Package ‘Rato’

May 7, 2026

Type Package

Title Resilience Analysis Toolkit (RATO)

Version 0.1.0

Author Victor Chavauty [aut, cre]

Maintainer Victor Chavauty <lesserfish@pm.me>

Description Collection of tools for the analysis of the resilience of dynamic networks. Created as a classroom project.

Encoding UTF-8

RoxygenNote 7.3.2

Imports deSolve, igraph, utils

Suggests animation, RColorBrewer

NeedsCompilation no

Repository CRAN

Date/Publication 2025-10-08 08:40:13 UTC

Contents

Barabasi.B_eff — *Average the product of the outgoing and incoming degrees of nodes*

Description

This is the average of the outgoing/incoming nodes, described by Barabasi in his paper 'Universal resilience patterns in complex networks'.

Usage

```
Barabasi.B_eff(M)
```

Arguments

`M`	Adjacency matrix of the network that describes the system

Value

A numeric value describing the Beta_eff described by Barabasi.

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")
g <- Rato::graph.from.csv(node_file, edge_file, sep=",", header=TRUE)

initial_parameters <- function(M){
 initial_params <- list('f' = 1, 'h' = 2, 'B' = 0.1, 'Beta' = Rato::Barabasi.B_eff(M))
  return(initial_params)
}

update_parameters <- function(params){
  M <- params$M
  params$Beta <- Rato::Barabasi.B_eff(M)
  return(params)
}

Rato::node.removal.thread( system = Rato::Michaelis.Menten.Fast
                         , M = g$M
                         , x0 = Rato::Barabasi.X_eff(g$M, g$initial_values)
                         , initial_params = initial_parameters
                         , update_params = update_parameters
                         , to.numeric = TRUE)
```

Barabasi.X_eff — *Average Nearest-Neighbor Activity*

Description

This is the average nearest-neight activity described by Barabasi in his paper 'Universal resilience patterns in complex networks'. It can be used to characterize the effective state of the system.

Usage

```
Barabasi.X_eff(M, x)
```

Arguments

`M`	Adjacency matrix of the network that describes the system
`x`	The value of all the nodes of the system

Value

A numeric value describing the effective state by the system by using the nearest-neighbour activity.

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")
g <- Rato::graph.from.csv(node_file, edge_file, sep=",", header=TRUE)

initial_parameters <- function(M){
 initial_params <- list('f' = 1, 'h' = 2, 'B' = 0.1, 'Beta' = Rato::Barabasi.B_eff(M))
  return(initial_params)
}

update_parameters <- function(params){
  M <- params$M
  params$Beta <- Rato::Barabasi.B_eff(M)
  return(params)
}

Rato::node.removal.thread( system = Rato::Michaelis.Menten.Fast
                          , M = g$M
                          , x0 = Rato::Barabasi.X_eff(g$M, g$initial_values)
                          , initial_params = initial_parameters
                          , update_params = update_parameters
                          , to.numeric = TRUE)
```

graph.from.csv *Loads a graph from Node / Edge CSV files*

Description

This function takes the file paths to two CSV files: one containing the node information of a network and the other containing the edge information of the network. It returns a list containing several useful data related to the network.

Usage

```
graph.from.csv(nodeinfo_path, edgeinfo_path, ...)
```

Arguments

`nodeinfo_path`	Path to a CSV file containing the node information. This CSV file should follow the convention of ‘node_name, initial_value‘.
`edgeinfo_path`	Path to a CSV file containing the edge information. This CSV file should follow the convention of ‘source_node_name, target_node_name, edge_weight‘.
`...`	Additional arguments passed to read.csv

Value

A list ‘L‘ with the following entries:

`M:`	the adjacency matrix of the network.
`initial_values:`	an array of initial values for the network.
`node_index:`	a function that takes a node name and returns its index value.
`node_value:`	a function that takes a node name and returns its initial value.
`edge_index:`	a function that takes two node names and returns the index value of the edge connecting them.
`edge_weight:`	a function that takes two node names and returns the weight of the edge connecting them.
`g:`	the igraph graph associated with the network.
`edges:`	an array of edges.
`nodes:`	an array of nodes.
`node_names:`	an array of node names.

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")
M <- graph.from.csv(node_file, edge_file)
```

Michaelis.Menten *Michaelis-Menten equations*

Description

The standard Michaelis-Menten equations

Usage

```
Michaelis.Menten(t, x, params)
```

Arguments

t	Current time
x	Current value of all nodes
params	List of parameters. Should include the adjacency matrix of the network ‘M‘, and parameters ‘f, h,‘ and ‘B‘

Value

A list with ‘dx‘

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")
g <- Rato::graph.from.csv(node_file, edge_file, sep=",", header=TRUE)

Rato::node.removal.thread(  Rato::Michaelis.Menten
                          , g$M
                          , g$initial_values
                          , initial_params = list('f' = 1, 'h'=2, 'B'=0.1))
```

Michaelis.Menten.Fast *Optimized Michaelis-Menten equations*

Description

The optimized Michaelis-Menten equations described by Barabasi in his paper ’Universal resilience patterns in complex networks’.

Usage

```
Michaelis.Menten.Fast(t, x, params)
```

Arguments

t	Current time
x	Current value of all nodes
params	List of parameters. Should include the B_eff matrix of the network as 'Beta', and parameters 'f, h,' and 'B'. To understand how to construct the matrix B_eff, please refer to the Barabasi paper.

Value

A list with 'dx'

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")
g <- Rato::graph.from.csv(node_file, edge_file, sep=",", header=TRUE)

initial_parameters <- function(M){
 initial_params <- list('f' = 1, 'h' = 2, 'B' = 0.1, 'Beta' = Rato::Barabasi.B_eff(M))
  return(initial_params)
}

update_parameters <- function(params){
  M <- params$M
  params$Beta <- Rato::Barabasi.B_eff(M)
  return(params)
}

Rato::node.removal.thread( system = Rato::Michaelis.Menten.Fast
                         , M = g$M
                         , x0 = Rato::Barabasi.X_eff(g$M, g$initial_values)
                         , initial_params = initial_parameters
                         , update_params = update_parameters
                         , to.numeric = TRUE)
```

---

node.raw.score *Node score in network dynamics*

Description

This function evaluates the significance of one or more nodes in a network given a system. It initially tests the dynamics of the network under a healthy state (i.e., without any perturbation). Then, it removes the specified nodes and runs the ODE again. The function compares the final state of the network under the perturbation to that of the healthy network to assess the importance of the nodes for network health.

Usage

```
node.raw.score(
  system,
  M,
  x0,
  nodes,
  initial_params = list(),
  times = seq(0, 100, 1),
  reduction = mean
)
```

Arguments

`system`	A function defining the system's dynamics: 'system(time, x, params)', which returns a list of state deltas 'dx'.
`M`	The initial adjacency matrix of the network.
`x0`	Initial conditions of the network's nodes (numeric vector).
`nodes`	Nodes of interest to be removed.
`initial_params`	Either a list of initial parameters or a function of type 'f(M) -> list' that takes the adjacency matrix of the network as input and returns the initial parameters of the system.
`times`	A vector of time points for the ODE solver.
`reduction`	A reduction function applied to the ODE solution. The output of this function must be a numeric value. The default is 'mean'.

Value

A list 'L' with the following entries:

- **L$score**: The score of the nodes.

- **L$normalized.score**: The normalized score of the nodes.

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")

g <- Rato::graph.from.csv(node_file, edge_file, sep=",", header=TRUE)


# Get the raw score of a node in the network
score <- Rato::node.raw.score(
    Rato::Michaelis.Menten  # Use the Michaelis-Menten equation
  , g$M                # The GRN as an adjacency matrix
  , g$initial_values  # The initial values of the genes
  , g$node_index("ko:K03171") # The gene of interest
 , initial_params = list('f' = 1, 'h'=2, 'B'=0.01) # Parameters of the Michaelis-Menten equation
)
```

`node.removal.thread` *Simulates Network Dynamics under Random Node Removal Perturbations.*

Description

This function simulates the behavior of a network undergoing multiple uniform removal of nodes. Starting with an initial healthy network, it calculates the network's trajectory by solving the system of ordinary differential equations (ODEs) after each perturbation. Nodes are removed randomly and uniformly, followed by dimension reduction using the specified reduction function.

Usage

```
node.removal.thread(
  system,
  M,
  x0,
  initial_params = list(),
  update_params = identity,
  reduction = identity,
  removal_order = NULL,
  ...
)
```

Arguments

`system`	A function defining the system's dynamics: 'system(time, x, params)' which returns a list of state deltas 'dx'.
`M`	The initial adjacency matrix of the network.
`x0`	Initial conditions of the network's nodes (numeric vector).
`initial_params`	Either a list of initial parameters, or a function of type 'f(M) -> list' that takes the adjacency matrix of the network as input and returns the initial parameters of the system.
`update_params`	A function of type 'f(list) -> list' that receives the list of parameters after each perturbation, and returns a new list of parameters. Defaults to 'identity'.
`reduction`	A reduction function applied to the ODE solution. This can be 'identity' (for all node states) or functions like 'mean' or 'median'. The function signature should be either 'f(numeric) -> numeric' or 'f(matrix) -> matrix', depending on whether 'only.final.state' is 'TRUE' or 'FALSE'.
`removal_order`	The removal order of the nodes. Leave NULL for a random removal order.
`...`	Additional arguments passed to the ODE solver (e.g., 'method', 'atol').

Value

Depending on ‘to.numeric‘, returns either a list or a numeric matrix, representing the system’s state across multiple perturbations. If ‘to.numeric‘ is ‘FALSE‘, the function returns a list ‘L‘, where ‘L[[i]]‘ represents the final state of the system after the ‘i‘-th perturbation. If ‘TRUE‘, the list is converted to a numeric matrix before being returned.#’

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")
g <- Rato::graph.from.csv(node_file, edge_file, sep=",", header=TRUE)

Rato::node.removal.thread( Rato::Michaelis.Menten
                          , g$M
                          , g$initial_values
                          , initial_params = list('f' = 1, 'h'=2, 'B'=0.1))
```

`node.score` *Node score in network dynamics with significance test*

Description

This function evaluates the significance of one or more nodes in a network given a system. It initially tests the dynamics of the network under a healthy state (i.e., without any perturbation). Then, it removes the specified nodes and runs the ODE again. The function compares the final state of the network under the perturbation to that of the healthy network to assess the importance of the nodes for network health. Additionally, it uses bootstrap methods to estimate a p-value associated with this score.

Usage

```
node.score(
  system,
  M,
  x0,
  nodes,
  initial_params = list(),
  times = seq(0, 100, 1),
  reduction = mean,
  bootstrap_iterations = 100,
  skip_equal_nodes = TRUE
)
```

Arguments

- `system` — A function defining the system's dynamics: 'system(time, x, params)', which returns a list of state deltas 'dx'.
- `M` — The initial adjacency matrix of the network.
- `x0` — Initial conditions of the network's nodes (numeric vector).
- `nodes` — Nodes of interest to be removed.
- `initial_params` — Either a list of initial parameters or a function of type 'f(M) -> list' that takes the adjacency matrix of the network as input and returns the initial parameters of the system.
- `times` — A vector of time points for the ODE solver.
- `reduction` — A reduction function applied to the ODE solution. The output of this function must be a numeric value. The default is 'mean'.
- `bootstrap_iterations` — The number of bootstrap iterations to run.
- `skip_equal_nodes` — Whether to check if the sampled nodes are equal to the input nodes in the bootstrap method. This should be 'TRUE' when nodes contain a single element or when the network is small.

Value

A list 'L' with the following entries:

- **L$score**: The score of the nodes.

- **L$normalized.score**: The normalized score of the nodes.

- **L$p.value**: A p-value associated with the significance of the score.

- **L$normalized.p.value**: A normalized p-value associated with the significance of the score.

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")

g <- Rato::graph.from.csv(node_file, edge_file, sep=",", header=TRUE)


# Get the raw score of a node in the network
score <- Rato::node.score(
    Rato::Michaelis.Menten  # Use the Michaelis-Menten equation
  , g$M                # The GRN as an adjacency matrix
  , g$initial_values  # The initial values of the genes
  , g$node_index("ko:K03171") # The gene of interest
 , initial_params = list('f' = 1, 'h'=2, 'B'=0.01) # Parameters of the Michaelis-Menten equation
)
```

perturbation.thread *Simulates Network Dynamics under Iterative Perturbations*

Description

This function simulates the behavior of a network undergoing multiple iterations of custom perturbations. Starting with an initial healthy network, it calculates the network's trajectory by solving the system of ordinary differential equations (ODEs) after each perturbation. Perturbations are applied iteratively, followed by dimension reduction using the specified reduction function.

Usage

```
perturbation.thread(
  system,
  M,
  x0,
  perturbation_function,
  initial_parameter_function,
  times = seq(0, 100, 1),
  reduction = identity,
  to.numeric = FALSE,
  only.final.state = TRUE
)
```

Arguments

`system` A function defining the system's dynamics: 'system(time, x, params)' which returns a list of state deltas 'dx'.

`M` The initial adjacency matrix of the network.

`x0` Initial conditions of the network's nodes (numeric vector).

`perturbation_function` A function defining the perturbation applied to the network. Its signature should be 'perturbation(params, iter)', where 'params' are the current parameters passed to 'system', and 'iter' is the current iteration. Access the current network state via 'params$M'. It should return updated parameters or 'NULL' if no further perturbations should be applied.

`initial_parameter_function` A function of type 'f(M) -> list' that takes the adjacency matrix of the network as input and returns the initial parameters of the system.

`times` A vector of time points for the ODE solver.

`reduction` A reduction function applied to the ODE solution. This can be 'identity' (for all node states) or functions like 'mean' or 'median'. The function signature should be either 'f(numeric) -> numeric' or 'f(matrix) -> matrix', depending on whether 'only.final.state' is 'TRUE' or 'FALSE'.

to.numeric
: Logical; if ‘TRUE‘, converts the final list to a numeric matrix. If ‘FALSE‘, returns a list ‘L‘, where ‘L[[i]]‘ is the network state after ‘i‘ perturbations, post-reduction.

only.final.state
: Logical; if ‘TRUE‘, applies reduction only to the final state of the system. If ‘FALSE‘, the reduction function is applied to the full ODE trajectory, which can be memory-intensive.

Value

Depending on ‘to.numeric‘, returns either a list or a numeric matrix, representing the system’s state across multiple perturbations. If ‘to.numeric‘ is ‘FALSE‘, the function returns a list ‘L‘, where ‘L[[i]]‘ represents the final state of the system after the ‘i‘-th perturbation. If ‘TRUE‘, the list is converted to a numeric matrix before being returned.

Examples

```
node_file <- system.file("extdata", "IL17.nodes.csv", package = "Rato")
edge_file <- system.file("extdata", "IL17.edges.csv", package = "Rato")
g <- Rato::graph.from.csv(node_file, edge_file, sep=",", header=TRUE)

initial_params = list('f' = 1, 'h'=2, 'B'=0.01)
removal_order = NULL
update_params = identity
reduction = identity
initial_parameter_function <- function(M) {

  if (is.list(initial_params)) {
    params <- initial_params
  }
  else if (is.function(initial_params)) {
    params <- initial_params(M)
  }
  n <- nrow(M) # The number of nodes of the graph

  if(is.null(removal_order)){
      removal_order <- sample(1:n, n)
  }

  if(is.null(params$M)){
    params$M = M
  }
  params$removal_order = removal_order
  return(params)
}
perturbation <- function(params, iter) {
  removal_order <- params$removal_order
  M <- params$M

  if(length(removal_order) > 0 ) {
    index <- removal_order[1]
    removal_order <- removal_order[-1]
```

```
    M[index, ] <- 0 # Remove entries
    M[, index] <- 0 # Remove entries

    params$M = M
    params$removal_order = removal_order

    # Add the list of removed indices to the parameter list.
    if(is.null(params$removed_indices)){
      params$removed_indices <- c(index)
    } else {
      removed_indices <- c(params$removed_indices, index)
      params$removed_indices <- removed_indices
    }

    output <- update_params(params)
    return(output) # Return updated parameters
  }

  # If we removed every single entry, just STOP.
  return(NULL)
}

Rato::perturbation.thread( Rato::Michaelis.Menten
                          , g$M
                          , g$initial_values
                          , initial_parameter_function = initial_parameter_function
                          , perturbation_function = perturbation)
```

Index