Package: rhype (via r-universe)

June 24, 2024

Title Work with Hypergraphs in R

Version 0.3.0.9000

Description Create and manipulate hypergraph objects. This early version of rhype allows for the output of matrices associated with the hypergraphs themselves. It also uses these matrices to calculate hypergraph spectra and perform spectral comparison. Functionality coming soon includes calculation of hyperpaths and hypergraph centrality measures.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.1

Imports igraph, Matrix, R6, RSpectra

Suggests covr, knitr, rmarkdown, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

Language en-US

VignetteBuilder knitr

Repository https://hwarden162.r-universe.dev

RemoteUrl https://github.com/hwarden162/rhype

RemoteRef HEAD

RemoteSha 53cb797b8d5c2bebacfeae5dce06c587a8987d63

Contents

`adjacency_matrix` *Find the Adjacency Matrix of a Hypergraph*

Description

An adjacency matrix is a square matrix with both rows and columns being indexed by vertices. For each entry, the number is proportional to the strength of the connection going from the vertex represented as the row and the vertex represented by the column. For undirected hypergraphs, this matrix is symmetric but this is usually not the case for directed.

Usage

```
adjacency_matrix(hype, normalise = FALSE, self_adj = FALSE, as_matrix = TRUE)
```

Arguments

hype	A hypergraph object
normalise	Whether the matrix should be normalised to either 1 or 0
self_adj	Whether self adjacency should be represented
as_matrix	Whether the output should be coerced into a simple matrix

Details

Great care should be taken when using a hypergraph with mixed positive and negative real coefficients as there is a chance no adjacency will be registered for two adjacenct vertices. rhype does not check for these cases and they must be checked for by the user.

Value

A matrix of adjacencies between vertices of a hypergraph.

Examples

```
h1 <- example_hype()
adjacency_matrix(h1)

h2 <- example_hype(oriented = TRUE, directed = TRUE)
adjacency_matrix(h2)
```

---

bootstrap_hype *Bootstrap A Hypergraph*

Description

Bootstrapping is a common statistical technique used to quantify uncertainty of calculations. This is an approximation of the bootstrap algorithm for hypergraphs. Bootstrapping is achieved by creating a new hypergraph where the vertices, hyperedges or both have themselves been bootstrapped, achieved using the `"vertex"`, `"hyperedge"` or `"both"` methods.

Usage

```
bootstrap_hype(hype, n = 1, method = "both")
```

Arguments

hype	A hypergraph object.
n	The number of bootstrapped hypergraphs required.
method	What method to use to calculate the bootstrapped hypergraphs

Value

A list of bootstrapped hypergraphs.

Examples

```
hype <- example_hype()
resamples <- bootstrap_hype(hype, n = 5)
lapply(resamples, incidence_matrix)
```

`cardinality` *Find The Cardinality Of Hyperedges In A Hypergraph*

Description

The cardinality of a hyperedge is the number of vertices that it contains.

Usage

```
cardinality(hype)
```

Arguments

`hype`	A hypergraph object.

Value

A vector of the cardinality of the hyperedges.

Examples

```
hype <- example_hype()
cardinality(hype)
```

`connectivity_graph` *Find The Connectivity Graph Of A Hypergraph*

Description

The connectivity graph is a graphical representation of a hypergraph, it has a vertex for each vertex and hyperedge in the hypergraph. Two hyperedges are connected in the connectivity graph if they both have a vertex in common, a vertex is connected to a hyperedge if the vertex is contained in the hyperedge and no vertices are connected by edges.

Usage

```
connectivity_graph(hype)
```

Arguments

hype	A hypergraph object.

Value

A graph object representing the hyperedge connectivity graph.

Examples

```
hype <- example_hype()
g <- connectivity_graph(hype)
print(g)
plot(g)
```

---

degree *Find the Degree of Vertices in a Hypergraph*

Description

The degree of a vertex is a way of expressing how many connections there are from a vertex to the rest of the hypergraph. The current version of rhype has three methods for computing degree.

Usage

```
degree(hype, method = NA)
```

Arguments

hype	A hypergraph object
method	The method for calculating degree. Out of `"vertex"`, `"vertex_simple"`, `"hyperedge"` and `"hyperedge_simple"`

Details

`"vertex"` counts the number of ways it is possible to move to another vertex. If there are multiple hyperedges connecting two vertices, then each of these hyperedges will be counted as a new way to move between these two vertices. For weighted hypergraphs or hypergraphs with real coefficients, the strength of connection between two vertices is a functions of the weights and real coefficients.

`"vertex_simple"` just counts the number of vertices it is possible to reach in one step from the given vertex, no matter how many hyperedges connect them.

`"hyperedge"` represents the strength with which a vertex connects with itself through the hyperedges it is a member of. This is taken from the work of Jurgen Jost and Raffaella Mulas doi:10.1016/j.aim.2019.05.025. For unweighted hypergraphs without real coefficients this is equivalent to `"hyperedge_simple"`.

`"hyperedge_simple"` just counts the number of hyperedges a vertex is a member of.

Value

A vector representing the degree of each vertex with respect to the given method.

Examples

```
h1 <- example_hype()
degree(h1)
```

`dual_hype` *Get The Dual Of A Hypergraph*

Description

The dual of a hypergraph has a vertex for each original hyperedge and a hyperedge for each original vertex. A vertex is a member of a hyperedge if the original hyperedge has the original vertex as a member.

Usage

```
dual_hype(hype)
```

Arguments

`hype`	A hypergraph object.

Value

A hypergraph object representing the dual of the hypergraph.

Examples

```
hype <- example_hype()
dual_hype(hype)
```

`eigenvector_centrality` *Calculate The Eigenvector Centrality Of A Hypergraph*

Description

To calculate the eigenvector centrality of a hypergraph, each vertex is assigned a value that is proportional to the sum of the value of its neighbours.

Usage

```
eigenvector_centrality(hype)
```

Arguments

hype	A hypergraph object

Value

A vector of values representing the eigenvector centrality of each node

Examples

```
h1 <- example_hype()
eigenvector_centrality(h1)
```

---

eigenvector_centrality_factor

Calculate The Eigenvector Centrality Scaling Factor Of A Hypergraph

---

Description

To calculate the eigenvector centrality of a hypergraph, each vertex is assigned a value that is proportional to the sum of the value of its neighbours. This function gives the scaling factor relating the value of each node to the sum of the value of its neighbours.

Usage

```
eigenvector_centrality_factor(hype)
```

Arguments

hype	A hypergraph object

Value

A number representing the scaling factor relating the value of each node to the sum of the value of its neighbours

Examples

```
h1 <- example_hype()
eigenvector_centrality_factor(h1)
```

example_hype *Generate an Example Hypergraph*

Description

Quickly generate an example hypergraph. Can be used for quickly testing and trialing examples.

Usage

```
example_hype(
  oriented = FALSE,
  directed = FALSE,
  vertex_weighted = FALSE,
  edge_weighted = FALSE,
  real_coef = FALSE
)
```

Arguments

oriented	Logical value representing whether the example hypergraph should be oriented
directed	Logical value representing whether the example hypergraph should be directed
vertex_weighted	Logical value representing whether the example hypergraph should have vertex weights
edge_weighted	Logical value representing whether the example hypergraph should have hyperedge weights
real_coef	Logical value representing whether the example hypergraph should have real coefficients relating vertices to hyperedges

Value

An example hypergraph with the given properties

Examples

```
h1 <- example_hype()
h2 <- example_hype(oriented = TRUE)
h3 <- example_hype(oriented = TRUE, directed = TRUE)
h4 <- example_hype(oriented = TRUE, directed = TRUE, real_coef = TRUE)
```

has_real_coef *Does a Hypergraph Have Real Coefficients*

Description

Takes a hypergraph object and returns whether there are real coefficients associating vertices to hyperedges.

Usage

```
has_real_coef(hype)
```

Arguments

hype	A hypergraph object.

Value

A logical value indicating whether there are real cofficients associating vertices to hyperedges.

Examples

```
h <- example_hype()
has_real_coef(h)
```

hyperedge_list *Get Hyperedge List*

Description

Take a hypergraph object and return its hyperedge list.

Usage

```
hyperedge_list(hype)
```

Arguments

hype	A hypergraph object

Value

A hyperedge list. See main documentation for more details on its structure

Examples

```
h <- example_hype()
hyperedge_list(h)
```

hyperedge_names *Get Hyperedge Names*

Description

Takes a hypergraph object and returns the names of the hyperedges.

Usage

```
hyperedge_names(hype)
```

Arguments

hype	A hypergraph object.

Value

A vector of strings representing the names of the the hyperedges. If the hyperedges have no names assocaited with them it will return NULL instead.

Examples

```
h <- example_hype()
hyperedge_names(h)
```

hyperedge_weights *Get Hyperedge Weights*

Description

Takes a hypergraph object and returns the weights associated with each hyperedge

Usage

```
hyperedge_weights(hype)
```

Arguments

hype	A hypergraph object.

Value

A vector of weights asssociated with the hyperedges. If the are no weights assicated with the hyperedges then NULL is returned instead.

Examples

```
h <- example_hype()
hyperedge_weights(h)
```

hype_from_edge_list *Create a Hypergraph From a Hyperedge List*

Description

Create a Hypergraph From a Hyperedge List

Usage

```
hype_from_edge_list(elist, directed = FALSE)
```

Arguments

elist	A hyperedge list. For an unoriented hypergraph, a hyperedge is just a vector of the vertices contained within the hyperedge. Each vertex is represented as a string. For an oriented hypergraph, each hyperedge is itself a list of two vectors. Each of these vectors contains strings representing the vertices contained in one end of the hyperedge. For a directed hypergraph, each hyperedge is also a list of two vectors. In the directed case, the first vector represents the vertices contained in the tail of the hyperedge and the second the vertices contained in the head. These two entries are also named `from` and `to`.
directed	A logical value representing whether the hypergraph should be directed.

Value

A hypergraph object with the given hyperedge structure.

Examples

```
l1 <- list(
  h1 = c("a", "b", "c"),
  h2 = c("c", "d", "e"),
  h3 = c("a", "e")
)
hype1 <- hype_from_edge_list(l1)

l2 <- list(
  h1 = list(
    c("a", "b"),
    c("b", "c")
  ),
  h2 = list(
    c("b", "c", "d"),
    c("e", "f")
  ),
  h3 = list(
    "f",
    "a"
  )
```

```
)
hype2 <- hype_from_edge_list(l2)
hype3 <- hype_from_edge_list(l2, directed = TRUE)
```

`hype_from_inc_mat` *Create a Hypergraph From an Incidence Matrix*

Description

Create a Hypergraph From an Incidence Matrix

Usage

```
hype_from_inc_mat(inc_mat, directed = FALSE, real_coef = FALSE)
```

Arguments

`inc_mat`	An incidence matrix or, for an oriented hypergraph, a list of two incidence matrices.
`directed`	A logical value representing whether the hypergraph should be directed.
`real_coef`	A logical value representing whether the hypergraph should have real coefficients associating vertices to hyperedges.

Value

A hypergraph object with the given incidence structure.

Examples

```
i1 <- matrix(
  c(1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0),
  nrow = 5,
  ncol = 3,
  dimnames = list(
    paste0("v", 1:5),
    paste0("h", 1:3)
  )
)
hype1 <- hype_from_inc_mat(i1)

i2 <- list(
  matrix(
    c(1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0),
    nrow = 4,
    ncol = 3,
    dimnames = list(
      paste0("v", 1:4),
      paste0("h", 1:3)
    )
```

```
  ),
  matrix(
    c(0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0),
    nrow = 4,
    ncol = 3,
    dimnames = list(
      paste0("v", 1:4),
      paste0("h", 1:3)
    )
  )
)
hype2 <- hype_from_inc_mat(i2)
hype3 <- hype_from_inc_mat(i2, directed = TRUE)
```

`hype_info` *Print More Detail About a Hypergraph*

Description

Get a more detailed printout of what is contained within a hypergraph object to understand more about its structure as a whole without having to repeatedly call other functions.

Usage

```
hype_info(
  hype,
  numv = TRUE,
  elist = TRUE,
  vnames = TRUE,
  vweights = TRUE,
  enames = TRUE,
  eweights = TRUE,
  weighted = TRUE,
  oriented = TRUE,
  directed = TRUE,
  real_coef = TRUE,
  inc_mat = TRUE
)
```

Arguments

`hype`	A hypergraph object
`numv`	A logical variable indicating whether information about the number of vertices should be printed
`elist`	A logical variable indicating whether information about the hyperedge list should be printed
`vnames`	A logical variable indicating whether information about the vertex names should be printed

vweights	A logical variable indicating whether information about the vertex weights should be printed
enames	A logical variable indicating whether information about the hyperedge names should be printed
eweights	A logical variable indicating whether information about the hyperedge weights should be printed
weighted	A logical variable indicating whether information about the hypergraph weighting should be printed
oriented	A logical variable indicating whether information about the hypergraph orientation should be printed
directed	A logical variable indicating whether information about the hypergraph direction should be printed
real_coef	A logical variable indicating whether information about the hypergraph real coefficients should be printed
inc_mat	A logical variable indicating whether information about the hypergraph incidence matrix should be printed

Details

This gives a more detailed look at the whole hypegraph object. It is intended solely to aid the user when using rhype and generally should not be included in final scripts. If a user wants to include this in their final script it is instead heavily encouraged that they use other rhype functions to generate their own bespoke messages.

Examples

```
hype1 <- example_hype()
hype_info(hype1)

hype2 <- example_hype(vertex_weighted = TRUE, edge_weighted = TRUE)
hype_info(hype2)

hype3 <- example_hype(oriented = TRUE, directed = TRUE, real_coef = TRUE)
hype_info(hype3)
```

---

hype_norm_lap_mat *Find the Hyperedge Normalised Laplacian Matrix of a Hypergraph*

Description

As defined by Jurgen Jost and Raffaella Mulas doi:10.1016/j.aim.2019.05.025

Usage

```
hype_norm_lap_mat(hype, as_matrix = TRUE)
```

Arguments

hype	A hypergraph object
as_matrix	Whether to coerce the result to a simple matrix

Value

The hyperedge normalised laplacian matrix of the hypergraph

Examples

```
h1 <- example_hype()
hype_norm_lap_mat(h1)
```

---

hype_order *Get The Order Of A Hypergraph*

Description

The order of a hypergraph is the number of vertices it has

Usage

```
hype_order(hype)
```

Arguments

hype	A hypergraph object

Value

A number representing the number of vertices in the hypergraph

Examples

```
hype <- example_hype()
hype_order(hype)
```

hype_size *Get The Size Of A Hypergraph*

Description

The size of a hypergraph is the number of hyperedges it contains

Usage

```
hype_size(hype)
```

Arguments

hype	A hypergraph object

Value

A number representing the number of hyperedges in a hypergraph

Examples

```
h1 <- example_hype()
hype_size(h1)
```

incidence_matrix *Find the Incidence Matrix of a Hypergraph*

Description

An incidence matrix has rows indexed by vertices and columns indexed by hyperedges. Each entry is non-zero if the associated vertex is a member of the associated hyperedge. For an oriented hypergraph, this returns a list of two matrices with the first representing incidence to one end of the hyperedges and the second representing incidence to the other end. For a directed hypergraph the first represents incidence to the tail of a hyperedge and the second represents incidence to the head.

Usage

```
incidence_matrix(hype, augment_oriented = TRUE, as_matrix = TRUE)
```

Arguments

hype	A hypergraph object
augment_oriented	Whether to augment an oriented hypergraph
as_matrix	Whether to coerce the result to a simple matrix

Details

It is hard to use the incidence matrices of oriented undirected hypergraphs in calculations. The `augment_oriented` option turns the hypergraph into a directed hypergraph, but each hyperedge is represented twice, once pointing in each direction. This is much easier to use for further calculations.

Value

An incidence matrix or a list of two incidence matrices.

Examples

```
h1 <- example_hype()
incidence_matrix(h1)

h2 <- example_hype(oriented = TRUE, directed = TRUE)
incidence_matrix(h2)
```

`is_directed` *Is a Hypergraph Directed*

Description

Takes a hypergraph object and returns whether the hyperedges are directed.

Usage

```
is_directed(hype)
```

Arguments

`hype`	A hyeprgraph object.

Value

A logical value indicating whether the hyperedges are directed.

Examples

```
h <- example_hype()
is_directed(h)
```

is_oriented *Is a Hypergraph Oriented*

Description

Takes a hypergraph object and returns whether the hyperedges are oriented.

Usage

```
is_oriented(hype)
```

Arguments

hype	A hypergraph object.

Value

A logical value indicating whether the hyperedges are oriented.

Examples

```
h <- example_hype()
is_oriented(h)
```

is_weighted *Is a Hypergraph Weighted*

Description

Takes a hypergraph object and returns whether a hypergraph has weights associated with its vertices or hyperedges.

Usage

```
is_weighted(hype)
```

Arguments

hype	A hypergraph object.

Value

A logical value indicating whether the hypergraph has weights associated with its vertices or hyperedges.

Examples

```
h <- example_hype()
is_weighted(h)
```

jackknife_hype *Jackknife A Hypergraph*

Description

Jackknifing is a resampling technique similar to bootstrapping, where many resamples are taken, each time leaving out one observation. For the abstraction to hypergraphs, the `"vertex"` method recreates the hypergraph leaving out one vertex, the `"hyperedge"` method recreates the hypergraph leaving out one hyperedge and the `"both"` method leaves out one of each.

Usage

```
jackknife_hype(hype, n = 1, method = "both")
```

Arguments

`hype`	A hypergraph object.
`n`	The number of hypergraphs to create.
`method`	The method to use to jackknife the hypergraphs.

Value

A list of jackknifed hypergraphs.

Examples

```
hype <- example_hype()
resamples <- jackknife_hype(hype, n = 5)
lapply(resamples, incidence_matrix)
```

laplacian_matrix *Find the Laplacian Matrix of a Hypergraph*

Description

Find the Laplacian Matrix of a Hypergraph

Usage

```
laplacian_matrix(hype, as_matrix = TRUE)
```

Arguments

`hype`	A hypergraph object
`as_matrix`	Whether to coerce the result to a simple matrix

Value

The laplacian matrix of the hypergraph

Examples

```
h1 <- example_hype()
laplacian_matrix(h1)
```

`max_cardinality` *Find The Maximum Cardinality Of A Hyperedge In A Hypergraph*

Description

The cardinality of a hyperedge is the number of vertices that it contains.

Usage

```
max_cardinality(hype)
```

Arguments

`hype`	A hypergraph object.

Value

The value of the maximum cardinality of a hyperedge in the hypergraph.

Examples

```
hype <- example_hype()
max_cardinality(hype)
```

`max_degree` *Find The Maximum Degree Of A Hypergraph*

Description

The degree of a vertex is a way of expressing how many connections there are from a vertex to the rest of the hypergraph. See the `degree` help documentation for more details.

Usage

```
max_degree(hype, method = NA)
```

Arguments

hype	A hypergraph object.
method	The method by which to calculate the degree, see degree help documentation for more information.

Value

The maximum value for the chosen degree among vertices of the hypergraph.

Examples

```
h <- example_hype()
max_degree(h)
```

---

min_cardinality *Find The Minimum Cardinality Of A Hyperedge In A Hypergraph*

Description

The cardinality of a hyperedge is the number of vertices that it contains.

Usage

```
min_cardinality(hype)
```

Arguments

hype	A hypergraph object.

Value

The value of the minimum cardinality of a hyperedge in the hypergraph.

Examples

```
hype <- example_hype()
min_cardinality(hype)
```

min_degree *Find The Minimum Degree Of A Hypergraph*

Description

The degree of a vertex is a way of expressing how many connections there are from a vertex to the rest of the hypergraph. See the degree help documentation for more details.

Usage

```
min_degree(hype, method = NA)
```

Arguments

hype	A hypergraph object.
method	The method by which to calculate the degree, see degree help documentation for more information.

Value

The maximum value for the chosen degree among vertices of the hypergraph.

Examples

```
h <- example_hype()
min_degree(h)
```

partial_hype *Generate A Partial Hypergraph*

Description

A partial hypergraph can be induced from a set of hyperedges. The partial hypergraph has all of the original vertices, but only the hyperedges used to induce it.

Usage

```
partial_hype(hype, hyperedges)
```

Arguments

hype	A hypergraph object.
hyperedges	A vector of the names of the hyperedges to be used to induce the partial hypergraph.

Value

A hypergraph object of the partial hypergraph.

Examples

```
hype <- example_hype()
partial_hype(hype, c("h1", "h2"))
```

---

pseudo_invert *Pseudo-Invert a Vector*

Description

Pseudo-inversion is where a vector has each non-zero element inverted and each zero element remains untouched. This is useful for pseudoinverting matrices that only have non-zero entries on the leading diagonal.

Usage

```
pseudo_invert(vec)
```

Arguments

vec	A vector of numbers

Value

A vector of pseudo-inverted numbers

---

shortest_hyperpaths *Find The Shortest Hyperpaths Between Two Vertices*

Description

A hyperpath is a set of hyperedges such that each consecutive pair of hyperedges contain at least vertex in common. A shortest hyperpath between two vertices is the smallest set of hyperedges that form a hyperpath such that one vertex is in the first hyperpath and the other vertex is in the last hyperpath.

Usage

```
shortest_hyperpaths(hype, from, to)
```

Arguments

hype	A hypergraph object.
from	The vertex that is the start of the hyperpath.
to	The vertex that is the end of the hyperpath,

Value

A list of shortest hyperpaths between the given vertices.

Examples

```
hype <- example_hype()
shortest_hyperpaths(hype, "v1", "v4")
```

---

shuffle_hype *Shuffle A Hypergraph*

Description

A hypergraph can be shuffled to slightly perturb its structure. These shuffled hypergraphs can then be used to estimate the uncertainty of calculations on the original hypergraph.

Usage

```
shuffle_hype(hype, n = 1, method = "hyperedge")
```

Arguments

hype	A hypergraph object.
n	The number of shuffled hypergraphs to calculate.
method	The method to use to shuffle the hypergraph.

Details

Two methods are used to shuffle a hypergraph, the "vertex" method keeps the degree of each vertex the same, randomly reassigning the hyperedges they are members of. The "hyperedge" method keeps the cardinality of each hyperedge the same, randomly reassigning the vertices that are members.

Value

A list of shuffled hypergraphs.

Examples

```
hype <- example_hype()
resamples <- shuffle_hype(hype, n = 5)
lapply(resamples, incidence_matrix)
```

spectra *Find the Spectra of a Hypergraph*

Description

Find the Spectra of a Hypergraph

Usage

```
spectra(hype, matrix = "laplacian", n = NULL)
```

Arguments

hype	A hypergraph object
matrix	The matrix to calculate the spectra with respect to. Out of "laplacian", "adjacency", "vert_norm_lap_mat" and "hype_norm_lap_mat"
n	The number of eigenvalues or eigenvectors to calculate. If left empty or as NULL all will be calculated.

Value

The eigen decomposition of the given matrix of the given hypergraph

Examples

```
h <- example_hype()
spectra(h)
```

spectral_distance *Find the Spectral Distance Between Two Hypergraphs*

Description

Find the Spectral Distance Between Two Hypergraphs

Usage

```
spectral_distance(hype1, hype2, matrix = "laplacian")
```

Arguments

hype1	A hypergraph object
hype2	A hypergraph object
matrix	The matrix to calculate the spectral distance with respect to. Out of "laplacian", "adjacency", "vert_norm_lap_mat" and "hype_norm_lap_mat"

Value

A number representing the spectral distance between the two hypergraphs with respect to the given matrix

Examples

```
h1 <- example_hype()
h2 <- example_hype()
spectral_distance(h1, h2)
```

`spectral_distance_disc` *Find the Spectral Distance From the Fully Disconnected Hypergraph*

Description

Find the Spectral Distance From the Fully Disconnected Hypergraph

Usage

```
spectral_distance_disc(hype, matrix = "vert_norm_lap_mat")
```

Arguments

`hype`	A hypergraph object
`matrix`	The matrix to calculate the spectra with respect to. Out of `"vert_norm_lap_mat"` and `"hype_norm_lap_mat"`

Value

The spectral distance from the disconnected hypergraph

Examples

```
h <- example_hype()
spectral_distance_disc(h)
```

support_graph *Find The Support Graph Of A Hypergraph*

Description

The support graph of a hypergraph is a graph that has a vertex for every vertex in the hypergraph. Two vertices are connected in the support graph if there is a hyperedge that connects them.

Usage

```
support_graph(hype, simple = TRUE)
```

Arguments

hype	A hypergraph object.
simple	Whether a simplified support graph should be created.

Details

If simple is set to FALSE then for unweighted hypergraphs without real coefficients the support graph has an edge connected vertices for each hyperedge connecting them in the hypergraph.

Value

The support graph of the hypergraph

Examples

```
hype <- example_hype()
g <- support_graph(hype)
print(g)
plot(g)
```

validate_hypergraph *Quickly Validate a Hypergraph*

Description

When using the rhype functions, the integrity of a hypergraph object should remain intact. However, as the properties of a hypergraph object are dependent on one another, it is possible in the case of an error or direct object manipulation by the user that a hypergraph object's integrity is corrupted. This will cause other rhype functions to either throw errors or to calculate incorrect answers. This function is not exhaustive but will perform multiple sanity checks on hypergraph objects and is a good place to start when debugging.

Usage

```
validate_hypergraph(hype, return = FALSE, verbose = TRUE)
```

Arguments

hype	A hypergraph object
return	A logical variable stating whether any output should be returned from the function
verbose	A logical variable indicating whether the function should output text to the screen

Value

Outputs text to screen of any problems found within the hypergraph object. If return is set to TRUE then a logical output will be returned. This logical output will be TRUE if it passed all of the tests, FALSE if it failed any test that proves the structure of the hypergraph is broken or NULL if it failed a test that most hypergraphs used practically should pass, but doesn't necessarily mean the hypergraph is broken, see text output for more details.

Examples

```
h <- example_hype()
validate_hypergraph(h)
```

---

vertex_names *Get Vertex Names*

Description

Takes a hypergraph object and returns the names of its vertices.

Usage

```
vertex_names(hype)
```

Arguments

hype	A hypergraph object.

Value

A vector of strings of vertex names

Examples

```
h <- example_hype()
vertex_names(h)
```

vertex_weights *Get Vertex Weights*

Description

Takes a hypergraph object and returns the weights associated with its vertices.

Usage

```
vertex_weights(hype)
```

Arguments

hype	A hypergraph object.

Value

A vector of weights associated with each vertex. If the hypergraph has no weights associated with its vertices it will return NULL instead.

Examples

```
h <- example_hype()
vertex_weights(h)
```

vert_norm_lap_mat *Find the Vertex Normalised Laplacian Matrix of a Hypergraph*

Description

As defined by Jurgen Jost and Raffaella Mulas doi:10.1016/j.aim.2019.05.025

Usage

```
vert_norm_lap_mat(hype, as_matrix = TRUE)
```

Arguments

hype	A hypergraph object
as_matrix	Whether to coerce the result to a simple matrix

Value

The vertex normalised laplacian matrix of the hypergraph

Examples

```
h1 <- example_hype()
vert_norm_lap_mat(h1)
```

Index