Package: causalnet (via r-universe)

May 14, 2026

Title Directed Causal Network Enumeration and Simulation

Version 0.1.0

Description Enumerate orientation-consistent directed networks from an undirected or partially directed skeleton, detect feedback loops, summarize topology, and simulate node dynamics via stochastic differential equations.

License GPL-3

URL https://github.com/KyuriP/causalnet

BugReports https://github.com/KyuriP/causalnet/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Imports stats, ggplot2, tidyr, scales, cowplot

Suggests testthat (>= 3.0.0), dplyr, ggbeeswarm, knitr, rmarkdown

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev

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

Date/Publication 2026-01-14 16:54:25 UTC

RemoteUrl https://github.com/kyurip/causalnet

RemoteRef HEAD

RemoteSha 61b7307e5be639a136cbb30c25affb8718a62e85

Contents

detect_feedback_loops 2
generate_directed_networks 2
get_sample_parameters 4
plot_dynamics 5

plot_network_metrics 6
simulate_dynamics 7
summarize_network_metrics 9

Index **11**

`detect_feedback_loops` *Detect Unique Feedback Loops in a Directed Network*

Description

Detects all directed cycles (including 2-node loops) and returns each as a unique set of nodes (ignores order and entry point).

Usage

```
detect_feedback_loops(adj_matrix, include_self_loops = FALSE, use_names = TRUE)
```

Arguments

`adj_matrix` Square directed adjacency (non-zero = edge).

`include_self_loops`
Logical; include 1-node self-loops (default FALSE).

`use_names` Logical; return node names if available (default TRUE).

Value

List of unique loops, each as a sorted vector (names if available, else indices).

`generate_directed_networks`
Generate Directed Networks Consistent with Constraints

Description

Enumerate all directed adjacency matrices that are consistent with a given undirected skeleton and optional direction constraints. Enumeration can optionally include bidirected edges and display a simple progress bar.

Usage

```
generate_directed_networks(
  adj_matrix,
  allow_bidirectional = TRUE,
  fixed_edges = NULL,
  max_networks = Inf,
  show_progress = interactive()
)
```

Arguments

- `adj_matrix` Symmetric binary (0/1) adjacency matrix giving the undirected skeleton. Only pairs with `adj_matrix[i, j] = 1` are considered for orientation; all other pairs remain 0.
- `allow_bidirectional` Logical. If `TRUE`, bidirected edges (`i <-> j`) are allowed during enumeration. Default: `TRUE`.
- `fixed_edges` Numeric matrix the same size as `adj_matrix` that encodes per-edge constraints (interpreted on the directed `i -> j` entry):
  - `1`: force `i -> j`
  - `2`: force `i <-> j` (both `i -> j` and `j -> i`)
  - `0` or `NA`: unconstrained

  To exclude an edge entirely, set `adj_matrix[i,j] = adj_matrix[j,i] = 0` in the skeleton; `fixed_edges` does not remove edges, it only constrains orientations of skeleton edges.
- `max_networks` Integer. Maximum number of networks to return. Use to cap output size when constraints are loose and the search space is large. Default: `Inf`.
- `show_progress` Logical. Show a text progress bar during enumeration. Default: `interactive()`.

Details

If the skeleton has m undirected edges, the number of orientation-consistent digraphs is at most 2^m when `allow_bidirectional = FALSE` and 3^m when `TRUE` (before applying constraints). Consider setting `max_networks` for exploratory use.

Value

A list of unique directed 0/1 adjacency matrices, each with the same dimensions and dimnames as `adj_matrix`.

See Also

`detect_feedback_loops`, `summarize_network_metrics`

Examples

```
skel <- matrix(0, 3, 3); skel[upper.tri(skel)] <- 1; skel <- skel + t(skel)
colnames(skel) <- rownames(skel) <- paste0("X", 1:3)
out <- generate_directed_networks(skel, allow_bidirectional = TRUE)
length(out)

# Force X1 -> X2 and X2 <-> X3:
F <- matrix(NA_real_, 3, 3, dimnames = dimnames(skel))
F["X1", "X2"] <- 1
F["X2", "X3"] <- 2
out2 <- generate_directed_networks(skel, fixed_edges = F)
length(out2)
```

get_sample_parameters *Generate Sample Parameters for Node Dynamics*

Description

Returns a list of simulation parameters (domain-agnostic: nodes can be any variables). If a parameter vector is not supplied, values are sampled i.i.d. from a uniform range.

Usage

```
get_sample_parameters(
  n_nodes,
  beta_range = c(-1.5, -1),
  alpha_range = c(0.05, 0.3),
  delta_range = c(1, 5),
  sigma_range = c(0.01, 0.1),
  beta = NULL,
  alpha_self = NULL,
  delta = NULL,
  sigma = NULL,
  nodes = NULL
)
```

Arguments

`n_nodes`	Integer number of nodes.
`beta_range`	Length-2 numeric range for baseline/exogenous drive (used if `beta` is NULL).
`alpha_range`	Length-2 numeric range for self-activation (used if `alpha_self` is NULL).
`delta_range`	Length-2 numeric range for nonlinear amplification (used if `delta` is NULL).
`sigma_range`	Length-2 numeric range for noise SD (used if `sigma` is NULL).
`beta`, `alpha_self`, `delta`, `sigma`	Optional fixed numeric vectors. If length-1, recycled to `n_nodes`.
`nodes`	Optional character vector of node names (length `n_nodes`) used to name outputs.

Value

A named list with elements `beta`, `alpha_self`, `delta`, `sigma` (each length `n_nodes`).

plot_dynamics *Plot Dynamics with Optional Stress Shading*

Description

Visualizes node/variable dynamics over time using ggplot2, with optional stress intervals and customizable styling.

Usage

```
plot_dynamics(
  S,
  stress_windows = NULL,
  title = "Dynamics",
  colors = NULL,
  legend_labels = NULL,
  show_lines = FALSE,
  line_width = 0.8,
  line_alpha = 1,
  base_size = 14,
  label_stress = TRUE,
  stress_label = "Stress Period",
  stress_fill = "gray60",
  stress_alpha = 0.2,
  stress_line_color = "gray40",
  y_label = "Level",
  legend_position = "right",
  y_limits = NULL
)
```

Arguments

`S`	Matrix (time x variables) of simulated states. If `attr(S,"time")` exists, it is used for the x-axis (continuous time). Otherwise the x-axis is step index `1:nrow(S)`.
`stress_windows`	Optional list of numeric `c(start, end)` intervals, or a 2-column matrix/data.frame with `start`,end. Units must match the x-axis (i.e., the "Time" used for plotting).
`title`	Plot title.
`colors`	Optional vector of line colors (length = #variables).
`legend_labels`	Optional vector of legend labels (length = #variables).
`show_lines`	If TRUE, draw dashed vertical lines instead of shaded rectangles.
`line_width`	Line width for trajectories.
`line_alpha`	Line transparency (0–1).
`base_size`	Base font size for theme.

`label_stress`	If TRUE and using shading, label each stress window.
`stress_label`	Text label (length 1 or length = #windows).
`stress_fill`	Fill color for shaded windows.
`stress_alpha`	Alpha for shaded windows.
`stress_line_color`	Color for dashed lines (if `show_lines = TRUE`).
`y_label`	Y-axis label.
`legend_position`	Legend position (e.g., `"right"`, `"bottom"`, `"none"`).
`y_limits`	Optional numeric length-2 vector for y-axis limits.

Value

A ggplot object.

`plot_network_metrics` *Generate ggplot objects summarizing network metrics*

Description

Produces a list of ggplot2 objects visualizing summary metrics across a list of directed networks.

Usage

```
plot_network_metrics(
  summary_df,
  n_bins = 6,
  fill_colors = c("skyblue", "darkgreen", "orange", "lightcoral"),
  base_size = 14,
  return_grid = TRUE,
  p2_style = c("auto", "jitter", "point", "beeswarm"),
  auto_threshold = 200
)
```

Arguments

`summary_df`	Data frame from `summarize_network_metrics()`.
`n_bins`	Number of histogram bins (default = 6).
`fill_colors`	Optional vector of 4 fill colors.
`base_size`	Base font size for plots (default = 14).
`return_grid`	If TRUE, returns cowplot grid; otherwise, returns list of plots.
`p2_style`	How to plot panel (b) (sigma_total by loop count). One of "auto" (default), "jitter", "point", or "beeswarm". If "auto", uses beeswarm when `nrow(summary_df)` < `auto_threshold`, otherwise jitter.
`auto_threshold`	Threshold for switching when `p2_style = "auto"` (default = 200).

Details

The "beeswarm" option uses `ggbeeswarm::geom_beeswarm()` if the ggbeeswarm package is installed. If not installed, the function falls back to jitter and emits a warning.

Value

A cowplot grid or a named list of ggplot2 objects.

Examples

```
## Not run:
summary_df <- summarize_network_metrics(nets)

# automatic switching
plot_network_metrics(summary_df, p2_style = "auto", auto_threshold = 200)

# force beeswarm
plot_network_metrics(summary_df, p2_style = "beeswarm")

# force jitter
plot_network_metrics(summary_df, p2_style = "jitter")

## End(Not run)
```

`simulate_dynamics` *Simulate network state dynamics via SDEs (nonlinear, linear, or custom)*

Description

Simulates the evolution of node states in a directed network using an Euler–Maruyama discretization of stochastic differential equations (SDEs). Choose the built-in nonlinear model, a linear alternative, or provide a custom update function.

Usage

```
simulate_dynamics(
  adj_matrix,
  params,
  t_max = 100,
  dt = 0.1,
  S0 = NULL,
  model_type = "nonlinear",
  model_fn = NULL,
  stress_event = NULL,
  boundary = c("auto", "reflect", "clamp", "none"),
  clamp = NULL
)
```

Arguments

adj_matrix
: Numeric matrix (square; directed adjacency). Interpreted as i -> j.

params
: Named list of model parameters. For `model_type = "nonlinear"`, requires vectors (length = n nodes):
  - `beta`: baseline/exogenous drive per node.
  - `alpha_self`: self-activation per node.
  - `delta`: nonlinear amplification of incoming effects.
  - `sigma`: noise SD per node.

  For `model_type = "linear"`, requires `beta`, `alpha_self`, `sigma`. For a custom model, include whatever your `model_fn` expects.

t_max
: Total simulated time (must be > 0).

dt
: Time step (must be > 0). The output has `floor(t_max/dt) + 1` rows.

S0
: Optional numeric vector of initial states (length = n). Defaults to 0.01.

model_type
: One of `"nonlinear"` (default), `"linear"`, or `NULL` when using a custom `model_fn`.

model_fn
: Optional function with signature `function(current, interaction, dt, ...)` returning a numeric vector of increments `dS`. Additional args are taken from `params`.

stress_event
: Optional function `f(time, state) -> numeric(n)` that returns an exogenous input vector added each step (e.g., shocks/perturbations).

boundary
: One of `"auto"`, `"reflect"`, `"clamp"`, `"none"`.
  - `"reflect"`: mirror overshoot back into `[clamp[1], clamp[2]]`.
  - `"clamp"`: hard-box to `[clamp[1], clamp[2]]`.
  - `"none"`: no bounding.
  - `"auto"`: pick a sensible default based on the model and `clamp`: nonlinear -> `boundary = "reflect"` (and if `clamp` is `NULL`, use `c(0, 1)`); linear/custom -> `boundary = "none"` unless a clamp range is supplied, in which case use `"clamp"`.

clamp
: Either `NULL` (no numeric range) or a length-2 numeric vector `c(min, max)` used by `"reflect"` or `"clamp"` to keep states within bounds.

Details

Direction convention. By default `adj[i, j] = 1` encodes a directed edge `i -> j`. Under this convention, the *incoming input* to node `j` is the dot product of column `j` with the current state; in vector form `t(adj) %*% state`. If your internal convention differs, transpose accordingly.

Integration uses Euler–Maruyama. The per-step diffusion term is added as $\sigma\sqrt{dt}\,Z$ with $Z \sim \mathcal{N}(0, I)$ (component-wise), i.e., `sigma * sqrt(dt) * rnorm(n)`.

Value

Numeric matrix of states over time (rows = time steps, cols = nodes). The time vector is attached as `attr(result, "time")`.

Boundary handling

- **Reflecting** avoids "sticky" edges by bouncing trajectories back inside the range, which is useful for bounded variables on `[0,1]`.
- **Clamping** is numerically simple but can create artificial absorbing states at the limits.
- For smoothly bounded dynamics, consider modeling on an unbounded latent scale and applying a link (e.g., logistic) instead of hard post-step bounds.

Examples

```
set.seed(1)
net <- matrix(c(0,1,0,0,
                0,0,1,0,
                0,0,0,1,
                1,0,0,0), 4, byrow = TRUE)

# Linear model, automatic boundary selection ("none" because no clamp supplied)
p_lin <- list(beta = rep(0.8, 4), alpha_self = rep(0.2, 4), sigma = rep(0.05, 4))
S1 <- simulate_dynamics(net, p_lin, model_type = "linear", boundary = "auto", t_max = 5, dt = 0.01)

# Linear model with a finite box -> "auto" switches to clamp on [0, 5]
S2 <- simulate_dynamics(net, p_lin, model_type = "linear",
                        boundary = "auto", clamp = c(0, 5), t_max = 5, dt = 0.01)

# Nonlinear model -> "auto" uses reflecting boundaries on [0,1]
p_nl <- list(beta = rep(0.2, 4), alpha_self = rep(0.2, 4),
             delta = rep(0.5, 4), sigma = rep(0.05, 4))
S3 <- simulate_dynamics(
  net, p_nl, model_type = "nonlinear",
  boundary = "auto", t_max = 5, dt = 0.01
)
```

---

summarize_network_metrics

Summarize Directed Network List

Description

Compute feedback loop and topology metrics for a list of directed networks.

Usage

```
summarize_network_metrics(net_list)
```

Arguments

`net_list`	A list of directed adjacency matrices (can be from any source).

Value

A data frame with one row per network and the following columns:

- net_id
- n_nodes
- n_edges
- num_loops (number of unique feedback loops)
- sigma_total (sum of SDs of in/out degrees)
- node_overlap_score
- avg_loop_size

Index

detect_feedback_loops, 2, *3*

generate_directed_networks, 2
get_sample_parameters, 4

plot_dynamics, 5
plot_network_metrics, 6

simulate_dynamics, 7
summarize_network_metrics, *3*, 9