

Package: fcmconfr (via r-universe)

September 17, 2026

Type Package

Title Fuzzy Cognitive Map Analysis in R

Version 0.2.3

Description fcmconfr streamlines the process of conducting dynamic simulations using Fuzzy Cognitive Maps (FCMs). The package supports multiple FCM types (conventional, Interval-Value Fuzzy Number (IVFN) and Triangular Fuzzy Number (TFN)) and provides tools for model aggregation and the propagation of uncertainty from individual FCMs through to the aggregate.

License GPL (>= 3) + file LICENSE

Encoding UTF-8

LazyData true

Suggests covr (>= 3.6.4), data.table (>= 1.15.4), graphics (>= 4.3.2), knitr (>= 1.44), Matrix (>= 1.6.1.1), rmarkdown (>= 2.25), testthat (>= 3.0.0), tibble (>= 3.2.1), tidyr (>= 1.3.0), vdiffr (>= 1.0.8)

Config/testthat/edition 3

Roxygen list(markdown = TRUE, roclets = c("`namespace", "`rd", "`srr::srr_stats_roclet"))

RoxygenNote 7.3.2

RdMacros Rdpack

Imports bslib (>= 0.8.0), checkmate (>= 2.3.2), cli (>= 3.6.2), ggplot2 (>= 3.4.2), igraph (>= 1.5.1), magrittr (>= 2.0.3), methods (>= 4.2.1), parallel (>= 4.2.1), pbapply (>= 1.7.2), Rdpack (>= 2.6.1), rlang (>= 1.1.4), shiny (>= 1.7.5), shinyWidgets (>= 0.8.7), stats (>= 4.2.1), visNetwork (>= 2.1.2),

Depends R (>= 2.10)

VignetteBuilder knitr

URL <https://bhroston.github.io/fcmconfr/>

BugReports <https://github.com/bhroston/fcmconfr/issues>

Config/pak/sysreqs cmake libglpk-dev make libuv1-dev libxml2-dev zlib1g-dev

Repository <https://ropensci-staging.r-universe.dev>

Date/Publication 2026-06-21 02:00:01 UTC

RemoteUrl <https://github.com/bhroston/fcmconfr>

RemoteRef HEAD

RemoteSha cca24f32b5f82468f2836c14598c15bc27978c86

Contents

aggregate_fcms	3
build_monte_carlo_fcms	6
c.ivfn	8
c.tfn	9
create_ivfn_fcm_from_conventional_fcm	9
create_tfn_fcm_from_conventional_fcm	10
defuzz_ivfn_or_tfn	11
estimate_fcm_lambda	12
fcm_view	14
fcmconfr	16
fcmconfr_gui	23
get_fcmconfr_inferences	25
infer_fcm	28
infer_fcm_set	30
ivfn	33
make_adj_matrix_w_ivfns	35
make_adj_matrix_w_tfns	36
plot.fcmconfr	37
plot.rtriangular_dist	42
print.aggregate	42
print.fcmconfr	43
print.ivfn	44
print.summary.fcmconfr	44
print.tfn	45
rtriangular_dist	46
sample_fcms	47
standardize_adj_matrices	48
subtract_ivfn	50
subtract_tfn	51
summary.fcmconfr	53
tfn	54

Index

56

aggregate_fcms

*Aggregate FCMs***Description**

Generate an aggregate adj. matrix from a list of adj. matrices. FCM aggregation works by calculating the mean/median edge weight for all edges across the input adj. matrices (i.e. the mean/median of the edge weight connecting A -> B across all maps, the mean/median of the edge weight connecting B -> C across all maps, and so on). The user may dictate whether to incorporate 0-valued edge weights in the mean/median calculations.

All input adj. matrices must have the same dimensions and concept names to generate an aggregate. Call [standardize_adj_matrices](#) to make sure all adj. matrices have the same dimensions.

Usage

```
aggregate_fcms(
  adj_matrices = list(),
  agg_function = c("mean", "median"),
  include_zeroes_in_sampling = FALSE,
  skip_checks = FALSE
)
```

Arguments

adj_matrices	[list()] A single adjacency matrix or a list of adjacency matrices (n x n) representing FCMs. Matrices can have conventional edge weights, IVFN edge weights or TFN edge weights.
agg_function	[character(1)] Choice of aggregation method (mean, median) for producing a single "collective" FCM from a group of individual FCMs. Omit this argument when analyzing a single FCM.
include_zeroes_in_sampling	[logical(1)] If TRUE, incorporate zero-weighted edges during FCM aggregation.
skip_checks	[logical(1)] FOR DEVELOPER USE ONLY. If TRUE, skip call to check_simulation_inputs (internal function)

Details

This function implements FCM aggregation methods used in Aminpour 2020 (<https://doi.org/10.1038/s41893-019-0467-z>) for use with conventional FCMs, and expands upon them for IVFN- and TFN-FCMs.

Conventional FCMs

The mean aggregate FCM of a set of conventional FCMs is:

$$A_{ij}^{aggregate} = \frac{\sum_{p=1}^N A_{ij}^{FCM_p}}{N}$$

where $A_{ij}^{aggregate}$ is the weight in the aggregate adjacency matrix for the element in row i and column j , and FCM_p is the weight for the element in row i and column j in the adjacency matrix of the p -th FCM.

If not including zero-valued edges in aggregation calculations, the mean aggregate equation becomes:

$$A_{ij}^{aggregate} = \frac{\sum_{p=1}^N A_{ij}^{FCM_p}}{\sum_{p=1}^N 1 | A_{ij}^{FCM_p} \neq 0}$$

The median aggregate FCM of a set of conventional FCMs is:

$$A_{ij}^{aggregate} = \text{Median}(A_{ij}^{FCM_1}, A_{ij}^{FCM_2}, \dots, A_{ij}^{FCM_N})$$

If not including zero-valued edges in aggregation calculations, the median aggregate equation becomes:

$$A_{ij}^{aggregate} = \text{Median}(A_{ij}^{FCM_1}, A_{ij}^{FCM_2}, \dots, A_{ij}^{FCM_N}) | A_{ij}^{FCM_p} \neq 0$$

IVFN-FCMs

The mean aggregate FCM of a set of IVFN-FCMs is:

$$\begin{aligned} A_{ij}^{aggregate} &= [A^L, A^U]_{ij}^{FCM_{aggregate}} \\ &= \left[\frac{\sum_{p=1}^N [A^L]_{ij}^{FCM_p}}{N}, \frac{\sum_{p=1}^N [A^U]_{ij}^{FCM_p}}{N} \right] \end{aligned}$$

where $[A^L, A^U]_{ij}^{FCM_{aggregate}}$ is the IVFN weight (with lower bound A^L and upper bound A^U) of the element in row i and column j of the aggregate, and $[A^L]_{ij}^{FCM_p}$ and $[A^U]_{ij}^{FCM_p}$ are the lower and upper bounds of the IVFN weight of the element in row i and column j for the p -th FCM adjacency matrix.

If not including zero-valued edges in aggregation calculations, the mean aggregate equation becomes:

$$\begin{aligned} A_{ij}^{aggregate} &= [A^L, A^U]_{ij}^{FCM_{aggregate}} \\ &= \left[\frac{\sum_{p=1}^N [A^L]_{ij}^{FCM_p}}{\sum_{p=1}^N 1 | [A^L]_{ij}^{FCM_p} \neq 0}, \frac{\sum_{p=1}^N [A^U]_{ij}^{FCM_p}}{\sum_{p=1}^N 1 | [A^U]_{ij}^{FCM_p} \neq 0} \right] \end{aligned}$$

The median aggregate FCM of a set of IVFN-FCMs is:

$$A_{ij}^{aggregate} = \begin{bmatrix} \text{Median} \left(A_{ij}^{FCM_1-L}, A_{ij}^{FCM_2-L}, \dots, A_{ij}^{FCM_N-L} \right), \\ \text{Median} \left(A_{ij}^{FCM_1-U}, A_{ij}^{FCM_2-U}, \dots, A_{ij}^{FCM_N-U} \right) \end{bmatrix}$$

If not including zero-valued edges in aggregation calculations, the median aggregate equation becomes:

$$A_{ij}^{aggregate} = \begin{bmatrix} \text{Median} \left(A_{ij}^{FCM_1-L}, A_{ij}^{FCM_2-L}, \dots, A_{ij}^{FCM_N-L} \right) | A_{ij}^{FCM_p-L} \neq A_{ij}^{FCM_p-U} \neq 0, \\ \text{Median} \left(A_{ij}^{FCM_1-U}, A_{ij}^{FCM_2-U}, \dots, A_{ij}^{FCM_N-U} \right) | A_{ij}^{FCM_p-L} \neq A_{ij}^{FCM_p-U} \neq 0 \end{bmatrix}$$

TFN-FCMs

The mean aggregate FCM of a set of TFN-FCMs is:

$$\begin{aligned} A_{ij}^{aggregate} &= [A^L, A^M, A^U]_{ij}^{FCM_{aggregate}} \\ &= \left[\frac{\sum_{p=1}^N [A^L]_{ij}^{FCM_p}}{N}, \frac{\sum_{p=1}^N [A^M]_{ij}^{FCM_p}}{N}, \frac{\sum_{p=1}^N [A^U]_{ij}^{FCM_p}}{N} \right] \end{aligned}$$

where $[A^L, A^M, A^U]_{ij}^{FCM_{aggregate}}$ is the TFN weight (with lower bound A^L , mode A^M , and upper bound A^U) of the element in row i and column j of the aggregate, and $[A^L]_{ij}^{FCM_p}$, $[A^M]_{ij}^{FCM_p}$, and $[A^U]_{ij}^{FCM_p}$ are the lower bound, mode, and upper bound of the TFN weight of the element in row i and column j for the p -th FCM adjacency matrix.

If not including zero-valued edges in aggregation calculations, the mean aggregate equation becomes:

$$\begin{aligned} A_{ij}^{aggregate} &= [A^L, A^M, A^U]_{ij}^{FCM_{aggregate}} \\ &= \left[\frac{\sum_{p=1}^N [A^L]_{ij}^{FCM_p}}{\sum_{p=1}^N 1|[A^L]_{ij}^{FCM_p} \neq 0}, \frac{\sum_{p=1}^N [A^M]_{ij}^{FCM_p}}{\sum_{p=1}^N 1|[A^M]_{ij}^{FCM_p} \neq 0}, \frac{\sum_{p=1}^N [A^U]_{ij}^{FCM_p}}{\sum_{p=1}^N 1|[A^U]_{ij}^{FCM_p} \neq 0} \right] \end{aligned}$$

The median aggregate FCM of a set of TFN-FCMs is:

$$\begin{aligned} A_{ij}^{aggregate} &= \left[\begin{aligned} &\text{Median} \left(A_{ij}^{FCM_1-L}, A_{ij}^{FCM_2-L}, \dots, A_{ij}^{FCM_N-L} \right), \\ &\text{Median} \left(A_{ij}^{FCM_1-M}, A_{ij}^{FCM_2-M}, \dots, A_{ij}^{FCM_N-M} \right), \\ &\text{Median} \left(A_{ij}^{FCM_1-U}, A_{ij}^{FCM_2-U}, \dots, A_{ij}^{FCM_N-U} \right) \end{aligned} \right] \end{aligned}$$

If not including zero-valued edges in aggregation calculations, the median aggregate equation becomes:

$$\begin{aligned} A_{ij}^{aggregate} &= \left[\begin{aligned} &\text{Median} \left(A_{ij}^{FCM_1-L}, A_{ij}^{FCM_2-L}, \dots, A_{ij}^{FCM_N-L} \right) | A_{ij}^{FCM_1-L} \neq A_{ij}^{FCM_1-M} \neq A_{ij}^{FCM_1-U} \neq 0, \\ &\text{Median} \left(A_{ij}^{FCM_1-M}, A_{ij}^{FCM_2-M}, \dots, A_{ij}^{FCM_N-M} \right) | A_{ij}^{FCM_1-L} \neq A_{ij}^{FCM_1-M} \neq A_{ij}^{FCM_1-U} \neq 0, \\ &\text{Median} \left(A_{ij}^{FCM_1-U}, A_{ij}^{FCM_2-U}, \dots, A_{ij}^{FCM_N-U} \right) | A_{ij}^{FCM_1-L} \neq A_{ij}^{FCM_1-M} \neq A_{ij}^{FCM_1-U} \neq 0 \end{aligned} \right] \end{aligned}$$

Value

[aggregate]

An an aggregate adj. matrix with edges represented as the same data types as the inputs (i.e. Numerics for, IVFNs, or TFNs)

References

Aminpour P, Gray SA, Jetter AJ, Introne JE, Singer A, Arlinghaus R (2020). "Wisdom of Stakeholder Crowds in Complex Social–Ecological Systems." *Nature Sustainability*, 3(3), 191–199. ISSN 2398-9629, doi:10.1038/s41893-019-0467-z.

See Also

Other aggregate_fcms: `print.aggregate()`

Examples

```
# Aggregate Conventional FCMS
aggregate_fcms(sample_fcms$simple_fcms$conventional_fcms,
               agg_function = "mean",
               include_zeroes_in_sampling = TRUE,
               skip_checks = FALSE)

# Aggregate IVFN FCMS
aggregate_fcms(sample_fcms$simple_fcms$ivfn_fcms,
               agg_function = "mean",
               include_zeroes_in_sampling = TRUE,
               skip_checks = FALSE)

# Aggregate TFN FCMS
aggregate_fcms(sample_fcms$simple_fcms$tfn_fcms,
               agg_function = "mean",
               include_zeroes_in_sampling = TRUE,
               skip_checks = FALSE)

# NOTE: If the lower bound of an edge weight is 0, but the upper bound is > 0,
# the assumption that a stakeholder simply did not include the edge is invalid.
# Thus, since the 0-lower bound is meaningful, it is included in mean/median
# calculations even if include_zeroes = FALSE
aggregate_fcms(sample_fcms$simple_fcms$ivfn_fcms, "mean", include_zeroes_in_sampling = FALSE)
```

build_monte_carlo_fcms

Build Monte Carlo FCMS

Description

This function generates N fcm adjacency matrices whose edge weights are sampled from edge values (that may be numeric, IVFNs, or TFNs) and stores them as a list of adjacency matrices.

The show_progress and parallel inputs change the functions called, but do NOT change the output!

These are allowed to be toggled on/off to improve user control at runtime.

Usage

```
build_monte_carlo_fcms(
  adj_matrices = list(),
  n_samples = 1000L,
  include_zeroes_in_sampling = TRUE,
  show_progress = TRUE,
  silent = FALSE,
  skip_checks = FALSE
)
```

Arguments

adj_matrices	[list()] A single adjacency matrix or a list of adjacency matrices (n x n) representing FCMs. Matrices can have conventional edge weights, IVFN edge weights or TFN edge weights.
n_samples	[integer() - Positive] The number of samples to draw with the selected sampling method. Also, the number of sampled models to generate.
include_zeroes_in_sampling	[logical(1)] If TRUE, incorporate zero-weighted edges during Monte Carlo sampling.
show_progress	[logical(1)] If TRUE, show progress bars and print runtime updates in the console when performing FCM simulations.
silent	[logical(1)] If TRUE, suppress warning and error messages.
skip_checks	[logical(1)] FOR DEVELOPER USE ONLY. If TRUE, skip call to check_simulation_inputs (internal function)

Details

For Conventional FCMs, edge weights are sampled from edge weights explicitly defined in the input FCMs.

For IVFN and TFN FCMs, edge weights are sampled from the combined distributions representative of the IVFN/TFN edge weights. For example, if an edge is given the following weights across two maps: $[0.4, 0.8]$ and $[0.5, 0.7]$, the samples will be drawn from the combined distribution: `sample(N, c(runif(N, 0.4, 0.8), runif(N, 0.5, 0.7)), replace = TRUE)`.

Value

[list()]
A list of empirical (Conventional) FCM adj. matrices generated via monte carlo methods

Examples

```
# Conventional FCMs
conventional_mc_fcms <- build_monte_carlo_fcms(
  adj_matrices = sample_fcms$simple_fcms$conventional_fcms[1:10],
  n_samples = 1000,
  include_zeroes_in_sampling = TRUE,
  show_progress = FALSE,
  silent = FALSE,
  skip_checks = FALSE
)

# IVFN FCMs
ivfn_mc_fcms <- build_monte_carlo_fcms(
```

```

adj_matrices = sample_fcms$simple_fcms$ivfn_fcms[1:10],
n_samples = 1000,
include_zeroes_in_sampling = TRUE,
show_progress = FALSE,
silent = FALSE,
skip_checks = FALSE
)

# TFN FCMS
tfn_mc_fcms <- build_monte_carlo_fcms(
  adj_matrices = sample_fcms$simple_fcms$tfn_fcms[1:10],
  n_samples = 1000,
  include_zeroes_in_sampling = TRUE,
  show_progress = FALSE,
  silent = FALSE,
  skip_checks = FALSE
)

```

c.ivfn

c.ivfn

Description

This forces the output of `c()` to the equivalent of `list()` only for inputs of type `ivfn`

Usage

```
## S3 method for class 'ivfn'
c(...)
```

Arguments

... a set of `ivfn` objects

Details

For `ivfn` objects, `c()` combines all of the lower and upper data into a single `ivfn` object, but `list()` returns the expected output of a list of distinct `ivfn` objects.

Value

[list()]
A list of `ivfn` objects

See Also

Other interval-valued-fuzzy-numbers: [create_ivfn_fcm_from_conventional_fcm\(\)](#), [ivfn\(\)](#), [make_adj_matrix_w_ivfn](#), [print.ivfn\(\)](#), [subtract_ivfn\(\)](#)

Examples

```
c(ivfn(0, 1), ivfn(0.2, 0.5))
```

c.tfn

c.tfn

Description

This forces the output of `c()` to the equivalent of `list()` only for inputs of type `tfn`

Usage

```
## S3 method for class 'tfn'
c(...)
```

Arguments

... a set of `tfn` objects

Details

For `tfn` objects, `c()` combines all of the lower, mode, and upper data into a single `tfn` object, but `list()` returns the expected output of a list of distinct `tfn` objects.

Value

a list of `tfn` objects

Examples

```
c(tfn(0, 0.5, 1), tfn(0.2, 0.4, 0.5))
```

create_ivfn_fcm_from_conventional_fcm

Create IVFN-FCM from Conventional FCM

Description

This creates an IVFN-FCM from a Conventional FCM by adding error to the Conventional FCM edge weights.

Usage

```
create_ivfn_fcm_from_conventional_fcm(
  conventional_adj_matrix = data.frame(),
  error = 0.2
)
```

Arguments

`conventional_adj_matrix`
 [list() or data.frame()]
 A single, conventional adjacency matrix (n x n) representing FCMs.

`error`
 [double(1)]
 The +/- error defining the lower and upper bounds of the IVFN edge weights created from the numeric edge weights

Value

[adj_matrix_w_ivfns]
 An adjacency matrix (of class 'ivfn') with edges represented as IVFNs

See Also

Other interval-valued-fuzzy-numbers: [c.ivfn\(\)](#), [ivfn\(\)](#), [make_adj_matrix_w_ivfns\(\)](#), [print.ivfn\(\)](#), [subtract_ivfn\(\)](#)

Examples

```
create_ivfn_fcm_from_conventional_fcm(
  sample_fcms$sample_fcms$conventional_fcms[[1]],
  error = 0.2
)
```

```
create_tfn_fcm_from_conventional_fcm
  Create TFN-FCM from Conventional FCM
```

Description

This creates a TFN-FCM from a Conventional FCM by adding error to the Conventional FCM edge weights. The Conventional FCM edge weights are used as the mode values for the TFN edge weights.

Usage

```
create_tfn_fcm_from_conventional_fcm(
  conventional_adj_matrix = data.frame(),
  error = 0.2
)
```

Arguments

`conventional_adj_matrix`
 [list() or data.frame()]
 A single, conventional adjacency matrix (n x n) representing FCMs.

`error`
 [double(1)]
 The +/- error defining the lower and upper bounds of the TFN edge weights created from the numeric edge weights.

Value

[adj_matrix_w_tfns]
 An adjacency matrix (of class 'tfn') with edges represented as TFNs

See Also

Other triangular-fuzzy-numbers: [make_adj_matrix_w_tfns\(\)](#), [print.tfn\(\)](#), [subtract_tfn\(\)](#), [tfn\(\)](#)

Examples

```
create_tfn_fcm_from_conventional_fcm(
  sample_fcms$sample_fcms$conventional_fcms[[1]],
  error = 0.2
)
```

defuzz_ivfn_or_tfn	<i>Defuzz (IVFN or TFN)</i>
--------------------	-----------------------------

Description

Convert a fuzzy number to a crisp value. For IVFNs, return the average of the upper and lower bounds. For TFNs, return the average of the lower bound, the mode, and the upper bound.

Usage

```
defuzz_ivfn_or_tfn(fuzzy_number)
```

Arguments

`fuzzy_number` [ivfn or tfn] An IVFN or TFN object

Details

Defuzz-ing an IVFN to a crisp number:

$$[x^L, x^U] \mapsto \frac{x^L + x^U}{2}$$

where x^L , and x^U are the lower and upper bounds of the IVFN.

Defuzz-ing a TFN to a crisp number:

$$[x^L, x^M, x^U] \mapsto \frac{x^L + x^M + x^U}{3}$$

where x^L , x^M , and x^U are the lower bound, mode, and upper bound of the TFN.

Value

[numeric(1)]

A crisp number representative of the input fuzzy number.

Examples

```
defuzz_ivfn_or_tfn(ivfn(-1, 1))
defuzz_ivfn_or_tfn(tfn(-1, 0, 1))
```

estimate_fcm_lambda	<i>Estimate lambda</i>
---------------------	------------------------

Description

This function calculates the largest possible lambda for sigmoid and tanh squashing functions that guarantees convergence of an FCM simulation. Lambda is estimated such that 'squashed' simulation values are contained within the near-linear region of the sigmoid or tanh functions.

Note: The lambda value estimated will vary for different FCMs.

This algorithm is intended for conventional FCMs. If an Interval Value Fuzzy Number (IVFN) FCM or a Triangular Fuzzy Number (TFN) FCM is provided, estimate_fcm_lambda() will create a conventional FCM by averaging the edge weights (upper and lower for IVFN-FCMs; and upper, mode, and lower for TFN-FCMs) for use in calculating lambda,

Usage

```
estimate_fcm_lambda(
  adj_matrix = data.frame(),
  squashing = c("sigmoid", "tanh")
)
```

Arguments

adj_matrix	[list() or data.frame()] A single adjacency matrix (n x n) representing FCMs. An adjacency matrix can have conventional edge weights, IVFN edge weights or TFN edge weights.
squashing	[character(1)] The squashing function used. Must be one of the following: 'tanh' or 'sigmoid'.

Details

This algorithm was first developed in Kottas et al. 2010 (https://doi.org/10.1007/978-3-642-03220-2_5), and expanded upon by Harmati et al. 2018 (<https://doi.org/10.1109/FUZZ-IEEE.2018.8491447>), and Koutsellis et al. 2022 (<https://doi.org/10.1007/s12351-022-00717-x>)

The function applies an algorithm that can be used to optimize lambda. It compares the lambda calculated based on the Frobenius-norm (λ') and S-norm (λ^*) of an adjacency matrix and selects the minimum.

$$\lambda_s < \min(\lambda'_s, \lambda_s^*)$$

$$\lambda_h < \min(\lambda'_h, \lambda_h^*)$$

where λ_s is the lambda calculation when using the sigmoid squashing function and λ_h is the lambda calculation when using tanh.

The equations for λ' were developed by Harmati et al. 2018 (<https://doi.org/10.1109/FUZZ-IEEE.2018.8491447>) and are given below.

$$\lambda_s < \lambda'_s = \frac{4}{\|\mathbf{W}\|_F}$$

$$\lambda_h < \lambda'_h = \frac{1}{\|\mathbf{W}\|_F}$$

where $\|\mathbf{W}\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^n (w_{ij}^2)}$ is the Frobenius norm of the adj. matrix (or `norm(x, type = "F")`).

The equations for λ^* were developed by Koutsellis et al. 2022 (<https://doi.org/10.1007/s12351-022-00717-x>). Unlike for λ' , the calculations for λ^* follow different steps based on whether calculating λ for the sigmoid or tanh squashing function.

For sigmoid:

$$\|x_i^k\|_{\max} = \max \left(\left| 0.211 \cdot \sum_{i=1}^p w_{ij}^+ + 0.789 \cdot \sum_{i=1}^p w_{ij}^- \right|, \left| 0.211 \cdot \sum_{i=1}^p w_{ij}^- + 0.789 \cdot \sum_{i=1}^p w_{ij}^+ \right| \right)$$

$$\|\mathbf{W}\|_s = \max_i (\|x_i^k\|_{\max})$$

$$\lambda_s < \lambda_s^* = \frac{1.317}{\|\mathbf{W}\|_s}$$

For tanh:

$$\lambda_h < \lambda_h^* = \frac{1}{\|\mathbf{W}\|_\infty}$$

$$\|\mathbf{W}\|_\infty = \max_i \sum_{j=1}^n |w_{ij}|$$

Finally, the maximum lambda that ensures convergence of the simulation for the input adjacency matrix is the minimum of λ' and λ^* .

Note: This is only algorithm included at present, but the code for estimate_fcm_lambda is organized to streamline the addition of new algorithms in the future.

Value

[double(1)]

The maximum lambda that ensures simulation convergence for the input FCM.

References

Kottas TL, Boutalis YS, Christodoulou MA (2010). “Fuzzy cognitive networks: Adaptive network estimation and control paradigms.” In Kacprzyk J, Glykas M (eds.), Fuzzy Cognitive Maps, volume 247, 89–134. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-03219-6 978-3-642-03220-2, doi:10.1007/978-3-642-03220-2_5, Series Title: Studies in Fuzziness and Soft Computing,

Harmati IA, Koczy LT (2018). “On the Existence and Uniqueness of Fixed Points of Fuzzy Set Valued Sigmoid Fuzzy Cognitive Maps.” In 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 1–7. ISBN 9781509060207, doi:10.1109/FUZZ-IEEE.2018.8491447,

Koutsellis T, Xexakis G, Koasidis K, Nikas A, Doukas H (2022). “Parameter analysis for sigmoid and hyperbolic transfer functions of fuzzy cognitive maps.” Operational Research, 22(5), 5733–5763. ISSN 1109-2858, 1866-1505, doi:10.1007/s12351-022-00717-x,

Examples

```
estimate_fcm_lambda(sample_fcms$simple_fcms$conventional_fcms[[1]], squashing = "sigmoid")
```

```
estimate_fcm_lambda(sample_fcms$simple_fcms$ivfn_fcms[[1]], squashing = "tanh")
```

```
estimate_fcm_lambda(sample_fcms$simple_fcms$tfn_fcms[[1]], squashing = "sigmoid")
```

Description

Display an FCM in the Viewer pane as an interactive visNetwork object. Use the shiny parameter to interact with the FCM visNetwork object and store its output in the global environment.

The shiny app provides the following functionalities: (1) interactive node placement, (2) the ability to toggle node labels and edge labels on and off, (3) the ability to change node size and shape, (4) the ability to change font size, (5) the ability to change the size of the arrowhead on directional edges, and (6) the ability to change the curvature of edges and the smoothing method used.

Usage

```
fcm_view(
  adj_matrix = data.frame(),
  fcm_visNetwork = list(),
  shiny = FALSE,
  ...
)
```

Arguments

adj_matrix	[list() or data.frame()] A single adjacency matrix (n x n) representing FCMs. An adjacency matrix can have conventional edge weights, IVFN edge weights or TFN edge weights.
fcm_visNetwork	An fcm_view visNetwork object output. fcm_view accepts either an adj_matrix or fcm_visNetwork input but NOT both.
shiny	View visNetwork output in an interactive shiny app. Allows the user to manipulate and save node locations and other plot characteristics. Outputs are saved as fcm_visNetwork objects.
...	For advanced use. Set alert_on_open = FALSE to remove the alert pop-up that describes how to save visNetwork outputs.

Value

a visNetwork object that plots FCM networks in the Viewer pane; if shiny = TRUE, plots FCM networks in a shiny app

See Also

Other utility: [get_fcmconfr_inferences\(\)](#)

Examples

```
adj_matrix <- sample_fcms$simple_fcms$conventional_fcms[[1]]
fcm_view(adj_matrix = adj_matrix, shiny = FALSE)

adj_matrix <- sample_fcms$simple_fcms$ivfn_fcms[[1]]
fcm_view(adj_matrix = adj_matrix, shiny = FALSE)

adj_matrix <- sample_fcms$simple_fcms$tfm_fcms[[1]]
```

```
fcm_view(adj_matrix = adj_matrix, shiny = FALSE)
```

fcmconfr

fcmconfr

Description

This is the primary function of the fcmconfr package. This function performs the following three analyses on the input fuzzy cognitive maps (FCMs).

1. **Dynamic Simulation:** The dynamic behavior of one or more input FCMs is evaluated in response to a perturbation. Two types of simulations can be performed, pulse and clamped.
 - **Pulse:** In pulse simulations one or more nodes are transiently perturbed (the scenario) and allowed to relax back to equilibrium over multiple iterations to understand the effect those nodes have on the network. Scenarios are established using the initial state vector (non-zero values perturb a node). All values in the clamping vector must be set to zero (Stylios, 1997).
 - **Clamped:** In clamped simulations, one or more nodes of interest are continuously activated (the scenario) and the result is compared to baseline conditions. Scenarios are established using the clamping vector (non-zero values clamp a node) and the baseline is established by setting all nodes in the initial state vector to one (i.e., it represents the response of the network to a transient pulse event involving all nodes) (Ozesmi & Ozesmi, 2003).

The final response of each node of an FCM in a scenario is known as an inference. Inferences can be calculated using the maximum (peak) or the final resting state (final) of a node. In pulse simulations inferences are the difference between either of these two measures and zero. In clamped simulations inferences are the difference between either of these measures in the scenario and their equivalent values in the baseline simulation. Simulations can be run using different activation functions (Kosko, Modified-Kosko, or Rescale) and squashing functions (sigmoid, tanh)

Conventional FCM simulations use the traditional FCM simulation algorithm as established in Kosko (1986); Stylios (1997); Ozesmi & Ozesmi (2003), etc. (See Details for equations)

IVFN- and TFN-FCM simulations use the algorithm from Yesil et al. (2014) where average values of IVFN and TFN elements in state vectors are passed to the IVFN- or TFN-FCM adjacency matrix. (See Details for equations)

2. **Model Aggregation:** Generates a single "collective" adjacency matrix from a list of adjacency matrices. Aggregation is performed by calculating the mean/median edge weight for all edges included in a set of input FCMs (i.e. the mean/median of the edge weight connecting A->B across all maps, the mean/median of the edge weight connecting B->C across all maps, and so on) (Aminpour et al., 2020). The user specifies whether 0-valued edge weights (reflecting the absence of a connection between two nodes) should be included when calculating the mean/median.

Aggregate analysis can be toggled off to reduce runtime.

3. Monte Carlo Analysis: This method assesses uncertainty in dynamic simulations by generating a distribution of possible inferences. Edge weights are randomly sampled from the input FCMs to construct a set of N Monte Carlo FCMs. Each FCM undergoes dynamic simulation, producing N inferences per node. These inferences represent the range of possible values each node may assume.

Min, Max, Median, and 25th/75th quantiles are estimated for each node. Bootstrapping (optional) can also be performed to estimate 95% confidence bounds about the mean or median inference for each node.

Monte Carlo analysis can be toggled off to reduce runtime.

fcmconfr can accommodate three types of FCMs. Note that each FCM type must be analyzed separately (i.e. they cannot be co-evaluated in a single input set). The three supported types of FCMs include:

1. Conventional FCMs, where edge weights are single-valued fuzzy numbers (Stylios, 1997).
2. Interval-Valued Fuzzy Number (IVFN) FCMs, which are an extension of conventional FCMs. IVFN-FCMs represent edge weights as ranges $[\min, \max]$, where any value within the range is equally as likely as any other (i.e., the probability distribution for each edge is uniform) (Moore & Lodwick, 2003; Hajek & Prochazka, 2016); and
3. Triangular Fuzzy Number (TFN) FCMs, which are an extension of IVFN-FCMs. TFN-FCMs also represent edge weights as ranges, but assume that one value within the range (the mode) is most likely and values closer to the central value are more likely than those further away (i.e., the probability distribution for each edge is triangular) (Yesil et al., 2014).

The fcmconfr package includes fcmconfr_gui, an interactive GUI tool that can be used to select appropriate values for each argument in fcmconfr().

Usage

```
fcmconfr(
  adj_matrices = list(),
  agg_function = c("mean", "median"),
  num_mc_fcms = 1000L,
  initial_state_vector = c(),
  clamping_vector = c(),
  activation = c("kosko", "modified-kosko", "rescale"),
  squashing = c("sigmoid", "tanh"),
  lambda = 1,
  point_of_inference = c("peak", "final"),
  max_iter = 100L,
  min_error = 1e-05,
  ci_centering_function = c("mean", "median"),
  confidence_interval = 0.95,
  num_ci_bootstraps = 1000L,
  show_progress = TRUE,
  parallel = FALSE,
  n_cores = 1L,
  run_agg_calcs = TRUE,
```

```

run_mc_calcs = TRUE,
run_ci_calcs = TRUE,
include_zeroes_in_sampling = FALSE,
include_sims_in_output = TRUE,
silent = FALSE
)

```

Arguments

<code>adj_matrices</code>	[list()] A single adjacency matrix or a list of adjacency matrices (n x n) representing FCMs. Matrices can have conventional edge weights, IVFN edge weights or TFN edge weights.
<code>agg_function</code>	[character(1)] Aggregate the adj. matrices into a single FCM by taking either the mean or median of the edge weights for edges included in multiple maps
<code>num_mc_fcms</code>	[integer(1) - Positive] The number of inferences to generate via Monte Carlo sampling. Omit this argument when analyzing a single, conventional FCM.
<code>initial_state_vector</code>	[vector("double")] A list of state values (one per node) at the start of an FCM simulation. In pulse simulations the <code>initial_state_vector</code> controls the scenario (i.e., a non-zero value is a transient perturbation). In clamped simulations all values in the <code>initial_state_vector</code> are set to 1.
<code>clamping_vector</code>	[vector("double")] A list of values (one per node) that indicates whether clamped simulations will be performed. In clamped simulations the <code>clamping_vector</code> controls the scenario (nodes assigned non-zero values will remain at those values for the entire simulation). In pulse simulations all values in the <code>clamping_vector</code> are set to 0.
<code>activation</code>	[character(1)] The activation function used. Must be one of the following: 'kosko', 'modified-kosko', or 'rescale'.
<code>squashing</code>	[character(1)] The squashing function used. Must be one of the following: 'tanh' or 'sigmoid'.
<code>lambda</code>	[double(1)] Positive A numeric value that defines the steepness of the squashing function's slope.
<code>point_of_inference</code>	[character(1)] Definition of an inference. The metric used to calculate the response of each node to a scenario of interest from a simulation timeseries. Must be one of the following: 'peak' (the maximum value) or 'final' (the state at equilibrium).
<code>max_iter</code>	[integer(1) - Positive] The maximum number of iterations to run (increase if the minimum error value is not achieved).

min_error	[double(1) - Positive] The error past which a simulation has converged and no further iterations are necessary. <i>Error equals the sum of the absolute value of the current state vector minus the previous state vector.</i>
ci_centering_function	[character(1)] Estimate confidence intervals about the "mean" or "median" of inferences from Monte Carlo simulations
confidence_interval	[double(1) - Positive (between 0 and 1)] Bootstrapped confidence level
num_ci_bootstraps	[integer(1) - Positive] Number of bootstrap draws
show_progress	[logical(1)] If TRUE, show progress bars and print runtime updates in the console when performing FCM simulations.
parallel	[logical(1)] If TRUE, utilize parallel processing.
n_cores	[integer(1) - Positive] The number of cores to use in parallel processing. If no input given, all available cores will be used.
run_agg_calcs	[logical(1)] If TRUE, run the code to generate and simulate an aggregate FCM generated from the input adj_matrices.
run_mc_calcs	[logical(1)] If TRUE, run the code to generate and simulate monte carlo-generated FCM sampled from the input adj_matrices
run_ci_calcs	TRUE/FALSE Run the code to estimate the 95 percent CI bounds about the means of the inferences of the monte carlo adj matrices
include_zeroes_in_sampling	[logical(1)] If TRUE, incorporate zeroes as intentionally-defined edge weights or ignore them when aggregating adjacency matrices and sampling for Monte Carlo FCMs.
include_sims_in_output	[logical(1)] If TRUE, include simulations and inferences in output. Set to FALSE to reduce output size.
silent	[logical(1)] If TRUE, suppress warning and error messages.

Details

FCM simulations are iterative applications of an activation function (f_A) that describes how node values at a particular state ($C^{(t)}$) are influenced by the edge weights defined in the adjacency matrix (w) to calculate the node values at the following state ($C^{(t+1)}$), and a squashing function (f_S) that restricts node values within a particular range.

In conventional FCM simulations the current state vector ($C^{(t)}$) is used to calculate the next state vector ($C^{(t+1)}$).

$$C_i^{(t+1)} = f_S \left(f_A \left(w, C_i^{(t)} \right) \right)$$

In IVFN- and TFN-FCM simulations, the current state vector has to be translated into crisp values prior to calculating the next state vector.

Formulas for crisp values:

$$\begin{aligned} \left(\overline{C}_i^{(t)} \right)_{IVFN} &= \frac{\left(C_i^{(t)} \right)_{lower} + \left(C_i^{(t)} \right)_{upper}}{2} \\ \left(\overline{C}_i^{(t)} \right)_{TFN} &= \frac{\left(C_i^{(t)} \right)_{lower} + \left(C_1^{(t)} \right)_{mode} + \left(C_i^{(t)} \right)_{upper}}{3} \end{aligned}$$

Formulas for next state vector:

$$\begin{aligned} \left(C_i^{(t+1)} \right)_{IVFN} &= \left[\left(C_i^{(t+1)} \right)_{lower}, \left(C_i^{(t+1)} \right)_{upper} \right] \text{ where } \begin{cases} \left(C_i^{(t+1)} \right)_{lower} = f_S \left(f_A \left(w_{lower}, \overline{C}_i^{(t)} \right) \right) \\ \left(C_i^{(t+1)} \right)_{upper} = f_S \left(f_A \left(w_{upper}, \overline{C}_i^{(t)} \right) \right) \end{cases} \\ \left(C_i^{(t+1)} \right)_{TFN} &= \left[\left(C_i^{(t+1)} \right)_{lower}, \left(C_i^{(t+1)} \right)_{upper} \right] \text{ where } \begin{cases} \left(C_i^{(t+1)} \right)_{lower} = f_S \left(f_A \left(w_{lower}, \overline{C}_i^{(t)} \right) \right) \\ \left(C_i^{(t+1)} \right)_{mode} = f_S \left(f_A \left(w_{mode}, \overline{C}_i^{(t)} \right) \right) \\ \left(C_i^{(t+1)} \right)_{upper} = f_S \left(f_A \left(w_{upper}, \overline{C}_i^{(t)} \right) \right) \end{cases} \end{aligned}$$

fcmconfr supports the Kosko, Modified-Kosko, and Rescale activation functions as defined here:

- Kosko (Kosko, 1986)

$$C_i^{(t+1)} = f \left(\sum_{\substack{j = i \\ i \neq j}}^M w_{ji} C_j^{(t)} \right)$$

- Modified-Kosko (Stylio & Groumpou, 2004)

$$C_i^{(t+1)} = f \left(\sum_{\substack{j = i \\ i \neq j}}^M w_{ji} C_j^{(t)} + C_i^{(t)} \right)$$

- Rescale (Papageorgiou, 2011; Dikopoulou, 2021)

$$C_i^{(t+1)} = f \left(\sum_{\substack{j = i \\ i \neq j}}^M w_{ji} \left(2C_j^{(t)} - 1 \right) + \left(2C_i^{(t)} - 1 \right) \right)$$

fcmconfr supports the sigmoid and hyperbolic tangent (tanh) squashing functions.

$$f_{sigmoid}(x) = \frac{1}{1 + e^{\lambda x}}$$

$$f_{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Value

A list of outputs generated from the individual_fcms simulations, aggregate_fcm analysis, and monte_carlo_fcms analysis. Bootstrap estimates of inferences from monte carlo analysis are included, as well as function inputs.

References

- Özesmi U, Özesmi S (2003). “A Participatory Approach to Ecosystem Conservation: Fuzzy Cognitive Maps and Stakeholder Group Analysis in Uluabat Lake, Turkey.” *Environmental Management*, 31(4), 518–531. ISSN 0364-152X, 1432-1009, doi:10.1007/s00267-002-2841-1.
- Aminpour P, Gray SA, Jetter AJ, Introne JE, Singer A, Arlinghaus R (2020). “Wisdom of Stakeholder Crowds in Complex Social–Ecological Systems.” *Nature Sustainability*, 3(3), 191–199. ISSN 2398-9629, doi:10.1038/s41893-019-0467-z.
- Stylios CD, Georgopoulos VC, Groumpos PP (1997). “Introducing the Theory of Fuzzy Cognitive Maps in Distributed Systems.” In *Proceedings of 12th IEEE International Symposium on Intelligent Control*, 55–60. doi:10.1109/ISIC.1997.626413.
- Moore R, Lodwick W (2003). “Interval Analysis and Fuzzy Set Theory.” *Fuzzy Sets and Systems*, 135(1), 5–9. ISSN 01650114, doi:10.1016/S0165-0114(02)00246-4.
- Hajek P, Prochazka O (2016). “Interval-valued fuzzy cognitive maps for supporting business decisions.” In *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 531–536. ISBN 978-1-5090-0626-7, doi:10.1109/FUZZ-IEEE.2016.7737732,
- Yesil E, Dodurka MF, Urbas L (2014). “Triangular fuzzy number representation of relations in Fuzzy Cognitive Maps.” In *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 1021–1028. ISBN 9781479920723, doi:10.1109/FUZZ-IEEE.2014.6891653,
- Stylios CD, Groumpos PP (2004). “Modeling complex systems using fuzzy cognitive maps.” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(1), 155–162. ISSN 1558-2426, doi:10.1109/TSMCA.2003.818878,
- Papageorgiou EI (2011). “A new methodology for Decisions in Medical Informatics using fuzzy cognitive maps based on fuzzy rule-extraction techniques.” *Applied Soft Computing*, 11(1), 500–513. ISSN 1568-4946, doi:10.1016/j.asoc.2009.12.010,
- Dikopoulou Z (2021). “Fuzzy Cognitive Maps.” In Dikopoulou Z (ed.), *Modeling and Simulating Complex Business Perceptions : Using Graphical Models and Fuzzy Cognitive Maps*, 27–42. Springer International Publishing, Cham. ISBN 9783030814960, doi:10.1007/978-3-030-81496-0_3,

Examples

```
# Conventional FCMs
ex_conventional_fcmconfr <- fcmconfr(
  adj_matrices = sample_fcms$simple_fcms$conventional_fcms[1:10],
  # Aggregation and Monte Carlo Sampling
  agg_function = 'mean',
  num_mc_fcms = 100L,
  # Simulation
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = 'modified-kosko',
  squashing = 'sigmoid',
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 100L,
  min_error = 1e-05,
  # Inference Estimation (bootstrap)
  ci_centering_function = "mean",
  confidence_interval = 0.95,
  # Runtime Options
  show_progress = TRUE,
  parallel = FALSE,
  n_cores = 1L,
  # Additional Options
  run_agg_calcs = TRUE,
  run_mc_calcs = TRUE,
  run_ci_calcs = TRUE,
  include_zeroes_in_sampling = FALSE,
  include_sims_in_output = TRUE
)
```

```
# IVFN FCM fcmconfr
ex_ivfn_fcmconfr <- fcmconfr(
  adj_matrices = sample_fcms$simple_fcms$ivfn_fcms[1:10],
  # Aggregation and Monte Carlo Sampling
  agg_function = 'mean',
  num_mc_fcms = 100L,
  # Simulation
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = 'modified-kosko',
  squashing = 'sigmoid',
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 100L,
  min_error = 1e-05,
  # Inference Estimation (bootstrap)
  ci_centering_function = "mean",
  confidence_interval = 0.95,
  # Runtime Options
  show_progress = TRUE,
```

```

parallel = FALSE,
n_cores = 1L,
# Additional Options
run_agg_calcs = TRUE,
run_mc_calcs = TRUE,
run_ci_calcs = TRUE,
include_zeroes_in_sampling = FALSE,
include_sims_in_output = TRUE
)

# TFN FCM fcmconfr
ex_tfn_fcmconfr <- fcmconfr(
  adj_matrices = sample_fcms$simple_fcms$tfnc_fcms[1:10],
  # Aggregation and Monte Carlo Sampling
  agg_function = 'mean',
  num_mc_fcms = 100L,
  # Simulation
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = 'modified-kosko',
  squashing = 'sigmoid',
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 100L,
  min_error = 1e-05,
  # Inference Estimation (bootstrap)
  ci_centering_function = "mean",
  confidence_interval = 0.95,
  # Runtime Options
  show_progress = TRUE,
  parallel = FALSE,
  n_cores = 1L,
  # Additional Options
  run_agg_calcs = TRUE,
  run_mc_calcs = TRUE,
  run_ci_calcs = TRUE,
  include_zeroes_in_sampling = FALSE,
  include_sims_in_output = TRUE
)

```

fcmconfr_gui

fcmconfr_gui

Description

fcmconfr_gui performs two tasks: (1) it launches a Shiny app that lets users interactively select parameters and (2) it outputs a corresponding call to [fcmconfr](#) that users can copy-and-paste to run in their own scripts.

No inputs are provided to fcmconfr_gui, but the local environment must have an individual FCM or a list object containing multiple FCMs in order for fcmconfr_gui to be used. Once fcmconfr_gui

is running the user selects the appropriate FCMs to analyze from a dropdown menu titled Adj. Matrix or List of Adj. Matrices.

The GUI interface is organized into four tabs: Data, Agg. and Monte Carlo Options, Simulation Options, and Runtime Options, which are described further below.

Data: In the Data tab, the user selects the FCMs they want to analyze (drop-down menu titled 'Adj. Matrix or List of Adj. Matrices'). It is also where they manipulate values in the Initial State (Pulse) Vector or Clamping Vector to identify the dynamic simulations they wish to explore. A common form of simulation is a "clamped" simulation where every value in the Initial State (Pulse) Vector is set to 1 and only the node(s)-of-interest are set to 1 in the Clamping Vector. In this configuration, the Initial State (Pulse) Vector is used to determine the network's state at equilibrium and the Clamping Vector is used to determine the network's state in response to continuous activation of one or more nodes. The difference between the two is a measure of the impact of those particular nodes on the system.

Agg. and Monte Carlo Option: In the Agg. and Monte Carlo Options tab, the user specifies their preferences regarding FCM aggregation and Monte Carlo sampling. Aggregation options are only available when datasets contain multiple FCMs. Monte Carlo sampling is available for list objects of conventional, IVFN-, and TFN-FCMs as well as individual IVFN-/TFN-FCMs, but not individual conventional FCMs. Aggregation and Monte Carlo options can be toggled off to speed up the simulation process.

- Aggregation options allow the user to specify whether multiple FCMs should be aggregated into a single collective model so that dynamic simulations can be performed on the aggregate in addition to individual FCMs. The user can specify the aggregation method (mean or median) and whether zero-weighted edges should be included in the aggregation process.
- Monte Carlo options allow the user to specify whether Monte Carlo sampling should be performed to estimate uncertainty in dynamic simulation outputs. If yes, the user must specify the number of FCMs that will be generated by Monte Carlo sampling (we recommend 1000 or more). Dynamic simulations will be performed on each of these FCMs, providing a range of inferences for each node, from which the median state and quantiles (25th, 75th) will be estimated. The user also has the option to perform nonparametric bootstrapping to estimate confidence bounds about the average inference for each node. Confidence limits are user specified (e.g., 95th percentile, 90th percentile, etc.).

Simulation Options: In the Simulation Options tab, the user specify the type of dynamic simulation to perform. This includes specifying (1) the activation function (Kosko, Modified-Kosko, or Rescale; default of Kosko), (2) the squashing function (sigmoid or hyperbolic tangent; default of sigmoid), (3) the squashing function's key parameter, lambda (default of 1), (4) the final simulation output (i.e., the peak estimate for each node or its final resting state; default of final) (5) the maximum number of iterations to perform per simulation (default of 100), and (6) the minimum acceptable error between iterations (default of $1 * 10^{-5}$)

Runtime Options: In the Runtime Options tab, the user may specify whether they want to use parallel processing and have a progress bar displayed at runtime. These options do not impact simulation outputs. They do, however, influence how long `fcmconfr` takes to run and what the user sees at runtime.

A brief summary of each parameter within the `fcmconfr_gui` is provided in a glossary stored in a side tab within the GUI. The side tab can be opened by clicking the arrow symbol in the top-right-hand-corner of the GUI.

Usage

```
fcmconfr_gui()
```

Value

On exit, this function outputs a copy-and-paste sample script to call [fcmconfr](#) with the selected inputs.

Examples

```
NULL
```

```
get_fcmconfr_inferences
```

Get inferences from an fcmconfr output

Description

Given an fcmconfr output object, return the inferences of all or just a specific analysis

Usage

```
get_fcmconfr_inferences(  
  fcmconfr_result_obj = list(),  
  analysis = c("individual", "aggregate", "mc")  
)
```

Arguments

fcmconfr_result_obj	[fcmconfr] A direct output from the fcmconfr function
analysis	[character()] The analysis for which inferences to return (HELP?)

Value

[list()] or data.frame()
A dataframe (or list of dataframes) of inferences from the selected analysis (analyses)

See Also

Other utility: [fcm_view\(\)](#)

Examples

```

ex_conventional_fcmconfr <- fcmconfr(
  adj_matrices = sample_fcms$simple_fcms$conventional_fcms,
  # adj_matrices = group_conventional_fcms,
  # Aggregation and Monte Carlo Sampling
  agg_function = 'mean',
  num_mc_fcms = 100L,
  # Simulation
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = 'rescale',
  squashing = 'sigmoid',
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 100L,
  min_error = 1e-05,
  # Inference Estimation (bootstrap)
  ci_centering_function = "mean",
  confidence_interval = 0.95,
  num_ci_bootstraps = 100L,
  # Runtime Options
  show_progress = TRUE,
  parallel = FALSE,
  n_cores = 1L,
  # Additional Options
  run_agg_calcs = TRUE,
  run_mc_calcs = TRUE,
  run_ci_calcs = TRUE,
  include_zeroes_in_sampling = TRUE,
  include_sims_in_output = TRUE
)

get_fcmconfr_inferences(ex_conventional_fcmconfr, analysis = c("individual"))
get_fcmconfr_inferences(ex_conventional_fcmconfr, analysis = c("individual", "aggregate"))
get_fcmconfr_inferences(ex_conventional_fcmconfr, analysis = c("individual", "aggregate", "mc"))
get_fcmconfr_inferences(ex_conventional_fcmconfr)

ex_ivfn_fcmconfr <- fcmconfr(
  adj_matrices = sample_fcms$simple_fcms$ivfn_fcms,
  # adj_matrices = group_ivfn_fcms,
  # Aggregation and Monte Carlo Sampling
  agg_function = 'mean',
  num_mc_fcms = 100L,
  # Simulation
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = 'rescale',
  squashing = 'sigmoid',
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 100L,
  min_error = 1e-05,

```

```

# Inference Estimation (bootstrap)
ci_centering_function = "mean",
confidence_interval = 0.95,
num_ci_bootstraps = 10L,
# Runtime Options
show_progress = TRUE,
parallel = FALSE,
n_cores = 1L,
# Additional Options
run_agg_calcs = TRUE,
run_mc_calcs = TRUE,
run_ci_calcs = TRUE,
include_zeroes_in_sampling = TRUE,
include_sims_in_output = TRUE
)
get_fcmconfr_inferences(ex_ivfn_fcmconfr, analysis = c("individual"))
get_fcmconfr_inferences(ex_ivfn_fcmconfr, analysis = c("individual", "aggregate"))
get_fcmconfr_inferences(ex_ivfn_fcmconfr, analysis = c("individual", "aggregate", "mc"))
get_fcmconfr_inferences(ex_ivfn_fcmconfr)

ex_tfn_fcmconfr <- fcmconfr(
  adj_matrices = sample_fcms$simple_fcms$tfn_fcms,
  # adj_matrices = group_tfn_fcms,
  # Aggregation and Monte Carlo Sampling
  agg_function = 'mean',
  num_mc_fcms = 1000L,
  # Simulation
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = 'rescale',
  squashing = 'sigmoid',
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 1000L,
  min_error = 1e-05,
  # Inference Estimation (bootstrap)
  ci_centering_function = "mean",
  confidence_interval = 0.95,
  num_ci_bootstraps = 1000L,
  # Runtime Options
  show_progress = TRUE,
  parallel = FALSE,
  n_cores = 1L,
  # Additional Options
  run_agg_calcs = TRUE,
  run_mc_calcs = TRUE,
  run_ci_calcs = TRUE,
  include_zeroes_in_sampling = TRUE,
  include_sims_in_output = TRUE
)
get_fcmconfr_inferences(ex_tfn_fcmconfr, analysis = c("individual"))
get_fcmconfr_inferences(ex_tfn_fcmconfr, analysis = c("individual", "aggregate"))

```

```
get_fcmconfr_inferences(ex_tfn_fcmconfr, analysis = c("individual", "aggregate", "mc"))
get_fcmconfr_inferences(ex_tfn_fcmconfr)
```

infer_fcm

Infer FCM

Description

This compares the baseline simulation of an fcm with the input scenario (scenario vector) to estimate how outputs change compared to the structural or expected behavior of the system.

Usage

```
infer_fcm(
  adj_matrix = data.frame(),
  initial_state_vector = c(),
  clamping_vector = c(),
  activation = c("kosko", "modified-kosko", "rescale"),
  squashing = c("sigmoid", "tanh"),
  lambda = 1,
  point_of_inference = c("peak", "final"),
  max_iter = 100L,
  min_error = 1e-05,
  skip_checks = FALSE
)
```

Arguments

adj_matrix	[list() or data.frame()] A single adjacency matrix (n x n) representing FCMs. An adjacency matrix can have conventional edge weights, IVFN edge weights or TFN edge weights.
initial_state_vector	[vector("double")] A list of state values (one per node) at the start of an FCM simulation. In pulse simulations the initial_state_vector controls the scenario (i.e., a non-zero value is a transient perturbation). In clamped simulations all values in the initial_state_vector are set to 1.
clamping_vector	[vector("double")] A list of values (one per node) that indicates whether clamped simulations will be performed. In clamped simulations the clamping_vector controls the scenario (nodes assigned non-zero values will remain at those values for the entire simulation). In pulse simulations all values in the clamping_vector are set to 0.
activation	[character(1)] The activation function used. Must be one of the following: 'kosko', 'modified-kosko', or 'rescale'.

squashing	[character(1)] The squashing function used. Must be one of the following: 'tanh' or 'sigmoid'.
lambda	[double(1)] Positive A numeric value that defines the steepness of the squashing function's slope.
point_of_inference	[character(1)] Definition of an inference. The metric used to calculate the response of each node to a scenario of interest from a simulation timeseries. Must be one of the following: 'peak' (the maximum value) or 'final' (the state at equilibrium).
max_iter	[integer(1) - Positive] The maximum number of iterations to run (increase if the minimum error value is not achieved).
min_error	[double(1) - Positive] The error past which a simulation has converged and no further iterations are necessary. <i>Error equals the sum of the absolute value of the current state vector minus the previous state vector.</i>
skip_checks	[logical(1)] FOR DEVELOPER USE ONLY. If TRUE, skip call to check_simulation_inputs (internal function)

Details

This function performs two fcm simulations and compares the output between the two. The first simulation considers the baseline activity where no nodes are "clamped" and the system behaves without any outside inputs. The second simulation considers a scenario where one or multiple nodes are "clamped" so that the system is reactive to additional inputs. The function returns the difference in simulation results between the scenario and baseline activity to understand how system manipulations compare to structural expectations of the system.

This function produces the same output as mental modeler for the following inputs:

- initial_state_vector = c(1, 1, ..., 1)
- activation = "kosko"
- squashing = either "sigmoid" or "tanh"
- lambda = 1

Value

[infer_conventional_fcm or infer_ivfn_or_tfn_fcm]

A list of FCM inference results (including baseline and simulation outputs)

Examples

```
# # Inference w/ Conventional FCMs
conventional_inference <- infer_fcm(sample_fcms$simple_fcms$conventional_fcms[[4]],
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = "modified-kosko",
```

```

squashing = "sigmoid",
lambda = 1.0,
point_of_inference = "final",
max_iter = 100L,
min_error = 1e-5,
skip_checks = FALSE)

# # Inference w/ Interval-Valued Fuzzy Numbers (IVFNs)
ivfn_inference <- infer_fcm(sample_fcms$simple_fcms$ivfn_fcms[[4]],
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = "kosko",
  squashing = "tanh",
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 100L,
  min_error = 1e-5,
  skip_checks = FALSE)

#

# # Inference w/ Triangular Fuzzy Numbers (TFNs)
tfn_inference <- infer_fcm(sample_fcms$simple_fcms$tfn_fcms[[4]],
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = "rescale",
  squashing = "sigmoid",
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 100L,
  min_error = 1e-5,
  skip_checks = FALSE)

```

infer_fcm_set

*Infer Multiple FCMs in a List***Description**

This function simulates a set of FCMs (Conventional, IVFN, and/or TFN) (whose edge weights were sampled using monte carlo methods) by iteratively calling the `infer_fcm` function for each adj. matrix in the set.

Usage

```

infer_fcm_set(
  adj_matrices = list(matrix()),
  initial_state_vector = c(),
  clamping_vector = c(),
  activation = c("kosko", "modified-kosko", "rescale"),
  squashing = c("sigmoid", "tanh"),

```

```

lambda = 1,
point_of_inference = c("peak", "final"),
max_iter = 100L,
min_error = 1e-05,
parallel = TRUE,
n_cores = 1L,
show_progress = TRUE,
include_sims_in_output = FALSE,
silent = FALSE,
skip_checks = FALSE
)

```

Arguments

adj_matrices	[list()] A single adjacency matrix or a list of adjacency matrices (n x n) representing FCMs. Matrices can have conventional edge weights, IVFN edge weights or TFN edge weights.
initial_state_vector	[vector("double")] A list of state values (one per node) at the start of an FCM simulation. In pulse simulations the initial_state_vector controls the scenario (i.e., a non-zero value is a transient perturbation). In clamped simulations all values in the initial_state_vector are set to 1.
clamping_vector	[vector("double")] A list of values (one per node) that indicates whether clamped simulations will be performed. In clamped simulations the clamping_vector controls the scenario (nodes assigned non-zero values will remain at those values for the entire simulation). In pulse simulations all values in the clamping_vector are set to 0.
activation	[character(1)] The activation function used. Must be one of the following: 'kosko', 'modified-kosko', or 'rescale'.
squashing	[character(1)] The squashing function used. Must be one of the following: 'tanh' or 'sigmoid'.
lambda	[double(1)] Positive A numeric value that defines the steepness of the squashing function's slope.
point_of_inference	[character(1)] Definition of an inference. The metric used to calculate the response of each node to a scenario of interest from a simulation timeseries. Must be one of the following: 'peak' (the maximum value) or 'final' (the state at equilibrium).
max_iter	[integer(1) - Positive] The maximum number of iterations to run (increase if the minimum error value is not achieved).

min_error	[double(1) - Positive] The error past which a simulation has converged and no further iterations are necessary. <i>Error equals the sum of the absolute value of the current state vector minus the previous state vector.</i>
parallel	[logical(1)] If TRUE, utilize parallel processing.
n_cores	[integer(1) - Positive] The number of cores to use in parallel processing. If no input given, all available cores will be used. Should be a positive integer.
show_progress	[logical(1)] If TRUE, show progress bars and print runtime updates in the console when performing FCM simulations.
include_sims_in_output	[logical(1)] If TRUE, include simulations and inferences in output. Set to FALSE to reduce output size.
silent	[logical(1)] If TRUE, suppress warning and error messages.
skip_checks	[logical(1)] FOR DEVELOPER USE ONLY. If TRUE, skip call to check_simulation_inputs (internal function)

Details

The show_progress and parallel inputs change how the function is run, but do NOT change the output! These are allowed to be toggled on/off to improve user control at runtime.

Value

[inference_of_fcm_set]
A list/data.frame of the inferences (and simulations if included) for each FCM in the set

Examples

```
# Set of Conventional FCMs
conventional_fcm_set_inferences <- infer_fcm_set(
  adj_matrices = sample_fcms$simple_fcms$conventional_fcms[1:5],
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = "modified-kosko",
  squashing = "sigmoid",
  lambda = 1.0,
  point_of_inference = "final",
  max_iter = 100L,
  min_error = 1e-5,
  parallel = FALSE,
  n_cores = 1L,
  show_progress = FALSE,
  include_sims_in_output = TRUE,
```

```

    silent = FALSE,
    skip_checks = FALSE
  )

  # Set of IVFN FCMs
  ivfn_fcm_set_inferences <- infer_fcm_set(
    adj_matrices = sample_fcms$simple_fcms$ivfn_fcms[1:5],
    initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
    clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
    activation = "modified-kosko",
    squashing = "sigmoid",
    lambda = 1.0,
    point_of_inference = "final",
    max_iter = 100L,
    min_error = 1e-5,
    parallel = FALSE,
    n_cores = 1L,
    show_progress = FALSE,
    include_sims_in_output = TRUE,
    silent = FALSE,
    skip_checks = FALSE
  )

  # Set of TFN FCMs
  tfn_fcm_set_inferences <- infer_fcm_set(
    adj_matrices = sample_fcms$simple_fcms$tfn_fcms[1:5],
    initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
    clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
    activation = "modified-kosko",
    squashing = "sigmoid",
    lambda = 1.0,
    point_of_inference = "final",
    max_iter = 100L,
    min_error = 1e-5,
    parallel = FALSE,
    n_cores = 1L,
    show_progress = FALSE,
    include_sims_in_output = TRUE,
    silent = FALSE,
    skip_checks = FALSE
  )

```

ivfn

Interval-Valued Fuzzy Number (IVFN) - S3 Class

Description

This constructs an interval-valued fuzzy number (ivfn) that represents a continuous, uniform distribution of values within a given range

Usage

```
ivfn(lower = double(), upper = double())
```

Arguments

lower	[double() - Unrestricted (positive or negative)] The lower bound of an IVFN
upper	[double() - Unrestricted (positive or negative)] The upper bound of an IVFN

Details

The IVFN class does not perform any operations on its input, rather it checks whether the input follows the defining criteria of IVFNs

For IVFNs, the lower bound must be less than or equal to the upper bound. If the lower bound and upper bound are equal, the IVFN represents a "crisp" numeric value.

$$\text{IVFN} : [x^L, x^U]$$

where x^L and x^U are the lower and upper bounds of the IVFN.

Value

[ivfn]
An interval-valued fuzzy number (IVFN)

References

Moore R, Lodwick W (2003). "Interval Analysis and Fuzzy Set Theory." Fuzzy Sets and Systems, 135(1), 5–9. ISSN 01650114, doi:[10.1016/S0165-0114\(02\)00246-4](https://doi.org/10.1016/S0165-0114(02)00246-4).

Dimuro GP (2011). "On Interval Fuzzy Numbers." In 2011 Workshop-School on Theoretical Computer Science, 3–8. ISBN 978-0-7695-4628-5 978-1-4673-0225-8, doi:[10.1109/WEIT.2011.19](https://doi.org/10.1109/WEIT.2011.19).

See Also

Other interval-valued-fuzzy-numbers: [c.ivfn\(\)](#), [create_ivfn_fcm_from_conventional_fcm\(\)](#), [make_adj_matrix_w_ivfns\(\)](#), [print.ivfn\(\)](#), [subtract_ivfn\(\)](#)

Examples

```
ivfn(lower = 0.0, upper = 1.0)
ivfn(-1.0, 1.0)
```

make_adj_matrix_w_ivfns

Create Adj. Matrix w/ Edges Represented as IVFNs

Description

This constructs an adjacency matrix with edges represented by interval-valued fuzzy numbers (IVFNs) from an adjacency matrix of lower bounds and an adjacency matrix of upper bounds

Usage

```
make_adj_matrix_w_ivfns(lower = data.frame(), upper = data.frame())
```

Arguments

lower	[list() or data.frame()] An n x n adjacency matrix that represents the lower limits of edges in an FCM
upper	[list() or data.frame()] An n x n adjacency matrix that represents the upper limits of edges in an FCM

Details

The input adjacency matrices must be square n x n matrices with the same dimensions. The input can be either matrix, data.table, tibble, or data.table type objects, but the output will always be a data.frame. This is for output readability. data.table and tibble objects work logically, but their outputs require additional steps to parse from the user's perspective.

If the input matrices have named columns, those names will be carried over in the IVFN-FCM adjacency matrix. Otherwise, generic node IDs will be used (C1, C2, ... Cn).

Value

[adj_matrix_w_ivfns]
An adjacency matrix (of class 'ivfn') with edges represented as IVFNs

See Also

Other interval-valued-fuzzy-numbers: [c.ivfn\(\)](#), [create_ivfn_fcm_from_conventional_fcm\(\)](#), [ivfn\(\)](#), [print.ivfn\(\)](#), [subtract_ivfn\(\)](#)

Examples

```
lower_adj_matrix <- data.frame(
  "A" = c(0, 0),
  "B" = c(0.4, 0)
)
upper_adj_matrix <- data.frame(
  "A" = c(0, 0),
  "B" = c(0.6, 0)
```

```
)
make_adj_matrix_w_ivfns(lower = lower_adj_matrix, upper = upper_adj_matrix)
```

```
make_adj_matrix_w_tfns
```

Create Adj. Matrix w/ Edges Represented as TFNs

Description

This constructs an adjacency matrix with edges represented by triangular fuzzy numbers (TFNs) from an adjacency matrix of lower bounds, an adjacency matrix of modes, and an adjacency matrix of upper bounds

Usage

```
make_adj_matrix_w_tfns(
  lower = data.frame(),
  mode = data.frame(),
  upper = data.frame()
)
```

Arguments

lower	[list() or data.frame()] An n x n adjacency matrix that represents the lower limits of edges in an FCM
mode	[list() or data.frame()] An n x n adjacency matrix that represents the modes (most likely values) of edges in an FCM
upper	[list() or data.frame()] An n x n adjacency matrix that represents the upper limits of edges in an FCM

Details

The input adjacency matrices must be square n x n matrices with the same dimensions. The input can be either matrix, data.table, tibble, or data.table type objects, but the output will always be a data.frame. This is for output readability. data.table and tibble objects work logically, but their outputs require additional steps to parse from the user's perspective.

If the input matrices have named columns, those names will be carried over in the triangular adjacency matrix. Otherwise, generic node IDs will be used (C1, C2, ... Cn).

Value

```
[adj_matrix_w_tfns]
An adjacency matrix (of class 'tfn') with edges represented as TFNs
```

See Also

Other triangular-fuzzy-numbers: [create_tfn_fcm_from_conventional_fcm\(\)](#), [print_tfn\(\)](#), [subtract_tfn\(\)](#), [tfn\(\)](#)

Examples

```
make_adj_matrix_w_tfns(
  lower = matrix(data = c(0, 0.2, 0, 0.5), nrow = 2, ncol = 2),
  mode = matrix(data = c(0, 0.3, 0, 0.6), nrow = 2, ncol = 2),
  upper = matrix(data = c(0, 0.4, 0, 0.7), nrow = 2, ncol = 2)
)

lower_adj_matrix <- data.frame(
  "A" = c(0, 0),
  "B" = c(0.4, 0)
)
mode_adj_matrix <- data.frame(
  "A" = c(0, 0),
  "B" = c(0.4, 0)
)
upper_adj_matrix <- data.frame(
  "A" = c(0, 0),
  "B" = c(0.6, 0)
)
make_adj_matrix_w_tfns(lower_adj_matrix, mode_adj_matrix, upper_adj_matrix)
```

plot.fcmconfr

Plot fcmconfr

Description

This plots the output of `fcmconfr()` using `ggplot`. Set `shiny = TRUE` to load plot in a Shiny app and toggle on/off results from different analyses.

Usage

```
## S3 method for class 'fcmconfr'
plot(
  x,
  include = "all",
  shiny = FALSE,
  filter_limit = 0.001,
  xlim = NA,
  coord_flip = FALSE,
  text_font_size = NA,
  mc_avg_and_CIs_color = "blue",
  mc_inferences_color = "blue",
  mc_inferences_size = 1,
```

```

mc_inferences_alpha = 0.3,
mc_inferences_shape = 3,
ind_inferences_color = "black",
ind_inferences_size = 1,
ind_inferences_alpha = 1,
ind_inferences_shape = 16,
agg_inferences_color = "red",
agg_inferences_size = 1,
agg_inferences_alpha = 1,
agg_inferences_shape = 17,
mc_avg_and_CIs_linewidth = 0.1,
ind_ivfn_and_tfn_linewidth = 0.1,
agg_ivfn_and_tfn_linewidth = 0.6,
...
)

```

Arguments

x	A direct output of the <code>fcmconfr</code> function
include	[character()] The concepts to include in the plot. By default, include = 'all' which does not exclude any concepts. Set to a vector of concept names to identify the only concepts to include in the plot.
shiny	[logical(1)] If TRUE, launch plot in a Shiny app to toggle on/off results from different analyses.
filter_limit	[double(1)] Only nodes with inferences above the filter_limit across any analysis will be plotted. This removes nodes with mostly 0-valued inferences indicating they were not impacted in the simulation.
xlim	[double(1)] The x-axis plot limits. xlim = NA lets ggplot determine the x-axis limits. xlim = c(lower_limit, upper_limit) for manual input limits. See ?ggplot2::xlim for additional info.
coord_flip	[logical(1)] Swap x- and y-axes (i.e. rotate plot). See ?ggplot2::coord_flip for additional info.
text_font_size	[double(1)] The font size of axis labels. text_font_size = NA lets ggplot determine the axis label font size.
mc_avg_and_CIs_color	[character(1)] Color of the crossbar (lines) indicating the avg inferences of empirical FCMs generated via Monte Carlo sampling and the confidence intervals about those averages.

`mc_inferences_color`
 [character(1)]
 Color of the points representing inferences of empirical FCMs generated via Monte Carlo sampling.

`mc_inferences_size`
 [double(1) - Positive]
 Size of the points for inferences of Monte Carlo FCMs. Must be greater than 0.

`mc_inferences_alpha`
 [double(1) - Positive (between 0 and 1)]
 Transparency of the points representing inferences of empirical FCMs generated via Monte Carlo sampling. Range from 0 to 1 (0: Transparent to 1: Opaque).

`mc_inferences_shape`
 [integer(1) or character(1)]
 Point shapes of the points representing inferences of empirical FCMs generated via Monte Carlo sampling. Accepts PCH point values and character strings.

`ind_inferences_color`
 [character(1)]
 Color of the points representing inferences of individual FCMs.

`ind_inferences_size`
 [double(1) - Positive]
 Size of the points for inferences of individual FCMs. Must be greater than 0.

`ind_inferences_alpha`
 [double(1) - Positive (between 0 and 1)]
 Transparency of the points representing inferences of individual FCMs. Range from 0 to 1 (0: Transparent to 1: Opaque).

`ind_inferences_shape`
 [integer(1) or character(1)]
 Point shapes of the points representing inferences of individual FCMs. Accepts PCH point values and character strings. Ignored for IVFN FCMs.

`agg_inferences_color`
 [character(1)]
 Color of the points representing inferences of the aggregate FCM

`agg_inferences_size`
 [double(1) - Positive]
 Size of the points for inferences of aggregate FCM. Must be greater than 0.

`agg_inferences_alpha`
 [double(1) - Positive (between 0 and 1)]
 Transparency of the points representing inferences of the aggregate FCM. Range from 0 to 1 (0: Transparent to 1: Opaque).

`agg_inferences_shape`
 [integer(1) or character(1)]
 Point shapes of the points representing inferences of the aggregate FCM. Accepts PCH point values and character strings. Ignored for IVFN FCMs.

`mc_avg_and_CIs_linewidth`
 [double(1) - Positive]
 Linewidth of lines representing the average (and confidence bounds the average) of the Monte Carlo FCMs inferences

```

ind_ivfn_and_tfn_linewidth
    [double(1) - Positive]
    Linewidth of lines representing inferences for analyses of individual IVFN- and
    TFN- FCMs.
agg_ivfn_and_tfn_linewidth
    [double(1) - Positive]
    Linewidth of lines representing inferences for analyses of aggregate IVFN- and
    TFN- FCMs
...
    Additional inputs

```

Details

Generates a generic plot visualizing `fcmconfr` results.

Value

A plot of an `fcmconfr` object's results

Examples

```

# Example using TFN FCMs fcmconfr
tfn_example_fcmconfr <- fcmconfr(
  adj_matrices = sample_fcms$simple_fcms$tfn_fcms,
  # adj_matrices = group_tfn_fcms,
  # Aggregation and Monte Carlo Sampling
  agg_function = 'mean',
  num_mc_fcms = 100,
  # Simulation
  initial_state_vector = c(1, 1, 1, 1, 1, 1, 1),
  clamping_vector = c(1, 0, 0, 0, 0, 0, 0),
  activation = 'rescale',
  squashing = 'sigmoid',
  lambda = 1,
  point_of_inference = "final",
  max_iter = 1000,
  min_error = 1e-05,
  # Inference Estimation (bootstrap)
  ci_centering_function = "mean",
  confidence_interval = 0.95,
  # Runtime Options
  show_progress = TRUE,
  parallel = FALSE,
  # Additional Options
  run_agg_calcs = TRUE,
  run_mc_calcs = TRUE,
  run_ci_calcs = TRUE,
  include_zeroes_in_sampling = TRUE,
  include_sims_in_output = TRUE
)

# Plot Defaults

```

```

plot(tfn_example_fcmconfr,
     interactive = FALSE, # Set to TRUE to open shiny app
     # Plot Formatting Parameters
     filter_limit = 1e-4,
     xlim = c(-1, 1),
     coord_flip = FALSE,
     text_font_size = NA,
     # Plot Aesthetic Parameters
     mc_avg_and_CIs_color = "blue",
     mc_inferences_color = "blue",
     mc_inferences_alpha = 0.3,
     mc_inferences_shape = 3,
     ind_inferences_color = "black",
     ind_inferences_alpha = 1,
     ind_inferences_shape = 16,
     agg_inferences_color = "red",
     agg_inferences_alpha = 1,
     agg_inferences_shape = 17,
     ind_ivfn_and_tfn_linewidth = 0.1,
     agg_ivfn_and_tfn_linewidth = 0.6
)

# Different from Plot Defaults
plot(tfn_example_fcmconfr,
     interactive = FALSE, # Set to TRUE to open shiny app
     # Plot Formatting Parameters
     filter_limit = 1e-4,
     xlim = c(-0.6, 0.6),
     coord_flip = FALSE,
     # Plot Aesthetic Parameters
     mc_avg_and_CIs_color = "darkblue",
     mc_inferences_color = "darkblue",
     mc_inferences_shape = 3,
     ind_inferences_color = "maroon",
     ind_inferences_shape = 16,
     agg_inferences_color = "grey",
     agg_inferences_shape = 17,
     ind_ivfn_and_tfn_linewidth = 0.1,
     agg_ivfn_and_tfn_linewidth = 0.6
)

# Plot Defaults w/ Shiny App
# plot(tfn_example_fcmconfr,
#      shiny = TRUE, # Set to TRUE to open shiny app
#      # Plot Formatting Parameters
#      filter_limit = 1e-4,
#      coord_flip = FALSE,
#      text_font_size = 12,
#      # Plot Aesthetic Parameters
#      mc_avg_and_CIs_color = "blue",
#      mc_inferences_color = "blue",
#      mc_inferences_shape = 3,

```

```
#      ind_inferences_color = "black",
#      ind_inferences_shape = 16,
#      agg_inferences_color = "red",
#      agg_inferences_shape = 17,
#      ind_ivfn_and_tfn_linewidth = 0.1,
#      agg_ivfn_and_tfn_linewidth = 0.6
# )
```

plot.rtriangular_dist *plot.rtriangular_dist*

Description

Plot rtriangular_dist distribution similar to how runif is plotted with the base plot function

Usage

```
## S3 method for class 'rtriangular_dist'
plot(x, ...)
```

Arguments

x	[rtriangular_dis] An rtriangular_dist object
...	additional inputs (leave empty)

Value

A plot of the triangular distribution generated by rtriangular_dist (in the vain of plot(runif(...)))

Examples

```
plot(rtriangular_dist(n = 1000L, lower = -1.0, mode = 0.0, upper = 1.0))
```

print.aggregate *Print method for aggregate*

Description

Print method for aggregate

Usage

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

Arguments

x	[aggregate] A direct output from aggregate_fcms
...	additional inputs

Value

[NULL]
A console printout of aggregate_fcm output

See Also

Other aggregate_fcms: [aggregate_fcms\(\)](#)

Examples

NULL

print.fcmconfr	<i>print.fcmconfr</i>
----------------	-----------------------

Description

This improves the readability of the fcmconfr output

Usage

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

Arguments

x	[fcmconfr] A direct output from the fcmconfr function
...	additional inputs

Details

Show the objects listed in the fcmconfr output \$inference and \$params, as well as \$bootstrap if present in output. Additionally, this prints descriptions/summaries of objects within each sub-list like inference_opts, bootstrap_input_opts, etc.

Value

A console printout of fcmconfr results

Examples

NULL

print.ivfn	<i>Print an Interval-Valued Fuzzy Number (IVFN) - S3 Class</i>
------------	--

Description

This prints an ivfn object

Usage

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

Arguments

x	[ivfn] a ivfn object
...	additional inputs

Value

[NULL]
A console printout of an IVFN object

See Also

Other interval-valued-fuzzy-numbers: [c.ivfn\(\)](#), [create_ivfn_fcm_from_conventional_fcm\(\)](#), [ivfn\(\)](#), [make_adj_matrix_w_ivfns\(\)](#), [subtract_ivfn\(\)](#)

Examples

```
print(ivfn(-1, 1))
```

print.summary.fcmconfr	<i>print.summary.fcmconfr</i>
------------------------	-------------------------------

Description

This prints a summary object of the fcmconfr output

Usage

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

Arguments

x	[summary.fcmconfr] A direct output from the summary.fcmconfr function
...	additional inputs

Details

Show the objects listed in the fcmconfr output \$inference and \$params, as well as \$bootstrap if present in output. Additionally, this prints descriptions/summaries of objects within each sub-list like inference_opts, bootstrap_input_opts, etc.

Value

A console printout (summary) of fcmconfr results

Examples

NULL

print.tfn	<i>Print a Triangular Fuzzy Number (TFN)</i>
-----------	--

Description

This improves the readability of the output

This prints a tfn object

Usage

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

Arguments

x	a tfn object
...	additional inputs

Value

A console printout of a TFN object

See Also

Other triangular-fuzzy-numbers: [create_tfn_fcm_from_conventional_fcm\(\)](#), [make_adj_matrix_w_tfns\(\)](#), [subtract_tfn\(\)](#), [tfn\(\)](#)

Examples

```
tfn(-1, 0, 1)
```

rtriangular_dist	<i>The Triangular Distribution</i>
------------------	------------------------------------

Description

This function generates a triangular distribution on the interval from lower to upper bounds with a given mode,

Usage

```
rtriangular_dist(
  n = integer(),
  lower = double(),
  mode = double(),
  upper = double()
)
```

Arguments

n	[integer(1) - Positive] The number of samples to draw from the triangular distribution
lower	[double(1) - Unrestricted (positive or negative)] The lower limit or minimum of the sample space
mode	[double(1) - Unrestricted (positive or negative)] The peak of the sample space
upper	[double(1) - Unrestricted (positive or negative)] The upper limit or maximum of the sample space

Value

[rtriangular_dist]
a vector of values representing a triangular distribution

Examples

```
rtriangular_dist(n = 100L, lower = -1.0, mode = 0.0, upper = 1.0)

x <- rtriangular_dist(n = 1000L, lower = -0.5, mode = 0.0, upper = 0.5)
# hist(x)

x <- rtriangular_dist(100L, 0.0, 0.0, 0.5)
# hist(x)
```

sample_fcms

sample_fcms

Description

An collection of FCM datasets based on stakeholder perceptions of rising salinization of a local reservoir in a socio-ecological system. This collection is separated into two sub-collections: one with larger FCMs that incorporate over 40 nodes, and one with smaller, simpler FCMs that greatly reduce the number of nodes ($n = 7$) per map. The sub-collection of smaller, simplified FCMs is included to hasten testing.

Usage

sample_fcms

Format

sample_fcms:

A collection of simple and large FCMs with conventional, ivfn, and tfn edge types

Details

\$simple_fcms: these FCMs are filtered versions of the larger FCMs which only include the following nodes, Guidance.Docs.for.Salt.Management, Public.Education.Programs, Salts.Added.during.Water.Treatment, Salts.Added.by.Winter.Maintenance.Activities, Ecosystem.Health, Public.Awareness.of.Reservoir.Salinization, and Salinization.of.the.Reservoir. (Renamd, but based on nodes in larger FCMs)

- \$conventional_fcms: A list of smaller, simplified conventional FCMs
- \$ivfn_fcms: A list of smaller, simplified ivfn FCMs
- \$tfn_fcms: A list of smaller, simplified tfn FCMs \$large_fcms: these FCMs are large ($n > 40$) and will likely increase runtimes but represent a more complete dataset
- \$conventional_fcms: A list of large conventional FCMs
- \$ivfn_fcms: A list of large ivfn FCMs
- \$tfn_fcms: A list of large tfn FCMs

Given in Rippey et al, (In Review); Not all stakeholder FCMs used here.

standardize_adj_matrices

Standardize a List of Adjacency Matrices

Description

Given a list of adj. matrices of different sizes, transforms the inputs into adj. matrices of the same size with 0-values added where edges are undefined or not included in maps.

Solves the problem where Map 1 has nodes A and B but Map 2 has nodes B and C and the user needs to analyse them both together.

Usage

```
standardize_adj_matrices(adj_matrices = list())
```

Arguments

adj_matrices [list()]
A single adjacency matrix or a list of adjacency matrices (n x n) representing FCMs. Matrices can have conventional edge weights, IVFN edge weights or TFN edge weights.

Value

[list()]
A list of adjacency matrices constructed from the input adj. matrices, that contain the same concepts (and dimensions). Rows/Columns of added concepts are all 0's.

Examples

```
# Standardize Conventional FCMs
adj_matrix_1 <- data.frame(
  "A" = c(0, 0),
  "B" = c(1, 0)
)
adj_matrix_2 <- data.frame(
  "A" = c(0, 0, 0),
  "B" = c(0.25, 0, 1),
  "C" = c(0, 0.7, 0)
)
adj_matrix_3 <- data.frame(
  "B" = c(0, 0),
  "D" = c(0.75, 0)
)
adj_matrix_4 <- data.frame(
  "A" = c(0, 0, 0.3, 0),
  "B" = c(0.5, 0, 0, 0.6),
  "E" = c(0, 0, 0, 0),
  "F" = c(1, 0, 1, 0)
```

```

)
adj_matrices_w_different_concepts <- list(
  adj_matrix_1, adj_matrix_2, adj_matrix_3, adj_matrix_4
)
standardize_adj_matrices(adj_matrices_w_different_concepts)

# Standardize IVFN FCMs
adj_matrix_1_lower <- data.frame(
  "A" = c(0, 0),
  "B" = c(0.8, 0)
)
adj_matrix_1_upper <- data.frame(
  "A" = c(0, 0),
  "B" = c(1, 0)
)
ivfn_adj_matrix_1 <- make_adj_matrix_w_ivfns(
  adj_matrix_1_lower, adj_matrix_1_upper
)

adj_matrix_2_lower <- data.frame(
  "A" = c(0, 0, 0),
  "B" = c(0.1, 0, 0.8),
  "C" = c(0, 0.6, 0)
)
adj_matrix_2_upper <- data.frame(
  "A" = c(0, 0, 0),
  "B" = c(0.3, 0, 1),
  "C" = c(0, 0.8, 0)
)
ivfn_adj_matrix_2 <- make_adj_matrix_w_ivfns(
  adj_matrix_2_lower, adj_matrix_2_upper
)

adj_matrices_w_different_concepts <- list(
  ivfn_adj_matrix_1, ivfn_adj_matrix_2
)
standardize_adj_matrices(adj_matrices_w_different_concepts)

# Standardize TFN FCMs
adj_matrix_1_lower <- data.frame(
  "A" = c(0, 0),
  "B" = c(0.8, 0)
)
adj_matrix_1_mode <- data.frame(
  "A" = c(0, 0),
  "B" = c(0.9, 0)
)
adj_matrix_1_upper <- data.frame(
  "A" = c(0, 0),
  "B" = c(1, 0)
)
tfn_adj_matrix_1 <- make_adj_matrix_w_tfns(

```

```

    adj_matrix_1_lower, adj_matrix_1_mode, adj_matrix_1_upper
  )

  adj_matrix_2_lower <- data.frame(
    "A" = c(0, 0, 0),
    "B" = c(0.1, 0, 0.8),
    "C" = c(0, 0.6, 0)
  )
  adj_matrix_2_mode <- data.frame(
    "A" = c(0, 0, 0),
    "B" = c(0.2, 0, 0.9),
    "C" = c(0, 0.7, 0)
  )
  adj_matrix_2_upper <- data.frame(
    "A" = c(0, 0, 0),
    "B" = c(0.3, 0, 1),
    "C" = c(0, 0.8, 0)
  )
  tfn_adj_matrix_2 <- make_adj_matrix_w_tfns(
    adj_matrix_2_lower, adj_matrix_2_mode, adj_matrix_2_upper
  )

  adj_matrices_w_different_concepts <- list(
    ivfn_adj_matrix_1, ivfn_adj_matrix_2
  )
  standardize_adj_matrices(adj_matrices_w_different_concepts)

```

subtract_ivfn

IVFN Subtraction (Interval Calculus)

Description

This subtracts one Interval-Value Fuzzy Number (IVFN) from another via interval calculus. An IVFN represented by the interval X:

$$X = (x_1, x_2) = [x_1; x_2] = \{x \in \mathbb{R} \mid x_1 \leq x \leq x_2\}$$

where x_1 and x_2 represent the lower and upper bounds, respectively.

The IVFN X may have another IVFN Y subtracted from it via:

$$X - Y = [x_1 - y_2; x_2 - y_1]$$

Usage

```
subtract_ivfn(ivfn_1 = ivfn(), ivfn_2 = ivfn())
```

Arguments

ivfn_1	[ivfn] An interval-value fuzzy number (ivfn) object
ivfn_2	[ivfn] An interval-value fuzzy number (ivfn) object

Details

It is not required for one IVFN to be "greater than" the other.

This difference may also be estimated by translating the IVFN's into their corresponding distributions (e.g. `ivfn(-1, 1) = runif(n, -1, 1)`), subtracting one distribution from the other, and estimating the minimum and maximum values of the difference distribution.

Value

[ivfn]
An IVFN object representing the subtraction of `ivfn_2` from `ivfn_1`

References

Moore R, Lodwick W (2003). "Interval Analysis and Fuzzy Set Theory." *Fuzzy Sets and Systems*, 135(1), 5–9. ISSN 01650114, doi:[10.1016/S0165-0114\(02\)00246-4](https://doi.org/10.1016/S0165-0114(02)00246-4).

Dimuro GP (2011). "On Interval Fuzzy Numbers." In 2011 Workshop-School on Theoretical Computer Science, 3–8. ISBN 978-0-7695-4628-5 978-1-4673-0225-8, doi:[10.1109/WEIT.2011.19](https://doi.org/10.1109/WEIT.2011.19).

Moore RE (1966). *Interval Analysis*, volume 4. prentice-Hall Englewood Cliffs.

See Also

Other interval-valued-fuzzy-numbers: `c.ivfn()`, `create_ivfn_fcm_from_conventional_fcm()`, `ivfn()`, `make_adj_matrix_w_ivfns()`, `print.ivfn()`

Examples

```
subtract_ivfn(ivfn(0.5, 0.8), ivfn(0.2, 0.5))
subtract_ivfn(ivfn(-0.5, 0.3), ivfn(0.4, 0.6))
subtract_ivfn(ivfn(-1, 1), ivfn(-0.5, 0.5))
```

Description

This subtracts one Triangular Fuzzy Number (TFN) from another via interval calculus. A TFN represented by the fuzzy set (triangular distribution):

$$X = (x_1, x_2, x_3) = \begin{cases} 0 & \text{for } x < x_1 \\ \frac{2(x - x_1)}{(x_3 - x_1)(x_2 - x_1)} & \text{for } x_1 \leq x < x_3 \\ \frac{x_3 - x_1}{2} & \text{for } x = x_2 \\ \frac{2(x_3 - x)}{(x_3 - x_1)(x_3 - x_2)} & \text{for } x_2 < x \leq x_3 \\ 0 & \text{for } x > x_3 \end{cases}$$

where x_1 and x_3 are the lower and upper bounds, respectively, and x_2 is the mode.

The TFN X may have another TFN Y subtracted from it via:

$$X - Y = (x_1 - y_3, x_2 - y_2, x_3 - y_1)$$

Usage

```
subtract_tfn(tfn_1 = tfn(), tfn_2 = tfn())
```

Arguments

tfn_1	[tfn] A triangular fuzzy number (tfn) object
tfn_2	[tfn] A triangular fuzzy number (tfn) object

Details

It is not required for one IVFN to be "greater than" the other.

This difference may also be estimated by translating the TFN's into their corresponding distributions (e.g. `tfn(-1, 0, 1) = EnvStats::rtri(n, min = -1, max = 1, mode = 0)`), subtracting one distribution from the other, and estimating the minimum, mode, and maximum values of the difference distribution.

Value

[tfn]
An TFN object representing the subtraction of `tfn_2` from `tfn_1`

References

- Chakraverty S, Sahoo DM, Mahato NR (2019). "Fuzzy Numbers." In Concepts of Soft Computing, 53–69. Springer Singapore, Singapore. ISBN 9789811374296, doi:10.1007/978-981-13-7430-2_3.
- Hanss M (ed.) (2005). Applied Fuzzy Arithmetic: An Introduction with Engineering Applications, SpringerLink BV"ucher. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-24201-7, doi:10.1007/b138914.
- Trillas E, Eciolaza L (2015). "Fuzzy Logic." In volume 320, chapter Fuzzy Arithmetic, 141–158. Springer International Publishing, Cham. ISBN 978-3-319-14202-9, doi:10.1007/978-3-319-14203-6_6

See Also

Other triangular-fuzzy-numbers: [create_tfn_fcm_from_conventional_fcm\(\)](#), [make_adj_matrix_w_tfns\(\)](#), [print.tfn\(\)](#), [tfn\(\)](#)

Examples

```
subtract_tfn(tfn(lower = 0.5, mode = 0.6, upper = 0.8), tfn(lower = 0.2, mode = 0.3, upper = 0.5))
subtract_tfn(tfn(-0.5, -0.2, 0.3), tfn(0.4, 0.5, 0.6))
subtract_tfn(tfn(-1, 0, 1), tfn(-0.5, 0, 0.5))
```

summary.fcmconfr	<i>summary.fcmconfr</i>
------------------	-------------------------

Description

This creates a summary object of the fcmconfr output

Usage

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

Arguments

object	[fcmconfr] A direct output from the fcmconfr function
...	additional inputs

Details

Show the objects listed in the fcmconfr output \$inference and \$params, as well as \$bootstrap if present in output. Additionally, this prints descriptions/summaries of objects within each sub-list like inference_opts, bootstrap_input_opts, etc.

Value

A summary object of fcmconfr results

Examples

```
NULL
```

tfn	<i>Triangular Fuzzy Number (TFN) - S3 Class</i>
-----	---

Description

This constructs an triangular fuzzy number (tfn) that represents a continuous, triangular distribution of values within a given range

Usage

```
tfn(lower = double(), mode = double(), upper = double())
```

Arguments

lower	[double(1) - Unrestricted (positive or negative)] The lower limit of a Triangular Number set (the lower value must be less than or equal to the upper value)
mode	[double(1) - Unrestricted (positive or negative)] The most likely value of a Triangular Number set
upper	[double(1) - Unrestricted (positive or negative)] The upper limit of a Triangular Number set (the upper value must be greater or equal to the lower value)

Details

The TFN class does not perform any operations on its input, rather it checks whether the input follows the defining criteria of TFNs

For TFNs, the lower bound must be less than or equal to the mode which must be less than or equal to the upper bound. If the lower bound, mode, and upper bound are equal, the TFN represents a "crisp" numeric value.

$$\mathbf{TFN} : [x^L, x^M, x^U]$$

where x^L , x^M , and x^U are the lower bound, mode, and upper bound of the TFN.

Value

[tfn]
A triangular fuzzy number (TFN)

References

Chakraverty S, Sahoo DM, Mahato NR (2019). "Fuzzy Numbers." In Concepts of Soft Computing, 53–69. Springer Singapore, Singapore. ISBN 9789811374296, doi:10.1007/978-981-13-7430-2_3.

Hanss M (ed.) (2005). Applied Fuzzy Arithmetic: An Introduction with Engineering Applications, SpringerLink B\''ucher. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-24201-7, doi:10.1007/b138914.

Trillas E, Eciolaza L (2015). “Fuzzy Logic.” In volume 320, chapter Fuzzy Arithmetic, 141–158. Springer International Publishing, Cham. ISBN 978-3-319-14202-9, doi:[10.1007/978-3-319-14203-6_6](https://doi.org/10.1007/978-3-319-14203-6_6)

See Also

Other triangular-fuzzy-numbers: [create_tfn_fcm_from_conventional_fcm\(\)](#), [make_adj_matrix_w_tfns\(\)](#), [print.tfn\(\)](#), [subtract_tfn\(\)](#)

Examples

```
tfn(lower = -1.0, mode = 0.0, upper = 1.0)
tfn(0, 0.5, 0.8)
tfn(-0.3, 0.1, 0.4)
```

Index

- * **aggregate_fcms**
 - aggregate_fcms, [3](#)
 - print.aggregate, [42](#)
- * **datasets**
 - sample_fcms, [47](#)
- * **infer_and_simulate_fcm**
 - infer_fcm_set, [30](#)
- * **interval-valued-fuzzy-numbers**
 - c.ivfn, [8](#)
 - create_ivfn_fcm_from_conventional_fcm, [9](#)
 - ivfn, [33](#)
 - make_adj_matrix_w_ivfns, [35](#)
 - print.ivfn, [44](#)
 - subtract_ivfn, [50](#)
- * **monte-carlo-model-generation-and-simulation**
 - build_monte_carlo_fcms, [6](#)
- * **triangular-fuzzy-numbers@family triangular-fuzzy-numbers**
 - c.tfn, [9](#)
- * **triangular-fuzzy-numbers**
 - create_tfn_fcm_from_conventional_fcm, [10](#)
 - make_adj_matrix_w_tfns, [36](#)
 - print.tfn, [45](#)
 - subtract_tfn, [51](#)
 - tfn, [54](#)
- * **utility**
 - fcm_view, [14](#)
 - get_fcmconfr_inferences, [25](#)
- * **utils-general**
 - standardize_adj_matrices, [48](#)

aggregate_fcms, [3](#), [43](#)

build_monte_carlo_fcms, [6](#)

c.ivfn, [8](#), [10](#), [34](#), [35](#), [44](#), [51](#)

c.tfn, [9](#)

create_ivfn_fcm_from_conventional_fcm, [8](#), [9](#), [34](#), [35](#), [44](#), [51](#)

create_tfn_fcm_from_conventional_fcm, [10](#), [37](#), [45](#), [53](#), [55](#)

defuzz_ivfn_or_tfn, [11](#)

estimate_fcm_lambda, [12](#)

fcm_view, [14](#), [25](#)

fcmconfr, [16](#), [23–25](#), [38](#), [40](#), [43](#), [53](#)

fcmconfr_gui, [23](#)

get_fcmconfr_inferences, [15](#), [25](#)

infer_fcm, [28](#)

infer_fcm_set, [30](#)

ivfn, [8](#), [10](#), [33](#), [35](#), [44](#), [51](#)

make_adj_matrix_w_ivfns, [8](#), [10](#), [34](#), [35](#), [44](#), [51](#)

make_adj_matrix_w_tfns, [11](#), [36](#), [45](#), [53](#), [55](#)

plot.fcmconfr, [37](#)

plot.rtriangular_dist, [42](#)

print.aggregate, [6](#), [42](#)

print.fcmconfr, [43](#)

print.ivfn, [8](#), [10](#), [34](#), [35](#), [44](#), [51](#)

print.summary.fcmconfr, [44](#)

print.tfn, [11](#), [37](#), [45](#), [53](#), [55](#)

rtriangular_dist, [46](#)

sample_fcms, [47](#)

standardize_adj_matrices, [3](#), [48](#)

subtract_ivfn, [8](#), [10](#), [34](#), [35](#), [44](#), [50](#)

subtract_tfn, [11](#), [37](#), [45](#), [51](#), [55](#)

summary.fcmconfr, [45](#), [53](#)

tfn, [11](#), [37](#), [45](#), [53](#), [54](#)