphs.

rich_core finds the boundary of the rich core of a graph, based on the decreasing order of vertex degree. It also calculates the degree that corresponds to that rank, and the core size relative to the total number of vertices in the graph.
Usage

rich_club_coeff(g, k = 1, weighted = FALSE, A = NULL)

rich_club_all(g, weighted = FALSE, A = NULL)

rich_club_norm(g, N = 100, rand = NULL, ...)

rich_core(g, weighted = FALSE, A = NULL)

Arguments

- **g**: An igraph graph object
- **k**: Integer; the minimum degree for including a vertex. Default: 1
- **weighted**: Logical indicating whether or not edge weights should be used. Default: FALSE
- **A**: A numeric matrix; the adjacency matrix of the input graph. Default: NULL
- **N**: Integer; the number of random graphs to generate. Default: 100
- **rand**: A list of igraph graph objects, if random graphs have already been generated. Default: NULL
- **...**: Other parameters (passed to sim.rand.graph.par)

Details

If random graphs have already been generated, you can supply a list as an argument.

For weighted graphs, the degree is substituted by a normalized weight:

\[
\text{ceiling}\left(\frac{A}{w_{\text{min}}}\right)
\]

where \(w_{\text{min}}\) is the minimum weight (that is greater than 0), and \(\text{ceiling}()\) is the ceiling function that rounds up to the nearest integer.

Value

- **rich_club_coeff**: a list with components:
  - **phi**: The rich club coefficient, \(\phi\).
  - **graph**: A subgraph containing only the rich club vertices.
  - **Nk, Ek**: The number of vertices/edges in the rich club graph.

- **rich_club_all**: a data.table with components:
  - **k**: A vector of all vertex degrees present in the original graph
  - **phi**: The rich-club coefficient
  - **Nk, Ek**: The number of vertices/edges in the rich club for each successive \(k\)

- **rich_club_norm**: a data table with columns:
  - **k**: Sequence of degrees
Rich Club

rand Rich-club coefficients for the random graphs
orig Rich-club coefficients for the original graph.
norm Normalized rich-club coefficients.
p P-values based on the distribution of rand
p.fdr The FDR-adjusted P-values
density The observed graph’s density
threshold,Group,name

rich_core - a data table with columns:
density The density of the graph.
rank The rank of the boundary for the rich core.
k.r The degree/strength of the vertex at the boundary.
core.size The size of the core relative to the graph size.
weighted Whether or not weights were used

Author(s)
Christopher G. Watson, <cgwatson@bu.edu>

References

See Also
Other Rich-club functions: plot_rich_norm, rich_club_attrs
Other Random graph functions: Random Graphs
rich_club_attrs

Assign graph attributes based on rich-club analysis

Description

Assigns vertex- and edge-level attributes based on the results of a rich-club analysis, based on a range of vertex degrees in which the rich-club coefficient was determined to be significantly greater than that of a set of random graphs (see rich_club_norm).

Usage

rich_club_attrs(g, deg.range = NULL, adj.vsize = FALSE)

Arguments

g An igraph graph object
deg.range Numeric vector of the range of degrees indicating inclusion in the rich-club; if the default NULL, it will be from 1 to the maximum degree in the graph
adj.vsize Logical indicating whether to adjust vertex size proportional to degree. Default: FALSE

Details

Vertices which are in the rich club will be assigned an attribute rich, taking on a binary value. Their colors (attribute color.rich) will be either red or gray. Their sizes (attribute size.rich) will either be 10 or will be proportional to their degree.

Edge attribute type.rich takes on three values: rich-club (if it connects two rich-club vertices), feeder (if it connects a rich- to a non-rich-club vertex), and local (if it connects two non-rich-club vertices). The color.rich attribute is red, orange, or green. Edge sizes (size.rich) will be largest for rich-club connections, then smaller for feeder, and smallest for local.

Value

An igraph graph object with additional attributes:

rich Binary indicating membership in the rich-club
type.rich Edge attribute indicating the type of connection
color.rich Edge and vertex attributes
size.rich Edge and vertex attributes

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

See Also

Other Rich-club functions: Rich Club, plot_rich_norm
robustness

Examples

```r
## Not run:
g <- rich_club_attrs(g, rich.dt[density == densities[N] & p.fdr < .01, range(k)])
## End(Not run)
```

robustness

Analysis of network robustness

Description

This function performs a “targeted attack” of a graph or a “random failure” analysis, calculating the size of the largest component after edge or vertex removal.

Usage

```r
robustness(g, type = c("vertex", "edge"), measure = c("btwn.cent", "degree", "random"), N = 1000)
```

Arguments

- `g`: An igraph graph object
- `type`: Character string; either 'vertex' or 'edge' removals. Default: vertex
- `measure`: Character string; sort by either 'btwn.cent' or 'degree', or choose 'random'. Default: 'btwn.cent'
- `N`: Integer; the number of iterations if 'random' is chosen. Default: 1e3

Details

In a targeted attack, it will sort the vertices by either degree or betweenness centrality (or sort edges by betweenness), and successively remove the top vertices/edges. Then it calculates the size of the largest component.

In a random failure analysis, vertices/edges are removed in a random order.

Value

Data table with elements:

- `type`: Character string describing the type of analysis performed
- `measure`: The input argument measure
- `comp.size`: The size of the largest component after edge/vertex removal
- `comp.pct`: Numeric vector of the ratio of maximal component size after each removal to the observed graph’s maximal component size
- `removed.pct`: Numeric vector of the ratio of vertices/edges removed
- `Group`: Character string indicating the subject group, if applicable
small.world

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References


---

**small.world**

*Calculate graph small-worldness*

**Description**

small.world calculates the normalized characteristic path length and clustering coefficient based on observed and random graphs, used to calculate the small-world coefficient $\sigma$.

**Usage**

```r
small.world(g.list, rand)
```

**Arguments**

- `g.list` A `brainGraphList` object or list of graphs
- `rand` List of (lists of) equivalent random graphs (output from `sim.rand.graph.par`)

**Value**

A `data.table` with the following components:

- `density` The range of density thresholds used.
- `N` The number of random graphs that were generated.
- `Lp, Lp.rand, Lp.norm` The observed, average random, and normalized characteristic path length.
- `Cp, Cp.rand, Cp.norm` The observed, average random, and normalized clustering coefficient.
- `sigma` The small-world measure of the graph.

**Author(s)**

Christopher G. Watson, <cgwatson@bu.edu>

**References**

**s_core**  
*Calculate the s-core of a network*

**Description**
Calculates the s-core decomposition of a network. This is analogous to the k-core decomposition, but takes into account the strength of vertices (i.e., in weighted networks). If an unweighted network is supplied, then the output of the function coreness is returned.

**Usage**
```r
s_core(g, W = NULL)
```

**Arguments**
- `g`: An igraph graph object
- `W`: Numeric matrix of edge weights (default: NULL)

**Details**
The s-core consists of all vertices $i$ with $s_i > s$, where $s$ is some threshold value. The $s_0$ core is the entire network, and the threshold value of the $s_n$ core is

$$s_{n-1} = \min_i s_i$$

for all vertices $i$ in the $s_{n-1}$ core.

Note that in networks with a wide distribution of vertex strengths, in which there are almost as many unique values as there are vertices, then several separate cores will have a single vertex. See the reference provided below.

**Value**
Integer vector of the vertices’ s-core membership

**Author(s)**
Christopher G. Watson, <cgwatson@bu.edu>

**References**

**See Also**
coreness
Vertex Roles

*Gateway coefficient, participation coefficient, and within-mod degree z-score*

**Description**

gateway_coeff calculates the gateway coefficient of each vertex, based on community membership.

part_coeff calculates the participation coefficient of each vertex, based on community membership.

within_module_deg_z_score is a measure of the connectivity from a given vertex to other vertices in its module/community.

**Usage**

gateway_coeff(g, memb, centr = c("btwn.cent", "degree", "strength"),
A = NULL, weighted = FALSE)

part_coeff(g, memb, A = NULL, weighted = FALSE)

within_module_deg_z_score(g, memb, A = NULL, weighted = FALSE)

**Arguments**

- **g**: An igraph graph object
- **memb**: A numeric vector of membership indices of each vertex
- **centr**: Character string; the type of centrality to use in calculating GC. Default: btwn.cent
- **A**: Numeric matrix; the adjacency matrix of the input graph. Default: NULL
- **weighted**: Logical indicating whether to calculate metrics using edge weights. Default: FALSE

**Details**

The gateway coefficient $G_i$ of vertex $i$ is:

$$G_i = 1 - \sum_{S=1}^{N_M} \left( \frac{\kappa_{iS}}{\kappa_i} \right)^2 (g_{iS})^2$$

where $\kappa_{iS}$ is the number of edges from vertex $i$ to vertices in module $S$, and $\kappa_i$ is the degree of vertex $i$. $N_M$ equals the number of modules. $g_{iS}$ is a weight, defined as:

$$g_{iS} = 1 - \bar{\kappa}_{iS}c_{iS}$$

where

$$\bar{\kappa}_{iS} = \sum_j \frac{\kappa_{iS}}{\kappa_{jS}}$$
for all nodes $j$ in node $i$’s module, and

$$c_iS = c_{iS} / \max(c_n)$$

The participation coefficient $P_i$ of vertex $i$ is:

$$P_i = 1 - \sum_{s=1}^{N_M} \left( \frac{\kappa_{is}}{\kappa_s} \right)^2$$

where $\kappa_{is}$ is the number of edges from vertex $i$ to vertices in module $s$, and $\kappa_s$ is the degree of vertex $i$. $N_M$ equals the number of modules.

As discussed in Guimera et al., $P_i = 0$ if vertex $i$ is connected only to vertices in the same module, and $P_i = 1$ if vertex $i$ is equally connected to all other modules.

The within-module degree z-score is:

$$z_i = \frac{\kappa_i - \bar{\kappa}_{s_i}}{\sigma_{\kappa_{s_i}}}$$

where $\kappa_i$ is the number of edges from vertex $i$ to vertices in the same module $s_i$, $\bar{\kappa}_{s_i}$ is the average of $\kappa$ over all vertices in $s_i$, and $\sigma_{\kappa_{s_i}}$ is the standard deviation.

Value

A vector of the participation coefficients, within-module degree z-scores, or gateway coefficients for each vertex of the graph.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References


---

**vif.bg_GLM**

*Variance inflation factors for bg_GLM objects*

**Description**

Variance inflation factors for bg_GLM objects

**Usage**

```r
vif.bg_GLM(mod, ...)```
vulnerability

Arguments

- **mod**: A bg_GLM object
- ...: Unused

Value

A named array of VIFs; names of the 3rd dimension are regions

---

### Calculate graph vulnerability

**Description**

This function calculates the *vulnerability* of the vertices of a graph. Here, vulnerability is considered to be the proportional drop in global efficiency when a given vertex is removed from the graph. The vulnerability of the graph is considered the maximum across all vertices.

**Usage**

```r
vulnerability(g, use.parallel = TRUE, weighted = FALSE)
```

**Arguments**

- **g**: An igraph graph object
- **use.parallel**: Logical indicating whether or not to use `foreach` (default: TRUE)
- **weighted**: Logical indicating whether weighted efficiency should be calculated (default: FALSE)

**Value**

A numeric vector of length equal to the vertex count of `g`

**Author(s)**

Christopher G. Watson, <cgwatson@bu.edu>

**References**


**See Also**

- `efficiency`
write_brainnet

Write files to be used for visualization with BrainNet Viewer

Description

Write the .node and .edge files necessary for visualization with the BrainNet Viewer software.

Usage

write_brainnet(g, vcolor = "none", vsize = "constant",
edge.wt = NULL, file.prefix = "")

Arguments

g The igraph graph object of interest
vcolor Character string indicating how to color the vertices (default: 'none')
vsize Character string indicating what size the vertices should be; can be any vertex-level attribute (default: 'constant')
edge.wt Character string indicating the edge attribute to use to return a weighted adjacency matrix (default: NULL)
file.prefix Character string for the basename of the .node and .edge files that are written

Details

For the .node file, there are 6 columns:

- Columns 1-3: Vertex x-, y-, and z-coordinates
- Column 4: Vertex color
- Column 5: Vertex size
- Column 6: Vertex label

The .edge file is the graph's associated adjacency matrix; a weighted adjacency matrix can be returned by using the edge.wt argument.

Author(s)

Christopher G. Watson, <cgwatson@bu.edu>

References

Examples

## Not run:
write_brainnet(g, vcolor='community', vsize='degree', edge wt='t.stat')

## End(Not run)
Index

* Centrality functions
  centr_betw_comm, 20
  centr_lev, 21

* GLM functions
  GLM, 39
  GLM design, 45
  GLM fits, 47
  mtpc, 72

* Graph creation functions
  brainGraphList, 14
  Creating_Graphs, 32
  Creating_Graphs_GLM, 35
  make_ego_brainGraph, 64

* Group analysis functions
  Bootstrapping, 8
  brainGraph_permute, 18
  GLM, 39
  Mediation, 69
  mtpc, 72
  NBS, 76

* Plotting functions
  plot.brainGraph, 79
  plot.brainGraphList, 81
  plot_brainGraph_multi, 83
  Plotting GLM graphs, 82

* Random graph functions
  Random Graphs, 89
  Rich Club, 97

* Rich-club functions
  plot_rich_norm, 86
  Rich Club, 97
  rich Club attrs, 100

* Structural covariance network functions
  Bootstrapping, 8
  brainGraph_permute, 18
  corr.matrix, 26
  import_scn, 58
  IndividualContributions, 59
  plot_volumetric, 88

  Residuals, 94

* datasets
  Brain Atlases, 10
  [.bg_GLM (GLM), 39
  [.brainGraphList (brainGraphList), 14
  [.brainGraph_resids (Residuals), 94
  [.corr_mats (corr.matrix), 26
  aal116 (Brain Atlases), 10
  aal2.120 (Brain Atlases), 10
  aal2.94 (Brain Atlases), 10
  aal90 (Brain Atlases), 10
  AIC, 52
  analysis_random_graphs (Random Graphs), 89
  Anova, 55
  anova.bg_GLM (GLM statistics), 53
  aop (IndividualContributions), 59
  apply_thresholds, 3
  as.POSIXct, 34
  as_atlas (Atlas Helpers), 4
  as_brainGraphList (brainGraphList), 14
  assortativity, 8
  Atlas Helpers, 4
  Attributes, 6
  backsolve, 48
  betweenness, 8
  bg_to_mediate, 69
  bg_to_mediate (Mediation), 69
  BIC, 52
  boot, 9
  boot.ci, 9
  Bootstrapping, 8, 19, 28, 43, 59, 61, 72, 75, 78, 89, 97
  Brain Atlases, 10
  brainGraph, 13
  brainGraph-methods, 13
  brainGraph-options (brainGraph), 13
  brainGraph-package (brainGraph), 13
brainGraph_boot (Bootstrapping), 8
brainGraph_GLM, 35, 37, 74, 75, 82, 94
brainGraph_GLM (GLM), 39
brainGraph_GLM_design, 41, 42, 70, 74, 77, 96
brainGraph_GLM_design (GLM design), 45
brainGraph_mediate, 70, 82
brainGraph_mediate (Mediation), 69
brainGraph_permute, 9, 18, 18, 28, 43, 59, 61, 72, 75, 78, 89, 97
brainGraphList, 14, 35, 37, 64
brainnetome (Brain Atlases), 10
brainsuite (Brain Atlases), 10
case.names.bg_GLM (GLM basic info), 44
case.names.brainGraph_resids (Residuals), 94
case.names.mtpc (mtpc), 72
case.names.NBS (NBS), 76
centr_betw, 8
centr_betw_comm, 20, 21
centr_eigen, 8
centr_lev, 20, 21
check_SID, 22
cluster_louvain, 8
coeff.bg_GLM (GLM statistics), 53
coeff_determ (GLM statistics), 53
coeff_table (GLM statistics), 53
coeff_var, 23
colMax (Matrix utilities), 66
colMaxAbs (Matrix utilities), 66
colMin (Matrix utilities), 66
communicability, 23
communities, 8
components, 8
confint.bg_GLM (GLM statistics), 53
contract, 25
contract_brainGraph, 24
cooks.distance.bg_GLM (GLM influence measures), 50
cor.diff.test, 25
coreness, 8, 103
corr.matrix, 9, 19, 26, 59–61, 89, 97
Count Edges, 29
count_homologous (Count Edges), 29
count_inter (Count Edges), 29
covratio.bg_GLM (GLM influence measures), 50
craddock200 (Brain Atlases), 10
create_atlas (Atlas Helpers), 4
create_mats, 3, 4, 30
Creating_Graphs, 17, 32, 37, 64
Creating_Graphs_GLM, 17, 35, 36, 64
data.table, 86
destrieux (Brain Atlases), 10
deviance.bg_GLM (GLM statistics), 53
df.residual.bg_GLM (GLM statistics), 53
df.residual.mtpc (mtpc), 72
df.residual.NBS (NBS), 76
dfbeta.bg_GLM (GLM influence measures), 50
dfbetas.bg_GLM (GLM influence measures), 50
dffits.bg_GLM (GLM influence measures), 50
diag, 66
diag_sq (Matrix utilities), 66
diameter, 8
distances, 8
dk (Brain Atlases), 10
dkt (Brain Atlases), 10
dosenbach160 (Brain Atlases), 10
edge_asymmetry, 37
edge.spatial_dist (Graph Distances), 56
efficiency, 38, 106
ego, 64
Extract.brainGraph_resids (Residuals), 94
Extract.brainGraphList
(brainGraphList), 14
Extract.corr_mats (corr.matrix), 26
extractAIC, 52
extractAIC.bg_GLM (GLM model selection), 52
fastLmBG (GLM fits), 47
fastLmBG_3d (GLM fits), 47
fastLmBG_3dY (GLM fits), 47
fastLmBG_3dY_1p (GLM fits), 47
fastLmBG_f (GLM fits), 47
fastLmBG_t (GLM fits), 47
fitted.bg_GLM (GLM statistics), 53
foreach, 89
formula.bg_GLM (GLM basic info), 44
formula.mtpc (mtpc), 72
formula.NBS (NBS), 76
gateway_coeff (Vertex Roles), 104
get.resid, 9, 18, 27, 60, 95
groupPage dimensions: 612.0x792.0
get.resid (Residuals), 94
groups.brainGraphList
get_thresholds (Matrix utilities), 66
IndividualContributions
GLM, 9, 19, 39, 46, 50, 52, 55, 72, 75, 76, 78
GLM basic info, 44
GLM design, 45
GLM fits, 47
GLM influence measures, 50
GLM model selection, 52
GLM statistics, 53
gordon333 (Brain Atlases), 10
Graph Data Tables, 55
Graph Distances, 56
group_size (Graph Data Tables), 55
graph_attr (Graph Data Tables), 55
graph_attr_names, 56
graph_from_data_frame, 56
groups.brainGraph_resids (Residuals), 94
groups.brainGraphList
(brainGraph-methods), 13
groups.corr_mats (brainGraph-methods), 13
guess_atlas (Atlas Helpers), 4
hatvalues.bg_GLM (GLM influence measures), 50
hcp_mmp1.0 (Brain Atlases), 10
hoa112 (Brain Atlases), 10
hubness, 57
import_scm, 9, 19, 22, 28, 58, 61, 89, 95, 97
IndividualContributions, 9, 19, 28, 59, 59, 89, 97
influence.bg_GLM (GLM influence measures), 50
influence.measures, 51, 97
inv (Inverse), 61
Inverse, 61
is.brainGraph (Creating_Graphs), 32
is.brainGraphList (brainGraphList), 14
is_binary (Matrix utilities), 66
keeping_degseq, 91
knn, 8
labels.bg_GLM (GLM basic info), 44
labels.mtpc (mtpc), 72
labels.NBS (NBS), 76
list.files, 31
lm, 48, 54
lm.influence, 51
logLik.bg_GLM (GLM model selection), 52
logLik.lm, 52
loo (IndividualContributions), 59
lpba40 (Brain Atlases), 10
make_auc_brainGraph, 35, 63
make_brainGraph, 89, 90
make_brainGraph (Creating_Graphs), 32
make_brainGraphList (brainGraphList), 14
make_brainGraphList.bg_GLM
(Creating_Graphs_GLM), 35
make_brainGraphList.mtpc
(Creating_Graphs_GLM), 35
make_brainGraphList.NBS
(Creating_Graphs_GLM), 35
make_eg_brainGraph, 17, 35, 37, 64
make_empty_brainGraph
(Creating_Graphs), 32
make_intersection_brainGraph, 65
Matrix utilities, 66
mean_distance_wt, 68
mediate, 69, 70, 72
Mediation, 9, 19, 43, 69, 75, 78
mtext, 80
mtpc, 9, 19, 35, 37, 43, 46, 50, 72, 72, 78, 82
NBS, 9, 19, 35, 37, 43, 72, 75, 76, 82, 93, 94
nobs.bg_GLM (GLM basic info), 44
nobs.brainGraph_resids (Residuals), 94
nobs.brainGraphList (brainGraphList), 14
nobs.mtpc (mtpc), 72
nobs.NBS (NBS), 76
nregions (brainGraph-methods), 13
nregions.bg_GLM (GLM basic info), 44
nregions.brainGraph_resids (Residuals), 94
nregions.corr_mats (corr.matrix), 26
nregions.mtpc (mtpc), 72
nregions.NBS (NBS), 76
options, 13
pad_zeros (check_sID), 22
par, 80, 82
part_coeff (Vertex Roles), 104
partition (randomise), 92
pinv (Inverse), 61
plot.bg_GLM (GLM), 39
plot.brainGraph, 79, 81, 83–85
plot.brainGraph_boot (Bootstrapping), 8
plot.brainGraph_GLM (Plotting GLM graphs), 82
plot.brainGraphmediate (Plotting GLM graphs), 82
plot.brainGraph_mtpc (Plotting GLM graphs), 82
plot.brainGraph_NBS (Plotting GLM graphs), 82
plot.brainGraph_permute (brainGraph_permute), 18
plot.brainGraph_resids (Residuals), 94
plot.brainGraphList, 80, 81, 83, 85
plot.common, 79
plot.corr_mats (corr.matrix), 26
plot.IC (IndividualContributions), 59
plot.igraph, 79
plot.lm, 40, 43
plot.mediate, 69
plot.mtpc (mtpc), 72
plot_brainGraph_multi, 80, 81, 83, 83
plot_global, 85
plot_rich_norm, 86, 99, 100
plot_vertex_measures, 87
plot_volumetric, 9, 19, 28, 59, 61, 88, 97
Plotting GLM graphs, 82
power264 (Brain Atlases), 10
print.bg_GLM (GLM), 39
print.brainGraphList (brainGraphList), 14
printCoefmat, 77
qqnorm, 97
qr.array (Matrix utilities), 66
qr.default, 67
qr.Q, 66
qr.R, 66
qr.Q2 (Matrix utilities), 66
qr_R2 (Matrix utilities), 66
Random Graphs, 89
randomise, 43, 92
randomise_3d (randomise), 92
rcorr, 28
region.names (brainGraph-methods), 13
region.names.bg_GLM (GLM basic info), 44
region.names.brainGraph_resids (Residuals), 94
region.names.corr_mats (corr.matrix), 26
region.names.mtpc (mtpc), 72
Residuals, 9, 19, 28, 59, 61, 89, 94
residuals.bg_GLM (GLM statistics), 53
rewire, 91
Rich Club, 97
rich俱乐部, 98
rich俱乐部 (Rich Club), 97
rich俱乐部_atr, 87, 99, 100
rich俱乐部_coeff, 97, 98
rich俱乐部_coeff (Rich Club), 97
rich俱乐部_norm, 98, 100
rich俱乐部_norm (Rich Club), 97
rich_core, 86, 87, 99
rich俱乐部 (Rich Club), 97
robustness, 101
rstandard.bg_GLM (GLM influence measures), 50
rstudent.bg_GLM (GLM influence measures), 50
s_core, 103
sample_degseq, 89, 91
set_brainGraph_attr, 15, 16, 33, 36
set_brainGraph_attr (Attributes), 6
sigma.bg_GLM (GLM statistics), 6
sim.rand.graph.clust, 89
sim.rand.graph.clust (Random Graphs), 89
sim.rand.graph.hqs (Random Graphs), 89
sim.rand.graph.par, 98, 102
sim.rand.graph.par (Random Graphs), 89
slicer (plot_brainGraph_multi), 83
small.world, 91, 102
solve, 48
sort, 67
summary.bg_GLM (GLM), 39
summary.brainGraph (Mediation), 69
summary.brainGraph (Creating Graphs), 32
summary.brainGraph_boot (Bootstrapping), 8
summary.brainGraph_permute (brainGraph_permute), 18
summary.brainGraph_resids, 96
summary.brainGraph_resids (Residuals), 94
summary.IC (IndividualContributions), 59
summary.lm, 53
INDEX

summary.mediate, 69
summary.mtpc (mtpc), 72
summary.NBS (NBS), 76
symm_mean (Matrix utilities), 66
symmetrize. 30
symmetrize (Matrix utilities), 66
tcrossprod, 62
terms.bg_GLM (GLM basic info), 44
terms.mtpc (mtpc), 72
terms.NBS (NBS), 76
transitivity, 8, 91

variable.names.bg_GLM (GLM basic info), 44
variable.names.mtpc (mtpc), 72
variable.names.NBS (NBS), 76
vcov.bg_GLM (GLM statistics), 53
Vertex Roles, 104
vertex_attr, 56
vertex_attr_dt (Graph Data Tables), 55
vertex_attr_names, 56
vertex_spatial_dist (Graph Distances), 56
vif.bg_GLM, 105
vulnerability, 106

within_module_deg_z_score (Vertex Roles), 104
write_brainnet, 107

xfm.weights, 9
xfm.weights (Attributes), 6