Package: QPress (via r-universe)

September 9, 2024

Type Package

Title Qualitative Network Analysis

Version 0.23

Date 2013-05-13

Description Qualitiative analysis of press perturbations of network models. Given a network model represented as a signed directed graphs, this package provide facilities for evaluating the impact of a press perturbation to the system through simulation.

Depends R(>= 3.2.0)

Imports ggplot2, shiny, shinythemes, tcltk, tcltk2, XML

License GPL-2

LazyData TRUE

Encoding UTF-8

RoxygenNote 7.2.1

Collate 'QPress-package.R' 'Community.R' 'text.R' 'tk.r' 'XML.R' 'shiny.R'

Suggests DiagrammeR, knitr, rmarkdown

VignetteBuilder knitr

Repository https://scar.r-universe.dev

RemoteUrl https://github.com/swotherspoon/QPress

RemoteRef HEAD

RemoteSha 699306e24d588c1b8254ba876b95d5608de87dc2

Contents

`QPress-package` *Qualitative Network Analysis*

Description

Qualitiative analysis of press perturbations of network models. Given a network model represented as a signed directed graphs, this package provide facilities for evaluating the impact of a press perturbation to the system through simulation.

This package provides facilities for simulating press perturbation scenarios for qualitative network models specified as signed directed graphs (signed digraphs).

Author(s)

Ben Raymond, Jessica Melbourne-Thomas, Simon Wotherspoon

B. Raymond, J. Melbourne-Thomas and S. Wotherspoon

`adjacency.image` *Adjacency Matrix Image*

Description

Display adjacency matrix of the directed graph as an image

Usage

```
adjacency.image(edges, required.groups = c(0), cex.axis = 1)
```

Arguments

`edges`	an edge list
`required.groups`	which edge groups should be included?
`cex.axis`	character expansion factor for the edge labels

Details

Display the matrix constructed by `adjacency.matrix` as an image.

Value

Returns the adjacency matrix for the directed graph.

See Also

`adjacency.matrix`

Examples

```
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D -> B"))
edges <- enforce.limitation(edges)
adjacency.image(edges)
```

`adjacency.matrix` *Adjacency Matrix*

Description

Adjacency matrix of the directed graph.

Usage

```
adjacency.matrix(edges, labels = FALSE, required.groups = c(0))
```

Arguments

`edges`	an edge list
`labels`	add row and column labels
`required.groups`	which edge groups should be included?

Details

This function converts an edge list to an adjacency matrix `A`, following the convention that `A[i,j]` represents the impact of node `j` on node `i`.

Value

Returns the adjacency matrix for the directed graph.

See Also

`adjacency.image`

Examples

```
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D -> B"))
edges <- enforce.limitation(edges)
adjacency.matrix(edges,labels=TRUE)
```

adjoint *Fedeew-Leverrier*

Description

Adjoint matrix and Characteristic Polynomial

Usage

```
adjoint(A)

charpoly(A)
```

Arguments

A	a square matrix

Details

These functions compute the adjoint matrix and characteristic polynomial of A by the Fedeew-Leverrier algorithm.

If A has integer elements and the computations are performed with integer arithmetic the result is exact.

Value

`adjoint` returns the adjoint matrix of `A`

`charpoly` returns the coefficients of the characteristic polynomial of `A` as a vector.

Examples

```
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D -> B"))
edges <- enforce.limitation(edges)
A <- adjacency.matrix(edges,labels=TRUE)
adjoint(A)
```

checkbox *Checkbox widgets*

Description

Construct a checkbox widget

Usage

```
checkbox(parent, label, initial = 0)

checkcolumn(parent, label, rows, label.rows = TRUE)
```

Arguments

parent	the parent window.
label	the label for the enclosing frame.
initial	the initial state of the checkbox
rows	the row labels
label.rows	whether to label rows.

Details

The checkbox function makes a single checkbox widget, while checkcolumn makes a widget containing a column of checkboxes.

Value

Returns an object of class checkbox or checkcolumn with elements

window	the widget
selected	function that returns the state of the checkboxes
state	the tclVars representing the state of the checkboxes

checkedges *Edge Selection Widget*

Description

Construct an edge selection widget

Usage

```
checkedges(parent, label, rows, edges, group = NULL, label.rows = TRUE)
```

Arguments

parent	the parent window
label	the label for the enclosing frame
rows	the labels for the rows (node names)
edges	an nx2 matrix that defines the edges
group	a numeric vector that groups edges
label.rows	whether to label rows

Details

Makes a widget consisting of a grid of check buttons that allow the user to select edges of the network.

Value

Returns an object of class checkedges with elements

window	the widget
selected	function that returns the state of the check buttons
state	the tclVars representing the state of the check buttons

---

community.sampler *Sampling Community Matrices*

Description

Construct functions to generate random community matrices

Usage

```
community.sampler(edges, required.groups = c(0))
```

Arguments

edges	an edge list
required.groups	a vector of integers specifying which groups of edges must always occur in the community matrix.

Details

Given an edge list that specifies a directed graph, this function constructs a list of functions that can be use to generate random community matrices corresponding to that directed graph.

Edges in the edge list that do not fall in a required group are considered uncertain, and may or may not be represented in the community matrix.

Random community matrices are generated in two stages, the first stage determines which of the uncertain edges will be included or excluded in subsequent simulations, while the second stage generates random matrices corresponding to the selected. The `select` function is a function of a single argument `p` that determines which of the uncertain edge pairs will be included in matrices generated by subsequent calls to `community`. This function always selects either neither or both edges of a pair and every uncertain pair has likelihood `p` of being selected. The `community` function is a function of no arguments that generates a random community matrix. The `weights` function is a function of a single argument `W` that returns those entries of the community matrix `W` that correspond to edges in the edge list.

Value

Returns a list with elements

`community()`	a function to generate a random community matrix
`select(p)`	a function that randomly selects which uncertain edges will be retained
`weights(W)`	a function that returns the (non-zero) weights as a vector
`edge.labels`	the labels of the edges
`uncertain.labels`	the labels of the uncertain edges

Examples

```
set.seed(32)
## Sample model
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D --> B"))
edges <- enforce.limitation(edges)
s <- community.sampler(edges)
## Force D --> B edge out
s$select(0)
## Generate community matrices
s$community()
s$community()
## Force D --> B edge in
s$select(1)
## Generate community matrices
s$community()
```

```
s$community()
## Select the uncertain D --> B edge with prob 0.6
s$select(0.6)
## Generate community matrices
s$community()
s$community()
```

`drop.nodes` *Drop one or more nodes from a system*

Description

This is an experimental function! Given a set of system simulation outputs (from `system.simulate`), it will drop one or more nodes and their associated edges, but leave all other elements of the system untouched. Each set of edge weights in `sim` is checked for stability after dropping the specified nodes, and any matrices representing unstable systems are removed from the returned set.

Usage

```
drop.nodes(sim, to.drop, method = "remove")
```

Arguments

`sim`	the result from `system.simulate`
`to.drop`	the names of the nodes to drop (check `node.labels(sim$edges)`)
`method`	either "remove" (the specified nodes will be fully removed from the system) or "zeros" (the specified nodes will be left in the system but all edges from or to these nodes (other than self-interactions) are set to zero).

Value

As for `system.simulate`

`enforce.limitation` *Self Limitation*

Description

Enforce self limitation

Usage

```
enforce.limitation(edges)
```

Arguments

`edges`	an edge list

Details

For stability, the majority of nodes of the directed graph should have a self limiting edge. This function adds a self limiting edge for every node to an existing edge list.

Value

Returns an edge list augmented with self limiting edges.

`grviz.digraph` *Export to DOT*

Description

Write a DOT specification of the model.

Usage

```
grviz.digraph(
  edges,
  name = "web",
  fontsize = 10,
  node.style = "filled",
  node.shape = "oval",
  node.color = "DarkOrange",
  edge.color = "DarkGrey"
)
```

Arguments

`edges`	An edge list
`name`	The name of the digraph
`fontsize`	Fontsize for node labels.
`node.style`	The node style.
`node.shape`	The node shape.
`node.color`	The node color.
`edge.color`	The edge color.

Details

Write a DOT specification of the model in a form suitable for use with `grViz` from **DiagrammeR**.

Value

Returns a string.

`impact.barplot` *Impact Barplot*

Description

Display the impact of a perturbation as a barplot

Usage

```
impact.barplot(sim, epsilon = 1e-05, main = "", cex.axis = 1)

impact.barplot0(
  sim,
  perturb = 0,
  monitor = NA,
  epsilon = 1e-05,
  main = "",
  cex.axis = 1
)
```

Arguments

`sim`	the result from `system.simulate`
`epsilon`	outomes below this in absolute magnitude are treated as zero.
`main`	text for plot title
`cex.axis`	character expansion factor for the edge labels
`perturb`	a named vector that indicates which nodes were perturbed and the relative magnitude of the perturbation.
`monitor`	n named vector of signs (-1,0,1) or NA that indicates the outcome of the perturbation.

Details

This control constructs a barplot that shows the fraction of simulations in which a positive (orange), negative (blue) or zero (off white) outcome occurs at each node following a given perturbation.

The user may specify the perturbation of the nodes, and any outcome known from monitoring the network, and then construct a barplot of the frequency table of outcomes at each node.

`impact.barplot0` is a non-interactive variant for programmatic use.

`impact.barplot.shiny` *Shiny Impact Barplot*

Description

A Shiny app to display the impact of a perturbation as a barplot

Usage

```
impact.barplot.shiny(sim, epsilon = 1e-05, main = "", cex.axis = 1)
```

Arguments

`sim`	the result from `system.simulate`
`epsilon`	outomes below this in absolute magnitude are treated as zero.
`main`	text for plot title
`cex.axis`	character expansion factor for the edge labels

Details

This control constructs a barplot that shows the fraction of simulations in which a positive (orange), negative (blue) or zero (off white) outcome occurs at each node following a given perturbation.

The user may specify the perturbation of the nodes, and any outcome known from monitoring the network, and then construct a barplot of the frequency table of outcomes at each node.

Examples

```
## Not run:
  set.seed(32)
  ## Sample model
  edges <- parse.digraph(c(
    "E *-> D",
    "D *-> C",
    "C -> E",
    "E *-> B",
    "B *-> A",
    "A -> E",
    "D --> B"))
  edges <- enforce.limitation(edges)
  sims <- system.simulate(10, edges)
  impact.barplot.shiny(sims)

## End(Not run)
```

impact.table *Impact Table*

Description

Tabulate the impact of every positive perturbation as table.

Usage

```
impact.table(sim, epsilon = 1e-05)
```

Arguments

sim	the result from system.simulate
epsilon	outomes below this in absolute magnitude are treated as zero.

Details

Crosstabulate the mean impact (positive or negative) at each node for a positive perturbation of each node. The k-th column corresponds to a perturbation of the k-th node, and shows the mean impact on each node.

Value

The crosstabulation as a matrix

interactive.selection *Interactive Selection Widget*

Description

Construct control widget

Usage

```
interactive.selection(
  action,
  nodes,
  edges = NULL,
  slider = NULL,
  checkbox = NULL,
  perturb = TRUE,
  monitor = TRUE
)
```

Arguments

action	function to perform the widgets action
nodes	node labels
edges	edge labels
slider	slider label
checkbox	checkbox label
perturb	should a node perturbation control be rendered
monitor	should a node monitoring control be rendered

Details

Constructs a toplevel window that allows the allowing the user to interactively select nodes to perturb/monitor, from a subset of models, and then perform a given action.

The action argument must be function of five arguments

- perturb the nodes that were perturbed
- monitor the outcome of the monitoring
- edge the edges to select
- check the state of a checkbutton
- slider the state of a slider

---

model.dia *Dia Representations*

Description

Read and write Dia representations of models

Usage

```
model.dia(file, labels = NULL)

write.dia(edges, file, width = 8, height = 2, self = TRUE)
```

Arguments

file	name of the file to read or write
labels	the sequence of labels to use for the nodes
edges	an edge list
width	width of the nodes in Dia
height	height of the nodes in Dia
self	should self edges be written.

Details

These functions read and write Dia representions of model toplogies.

These functions should be used with care as no attempt is made to test for model mis-specification. The `model.dia` function only recognizes node shapes "Flowchart - Ellipse", "Flowchart - Box" and "Flowchart - Terminal", line types "Standard - Arc", "Standard - ZigZagLine" and "Standard - Line", and arrow types 8, 1 and 5. Other node shapes, line or arrow types will be silently ignored leading to a mispecified model.

Value

The `model.dia` function returns an edge list.

See Also

`read.digraph`

`node.labels` *Node and Edge Labels*

Description

Extract labels for the nodes and edges of the directed graph.

Usage

```
node.labels(edges)

edge.labels(edges, reverse = FALSE)
```

Arguments

`edges`	an edge list
`reverse`	reverse the direction of edges

Details

These functions construct meaningful labels for the nodes and edges from an edge list.

Value

Return a vector of node or edge labels

parse.edge *Indices of (Directed) Edges*

Description

Parse a text representation of (directed) edges, return the index of the directed edge within the edge list.

Usage

```
parse.edge(lines, edges)
```

Arguments

lines	a vector of strings representing directed edges
edges	an edge list

Details

Each directed edge is represented as a string consisting of two node labels separated by an arrow, where the arrow consists of a sequence of dashes "-" followed by one of the character sequences ">","*","<>". The number of dashes used in the arrow is ignored.

Value

the indices of the directed edges within the edge list

Examples

```
## Sample model
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D --> B"))
edges <- enforce.limitation(edges)
parse.edge(c("E->D","D-*E","A-*B"),edges)
```

press.impact *Response to Press Perturbation*

Description

Construct a function to calculate response to perturbation.

Usage

```
press.impact(edges, perturb, monitor = NULL)
```

Arguments

edges	an edge list.
perturb	a named vector that indicates which nodes were perturbed and the relative magnitude of the perturbation.
monitor	n named vector that indicates the subset of nodes to monitor.

Details

Given the an edge list that specifies a directed graph, a set of nodes to perturb and a set of nodes to monitor, `press.impact` constructs a function of a single argument `W` that determines the response of the monitored nodes to the perturbation for a simulated community matrix `W`.

Value

Returns a function that when applied to a community matrix calculates the response to a press perturbation.

Examples

```
set.seed(32)
## Sample model
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D --> B"))
edges <- enforce.limitation(edges)
s <- community.sampler(edges)
s$select(0.5)
## Perturb D, monitor C
f <- press.impact(edges,perturb=c(D=1),monitor=c(C=0))
W <- s$community()
f(W)
```

```
W <- s$community()
f(W)
## Perturb D, monitor all
f <- press.impact(edges,perturb=c(D=1))
W <- s$community()
f(W)
W <- s$community()
f(W)
```

`press.validate` *Validation Criterion*

Description

Construct a function to test a validation criterion

Usage

```
press.validate(edges, perturb, monitor, epsilon = 1e-05)
```

Arguments

`edges`	an edge list
`perturb`	a named vector that indicates which nodes were perturbed and the relative magnitude of the perturbation.
`monitor`	n named vector of signs (-1,0,1) that indicates the outcome of the perturbation.
`epsilon`	outomes below this in absolute magnitude are treated as zero.

Details

Given the an edge list that specifies a directed graph, a set of nodes to perturb and a set of nodes to monitor, `press.validate` constructs a function of a single argument `W` to test whether the response to perturbation of the system represented by the community matrix `W` matches an observed outcome. The outcome is only specified up to sign (-1, 0 or +1), where outcomes smaller than `epsilon` are treated as zero.

Value

Returns a function that when applied to a community matrix determines whether the matrix is consistent with the given validation criterion.

Examples

```
set.seed(32)
## Sample model
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D --> B"))
edges <- enforce.limitation(edges)
s <- community.sampler(edges)
s$select(0.5)
## Perturb D, B and C must decrease
f <- press.validate(edges,perturb=c(D=1),monitor=c(B=-1,C=-1))
W <- s$community()
f(W)
W <- s$community()
f(W)
```

---

radiogrid *Grid of Radio Buttons*

Description

Construct a grid of radio buttons to select from a range of options that are common to many items.

Usage

```
radiogrid(parent, label, rows, choices, initial = 1, label.rows = TRUE)
```

Arguments

parent	the parent window
label	the label for the enclosing frame
rows	the labels for the rows/items
choices	the labels for the columns/choices
initial	the initial selection
label.rows	whether to label rows

Value

Returns an object of class radiogrid with elements

window	the widget
selected	function that returns the state of the radiobuttons
state	the tclVars representing the state of the radiobuttons

`read.digraph` *Text Representations of Models*

Description

Read and write text representations of models

Usage

```
read.digraph(file, labels = NULL)

parse.digraph(lines, labels = NULL)

deparse.digraph(edges)

write.digraph(edges, file = "")
```

Arguments

`file`	the name of the file to read or write
`labels`	the sequence of labels to use for the nodes
`lines`	a string representation of the model
`edges`	an edge list.

Details

The functions `read.digraph` and `parse.digraph` read a model description from a text file and a string respectively, while `write.digraph` writes a text representation of the model to and file.

These functions recognize the following text format. Each line corresponds to an edge, and must consist of two node labels separated by an arrow. An arrow consists of one of the character sequences "<","*","<>" or "" on the left and ">","*","<>" or "" on the right, separated by a sequence of dashes "-". The number of dashes used in the arrow defines the group number of the edge.

Value

The `write.digraph` function invisibly returns the text that was written to the file.

The functions `read.digraph` and `parse.digraph` return an edge list - a data frame with columns

`From`	a factor indicating the origin of each edge (the node that effects)
`To`	a factor indicating the destination of each edge (the node that is effected)
`Group`	an integer vector that indicates the group each edge belongs to
`Type`	a factor indicating the edge type - "N" (negative) ,"P" (positive),"U" (unknown) or "Z" (zero)
`Pair`	an integer vector that indicates the pairing of directed edges

Each edge of the text specification is separated into two directed edges, and every row of an edge list corresponds to a single directed edge.

Examples

```
edges <- parse.digraph(c("A <-* B","C *-> A","C <- D",
    "D -> B","B *--* C","A <--- D"))
edges
deparse.digraph(edges)
```

`retain.groups` *Edge Subsets*

Description

Subset an edge list

Usage

```
retain.groups(edges, groups)

retain.nodes(edges, nodes)
```

Arguments

`edges`	an edge list
`groups`	the groups to retain in the subset
`nodes`	the nodes to retain in the subset

Details

These functions extract a subset of an edge list containing only edges in a specified group, or incident with a specified set of nodes.

Value

`retain.groups` returns an edge list containing only edges from the specified groups.

`retain.nodes` returns an edge list containing only edges incident on the specified nodes.

Examples

```
edges <- parse.digraph(c("A *-> B","B *-> C","C *--> D"))
write.digraph(retain.groups(edges,c(0)))
```

signum *Sign classification*

Description

Classify the sign of the elements of a vector

Usage

```
signum(x, epsilon = 1e-05)
```

Arguments

x	vector of values to test
epsilon	magnitude threshold

Details

Calculates the sign of the elements of then vector x, except that values less than epsilon in magnitude are rounded down to zero.

Value

Returns a vector with elements +1,0 or -1.

Examples

```
signum(c(-40,-3,-0.1E-8,0,2,5))
```

slider *Slider Widgets*

Description

Construct a slider widget.

Usage

```
slider(parent, initial = 1, from = 0, to = 100, orient = "horizontal")
```

Arguments

parent	the parent window
initial	the initial values of the sliders
from	minimum slider values
to	maximum slider value
orient	slider orientation

Details

The `slider` function creates a widget containing a single horizontal slider.

Value

Returns an object of class slider with elements

`window`	the widget
`selected`	function that returns the state of the sliders
`state`	the tclVars representing the state of the sliders

`stable.community` *System Stability*

Description

Test community matrix stability

Usage

```
stable.community(W)
```

Arguments

`W`	a simulated community matrix

Details

The system is stable if the eigenvalues of community matrix all have negative real part. This function tests the eigenvalues of a simulated community matrix to determine the stability of the repreeented system.

Value

Returns TRUE if the system is stable, FALSE otherwise.

Examples

```
set.seed(32)
## Sample model
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D --> B"))
```

```
edges <- enforce.limitation(edges)
s <- community.sampler(edges)
s$select(0)
## First sample is stable
W <- s$community()
stable.community(W)
## Second is not
W <- s$community()
stable.community(W)
```

`system.simulate` *Simulate System*

Description

Simulate Inverse Community Matrices for a Network

Usage

```
system.simulate(
  n.sims,
  edges,
  required.groups = c(0),
  sampler = community.sampler(edges, required.groups),
  validators = NULL
)
```

Arguments

`n.sims`	number of matrices to simulate.
`edges`	an edge list.
`required.groups`	a vector of integers specifying which groups of edges must always occur in the community matrix.
`sampler`	the sampler object used to generate random weights (see `community.sampler`)
`validators`	an (optional) list of validation functions generated by `press.validate`.

Details

Generate sets of edge weights and the inverse community matrices given a directed graph and validation criteria by rejection sampling. Matrices with a pattern of signs consistent with the given model are generated, and only the matrices that correspond to stable equilibria and consistent with the given validation criteria are retained. For matrices retained in the sample, the matrix is inverted, and the *inverse* community matrix, and the weights that define the community matrix are returned. The function also returns the total number of matrices generated, the number of these that are stable and the number that are ultimately accepted for the sample.

The output of this function may be passed to the interactive exploratory tools.

This function is a simple wrapper for `community.sampler`, `stable.community` and the functions generated by `press.validate`.

Value

Returns a list with elements

`edges`	The edge list
`A`	A list of inverse community matrices
`w`	A matrix of the corresponding edge weights
`total`	The total number of matrices generated
`stable`	The number of stable matrices generated
`accepted`	The number of matrices accepted for the sample

Examples

```
set.seed(32)
## Sample model
edges <- parse.digraph(c(
  "E *-> D",
  "D *-> C",
  "C -> E",
  "E *-> B",
  "B *-> A",
  "A -> E",
  "D --> B"))
edges <- enforce.limitation(edges)
sims <- system.simulate(10,edges,
                        validators=list(
                          press.validate(edges,
                                         perturb=c(D=1),
                                         monitor=c(D=1)),
                          press.validate(edges,
                                         perturb=c(D=1),
                                         monitor=c(B=-1,C=1))))
```

---

weight.density *Weight Density Plots*

Description

Display weights of valid and invalid matrices as a density plots

Usage

```
weight.density(sim, epsilon = 1e-05, main = "")

weight.density0(
  sim,
  perturb,
  monitor,
  edges,
  smooth = 1,
  epsilon = 1e-05,
  main = ""
)
```

Arguments

sim	the result from system.simulate
epsilon	outomes below this in absolute magnitude are treated as zero.
main	text for plot title
perturb	a named vector that indicates which nodes were perturbed and the relative magnitude of the perturbation.
monitor	n named vector of signs (-1,0,1) or NA that indicates the outcome of the perturbation.
edges	logical vector indicating which edges to plot.
smooth	double in the range [0,1] controlling the level of smoothing applied.

Details

This control constructs density plots that show the distribution of selected edge weights for the cases that meet the selected validation criteria (blue), and those that do not (red), following a given perturbation.

The slider controls the level of smoothing of the densities. Edges are labelled by pairs of integers for compactness, where the integer codes correspond to the ordering of the node labels.

weight.density0 is a non-interactive variant for programmatic use.

---

weight.density.shiny *Shiny Weight Density Plots*

Description

Shiny app to display weights of valid and invalid matrices as a density plots

Usage

```
weight.density.shiny(sim, epsilon = 1e-05, main = "")
```

Arguments

sim	the result from system.simulate
epsilon	outomes below this in absolute magnitude are treated as zero.
main	text for plot title

Details

This control constructs density plots that show the distribution of selected edge weights for the cases that meet the selected validation criteria (blue), and those that do not (red), following a given perturbation.

The slider controls the level of smoothing of the densities. Edges are labelled by pairs of integers for compactness, where the integer codes correspond to the ordering of the node labels.

weight.density0 is a non-interactive variant for programmatic use.

Index