Package 'tma'

September 19, 2025

Title Transmodal Analysis (TMA)

Type Package

Author Cody L Marquart [aut, cre] (ORCID:
<https://orcid.org/0000-0002-3387-6792>),
Muhammad Hasnat Ashiq [aut],
David Williamson Shaffer [aut]

Maintainer Cody L Marquart <cody.marquart@wisc.edu>

Version 0.3.1

Encoding UTF-8

Description A robust computational framework for analyzing complex multimodal data. Extends existing state-dependent models to account for diverse data streams, addressing challenges such as varying temporal scales and learner characteristics to improve the robustness and interpretability of findings. For methodological details, see Shaffer, Wang, and Ruis (2025) ``Transmodal Analysis'' <doi:10.18608/jla.2025.8423>.

LazyData TRUE

Depends R (>= 3.6), data.table

License GPL-3

Imports stats, methods, rlang, Rcpp

Suggests testthat (>= 2.1.0), rstudioapi, knitr, rmarkdown, jsonlite

RoxygenNote 7.3.2

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Repository CRAN

Date/Publication 2025-09-19 13:50:02 UTC

Contents

`accumulate` *Accumulate Connections from a Multidimensional Array and Context Model*

Description

This function processes a context model and a multidimensional array of window/weight parameters to compute connection counts for each unit of analysis. It applies the context model to the array, using sender, receiver, and mode columns (as defined in the array attributes), and accumulates co-occurrence or adjacency matrices for each unit. The result is a set of connection counts and row-level connection matrices, suitable for network analysis (e.g., ENA/ONA).

Usage

```
accumulate(
  context_model,
  codes,
  tensor = context_tensor(context_model$model$raw.input),
  time_column = NULL,
  ordered = FALSE,
  binary = TRUE
)
```

Arguments

`context_model`	A context model object (as produced by 'tma::contexts') containing contexts for each unit of analysis.
`codes`	Character vector of code names to use for constructing adjacency matrices.
`tensor`	A multidimensional array (see 'context_tensor') containing window and weight values for each sender/receiver/mode combination. Defaults to an array generated from the context model's raw input.
`time_column`	Character string giving the name of the time column in the context model. If NULL, uses the default context column ID.
`ordered`	Logical; if TRUE (default), computes ordered adjacency matrices (ONA); if FALSE, computes unordered (ENA-style) matrices.
`binary`	Logical; if TRUE (default), binarizes the connection counts (not currently implemented in this function).

Details

This function is used to perform accumulation of network connections for each unit, based on the context model and tensor parameters. It supports both ordered and unordered accumulation, and returns results suitable for further network analysis or visualization.

Value

The input ‘context_model‘ with additional fields:

`connection.counts`
: A data.table of accumulated connection counts for each unit.

`model$row.connection.counts`
: A data.table of row-level connection matrices for each unit.

`meta.data`
: A data.table of metadata columns for each unit.

The class of the returned object is updated to reflect the type of accumulation (ordered or unordered).

`accumulate_contexts` *accumulate_contexts*

Description

accumulate_contexts

Usage

```
accumulate_contexts(
  x,
  codes,
  decay.function = decay(simple_window, window_size = 4),
  time.column = NULL,
  mode.column = NULL,
  mask = NULL,
  weight.by = sqrt,
  norm.by = `_sphere_norm`,
  meta.data = NULL,
  return.dena.set = FALSE,
  return.ena.set = TRUE,
  context_filter = NULL,
  summarize_ground_using = colSums,
  calculate_adj_vectors_using = ground_response_crossprod,
  ground_effect_function = function(x, y) crossprod(t(x), y),
  accumulate_unit_vectors_by = colSums
)
```

Arguments

`x`	TBD
`codes`	TBD
`decay.function`	TBD
`time.column`	TBD
`mode.column`	TBD

mask	TBD
weight.by	TBD
norm.by	TBD
meta.data	TBD
return.dena.set	TBD
return.ena.set	TBD
context_filter	TBD
summarize_ground_using	TBD
calculate_adj_vectors_using	TBD
ground_effect_function	TBD
accumulate_unit_vectors_by	TBD

Value

ENA set

`accumulate_networks` *fast accumulate networks*

Description

fast accumulate networks

Usage

```
accumulate_networks(
  x,
  code_cols,
  decay_function,
  time_col = -1L,
  ordered = TRUE
)
```

Arguments

x	TBD
code_cols	TBD
decay_function	TBD
time_col	TBD
ordered	TBD

accumulate_threads *accumulate_threads*

Description

accumulate_threads

Usage

```
accumulate_threads(
  data,
  units_by,
  conversation_rules,
  code_cols,
  ...,
  conversation_splits = NULL,
  as_directed = FALSE,
  window_size = 4,
  meta_data = units_by
)
```

Arguments

data	TBD
units_by	TBD
conversation_rules	TBD
code_cols	TBD
...	TBD
conversation_splits	TBD
as_directed	TBD
window_size	TBD
meta_data	TBD

Value

TBD

adjacency_key *Adjacency Key*

Description

Adjacency Key

Usage

```
adjacency_key(codes, upper = TRUE)
```

Arguments

codes	character vector
upper	logical, default, TRUE, returns the key for the upper triangle, FALSE - not implemented yet

Value

matrix

apply_tensor *Apply windowing and weighting to context data for network accumulation (C++ backend)*

Description

This function implements the core logic for accumulating network connections using a multidimensional parameter array (context_tensor), efficiently applying window and weight parameters to each response line in a unit's context. It is designed for use in the TMA package to speed up accumulation calculations by leveraging C++ and Armadillo for matrix operations.

Usage

```
apply_tensor(
  tensor,
  dims,
  dims_sender,
  dims_receiver,
  dims_mode,
  context_matrix,
  unit_rows,
  codes,
  times,
  ordered = TRUE
)
```

Arguments

tensor	NumericVector. The multi-dimensional context_tensor array created in R can be supplied as-is; it is automatically converted to a 1D vector when passed to this function via Rcpp.
dims	IntegerVector. The dimensions of the original context_tensor array.
dims_sender	std::vector<int>. Indices of sender dimensions in the array.
dims_receiver	std::vector<int>. Indices of receiver dimensions in the array.
dims_mode	std::vector<int>. Indices of mode dimensions in the array.
context_matrix	NumericMatrix. Matrix representation of the context for a single unit (rows = context lines, columns = factors).
unit_rows	std::vector<int>. Indices of the response rows for the unit in the context.
codes	arma::mat. Matrix of codes (nrow = context lines, ncol = number of codes).
times	NumericVector. Vector of time values for each context line.
ordered	bool. If TRUE, returns a full adjacency matrix; if FALSE, returns only upper triangle (ENA style).

Value

A list with two elements: - row_connection_counts: A matrix of connection counts for each response line (rows = response lines, columns = connections). - connection_counts: A vector of accumulated connection counts for the unit (length = number of connections).

---

as.character.adjacency.key

Convert Adjacency Key to Character (S3 method)

Description

This S3 method converts an adjacency key object (typically a 2-row matrix or list of pairs) into a character vector, concatenating each pair with ' & '.

Usage

```
## S3 method for class 'adjacency.key'
as.character(x, ...)
```

Arguments

x	An adjacency key object (matrix or list) to convert to character.
...	Additional arguments (unused).

Value

A character vector where each element is a concatenation of the adjacency key pair.

as.double.adjacency.key *Convert Adjacency Key to Double (S3 method)*

Description

This S3 method converts an adjacency key object to a numeric (double) vector, applying as.numeric to each element.

Usage

```
## S3 method for class 'adjacency.key'
as.double(x, ...)
```

Arguments

x	An adjacency key object (matrix or list) to convert to numeric.
...	Additional arguments (unused).

Value

A numeric vector representation of the adjacency key.

as.matrix.ena.matrix *Matrix without metadata*

Description

Matrix without metadata

Usage

```
## S3 method for class 'ena.matrix'
as.matrix(x, ...)
```

Arguments

x	Object to convert to a matrix
...	additional arguments to be passed to or from methods

Value

matrix

as.matrix.network.connections *Convert Network Connections to Matrix (S3 method)*

Description

This S3 method extracts the connection columns from a network connections object and returns them as a numeric matrix. It is used to facilitate matrix operations on network connection data.

Usage

```
## S3 method for class 'network.connections'
as.matrix(x, ...)
```

Arguments

x	An object of class "network.connections" (or compatible data.table/data.frame) containing connection columns (of class "network.connection").
...	Additional arguments passed to 'as.matrix'.

Value

A numeric matrix of network connections (rows = units/contexts, columns = connections).

as.network.connection *Re-class vector as network.connection*

Description

Re-class vector as network.connection

Usage

```
as.network.connection(x)
```

Arguments

x	Vector to re-class

Value

re-classed vector

as.qe.code *Convert a vector to 'qe.code' class*

Description

This function converts a vector to the 'qe.code' class. If the vector is a factor, it is first converted to a character vector.

Usage

```
as.qe.code(x)
```

Arguments

x	A vector. The vector to be converted to 'qe.code' class.

Value

The modified vector with the 'qe.code' class.

Examples

```
vec <- factor(c("A", "B", "C"))
vec <- as.qe.code(vec)
class(vec) # Should show 'qe.code' along with other classes
```

as.qe.data *Convert an object to 'qe.data' class*

Description

This function converts an object to the 'qe.data' class. If the object is not a data.frame or matrix, it is first converted to a data.table.

Usage

```
as.qe.data(x)
```

Arguments

x	An object. The object to be converted to 'qe.data' class.

Value

The modified object with the 'qe.data' class.

Examples

```
library(data.table)

dt <- data.table(
  ID = 1:5,
  Name = c("Alice", "Bob", "Charlie", "David", "Eve"),
  Age = c(25, 30, 35, 40, 45),
  Score = c(85, 90, 95, 80, 75)
)
dt <- as.qe.data(dt);
class(dt) # Should show 'qe.data' along with other classes
```

---

as.qe.horizon *Convert a vector to 'qe.horizon' class*

Description

This function converts a vector to the 'qe.horizon' class. If the vector is a factor, it is first converted to a character vector.

Usage

```
as.qe.horizon(x)
```

Arguments

x	A vector. The vector to be converted to 'qe.horizon' class.

Value

The modified vector with the 'qe.horizon' class.

Examples

```
vec <- factor(c("A", "B", "C"))
vec <- as.qe.horizon(vec)
class(vec) # Should show 'qe.horizon' along with other classes
```

as.qe.metadata *Convert a vector to 'qe.metadata' class*

Description

This function converts a vector to the 'qe.metadata' class. If the vector is a factor, it is first converted to a character vector.

Usage

```
as.qe.metadata(x)
```

Arguments

x	A vector. The vector to be converted to 'qe.metadata' class.

Value

The modified vector with the 'qe.metadata' class.

Examples

```
vec <- factor(c("A", "B", "C"))
vec <- as.qe.metadata(vec)
class(vec) # Should show 'qe.metadata' along with other classes
```

as.qe.unit *Convert a vector to 'qe.unit' class*

Description

This function converts a vector to the 'qe.unit' class. If the vector is a factor, it is first converted to a character vector.

Usage

```
as.qe.unit(x)
```

Arguments

x	A vector. The vector to be converted to 'qe.unit' class.

Value

The modified vector with the 'qe.unit' class.

Examples

```
vec <- factor(c("A", "B", "C"))
vec <- as.qe.unit(vec)
class(vec) # Should show 'qe.unit' along with other classes
```

---

as.undirected.vector *Extract Upper Triangular Elements*

Description

This function extracts the elements from the upper triangular part of a matrix.

Usage

```
as.undirected.vector(x, diag = FALSE)
```

Arguments

x	A numeric matrix from which to extract upper triangular elements.
diag	A logical value indicating whether to include the diagonal elements. Defaults to FALSE.

Value

A vector containing the upper triangular elements of the matrix.

---

as.unordered *Convert to Unordered Factor*

Description

This function is a generic method to convert an object to an unordered factor. It dispatches methods based on the class of the input object.

Usage

```
as.unordered(x)
```

Arguments

x	An object to be converted to an unordered factor.

Value

An unordered factor representation of the input object.

as.unordered.default *Default Method for as.unordered*

Description

This function provides the default method for handling the input x when no specific method is available.

Usage

```
## Default S3 method:
as.unordered(x)
```

Arguments

x	Any object that you want to apply the default method to.

Value

The input object x, unchanged.

as.unordered.ordered.ena.connections *Unorder Connections in a Matrix*

Description

This function takes a matrix and creates an unordered version of its connections, combining upper and lower triangular elements.

Usage

```
## S3 method for class 'ordered.ena.connections'
as.unordered(x)
```

Arguments

x	A matrix or data frame containing the connections. The input should be a square matrix.

Value

A data.table with ordered connections, reclassified as "unordered.ena.connections", "ena.connections", and "ena.matrix".

as.unordered.ordered.row.connections

Convert Ordered Row Connections to Unordered (S3 method)

Description

This S3 method takes a matrix or data frame of ordered row connections (e.g., from ONA) and produces an unordered version by summing upper and lower triangular elements for each connection.

Usage

```
## S3 method for class 'ordered.row.connections'
as.unordered(x)
```

Arguments

x	An object of class "ordered.row.connections" (or compatible matrix/data.frame) containing ordered connection data. The input should be a square matrix or have square number of columns.

Value

A data.table with unordered row connections, reclassified as "unordered.row.connections", "row.connections", and "ena.matrix".

choose_two

n choose 2

Description

n choose 2

Usage

```
choose_two(n)
```

Arguments

n	int

`colSums.ena.matrix` *Column Sums for ENA Matrices (S3 method)*

Description

This S3 method computes column sums for ENA matrix objects, with optional binarization. It is used to summarize connection counts across rows (e.g., for each unit or context).

Usage

```
colSums.ena.matrix(x, na.rm = FALSE, dims = 1L, binary = FALSE)
```

Arguments

`x`	An object of class "ena.matrix" (or compatible matrix/data.frame) containing connection data.
`na.rm`	Logical; whether to remove missing values (passed to ‘colSums‘).
`dims`	Integer; which dimensions to sum over (passed to ‘colSums‘).
`binary`	Logical; if TRUE, binarizes the matrix before summing (i.e., all nonzero values become 1).

Value

A numeric vector of column sums for the matrix.

`contexts` *Create Contexts for Units of Analysis*

Description

This function generates context data for each unit of analysis in your dataset, applying subsetting rules ("hoo rules") and optional splitting rules to organize the data for network accumulation.

Usage

```
contexts(x, hoo_rules, units_by = NULL, split_rules = NULL)
```

Arguments

`x`	A data.frame or TMA model object containing the raw input data.
`hoo_rules`	A list of logical expressions (see [rules()]) specifying how to subset the data for each context/unit.
`units_by`	Character vector of column names to use for defining units of analysis (e.g., c("userID", "condition")).
`split_rules`	Optional. Either a function or an expression specifying how to further split each context (e.g., by time period or other grouping variable).

Details

This function is a core part of the TMA workflow. It first applies the specified ‘hoo_rules‘ to subset the data for each unit, then (optionally) applies ‘split_rules‘ to further divide each context. The resulting contexts are used in subsequent accumulation and network analysis steps.

Value

A TMA model object with updated contexts for each unit, where each context is a data.table containing only the relevant rows for that unit and context. The object includes attributes for unit labels and context row indices.

Examples

```
data(test_mockdata, package = "tma")
mock_data <- test_mockdata[test_mockdata$chatGroup == "PAM",]
unit_cols <- c("userID", "condition")
codes <- c("A", "B", "C")
HOO_rules_model <- rules(
  modality %in% "chat" & chatGroup %in% UNIT$chatGroup & condition %in% UNIT$condition,
  modality %in% "resource" & userID %in% UNIT$userID & condition %in% UNIT$condition
)

context_model <- contexts(
  x = mock_data,
  units = unit_cols,
  hoo_rules = HOO_rules_model
)
str(context_model$model$contexts)
```

---

context_tensor *Generate a multidimensional array for window and weight parameters*

Description

This function constructs a multidimensional array representing all combinations of sender(s), receiver(s), and mode(s), with an additional axis for weight and window parameters. The resulting array can be used to efficiently look up or modify window and weight values for each unique combination in your data, which is useful for network accumulation and modeling.

Usage

```
context_tensor(
  df,
  sender_cols = NULL,
  receiver_cols = NULL,
  mode_column = ATTR_NAMES$CONTEXT_ID,
  default_window = 1,
  default_weight = 1
)
```

Arguments

df	A data.frame containing the data to extract unique values for senders, receivers, and modes.
sender_cols	Character vector of column names in ‘df‘ to use as sender(s). Can be empty or NULL if not applicable.
receiver_cols	Character vector of column names in ‘df‘ to use as receiver(s). Can be empty or NULL if not applicable.
mode_column	Character string giving the column name in ‘df‘ to use as the mode (e.g., modality, channel). Can be empty or NULL if not applicable.
default_window	Numeric value to use as the default window for all combinations (default: 1).
default_weight	Numeric value to use as the default weight for all combinations (default: 1).

Value

A multidimensional array with dimensions [sender(s), receiver(s), mode(s), weight/window], where the last axis has two levels: "weight" and "window". The array is initialized with the default weight and window values, and has named dimensions for easy indexing.

Examples

```
df <- data.frame(sender = c("A", "B"), receiver = c("X", "Y"), mode = c("chat", "resource"))
arr <- context_tensor(df, sender_cols = "sender", receiver_cols = "receiver", mode_column = "mode")
arr["A", "X", "chat", "weight"] # Access the weight for sender A, receiver X, mode chat
```

---

conversation_rules *Conversation rules*

Description

Conversation rules

Usage

```
conversation_rules(...)
```

Arguments

...	list of rules

Value

callable expressions, see ‘rlang::exprs‘

decay *Internal: Decay function factory (legacy)*

Description

Internal factory for creating decay functions, used in TMA v0.1.0. Not exported. Kept for backward compatibility with legacy code.

Usage

```
decay(what, ...)
```

Arguments

what	Function to use as the decay kernel.
...	Named parameters to pass to ‘what‘.

Value

A function that applies the specified decay kernel to its input.

find_meta_cols *Find metadata columns*

Description

Find metadata columns

Usage

```
find_meta_cols(x)
```

Arguments

x	data.table (or frame) to search for columns of class ena.metadata

Value

logical vector

hoo *Apply a Subsetting Rule to TMA Contexts (Internal)*

Description

Internal helper to apply a logical subsetting rule ("hoo rule") to each unit's context in a TMA model object. Updates the contexts for each unit by including only rows that match the rule.

Usage

```
hoo(x, ..., rule = NULL)
```

Arguments

x	A TMA model object as produced by [units()].
...	Logical expression(s) specifying the subsetting rule to apply. If not provided, uses the 'rule' argument.
rule	A single logical expression to use as the subsetting rule (alternative to ...).

Value

The input TMA model object with updated contexts for each unit, where each context contains only rows matching the rule.

is.qe.code *Check if an object is of class 'qe.code'*

Description

This function checks if an object is of class 'qe.code'.

Usage

```
is.qe.code(x)
```

Arguments

x	An object. The object to be checked.

Value

A logical value. TRUE if the object is of class 'qe.code', otherwise FALSE.

Examples

```
dt <- 1:5
class(dt) <- c("qe.code", class(dt))
is.qe.code(dt) # Should return TRUE
```

is.qe.data *Check if an object is of class 'qe.data'*

Description

This function checks if an object is of class 'qe.data'.

Usage

```
is.qe.data(x)
```

Arguments

x	An object. The object to be checked.

Value

A logical value. TRUE if the object is of class 'qe.data', otherwise FALSE.

Examples

```
library(data.table)

dt <- data.table(ID = 1:5)
class(dt) <- c("qe.data", class(dt))
is.qe.data(dt) # Should return TRUE
```

is.qe.horizon *Check if an object is of class 'qe.horizon'*

Description

This function checks if an object is of class 'qe.horizon'.

Usage

```
is.qe.horizon(x)
```

Arguments

x	An object. The object to be checked.

Value

A logical value. TRUE if the object is of class 'qe.horizon', otherwise FALSE.

Examples

```
dt <- 1:5
class(dt) <- c("qe.horizon", class(dt))
is.qe.horizon(dt) # Should return TRUE
```

`is.qe.metadata` *Check if an object is of class 'qe.metadata'*

Description

This function checks if an object is of class 'qe.metadata'.

Usage

```
is.qe.metadata(x)
```

Arguments

`x`	An object. The object to be checked.

Value

A logical value. TRUE if the object is of class 'qe.metadata', otherwise FALSE.

Examples

```
dt <- 1:5
class(dt) <- c("qe.metadata", class(dt))
is.qe.metadata(dt) # Should return TRUE
```

`is.qe.unit` *Check if an object is of class 'qe.unit'*

Description

This function checks if an object is of class 'qe.unit'.

Usage

```
is.qe.unit(x)
```

Arguments

`x`	An object. The object to be checked.

Value

A logical value. TRUE if the object is of class 'qe.unit', otherwise FALSE.

Examples

```
dt <- 1:5
class(dt) <- c("qe.unit", class(dt))
is.qe.unit(dt) # Should return TRUE
```

`names.network.connections` *Title*

Description

Title

Usage

```
## S3 method for class 'network.connections'
names(x)
```

Arguments

`x`	TBD

Value

TBD

`namesToAdjacencyKey` *Names to Adjacency Key*

Description

Convert a vector of strings, representing the names of a square matrix, to an adjacency key matrix.

Usage

```
namesToAdjacencyKey(vector, upper_triangle = TRUE)
```

Arguments

`vector`	Vector representing the names of a square matrix.
`upper_triangle`	Not Implemented.

Details

Returns a matrix with 2 rows and choose(length(vector), 2) columns, where each column represents a unique pair of names from the input vector, corresponding to the upper triangle of a square matrix.

Value

A character matrix with 2 rows and choose(length(vector), 2) columns. Each column contains a pair of names representing a unique adjacency (edge) between nodes in the original square matrix.

---

print.network.matrix *Print Method for Network Matrix (S3 method)*

Description

This S3 method prints a network matrix object, optionally including metadata. It adjusts the class and attaches adjacency key names for improved readability.

Usage

```
## S3 method for class 'network.matrix'
print(x, include.meta = TRUE, ...)
```

Arguments

x	An object of class "network.matrix" to print.
include.meta	Logical; whether to include metadata in the printout (currently not used).
...	Additional arguments passed to lower-level print methods.

Value

Invisibly returns the printed object.

---

remove_meta_data *Remove meta columns from a data.table or data.frame*

Description

This function removes columns of class ‘ena.meta.data‘ from the input object.

Usage

```
remove_meta_data(x)
```

Arguments

x	A ‘data.table‘ or ‘data.frame‘ object from which meta columns should be removed.

Value

A ‘data.frame‘ with columns of class ‘ena.meta.data‘ removed.

RS.data *Coded Rescushell Chat Data*

Description

A dataset containing sample chat data from the Rescushell Virtual Internship

Usage

```
RS.data
```

Format

An object of class `data.frame` with 3824 rows and 21 columns.

RS.data.multimodal *Coded Rescushell multi-modal Data*

Description

A dataset containing sample chat data from the Rescushell Virtual Internship with multiple modalities

Usage

```
RS.data.multimodal
```

Format

An object of class `data.table` (inherits from `data.frame`) with 6641 rows and 19 columns.

rules *Capture Subsetting Rules as Expressions*

Description

Allows users to supply conditions for subsetting rows from their data. The collected unevaluated expressions are intended to be used as the ‘hoo_rules‘ parameter in the ‘contexts()‘ function within the TMA workflow.

Usage

```
rules(...)
```

Arguments

...	Logical expressions specifying the conditions for subsetting data. These expressions are captured unevaluated and returned as a list.

Value

A list of unevaluated expressions representing subsetting rules.

Examples

```
rules(
  modality %in% "chat" & chatGroup %in% UNIT$chatGroup & condition %in% UNIT$condition,
  modality %in% "resource" & userID %in% UNIT$userID & condition %in% UNIT$condition
)
```

simple_window *Internal: Simple window decay (legacy)*

Description

Internal helper for window decay, used in TMA v0.1.0. Not exported. Kept for backward compatibility with legacy decay function creation.

Usage

```
simple_window(x, args = NULL)
```

Arguments

x	Numeric vector of time differences.
args	List of arguments (expects ‘window_size‘).

Value

Numeric vector (0/1) indicating whether each value is within the window.

`test_mockdata` *Sample Data*

Description

A small sample dataset

Usage

```
test_mockdata
```

Format

An object of class `data.table` (inherits from `data.frame`) with 24 rows and 12 columns.

`test_reddit` *Sample Data*

Description

A dataset containing threads

Usage

```
test_reddit
```

Format

An object of class `data.frame` with 463 rows and 19 columns.

`test_reddit2` *Sample Data*

Description

A dataset containing threads

Usage

```
test_reddit2
```

Format

An object of class `data.frame` with 776 rows and 15 columns.

test_smalldata *Sample Data*

Description

A small sample dataset

Usage

```
test_smalldata
```

Format

An object of class `data.frame` with 12 rows and 8 columns.

tma *TMA for ENA*

Description

TMA for ENA. rENA is used to create and visualize network models of discourse and other phenomena from coded data using Epistemic Network Analysis (ENA).

tma.conversations *Find conversations by unit*

Description

Identify and extract rows corresponding to conversations for specified units in a dataset or context model. Useful for subsetting and analyzing conversational windows in network analysis.

Usage

```
tma.conversations(
  x,
  units,
  units.by = NULL,
  codes = NULL,
  conversation.by = NULL,
  window = 4,
  conversation.exclude = c(),
  id_col = "QEUNIT"
)
```

Arguments

x	A data.frame or context model containing conversation data.
units	Character vector of unit identifiers to extract conversations for.
units.by	Character vector of column names specifying unit grouping (default: from context model attributes).
codes	Character vector of code columns to use for identifying coded rows.
conversation.by	Character vector of column names to group by conversation.
window	Integer; window size for co-occurrence (default: 4).
conversation.exclude	Character vector of conversation keys to exclude.
id_col	Character; column name for unit IDs (default: "QEUNIT").

Details

This function groups rows by conversation (using ‘conversation.by‘ columns), identifies which rows are associated with the specified units and codes, and returns indices for each conversation, as well as metadata about which rows to include or exclude.

Value

A list with elements:

conversations	List of row indices for each conversation.
unitConvs	Unique conversation keys for the specified units.
allRows	All row indices included for the units.
unitRows	Row indices for the units with codes.
convRows	All row indices for the unit's conversations.
toRemove	Rows not meeting co-occurrence criteria.

---

units *Set Units of Analysis for a TMA Model*

Description

Internal helper to initialize and label units of analysis in a TMA model object. Given a data.frame and a set of columns, this function creates a model structure with unit labels and context placeholders for each unique unit.

Usage

```
units(x, by)
```

Arguments

x	A data.frame or TMA model object containing the raw input data.
by	Character vector of column names to use for defining units of analysis (e.g., c("userID", "condition")).

Value

A TMA model object with unit labels and empty context slots for each unit.

---

view *Interactive Conversation Viewer*

Description

Launch an interactive HTML viewer for conversations and codes for a specified unit or set of units. Useful for exploring and validating conversation windows and code assignments in the TMA workflow.

Usage

```
view(
  x,
  wh,
  text_col = "text",
  units.by = x$`_function.params`$units.by,
  conversation.by = x$`_function.params`$conversation.by,
  codes = x$rotation$codes,
  window_size = x$`_function.params`$window_size,
  more_cols = NULL,
  in_browser = FALSE,
  id_col = "QEUNIT"
)
```

Arguments

x	A context model or data.frame containing conversation data.
wh	Character or integer; unit(s) to view.
text_col	Character; column name for text (default: "text").
units.by	Character vector of unit grouping columns (default: from context model attributes).
conversation.by	Character vector of conversation grouping columns (default: from context model attributes).
codes	Character vector of code columns (default: from context model attributes).
window_size	Integer; window size for co-occurrence (default: from context model attributes).

more_cols	Character vector of additional columns to include in the viewer.
in_browser	Logical; if TRUE, open in system browser, otherwise use RStudio viewer (default: FALSE).
id_col	Character; column name for unit IDs (default: "QEUNIT").

Value

A list containing the viewer data and metadata (invisibly). The function is called for its side effect of launching the viewer.

---

windows_weights *Deprecated Alias for context_tensor*

Description

Use `context_tensor()` instead. This alias will be removed in a future release.

Usage

```
windows_weights(...)
```

Arguments

...	Arguments passed to `context_tensor()`.

Value

A multidimensional array with dimensions [sender(s), receiver(s), mode(s), weight/window], where the last axis has two levels: "weight" and "window". The array is initialized with the default weight and window values, and has named dimensions for easy indexing.

See Also

context_tensor

`$.network.matrix` *Extract Metadata or Columns from Network Matrix (S3 method)*

Description

This S3 method allows convenient extraction of metadata columns from a network matrix object using the $ operator. If the requested column is metadata, it is returned from the model's meta.data; otherwise, the standard extraction is performed.

Usage

```
## S3 method for class 'network.matrix'
x$i
```

Arguments

`x`	An object of class "network.matrix".
`i`	Name of the column or metadata field to extract.

Value

The requested column or metadata field from the network matrix.

Index