Package 'ISMtools'

March 12, 2026

Type Package

Title Interpretive Structural Modelling Analysis Tools

Version 0.1.0

Description A comprehensive toolkit for Interpretive Structural Modelling (ISM) analysis. Provides functions for creating adjacency matrices from various input formats including SSIM (Structural Self-Interaction Matrix), computing reachability matrices using Warshall's algorithm, performing hierarchical level partitioning, MICMAC (Cross-Impact Matrix Multiplication Applied to Classification) analysis, and visualizing ISM structures through both static and interactive diagrams. ISM is a methodology for identifying and summarizing relationships among specific elements which define an issue or problem, as described in Warfield (1974) <doi:10.1109/TSMC.1974.5408524>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports graphics, stats

Suggests igraph, visNetwork, viridis, testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.2

Config/testthat/edition 3

NeedsCompilation no

Author Yi Tang [aut, cre] (ORCID: <https://orcid.org/0000-0002-4374-6334>)

Maintainer Yi Tang <tangyi@cidp.edu.cn>

Repository CRAN

Date/Publication 2026-03-12 19:30:08 UTC

Contents

`ISMtools-package` *ISMtools: Interpretive Structural Modelling Analysis Tools*

Description

A comprehensive toolkit for Interpretive Structural Modelling (ISM) and MICMAC analysis. ISM is a methodology for identifying and summarizing relationships among specific elements which define an issue or problem.

Input Functions

`create_relation_matrix` Create or convert adjacency matrices

`convert_to_matrix` Convert edge lists to adjacency matrices

`create_ssim` Create SSIM (Structural Self-Interaction Matrix) template

`ssim_to_matrix` Convert SSIM (V/A/X/O) to adjacency matrix

Core Analysis Functions

`compute_reachability` Calculate reachability matrix using Warshall's algorithm

`level_partitioning` Perform hierarchical level decomposition

`micmac_analysis` MICMAC analysis (driving/dependence power)

`extract_direct_edges` Transitive reduction for clean diagrams

Visualization Functions

`plot_ism` Static ISM structure visualization (base R or igraph)

`plot_interactive_ism` Interactive ISM visualization (requires visNetwork)

`plot_micmac` MICMAC scatter plot

Typical Workflow

1. Create SSIM using `create_ssim` or adjacency matrix using `create_relation_matrix`
2. Convert SSIM to adjacency matrix using `ssim_to_matrix` (if applicable)
3. Compute reachability matrix using `compute_reachability`
4. Perform level partitioning using `level_partitioning`
5. Perform MICMAC analysis using `micmac_analysis`
6. Visualize results using `plot_ism` and `plot_micmac`

Example Data

The package includes an example dataset `ism_example` containing a project risk factors adjacency matrix for demonstration purposes.

Author(s)

Maintainer: Yi Tang <tangyi@cidp.edu.cn> (ORCID)

References

Warfield, J. N. (1974). Developing interconnection matrices in structural modeling. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-4(1), 81-87. doi:10.1109/TSMC.1974.5408524

Sage, A. P. (1977). *Interpretive Structural Modeling: Methodology for Large-scale Systems*. McGraw-Hill.

`compute_reachability` *Compute Reachability Matrix*

Description

Calculates the reachability matrix from an adjacency matrix using Warshall's algorithm. The reachability matrix shows all direct and indirect relationships between elements in a system.

Usage

```
compute_reachability(adj_matrix, include_self = TRUE)
```

Arguments

`adj_matrix`	A square adjacency matrix (n x n) containing only 0s and 1s. A value of 1 at position (i,j) indicates that element i directly influences element j.
`include_self`	Logical. If `TRUE` (default), the diagonal elements are set to 1, indicating that each element can reach itself (self-reachability). This is the standard ISM convention.

Details

The function implements Warshall's algorithm for computing the transitive closure of a directed graph. The time complexity is O(n^3) where n is the number of elements.

In standard ISM methodology, the reachability matrix is defined as:

$$R = (A + I)^k$$

where A is the adjacency matrix, I is the identity matrix, and k is the smallest integer such that $(A + I)^k = (A + I)^{k+1}$.

The diagonal elements being 1 (self-reachability) is essential for correct level partitioning in ISM analysis.

Value

A reachability matrix of the same dimension as the input.

- A value of 1 at position (i,j) indicates that element i can reach element j either directly or through intermediate elements.
- When `include_self = TRUE`, diagonal elements are always 1.

References

Warfield, J. N. (1974). Developing interconnection matrices in structural modeling. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-4(1), 81-87. doi:10.1109/TSMC.1974.5408524

See Also

`create_relation_matrix` for creating adjacency matrices, `level_partitioning` for hierarchical decomposition, `plot_ism` for visualization.

Examples

```
# Create a 4x4 adjacency matrix
adj_matrix <- matrix(c(0, 1, 0, 0,
                       0, 0, 1, 1,
                       0, 0, 0, 0,
                       0, 0, 0, 0),
                     nrow = 4, byrow = TRUE)

# Compute reachability matrix (with self-reachability)
reach_matrix <- compute_reachability(adj_matrix)
print(reach_matrix)
```

```
# Note: diagonal elements are 1 (self-reachability)
diag(reach_matrix)

# With named elements
rownames(adj_matrix) <- colnames(adj_matrix) <- c("A", "B", "C", "D")
reach_matrix <- compute_reachability(adj_matrix)
print(reach_matrix)
```

`convert_to_matrix` *Convert Input to Adjacency Matrix*

Description

Converts various input formats (data.frame, matrix) to a standard adjacency matrix suitable for ISM analysis. Performs validation and provides warnings for potential issues.

Usage

```
convert_to_matrix(x, from = 1, to = 2, nodes = NULL, validate = TRUE)
```

Arguments

`x`	Input data. Can be: • A two-column matrix representing an edge list • A square matrix (adjacency matrix) • A data.frame with source and target columns
`from`	Column name or index for source nodes (if x is a data.frame). Default is 1.
`to`	Column name or index for target nodes (if x is a data.frame). Default is 2.
`nodes`	Optional character vector of predefined node names. If provided, ensures the resulting matrix includes all specified nodes.
`validate`	Logical. If `TRUE` (default), validates the input and provides warnings for potential issues.

Details

This function provides flexible input handling for ISM analysis. It can convert edge lists (either as data frames or two-column matrices) into adjacency matrices, or validate and return existing adjacency matrices.

When `validate = TRUE`, the function checks:

- Square matrices contain only 0s and 1s (or logical values)
- Edge list nodes exist in the predefined node list (if provided)
- Column names exist in data frames

Value

A square numeric adjacency matrix with node names as row and column names. A value of 1 at position (i,j) indicates a directed edge from node i to node j.

See Also

`create_relation_matrix` for a higher-level interface, `ssim_to_matrix` for SSIM conversion, `compute_reachability` for computing reachability matrices.

Examples

```
# From data frame edge list
edge_df <- data.frame(source = c("A", "B"), target = c("B", "C"))
convert_to_matrix(edge_df, from = "source", to = "target")

# From matrix edge list
edge_mat <- matrix(c("A", "B", "B", "C"), ncol = 2, byrow = TRUE)
convert_to_matrix(edge_mat)

# Existing adjacency matrix (validated and returned)
adj_mat <- matrix(c(0, 1, 0, 0,
                    0, 0, 1, 1,
                    0, 0, 0, 0,
                    0, 0, 0, 0), nrow = 4, byrow = TRUE)
convert_to_matrix(adj_mat)

# With predefined nodes (ensures all nodes are included)
edge_df <- data.frame(source = "A", target = "B")
convert_to_matrix(edge_df, from = "source", to = "target",
                  nodes = c("A", "B", "C", "D"))
```

`create_relation_matrix`

Create or Convert to Adjacency Matrix

Description

Creates a new adjacency matrix or converts existing data to an adjacency matrix format suitable for ISM analysis.

Usage

```
create_relation_matrix(input, from = 1, to = 2, nodes = NULL)
```

Arguments

`input`	Input data. Accepts: • Integer n: creates an n x n zero matrix • Matrix: validates and returns (or converts from edge list) • Data frame: converts from edge list format
`from`	Column position or name for source nodes (when input is an edge list). Default is 1.
`to`	Column position or name for target nodes (when input is an edge list). Default is 2.
`nodes`	Optional character vector of predefined node names. Ensures matrix completeness when converting from edge lists.

Details

This function provides a unified interface for creating adjacency matrices in ISM analysis. It handles three common scenarios:

1. Creating an empty matrix of specified size for manual population
2. Converting edge list data (data.frame or matrix) to adjacency format
3. Validating existing adjacency matrices

Value

A square adjacency matrix with node labels as dimnames. A value of 1 at position (i,j) indicates a directed relationship from element i to element j.

See Also

`convert_to_matrix` for the underlying conversion logic, `compute_reachability` for computing reachability matrices.

Examples

```
# Create a 3x3 zero matrix
create_relation_matrix(3)

# Create with custom node labels
create_relation_matrix(3, nodes = c("A", "B", "C"))

# Convert edge list data.frame
edge_df <- data.frame(source = c("A", "B"), target = c("B", "C"))
create_relation_matrix(edge_df, from = "source", to = "target",
                       nodes = LETTERS[1:4])

# Validate existing matrix
existing_mat <- matrix(c(0, 1, 1, 0), nrow = 2)
create_relation_matrix(existing_mat)
```

`create_ssim` *Create SSIM (Structural Self-Interaction Matrix)*

Description

Creates an empty SSIM template or converts existing data to SSIM format. SSIM uses V/A/X/O notation to describe pairwise relationships between elements.

Usage

```
create_ssim(n, labels = NULL)
```

Arguments

`n`	Number of elements, or a character vector of element names.
`labels`	Optional character vector of element labels. If `n` is a character vector, this parameter is ignored.

Details

SSIM (Structural Self-Interaction Matrix) is the standard input format for ISM analysis. For each pair of elements (i, j) where i < j, the relationship is coded as:

V Element i influences element j (i -> j)

A Element j influences element i (j -> i)

X Both elements influence each other (i <-> j)

O No relationship between elements

Value

A character matrix of dimension n x n with:

- Upper triangle: empty strings (to be filled with V/A/X/O)
- Diagonal: "X" (self-relation)
- Lower triangle: "-" (mirror of upper, not used directly)

See Also

`ssim_to_matrix` for converting SSIM to adjacency matrix, `create_relation_matrix` for direct matrix creation.

Examples

```
# Create empty 4x4 SSIM
ssim <- create_ssim(4)
print(ssim)

# Create with labels
ssim <- create_ssim(c("Budget", "Resources", "Quality", "Success"))
print(ssim)

# Fill in relationships
ssim["Budget", "Resources"] <- "V"
ssim["Budget", "Quality"] <- "V"
ssim["Resources", "Quality"] <- "V"
ssim["Quality", "Success"] <- "V"
print(ssim)
```

---

extract_direct_edges *Extract Direct Edges (Transitive Reduction)*

Description

Performs transitive reduction on a reachability matrix to extract only the direct (essential) edges, removing edges that can be inferred through transitive paths.

Usage

```
extract_direct_edges(reach_matrix, adj_matrix = NULL)
```

Arguments

reach_matrix	A square reachability matrix (n x n) with 0/1 entries.
adj_matrix	Optional. The original adjacency matrix. If provided, the function will use it to identify direct edges more accurately.

Details

Transitive reduction removes redundant edges from a directed graph while preserving reachability. An edge (i,j) is considered redundant (transitive) if there exists another path from i to j through intermediate nodes.

For example, if A->B->C and A->C, the edge A->C is transitive and will be removed, leaving only A->B and B->C.

This is essential for creating clean ISM diagrams suitable for publications, as showing all reachability edges would result in cluttered graphs.

Value

A matrix of the same dimension containing only direct edges (1s where there is a direct relationship that cannot be inferred from other paths).

See Also

identify_transitive_edges for identifying (not removing) transitive edges, plot_ism which uses this function internally.

Examples

```
# Create adjacency matrix with a transitive edge
# A -> B -> C, and A -> C (transitive)
adj <- matrix(c(0, 1, 1,
                0, 0, 1,
                0, 0, 0), nrow = 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

# Compute reachability
reach <- compute_reachability(adj)
print(reach)

# Extract direct edges only
direct <- extract_direct_edges(reach)
print(direct)
# A->C is removed because it's transitive through B
```

---

identify_transitive_edges

Identify Transitive Edges

Description

Identifies which edges in a reachability matrix are transitive (can be inferred from other paths) versus direct (essential).

Usage

```
identify_transitive_edges(reach_matrix, adj_matrix = NULL)
```

Arguments

reach_matrix	A square reachability matrix (n x n) with 0/1 entries.
adj_matrix	Optional. The original adjacency matrix for comparison.

Details

This function is useful for understanding the structure of relationships and for creating visualizations where transitive edges are shown differently (e.g., as dashed lines) from direct edges.

Value

A data frame with columns:

- `from`: source node index
- `to`: target node index
- `from_label`: source node label (if available)
- `to_label`: target node label (if available)
- `type`: "direct" or "transitive"

See Also

`extract_direct_edges` for removing transitive edges, `plot_ism` for visualization.

Examples

```
adj <- matrix(c(0, 1, 1,
                0, 0, 1,
                0, 0, 0), nrow = 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

reach <- compute_reachability(adj)
edges <- identify_transitive_edges(reach)
print(edges)
```

---

ism_example *Example ISM Data: Project Risk Factors*

Description

A dataset containing an adjacency matrix representing relationships between project risk factors. This example is useful for demonstrating ISM analysis workflow.

Usage

```
data(ism_example)
```

Format

A 7x7 numeric matrix with row and column names F1 through F7, representing the following risk factors:

F1 Budget Constraints - financial limitations

F2 Resource Shortage - insufficient human or material resources

F3 Technical Complexity - challenging technical requirements

F4 Poor Communication - inadequate information flow

F5 Schedule Pressure - tight deadlines

F6 Quality Issues - defects and quality problems

F7 Project Failure - ultimate negative outcome

The matrix values indicate direct influence relationships:

- 1: Factor in row directly influences factor in column
- 0: No direct influence

The relationships encoded are:

- F1 (Budget) -> F2 (Resource), F5 (Schedule)
- F2 (Resource) -> F3 (Technical), F6 (Quality)
- F3 (Technical) -> F6 (Quality)
- F4 (Communication) -> F2, F3, F6
- F5 (Schedule) -> F4 (Communication), F6 (Quality)
- F6 (Quality) -> F7 (Failure)

Source

Hypothetical example for demonstration purposes, based on common project management risk factor relationships.

See Also

compute_reachability, level_partitioning, micmac_analysis

Examples

```
# Load the example data
data(ism_example)

# View the adjacency matrix
print(ism_example)

# Compute reachability matrix
reach <- compute_reachability(ism_example)
print(reach)

# Perform level partitioning
levels <- level_partitioning(reach)
print(levels)

# MICMAC analysis
micmac <- micmac_analysis(reach)
print(micmac)
```

`level_partitioning` *Level Partitioning for ISM*

Description

Performs hierarchical level decomposition of system elements based on their reachability and antecedent sets. This is a core step in Interpretive Structural Modelling (ISM) analysis.

Usage

```
level_partitioning(reach_matrix)
```

Arguments

`reach_matrix`	A square reachability matrix (n x n) with 0/1 entries, typically computed using `compute_reachability`. The diagonal should be 1 (self-reachability).

Details

The algorithm implements the standard ISM level partitioning procedure:

1. For each remaining element i, compute:
   - **Reachability set R(i)**: elements that i can reach (within remaining set)
   - **Antecedent set A(i)**: elements that can reach i (within remaining set)
2. An element belongs to the current (top) level if: $R(i) \cap A(i) = R(i)$, i.e., the reachability set equals the intersection
3. Remove top-level elements and repeat until all elements are assigned

This implementation correctly operates on the **remaining subset** at each iteration, which is essential for correct level assignment.

Value

An object of class `ism_levels`, which is a list containing:

- Each element is a vector of node indices belonging to that level
- Level 1 is the top level (outcomes/dependent variables)
- Higher numbered levels are lower in the hierarchy (drivers/independent variables)
- Attribute `labels`: node names if the input matrix has dimnames

References

Warfield, J. N. (1974). Developing interconnection matrices in structural modeling. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-4(1), 81-87. doi:10.1109/TSMC.1974.5408524

Sage, A. P. (1977). *Interpretive Structural Modeling: Methodology for Large-scale Systems*. McGraw-Hill.

See Also

compute_reachability for computing reachability matrices, plot_ism for visualization, plot_interactive_ism for interactive visualization, micmac_analysis for MICMAC analysis.

Examples

```
# Create adjacency matrix
adj_matrix <- matrix(c(0, 1, 0, 0,
                       0, 0, 1, 1,
                       0, 0, 0, 0,
                       0, 0, 0, 0), nrow = 4, byrow = TRUE)
rownames(adj_matrix) <- colnames(adj_matrix) <- c("A", "B", "C", "D")

# Compute reachability matrix
reach_mat <- compute_reachability(adj_matrix)

# Perform level partitioning
levels <- level_partitioning(reach_mat)
print(levels)

# Access specific levels
levels[[1]]  # Top level elements (outcomes)
levels[[length(levels)]]  # Bottom level elements (root causes)
```

---

micmac_analysis *MICMAC Analysis*

Description

Performs MICMAC (Cross-Impact Matrix Multiplication Applied to Classification) analysis on a reachability matrix to classify elements based on their driving power and dependence power.

Usage

```
micmac_analysis(reach_matrix)
```

Arguments

reach_matrix	A square reachability matrix (n x n) with 0/1 entries, typically computed using compute_reachability.

Details

MICMAC analysis is a complementary technique to ISM that helps identify the key drivers and dependent variables in a system.

Driving Power: The number of elements that a given element can reach (row sum of reachability matrix).

Dependence Power: The number of elements that can reach a given element (column sum of reachability matrix).

Elements are classified into four clusters based on whether their driving and dependence powers are above or below the median values.

Value

An object of class `micmac_result`, which is a data frame with:

- `node`: node index
- `label`: node label (from matrix dimnames or numeric)
- `driving_power`: number of elements this node can reach
- `dependence_power`: number of elements that can reach this node
- `cluster`: classification into one of four clusters

The four clusters are:

I - Autonomous Low driving power, low dependence. Disconnected from system.

II - Dependent Low driving power, high dependence. Outcomes/results.

III - Linkage High driving power, high dependence. Unstable, key connectors.

IV - Independent High driving power, low dependence. Root causes/drivers.

References

Duperrin, J. C., & Godet, M. (1973). Methode de hierarchisation des elements d'un systeme. Rapport economique du CEA, R-45-41.

Warfield, J. N. (1974). Developing interconnection matrices in structural modeling. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-4(1), 81-87. doi:10.1109/TSMC.1974.5408524

See Also

`plot_micmac` for visualization, `compute_reachability` for computing reachability matrices, `level_partitioning` for hierarchical decomposition.

Examples

```
# Create adjacency matrix
adj <- matrix(c(0, 1, 0, 0, 0,
                0, 0, 1, 0, 0,
                0, 0, 0, 1, 1,
                0, 0, 0, 0, 0,
                0, 0, 0, 0, 0), nrow = 5, byrow = TRUE)
rownames(adj) <- colnames(adj) <- paste0("F", 1:5)

# Compute reachability and MICMAC
reach <- compute_reachability(adj)
micmac <- micmac_analysis(reach)
print(micmac)
```

```
# View cluster distribution
table(micmac$cluster)
```

`plot_interactive_ism` *Interactive ISM Visualization*

Description

Generates interactive Interpretive Structural Model diagrams with node dragging, zooming, and level-based filtering capabilities. Requires the visNetwork package (suggested dependency).

Usage

```
plot_interactive_ism(
  reach_matrix,
  node_labels = NULL,
  level_result = NULL,
  show_transitive = FALSE,
  direction = c("UD", "LR", "DU", "RL")
)
```

Arguments

`reach_matrix`	A reachability matrix (n x n) representing the ISM relationships.
`node_labels`	A character vector of length n specifying custom node labels. If `NULL`, uses matrix row names or numeric indices.
`level_result`	A list of class `ism_levels` representing level partitioning results from `level_partitioning`. If `NULL`, level partitioning is computed automatically.
`show_transitive`	Logical. If `FALSE` (default), transitive edges are removed for cleaner visualization. If `TRUE`, all edges are shown.
`direction`	Layout direction for visualization. Options: • `"UD"`: Up-down (default) • `"LR"`: Left-right • `"DU"`: Down-up • `"RL"`: Right-left

Details

This function requires the **visNetwork** package. If **viridis** is available, it will be used for level-based coloring; otherwise, a default color palette is used.

The interactive visualization provides several features:

- **Drag nodes**: Click and drag to reposition nodes
- **Zoom**: Use mouse wheel or navigation buttons

- **Highlight**: Hover over nodes to highlight connections
- **Filter**: Select nodes by ID or filter by level group
- **Keyboard navigation**: Use arrow keys to navigate

By default, transitive edges are removed using `extract_direct_edges` to produce cleaner diagrams. Set `show_transitive = TRUE` to show all edges.

Value

An interactive `visNetwork` object that can be displayed in RStudio Viewer, Shiny apps, or web browsers.

See Also

`plot_ism` for static visualization, `level_partitioning` for computing hierarchical levels, `compute_reachability` for computing reachability matrices, `extract_direct_edges` for transitive reduction.

Examples

```
# Create sample data
adj_matrix <- matrix(c(0, 1, 0, 0,
                       0, 0, 1, 1,
                       0, 0, 0, 0,
                       0, 0, 0, 0), nrow = 4, byrow = TRUE)
rownames(adj_matrix) <- colnames(adj_matrix) <- LETTERS[1:4]
reach_matrix <- compute_reachability(adj_matrix)
levels <- level_partitioning(reach_matrix)

# Basic interactive plot (requires visNetwork)
if (requireNamespace("visNetwork", quietly = TRUE)) {
  plot_interactive_ism(reach_matrix)

  # With custom labels and levels
  plot_interactive_ism(reach_matrix,
                       node_labels = c("Factor A", "Factor B",
                                       "Factor C", "Factor D"),
                       level_result = levels)
}
```

`plot_ism` *Plot ISM Structure*

Description

Visualizes the Interpretive Structural Model (ISM) as a hierarchical diagram. By default, only essential edges are shown (transitive edges removed) for cleaner visualization suitable for publications.

Usage

```
plot_ism(
  reach_matrix,
  levels = NULL,
  show_transitive = FALSE,
  use_igraph = FALSE,
  node_labels = NULL,
  main = "ISM Hierarchical Structure",
  ...
)
```

Arguments

reach_matrix	A square reachability matrix (n x n) with 0/1 entries, typically computed using compute_reachability.
levels	Optional. An object of class ism_levels from level_partitioning. If NULL, it will be computed automatically.
show_transitive	Logical. If FALSE (default), transitive edges are removed for cleaner visualization. If TRUE, all edges are shown.
use_igraph	Logical. If TRUE and igraph is available, use igraph for plotting. If FALSE (default), use base R graphics.
node_labels	Optional character vector of node labels. If NULL, uses matrix row names or numeric indices.
main	Title for the plot. Default is "ISM Hierarchical Structure".
...	Additional arguments passed to plotting functions.

Details

The function performs transitive reduction by default, removing edges that can be inferred from other paths. This produces cleaner diagrams that are more suitable for academic publications and presentations.

Two plotting backends are available:

- **Base R** (default): No additional packages required. Nodes are arranged in horizontal levels with arrows showing relationships.
- **igraph**: Requires the igraph package. Provides more sophisticated graph layouts and styling options.

Value

Invisibly returns a list containing:

- levels: the level partitioning result
- edges: data frame of edges used in the plot
- reduced_matrix: the adjacency matrix after transitive reduction

See Also

plot_interactive_ism for interactive visualization, compute_reachability for computing reachability matrices, level_partitioning for hierarchical decomposition, extract_direct_edges for transitive reduction.

Examples

```
# Create adjacency matrix
adj <- matrix(c(0, 1, 0, 0,
                0, 0, 1, 1,
                0, 0, 0, 0,
                0, 0, 0, 0), nrow = 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Compute reachability
reach <- compute_reachability(adj)

# Plot ISM structure (base R, transitive edges removed)
plot_ism(reach)

# Show all edges including transitive ones
plot_ism(reach, show_transitive = TRUE)


# Use igraph if available
plot_ism(reach, use_igraph = TRUE)
```

---

plot_micmac *Plot MICMAC Diagram*

Description

Creates a scatter plot showing the MICMAC classification of elements based on their driving power and dependence power.

Usage

```
plot_micmac(micmac_result, show_labels = TRUE, main = "MICMAC Analysis", ...)
```

Arguments

micmac_result	An object of class micmac_result from micmac_analysis.
show_labels	Logical. If TRUE (default), show node labels on the plot.
main	Title for the plot. Default is "MICMAC Analysis".
...	Additional arguments passed to plot.

Details

The plot is divided into four quadrants:

Quadrant I (bottom-left) Autonomous variables - weak drivers, weak dependence

Quadrant II (bottom-right) Dependent variables - weak drivers, strong dependence

Quadrant III (top-right) Linkage variables - strong drivers, strong dependence

Quadrant IV (top-left) Independent variables - strong drivers, weak dependence

Value

Invisibly returns the micmac_result object.

See Also

`micmac_analysis` for computing MICMAC results.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 0,
                0, 0, 1, 0, 0,
                0, 0, 0, 1, 1,
                0, 0, 0, 0, 0,
                0, 0, 0, 0, 0), nrow = 5, byrow = TRUE)
rownames(adj) <- colnames(adj) <- paste0("F", 1:5)

reach <- compute_reachability(adj)
micmac <- micmac_analysis(reach)
plot_micmac(micmac)
```

---

print.ism_levels *Print ISM Levels*

Description

Print method for objects of class `ism_levels` created by `level_partitioning`.

Usage

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

Arguments

`x`	An object of class `ism_levels`
`...`	Additional arguments (ignored)

Value

Invisibly returns the input object

print.micmac_result *Print MICMAC Results*

Description

Print MICMAC Results

Usage

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

Arguments

x	An object of class micmac_result
...	Additional arguments (ignored)

Value

Invisibly returns the input object

print.ssim_matrix *Print SSIM Matrix*

Description

Print SSIM Matrix

Usage

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

Arguments

x	An object of class ssim_matrix
...	Additional arguments passed to print.default

Value

Invisibly returns the input object

`ssim_to_matrix` *Convert SSIM to Adjacency Matrix*

Description

Converts a Structural Self-Interaction Matrix (SSIM) with V/A/X/O notation to a binary adjacency matrix suitable for ISM analysis.

Usage

```
ssim_to_matrix(ssim, validate = TRUE)
```

Arguments

`ssim`	A square character matrix with V/A/X/O values in the upper triangle. Can be created using `create_ssim`.
`validate`	Logical. If `TRUE` (default), validates that all upper triangle entries are valid (V/A/X/O).

Details

The conversion rules are:

V at (i,j) Sets adj[i,j] = 1 (i influences j)

A at (i,j) Sets adj[j,i] = 1 (j influences i)

X at (i,j) Sets adj[i,j] = 1 AND adj[j,i] = 1 (mutual influence)

O at (i,j) No edges added

Value

A square numeric adjacency matrix where:

- 1 at position (i,j) indicates element i influences element j
- 0 indicates no direct influence

See Also

`create_ssim` for creating SSIM templates, `compute_reachability` for the next step in ISM analysis.

Examples

```
# Create and fill SSIM
ssim <- create_ssim(c("Budget", "Resources", "Quality", "Success"))
ssim["Budget", "Resources"] <- "V"
ssim["Budget", "Quality"] <- "V"
ssim["Resources", "Quality"] <- "V"
ssim["Quality", "Success"] <- "V"

# Convert to adjacency matrix
adj <- ssim_to_matrix(ssim)
print(adj)

# Continue with ISM analysis
reach <- compute_reachability(adj)
levels <- level_partitioning(reach)
```

---

summary.ism_levels *Summary of ISM Levels*

Description

Summary of ISM Levels

Usage

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

Arguments

`object`	An object of class `ism_levels`
`...`	Additional arguments (ignored)

Value

A data frame with level information

Index