Package 'iglm'

April 23, 2026

Type Package

Title Regression under Interference in Connected Populations

Version 1.2.4

Date 2026-04-10

Description An implementation of generalized linear models (GLMs) for studying relationships among attributes in connected populations, where responses of connected units can be dependent, as introduced by Fritz et al. (2025) <doi:10.1080/01621459.2025.2565851>. 'igml' extends GLMs for independent responses to dependent responses and can be used for studying spillover in connected populations and other network-mediated phenomena.

License GPL-3

Imports Rcpp (>= 1.0.8), R6, MASS, RcppArmadillo, Matrix, parallel, methods, coda, igraph, ragg

Suggests rmarkdown, knitr, testthat

Depends RcppProgress, R (>= 3.5.0)

LinkingTo Rcpp, RcppProgress, RcppArmadillo, Matrix

RoxygenNote 7.3.3

Encoding UTF-8

VignetteBuilder knitr

LazyData false

SystemRequirements C++17

NeedsCompilation yes

Author Cornelius Fritz [aut, cre], Michael Schweinberger [aut]

Maintainer Cornelius Fritz <corneliusfritz2010@gmail.com>

Repository CRAN

Date/Publication 2026-04-23 11:10:03 UTC

Contents

`control.iglm` *Set Control Parameters for iglm Estimation*

Description

Create a list of control parameters for the ‘iglm‘ estimation algorithm.

Usage

```
control.iglm(
  estimate_model = TRUE,
  display_progress = FALSE,
  return_samples = TRUE,
  offset_nonoverlap = 0,
  var_method = "Mean-value",
  non_stop = FALSE,
  tol = 0.001,
  max_it = 100,
  return_x = FALSE,
  return_y = FALSE,
  return_z = FALSE,
  accelerated = TRUE,
  exact = TRUE
)
```

Arguments

estimate_model	(logical) If 'TRUE' (default), the main model parameters are estimated. If 'FALSE', estimation is skipped and only the preprocessing is done.
display_progress	(logical) If 'TRUE', display progress messages or a progress bar during estimation. Default is 'FALSE'.
return_samples	(logical). If TRUE (default), return simulated network/attribute samples (i.e., iglm.data objects) generated during estimation (if applicable).
offset_nonoverlap	(numeric) A value added to the linear predictor for dyads not in the 'overlap' set. Default is '0'.
var_method	(string) Method for variance estimation. Options are "Mean-value" (default), "Godambe", and "Hessian". The mean-value version is described in Section 3.3 of Fritz et al. (2025), the Godambe method is described in Schmid and Hunter (2023), and the "Hessian" option just assumes that the pseudo likelihood is the correct likelihood.
non_stop	(logical) If 'TRUE', the estimation algorithm continues until 'max_it' iterations, ignoring the 'tol' convergence criterion. Default is 'FALSE'.
tol	(numeric) The tolerance level for convergence. The estimation stops when the change in coefficients between iterations is less than 'tol'. Default is '0.001'.
max_it	(integer) The maximum number of iterations for the estimation algorithm. Default is '100'.
return_x	(logical). If TRUE, return the change statistics for the x attribute Default is FALSE. from samples. Default is 'FALSE'. (Note: 'return_samples=TRUE' likely implies this).
return_y	(logical). If TRUE, return the change statistics for the y attribute Default is FALSE.
return_z	(logical). If TRUE, return the change statistics for the z network. Default is FALSE.
accelerated	(logical) If 'TRUE' (default), an accelerated MM algorithm is used based on a Quasi Newton scheme described in the Supplemental Material of Fritz et al (2025).
exact	(logical) If 'TRUE', the pseudo Fisher information is calculated exact for assessing the uncertainty of the estimates. Default is 'FALSE'.

Value

A list object of class '"control.iglm"' containing the specified control parameters.

References

Fritz, C., Schweinberger, M. , Bhadra S., and D. R. Hunter (2025). A Regression Framework for Studying Relationships among Attributes under Network Interference. Journal of the American Statistical Association, to appear.

Schmid, C.S. and D. R. Hunter (2023). Computing Pseudolikelihood Estimators for Exponential-Family Random Graph Models. Journal of Data Science

copenhagen *Copenhagen Network Study*

Description

A preprocessed dataset containing social ties, physical proximity, and nodal attributes for a subset of participants in the Copenhagen Networks Study. The object is provided as an `iglm.data` class.

Usage

```
data(copenhagen)
```

Format

The `iglm.data` provides the following information:

z_network A $E \times 2$ integer matrix representing the undirected friendship network (Z).

x_attribute A logical/binomial vector of length N indicating gender (1 for female, 0 for male).

y_attribute A numeric vector of length N representing the log-transformed total call duration in minutes: $y_i = \log(\frac{\text{seconds}}{60})$.

neighborhood A matrix defining the proximity-based constraint space. Pairs are included if their cumulative physical proximity exceeded 24 hours during the observation period.

fix_x Boolean `TRUE`, indicating that the x attribute is treated as exogenous.

Details

The following preprocessing steps were carried out:

- **Temporal Aggregation:** Proximity data (Bluetooth scans) were aggregated into sessions. A session break was defined by any temporal gap exceeding 10 minutes.
- **Recursive Pruning:** A recursive filter removed actors with missing gender information or isolated actors in either the friendship (`z_network`) or proximity (`neighborhood`) networks,

References

Sapiezynski, P., Stopczynski, A., Lassen, D. D. and Lehmann, S. (2019), Interaction data from the Copenhagen Networks Study. Scientific Data 6(1), 315.

create_userterms_skeleton *Generate the Skeleton for an R Package to Implement Additional iglm Terms*

Description

This function generates the directory structure and source files for a new R package named `iglm.userterms` (or whatever name is provided in the parameter `pkg_name`). This auxiliary package serves as a template for extending the `iglm` framework to user-defined sufficient statistics. By compiling this package, users can link custom C++ implementations of change statistics directly with the `iglm` package, enabling seamless integration of new model terms.

Usage

```
create_userterms_skeleton(path = ".", pkg_name = "iglm.userterms")
```

Arguments

`path`	A character string specifying the path where the package directory should be created. Defaults to the current working directory (`"."`).
`pkg_name`	A character string specifying the name of the package to be created.

Details

The function creates a directory with the name specified in `pkg_name` at the specified location. As an example for a possible statistic, the statistic counting mutual connections in the network is implemented. After defining all possible change-statistics in the c++ function (this has to include a change for `z_ij` (network), `x_i` (attribute x), and `y_i` (attribute y) all toggling from 0 to 1), the function has to be registered using the `EFFECT_REGISTER` macro. After compiling the package, users have to load the package using `library(pkg_name)` before using it in `iglm`.

iglm *Construct an iglm Model Specification Object*

Description

`R` package `iglm` implements generalized linear models (GLMs) for studying relationships among attributes in connected populations, where responses of connected units can be dependent. It extends GLMs for independent responses to dependent responses and can be used for studying spillover in connected populations and other network-mediated phenomena. It is based on a joint probability model for dependent responses (Y) and connections (Z) conditional on predictors (X).

Usage

```
iglm(
  formula = NULL,
  coef = NULL,
  coef_degrees = NULL,
  sampler = NULL,
  control = NULL,
  name = NULL,
  file = NULL
)
```

Arguments

formula	A model 'formula' object. The left-hand side should be the name of a 'iglm.data' object available in the calling environment. See iglm-terms for details on specifying the right-hand side terms.
coef	Optional numeric vector of initial coefficients for the structural (non-degrees) terms in 'formula'. If 'NULL', coefficients are initialized to zero. Length must match the number of terms.
coef_degrees	Optional numeric vector specifying the initial degrees coefficients. Required if 'formula' includes degrees terms, otherwise should be 'NULL'. Length must match 'n_actor' (for undirected) or '2 * n_actor' (for directed).
sampler	An object of class sampler.iglm, controlling the MCMC sampling scheme. If 'NULL', default sampler settings will be used.
control	An object of class control.iglm, specifying parameters for the estimation algorithm. If 'NULL', default control settings will be used.
name	Optional character string specifying a name for the model.
file	Optional character string specifying a file path to load a previously saved iglm.object from disk (in RDS format). If provided, other arguments are ignored and the object is loaded from the file.

Value

An object of class iglm.object.

Model Formulation

The joint probability density is specified as

$$f_\theta(y, z, x) \propto \Big[\prod_{i=1}^{N} a_x(x_i)\, a_y(y_i) \exp(\theta_g^T \mathbf{g}_i(x_i^*, y_i^*))\Big] \times \Big[\prod_{i \neq j} a_z(z_{i,j}) \exp(\theta_h^T \mathbf{h}_{i,j}(x_i^*, x_j^*, y_i^*, y_j^*, z))\Big],$$

which is defined by two distinct sets of user-specified features:

- $\mathbf{g}_i(x_i^*, y_i^*) = (g_i(x_i^*, y_i^*))$: A vector of unit-level functions (or "g-terms") that describe the relationship between an individual actor i's predictors (x_i) and their own response (y_i).

- $\mathbf{h}_{i,j}(x_i^*, x_j^*, y_i^*, y_j^*, z) = (h_{i,j}(x_i^*, x_j^*, y_i^*, y_j^*, z))$: A vector of pair-level functions (or "h-terms") that specify how the connections (z) and responses (y_i, y_j) of a pair of units $\{i, j\}$ depend on each other and the wider network structure.

This separation allows the model to simultaneously capture individual-level behavior (via g_i) and dyadic, network-based dependencies (via $h_{i,j}$), including local dependence limited to overlapping neighborhoods. This help page documents the various statistics available in 'iglm', corresponding to the g_i (attribute-level) and $h_{i,j}$ (pair-level) components of the joint model. See `iglm-terms` for details on specifying all model terms via the formula interface.

References

Fritz, C., Schweinberger, M., Bhadra, S., and D.R. Hunter (2025). A Regression Framework for Studying Relationships among Attributes under Network Interference. Journal of the American Statistical Association, to appear.

Schweinberger, M. and M.S. Handcock (2015). Local Dependence in Random Graph Models: Characterization, Properties, and Statistical Inference. Journal of the Royal Statistical Society, Series B (Statistical Methodology), 7, 647-676.

Schweinberger, M. and J.R. Stewart (2020). Concentration and Consistency Results for Canonical and Curved Exponential-Family Models of Random Graphs. The Annals of Statistics, 48, 374-396.

Stewart, J.R. and M. Schweinberger (2025). Pseudo-Likelihood-Based M-Estimation of Random Graphs with Dependent Edges and Parameter Vectors of Increasing Dimension. The Annals of Statistics, to appear.

Examples

```
# Example usage:
# Create a iglm.data data object (example)
n_actor <- 50
neighborhood <- matrix(1, nrow = n_actor, ncol = n_actor)
xyz_obj <- iglm.data(
  neighborhood = neighborhood, directed = FALSE,
  type_x = "binomial", type_y = "binomial"
)
# Define ground truth coefficients
gt_coef <- c("edges_local" = 3, "attribute_y" = -1, "attribute_x" = -1)
gt_coef_pop <- rnorm(n = n_actor, -2, 1)
# Define MCMC sampler
sampler_new <- sampler.iglm(
  n_burn_in = 100, n_simulation = 10,
  sampler_x = sampler.net.attr(n_proposals = n_actor * 10),
  sampler_y = sampler.net.attr(n_proposals = n_actor * 10),
  sampler_z = sampler.net.attr(n_proposals = sum(neighborhood > 0) * 10),
  init_empty = FALSE
)
# Create iglm model specification
model_tmp_new <- iglm(
  formula = xyz_obj ~ edges(mode = "local") +
    attribute_y + attribute_x + degrees,
  coef = gt_coef,
```

```
  coef_degrees = gt_coef_pop,
  sampler = sampler_new,
  control = control.iglm(
    accelerated = FALSE,
    max_it = 200, display_progress = FALSE
  )
)
# Simulate from the model
model_tmp_new$simulate()
model_tmp_new$set_target(model_tmp_new$get_samples()[[1]])

# Estimate model parameters
model_tmp_new$estimate()

# Model Assessment
model_tmp_new$assess(formula = ~degree_distribution)
model_tmp_new$results$plot(model_assessment = TRUE)
```

`iglm-terms` *Model Specification for iglm Terms*

Description

The help pages of `iglm` describe the model with details on model fitting and estimation. Generally, a model is specified via it's sufficient statistics, that can be further decomposed into two parts:

- $\mathbf{g}_i(x_i^*, y_i^*) = \mathbf{g}_i(x_i, y_i) = (g_i(x_i, y_i))$: A vector of unit-level functions (or "g-terms") that describe the relationship between an individual actor i's predictors (x_i) and their own response (y_i).
- $\mathbf{h}_{i,j}(x_i^*, x_j^*, y_i^*, y_j^*, z) = \mathbf{h}_{i,j}(x, y, z) = (h_{i,j}(x, y, z))$: A vector of pair-level functions (or "h-terms") that specify how the connections (z) and responses (y_i, y_j) of a pair of units $\{i, j\}$ depend on each other and the wider network structure.

Each term defines a component for the model's features, which are a sum of unit-level components, $\sum_i g_i(x_i, y_i)$, and/or pair-level components, $\sum_{i \neq j} h_{i,j}(x, y, z)$. The implemented terms are grouped into three categories:

1. **Attribute Terms**: Depend only on individual attributes x_i or y_i.
2. **Network Terms**: Depend only on the connections $z_{i,j}$.
3. **Joint Attribute/Network Terms**: Depend on both individual attributes and connections.

`degrees`: Degrees: Specifies node-level fixed effects. Estimation requires an MM algorithm constraint.

`edges(mode = "global")`: Edges: Captures the baseline propensity of tie formation $z_{i,j}$, partitioned by structural boundary $c_{i,j}$.

- `global`: $h_{i,j}(x, y, z) = z_{i,j}$

- `local`: $h_{i,j}(x, y, z) = c_{i,j} z_{i,j}$
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) z_{i,j}$

`mutual(mode = "global")`: Mutual Reciprocity: Evaluates reciprocal tie formation in directed networks.

- `global`: $h_{i,j}(x, y, z) = z_{i,j} z_{j,i}$ (for $i < j$)
- `local`: $h_{i,j}(x, y, z) = c_{i,j} z_{i,j} z_{j,i}$ (for $i < j$)
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) z_{i,j} z_{j,i}$ (for $i < j$)

`cov_z(data, mode = "global")`: Dyadic Covariate: Exogenous dyadic covariate $w_{i,j}$ influence on edge formation.

- `global`: $h_{i,j}(x, y, z) = w_{i,j} z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} w_{i,j} z_{i,j}$
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) w_{i,j} z_{i,j}$

`cov_z_out(data, mode = "global")`: Covariate Sender: Exogenous monadic covariate v_i influence on generating an outgoing tie.

- `global`: $h_{i,j}(x, y, z) = v_i z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} v_i z_{i,j}$
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) v_i z_{i,j}$

`cov_z_in(data, mode = "global")`: Covariate Receiver: Exogenous monadic covariate v_j influence on receiving an incoming tie.

- `global`: $h_{i,j}(x, y, z) = v_j z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} v_j z_{i,j}$
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) v_j z_{i,j}$

`cov_x(data = v)`: Nodal Covariate (X): Effect of a unit-level exogenous covariate v_i on endogenous attribute x_i. $g_i(x_i, y_i) = v_i x_i$

`cov_y(data = v)`: Nodal Covariate (Y): Effect of a unit-level exogenous covariate v_i on endogenous attribute y_i. $g_i(x_i, y_i) = v_i y_i$

`attribute_xy(mode = "global")`: Nodal Attribute Interaction (X-Y): Interaction of attributes x_i and y_i.

- `global`: $g_i(x_i, y_i) = x_i y_i$
- `local`: $g_i(x_i, y_i) = x_i \sum_{j \in \mathcal{N}_i} y_j + y_i \sum_{j \in \mathcal{N}_i} x_j$
- `alocal`: $g_i(x_i, y_i) = x_i \sum_{j \notin \mathcal{N}_i} y_j + y_i \sum_{j \notin \mathcal{N}_i} x_j$

`attribute_yz(mode = "local")`: Attribute Sum (Y-Z): Models the additive effect of y_i and y_j on edge formation within local neighborhoods.

`attribute_xz(mode = "local")`: Attribute Sum (X-Z): Models the additive effect of x_i and x_j on edge formation within local neighborhoods.

`inedges_y(mode = "global")`: Attribute In-Degree (Y-Z): Influence of endogenous y_j on in-degree reception.

- `global`: $h_{i,j}(x, y, z) = y_j z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} y_j z_{i,j}$
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) y_j z_{i,j}$

`outedges_y(mode = "global")`: Attribute Out-Degree (Y-Z): Influence of endogenous y_i on out-degree formation.

- `global`: $h_{i,j}(x, y, z) = y_i z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} y_i z_{i,j}$
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) y_i z_{i,j}$

`inedges_x(mode = "global")`: Attribute In-Degree (X-Z): Influence of endogenous x_j on in-degree reception.

- `global`: $h_{i,j}(x, y, z) = x_j z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} x_j z_{i,j}$
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) x_j z_{i,j}$

`outedges_x(mode = "global")`: Attribute Out-Degree (X-Z): Influence of endogenous x_i on out-degree formation.

- `global`: $h_{i,j}(x, y, z) = x_i z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} x_i z_{i,j}$
- `alocal`: $h_{i,j}(x, y, z) = (1 - c_{i,j}) x_i z_{i,j}$

`attribute_x`: Attribute (X): Intercept for attribute x. $g_i(x_i, y_i) = x_i$

`attribute_y`: Attribute (Y): Intercept for attribute y. $g_i(x_i, y_i) = y_i$

`edges_x_match(mode = "global")`: Attribute Match (X-Z): Models homophily/matching on the binary attribute x.

- `global`: $h_{i,j}(x, y, z) = \mathbb{I}(x_i = x_j) z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} \mathbb{I}(x_i = x_j) z_{i,j}$

`edges_y_match(mode = "global")`: Attribute Match (Y-Z): Models homophily/matching on the binary attribute y.

- `global`: $h_{i,j}(x, y, z) = \mathbb{I}(y_i = y_j) z_{i,j}$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} \mathbb{I}(y_i = y_j) z_{i,j}$

`spillover_yy_scaled(mode = "global")`: Scaled Y-Y-Z Outcome Spillover: Normalizes the y-outcome spillover influence by the relevant out-degree topology.

- `global`: $h_{i,j}(x, y, z) = y_i y_j z_{i,j} / \text{deg}(i)$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} y_i y_j z_{i,j} / \text{deg}(i, \text{local})$

`spillover_xx_scaled(mode = "global")`: Scaled X-X-Z Outcome Spillover: Normalizes the x-outcome spillover influence by the relevant out-degree topology.

- `global`: $h_{i,j}(x, y, z) = x_i x_j z_{i,j} / \text{deg}(i)$

- `local`: $h_{i,j}(x, y, z) = c_{i,j} x_i x_j z_{i,j} / \text{deg}(i, \text{local})$

`spillover_yx_scaled(mode = "global")`: Scaled Y-X-Z Treatment Spillover: Normalizes cross-attribute $y_i \to x_j$ spillover influence.

- `global`: $h_{i,j}(x, y, z) = y_i x_j z_{i,j} / \text{deg}(i)$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} y_i x_j z_{i,j} / \text{deg}(i, \text{local})$

`spillover_xy_scaled(mode = "global")`: Scaled X-Y-Z Treatment Spillover: Normalizes cross-attribute $x_i \to y_j$ spillover influence.

- `global`: $h_{i,j}(x, y, z) = x_i y_j z_{i,j} / \text{deg}(i)$
- `local`: $h_{i,j}(x, y, z) = c_{i,j} x_i y_j z_{i,j} / \text{deg}(i, \text{local})$

`gwesp(data, mode = "global", variant = "OSP", decay = 0)`: Geometrically Weighted Edgewise-Shared Partners: Models triadic closure propensity conditioning on existing edges. Types dictate path constraint: OTP, ITP, OSP, ISP for directed; symm for undirected.

`gwdsp(data, mode = "global", variant = "OSP", decay = 0)`: Geometrically Weighted Dyadwise-Shared Partners: Models triadic potential irrespective of the closing edge. Types dictate path constraint: OTP, ITP, OSP, ISP for directed; symm for undirected.

`gwdegree(mode = "global", decay = 0)`: Geometrically Weighted Degree: Captures the degree distribution utilizing an exponential decay parameter.

`gwidegree(mode = "global", decay = 0)`: Geometrically Weighted In-Degree: Captures the in-degree distribution utilizing an exponential decay parameter.

`gwodegree(mode = "global", decay = 0)`: Geometrically Weighted Out-Degree: Captures the out-degree distribution utilizing an exponential decay parameter.

`spillover_yc_symm(data = v, mode = "local")`: Symmetric Y-C-Z Treatment Spillover: Bidirectional mapping of exogenous covariate v and endogenous trait y interaction.

`spillover_xy(mode = "local")`: Directed X-Y-Z Treatment Spillover: Maps cross-attribute $x_i \to y_j$ treatment assignment.

`spillover_yc(mode = "local")`: Directed Y-C-Z Treatment Spillover: Exogenous covariate v interacting with endogenous trait y.

`spillover_yx(mode = "local")`: Directed Y-X-Z Treatment Spillover: Maps cross-attribute $y_i \to x_j$ treatment assignment.

`spillover_yy(mode = "local")`: Symmetric Y-Y-Z Outcome Spillover: Propagates y-outcome spillover effects.

`spillover_xx(mode = "local")`: Symmetric X-X-Z Outcome Spillover: Propagates x-outcome spillover effects.

`transitive`: Transitivity (Local): Indicator evaluating the presence of a local transitive triad configuration.

`nonisolates`: Non-Isolates: Captures frequency of nodes with degree strictly greater than zero.

`isolates`: Isolates: Captures frequency of nodes with degree zero.

Category 1

Attribute Terms:

Below is a detailed description of terms that depend only on nodal attributes:

- attribute_x-term, attribute_y-term
- cov_x-term, cov_y-term
- attribute_xy-term

Category 2

Network Terms:

Below is a detailed description of terms that depend only on the network structure:

- edges-term, mutual-term
- cov_z-term, cov_z_in-term, cov_z_out-term
- degrees-term
- gwdegree-term, gwidegree-term, gwodegree-term
- gwesp-term, gwdsp-term
- transitive-term, nonisolates-term, isolates-term

Category 3

Joint Attribute/Network Terms:

Below is a detailed description of terms that depend on both attributes and the network:

- attribute_xz-term, attribute_yz-term
- inedges_x-term, inedges_y-term, outedges_x-term, outedges_y-term
- edges_x_match-term, edges_y_match-term
- spillover_xx-term, spillover_yy-term
- spillover_yx-term, spillover_xy-term, spillover_yc-term, spillover_yc_symm-term

References

Fritz, C., Schweinberger, M., Bhadra, S., and D.R. Hunter (2025). A Regression Framework for Studying Relationships among Attributes under Network Interference. Journal of the American Statistical Association, to appear.

Schweinberger, M. and M.S. Handcock (2015). Local Dependence in Random Graph Models: Characterization, Properties, and Statistical Inference. Journal of the Royal Statistical Society, Series B (Statistical Methodology), 7, 647-676.

Schweinberger, M. and J.R. Stewart (2020). Concentration and Consistency Results for Canonical and Curved Exponential-Family Models of Random Graphs. The Annals of Statistics, 48, 374-396.

Stewart, J.R. and M. Schweinberger (2025). Pseudo-Likelihood-Based M-Estimation of Random Graphs with Dependent Edges and Parameter Vectors of Increasing Dimension. The Annals of Statistics, to appear.

iglm.data *Constructor for the iglm.data R6 object*

Description

Creates a ‘iglm.data‘ object, which stores network and attribute data. This function acts as a user-friendly interface to the ‘iglm.data‘ R6 class generator. It handles data input, infers parameters like the number of actors (‘n_actor‘) and network directedness (‘directed‘) if not explicitly provided, processes network data into a consistent edgelist format, calculates the overlap relation based on an optional neighborhood definition, and performs extensive validation of all inputs.

Usage

```
iglm.data(
  x_attribute = NULL,
  y_attribute = NULL,
  z_network = NULL,
  neighborhood = NULL,
  directed = TRUE,
  n_actor = NA,
  type_x = "binomial",
  type_y = "binomial",
  scale_x = 1,
  scale_y = 1,
  fix_x = FALSE,
  fix_z = FALSE,
  fix_z_alocal = FALSE,
  return_neighborhood = TRUE,
  file = NULL
)
```

Arguments

x_attribute	A numeric vector for the first unit-level attribute.
y_attribute	A numeric vector for the second unit-level attribute.
z_network	A matrix representing the network. Can be a 2-column edgelist or a square adjacency matrix.
neighborhood	An optional matrix for the neighborhood representing local dependence. Can be a 2-column edgelist or a square adjacency matrix. A tie in ‘neighborhood‘ between actor i and j indicates that j is in the neighborhood of i, implying dependence between the respective actors.
directed	A logical value indicating if ‘z_network‘ is directed. If ‘NA‘ (default), directedness is inferred from the symmetry of ‘z_network‘.
n_actor	An integer for the number of actors in the system. If ‘NA‘ (default), ‘n_actor‘ is inferred from the attributes or network matrices.

`type_x` Character string for the type of ‘x_attribute‘. Must be one of ‘"binomial"‘, ‘"poisson"‘, or ‘"normal"‘. Default is ‘"binomial"‘.

`type_y` Character string for the type of ‘y_attribute‘. Must be one of ‘"binomial"‘, ‘"poisson"‘, or ‘"normal"‘. Default is ‘"binomial"‘.

`scale_x` A positive numeric value for scaling (e.g., variance for "normal" type). Default is 1.

`scale_y` A positive numeric value for scaling (e.g., variance for "normal" type). Default is 1.

`fix_x` (logical) If ‘TRUE‘, the ’x’ predictor is held fixed during estimation/simulation (fixed design in regression). Default is ‘FALSE‘.

`fix_z` (logical) If ‘TRUE‘, the ’z’ network is held fixed during estimation/simulation (fixed network design). Default is ‘FALSE‘. Setting this to TRUE, allows practicioners to estimate autologistic actor attribute models, which were introduced in binary settings in Daraganova, G., & Robins, G. (2013).

`fix_z_alocal` (logical) If ‘TRUE‘, edges outside the overlap region are fixed, else they are random (default).

`return_neighborhood` Logical. If ‘TRUE‘ (default) and ‘neighborhood‘ is ‘NULL‘, a full neighborhood (all dyads) is generated implying global dependence. If ‘FALSE‘, no neighborhood is set.

`file` (character) Optional file path to load a saved ‘iglm.data‘ object state.

Value

An object of class ‘iglm.data‘ (and ‘R6‘).

References

Fritz, C., Schweinberger, M. , Bhadra S., and D. R. Hunter (2025). A Regression Framework for Studying Relationships among Attributes under Network Interference. Journal of the American Statistical Association, to appear.

Daraganova, G., and Robins, G. (2013). Exponential random graph models for social networks: Theory, methods and applications, 102-114. Cambridge University Press.

Examples

```
data("state_twitter")
state_twitter
state_twitter$iglm.data$degree_distribution(prob = FALSE, plot = TRUE)
state_twitter$iglm.data$geodesic_distances_distribution(prob = FALSE, plot = TRUE)
state_twitter$iglm.data$mean_x()
state_twitter$iglm.data$mean_y()


# Generate a small iglm data object either via adjacency matrix or edgelist
tmp_adjacency <- iglm.data(
  z_network = matrix(c(
```

```
    0, 1, 1, 0,
    1, 0, 0, 1,
    1, 0, 0, 1,
    0, 1, 1, 0
  ), nrow = 4, byrow = TRUE),
  directed = FALSE,
  n_actor = 4,
  type_x = "binomial",
  type_y = "binomial"
)


tmp_edgelist <- iglm.data(
  z_network = tmp_adjacency$z_network,
  directed = FALSE,
  n_actor = 4,
  type_x = "binomial",
  type_y = "binomial"
)

tmp_edgelist$mean_z()
tmp_adjacency$mean_z()
```

`iglm.data_generator` *Networks with Unit-Level Attributes (R6 Class)*

Description

The 'iglm.data' class is a container for storing, validating, and analyzing unit-level attributes (x_attribute, y_attribute) and connections (z_network).

Active bindings

`x_attribute` ('numeric') The vector for the first unit-level attribute.

`y_attribute` ('numeric') The vector for the second unit-level attribute.

`z_network` ('matrix') The primary network structure as a 2-column integer edgelist.

`neighborhood` ('matrix') Read-only. The secondary/neighborhood structure as a 2-column integer edgelist. An empty matrix if not provided.

`overlap` ('matrix') Read-only. The calculated overlap relation (dyads with shared neighbors in 'neighborhood') as a 2-column integer edgelist. An empty matrix if overlap hasn't been computed or is not available.

`directed` ('logical') Indicates if the 'z_network' is treated as directed.

`n_actor` ('integer') The total number of actors (nodes) in the network.

`type_x` ('character') The specified distribution type for the 'x_attribute'.

`type_y` ('character') The specified distribution type for the 'y_attribute'.

`scale_x` ('numeric') The scale parameter associated with the 'x_attribute'.

`scale_y` (‘numeric‘) The scale parameter associated with the ‘y_attribute‘.

`fix_x` (‘logical‘) Indicates if the ‘x_attribute‘ is fixed during estimation/simulation.

`fix_z` (‘logical‘) RIndicates if the ‘z_network‘ is fixed during estimation/simulation.

`descriptives` (‘list‘)A list storing computed descriptive statistics for the network and attributes.

`fix_z_alocal` (‘logical‘) Flag indicating whether nonoverlap edges are treated as random.

Methods

Public methods:

- `iglm.data_generator$new()`
- `iglm.data_generator$set_z_network()`
- `iglm.data_generator$set_type_x()`
- `iglm.data_generator$set_type_y()`
- `iglm.data_generator$set_scale_x()`
- `iglm.data_generator$set_scale_y()`
- `iglm.data_generator$set_x_attribute()`
- `iglm.data_generator$set_y_attribute()`
- `iglm.data_generator$gather()`
- `iglm.data_generator$set_fix_z_alocal()`
- `iglm.data_generator$delete_isolates()`
- `iglm.data_generator$save()`
- `iglm.data_generator$set_fix_x()`
- `iglm.data_generator$set_fix_z()`
- `iglm.data_generator$mean_z()`
- `iglm.data_generator$mean_x()`
- `iglm.data_generator$mean_y()`
- `iglm.data_generator$x_distribution()`
- `iglm.data_generator$y_distribution()`
- `iglm.data_generator$edgewise_shared_partner()`
- `iglm.data_generator$set_neighborhood_overlap()`
- `iglm.data_generator$dyadwise_shared_partner()`
- `iglm.data_generator$geodesic_distances_distribution()`
- `iglm.data_generator$geodesic_distances()`
- `iglm.data_generator$edgewise_shared_partner_distribution()`
- `iglm.data_generator$dyadwise_shared_partner_distribution()`
- `iglm.data_generator$degree_distribution()`
- `iglm.data_generator$degree()`
- `iglm.data_generator$spillover_degree_distribution()`
- `iglm.data_generator$plot()`
- `iglm.data_generator$print()`
- `iglm.data_generator$clone()`

Method `new()`**:** Create a new ‘iglm.data‘ object, that includes data on two attributes and one network.

Usage:

```
iglm.data_generator$new(
  x_attribute = NULL,
  y_attribute = NULL,
  z_network = NULL,
  neighborhood = NULL,
  directed = NA,
  n_actor = NA,
  type_x = "binomial",
  type_y = "binomial",
  scale_x = 1,
  scale_y = 1,
  fix_x = FALSE,
  fix_z = FALSE,
  fix_z_alocal = TRUE,
  return_neighborhood = TRUE,
  file = NULL
)
```

Arguments:

`x_attribute` A numeric vector for the first unit-level attribute.

`y_attribute` A numeric vector for the second unit-level attribute.

`z_network` A matrix representing the network. Can be a 2-column edgelist or a square adjacency matrix.

`neighborhood` An optional matrix for the neighborhood representing local dependence. Can be a 2-column edgelist or a square adjacency matrix. A tie in ‘neighborhood‘ between actor i and j indicates that j is in the neighborhood of i, implying dependence between the respective actors.

`directed` A logical value indicating if ‘z_network‘ is directed. If ‘NA‘ (default), directedness is inferred from the symmetry of ‘z_network‘.

`n_actor` An integer for the number of actors in the system. If ‘NA‘ (default), ‘n_actor‘ is inferred from the attributes or network matrices.

`type_x` Character string for the type of ‘x_attribute‘. Must be one of ‘"binomial"‘, ‘"poisson"‘, or ‘"normal"‘. Default is ‘"binomial"‘.

`type_y` Character string for the type of ‘y_attribute‘. Must be one of ‘"binomial"‘, ‘"poisson"‘, or ‘"normal"‘. Default is ‘"binomial"‘.

`scale_x` A positive numeric value for scaling (e.g., variance for "normal" type). Default is 1.

`scale_y` A positive numeric value for scaling (e.g., variance for "normal" type). Default is 1.

`fix_x` Logical. If ‘TRUE‘, the ‘x_attribute‘ is treated as fixed during model estimation and simulation. Default is ‘FALSE‘.

`fix_z` Logical. If ‘TRUE‘, the ‘z_network‘ is treated as fixed during model estimation and simulation. Default is ‘FALSE‘.

`fix_z_alocal` Logical. If ‘TRUE‘ (default), alocal dyads in the neighborhood are fixed.

`return_neighborhood` Logical. If ‘TRUE‘ (default) and ‘neighborhood‘ is ‘NULL‘, a full neighborhood (all dyads) is generated implying global dependence. If ‘FALSE‘, no neighborhood is set.

`file` (character) Optional file path to load a saved ‘iglm.data‘ object state.

Returns: A new ‘iglm.data‘ object.

Method `set_z_network()`**:** Sets the ‘z_network‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_z_network(z_network)
```

Arguments:

`z_network` A matrix representing the network. Can be a 2-column edgelist or a square adjacency matrix. @return The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `set_type_x()`**:** Sets the ‘type_x‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_type_x(type_x)
```

Arguments:

`type_x` A character string for the type of ‘x_attribute‘. Must be one of ‘"binomial"‘, ‘"poisson"‘, or ‘"normal"‘. @return The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `set_type_y()`**:** Sets the ‘type_y‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_type_y(type_y)
```

Arguments:

`type_y` A character string for the type of ‘y_attribute‘. Must be one of ‘"binomial"‘, ‘"poisson"‘, or ‘"normal"‘.

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `set_scale_x()`**:** Sets the ‘scale_x‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_scale_x(scale_x)
```

Arguments:

`scale_x` A positive numeric value for scaling (e.g., variance for "normal" type).

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `set_scale_y()`**:** Sets the ‘scale_y‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_scale_y(scale_y)
```

Arguments:

`scale_y` A positive numeric value for scaling (e.g., variance for "normal" type).

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `set_x_attribute()`**:** Sets the ‘x_attribute‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_x_attribute(x_attribute)
```

Arguments:

`x_attribute` A numeric vector for the first unit-level attribute.

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `set_y_attribute()`**:** Sets the ‘y_attribute‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_y_attribute(y_attribute)
```

Arguments:

`y_attribute` A numeric vector for the first unit-level attribute.

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `gather()`**:** Gathers the current state of the ‘iglm.data‘ object into a list. This includes all attributes, network, and configuration details necessary to reconstruct the object later.

Usage:

```
iglm.data_generator$gather()
```

Returns: A list containing the current state of the ‘iglm.data‘ object.

Method `set_fix_z_alocal()`**:** Sets the option whether alocal edges are fixed or not.

Usage:

```
iglm.data_generator$set_fix_z_alocal(fix_z_alocal)
```

Arguments:

`fix_z_alocal` A logical value indicating whether alocal edges should be treated as fixed or not.

Method `delete_isolates()`**:** Deletes isolates from the ‘z_network‘ and updates the attributes and neighborhood accordingly. Isolates are actors that do not have any connections in the ‘z_network‘. This method identifies such actors, removes them from the attributes and neighborhood, and updates the ‘z_network‘ to reflect the new actor indices.

Usage:

```
iglm.data_generator$delete_isolates()
```

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `save()`**:** Saves the current state of the ‘iglm.data‘ object to a specified file path in RDS format. This includes all attributes, network, and configuration details necessary to reconstruct the object later.

Usage:

```
iglm.data_generator$save(file)
```

Arguments:

`file` (character) The file where the object state should be saved. Must have a .rds extension.

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `set_fix_x()`**:** Sets the ‘fix_x‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_fix_x(fix_x)
```

Arguments:

`fix_x` A logical value indicating if ‘x_attribute‘ is fixed or random.

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `set_fix_z()`**:** Sets the ‘fix_z‘ of the ‘iglm.data‘ object.

Usage:

```
iglm.data_generator$set_fix_z(fix_z)
```

Arguments:

`fix_z` A logical value indicating if ‘z_network‘ is fixed or random.

Returns: The ‘iglm.data‘ object itself (‘self‘), invisibly.

Method `mean_z()`**:** Calculates the density of the ‘z_network‘.

Usage:

```
iglm.data_generator$mean_z()
```

Returns: A numeric value for the network density.

Method `mean_x()`**:** Calculates the mean of the ‘x_attribute‘.

Usage:

```
iglm.data_generator$mean_x()
```

Returns: A numeric value for the mean of ‘x_attribute‘.

Method `mean_y()`**:** Calculates the mean of the ‘y_attribute‘.

Usage:

```
iglm.data_generator$mean_y()
```

Returns: A numeric value for the mean of ‘y_attribute‘.

Method `x_distribution()`**:** Calculates the distribution of the ‘x_attribute‘.

Usage:

```
iglm.data_generator$x_distribution(
  value_range = NULL,
  prob = TRUE,
  plot = TRUE
)
```

Arguments:

`value_range` (numeric vector) Optional range of values to consider for the distribution. If ‘NULL‘ (default), the range is inferred from the data.

`prob` (logical) If ‘TRUE‘ (default), returns probabilities; if ‘FALSE‘, returns frequencies.

`plot` (logical) If ‘TRUE‘ (default), plots the distribution using a density plot for continuous data or a bar plot for discrete data.

Returns: A numeric vector representing the distribution of ‘x_attribute‘ (invisible).

Method `y_distribution()`**:** Calculates the distribution of the ‘y_attribute‘.

Usage:

```
iglm.data_generator$y_distribution(
  value_range = NULL,
  prob = TRUE,
  plot = TRUE
)
```

Arguments:

`value_range` (numeric vector) Optional range of values to consider for the distribution. If 'NULL' (default), the range is inferred from the data.

`prob` (logical) If 'TRUE' (default), returns probabilities; if 'FALSE', returns frequencies.

`plot` (logical) If 'TRUE' (default), plots the distribution using a density plot for continuous data or a bar plot for discrete data.

Returns: A numeric vector representing the distribution of 'y_attribute' (invisible).

Method `edgewise_shared_partner()`**:** Calculates the matrix of edgewise shared partners. This is a two-path matrix (e.g., $A A^T$ or $A^T A$).

Usage:

```
iglm.data_generator$edgewise_shared_partner(type = "ALL")
```

Arguments:

`type` (character) The type of two-path to calculate for directed networks. Ignored if network is undirected. Must be one of: '"OTP"' (Outgoing Two-Path, $z_{i,j}\, z_{i,h}\, z_{j,h}$), '"ISP"' (Ingoing Shared Partner, $z_{i,j}\, z_{h,i}\, z_{j,h}$), '"OSP"' (Outgoing Shared Partner, $z_{i,j}\, z_{i,h}\, z_{j,h}$), '"ITP"' (Incoming Two-Path, $z_{i,j}\, z_{h,i}\, z_{j,h}$), '"ALL"' (Any one of the above). Default is '"ALL"'.

Returns: A sparse matrix ('dgCMatrix') of shared partner counts.

Method `set_neighborhood_overlap()`**:** Sets the neighborhood and overlap matrices.

Usage:

```
iglm.data_generator$set_neighborhood_overlap(neighborhood, overlap)
```

Arguments:

`neighborhood` A matrix for a secondary neighborhood. Can be a 2-column edgelist or a square adjacency matrix.

`overlap` A matrix for the overlap network. Can be a 2-column edgelist or a square adjacency matrix.

Returns: None. Updates the internal neighborhood and overlap matrices.

Method `dyadwise_shared_partner()`**:** Calculates the matrix of dyadwise shared partners.

Usage:

```
iglm.data_generator$dyadwise_shared_partner(type = "ALL")
```

Arguments:

`type` (character) The type of two-path to calculate for directed networks. Ignored if network is undirected. Must be one of: '"OTP"' (Outgoing Two-Path, $z_{i,h}\, z_{j,h}$), '"ISP"' (Ingoing Shared Partner, $z_{h,i}\, z_{j,h}$), '"OSP"' (Outgoing Shared Partner, $z_{i,h}\, z_{j,h}$), '"ITP"' (Incoming Two-Path, $z_{h,i}\, z_{j,h}$), '"ALL"' (Any one of the above). Default is '"ALL"'.

Returns: A sparse matrix (‘dgCMatrix‘) of shared partner counts.

Method `geodesic_distances_distribution()`**:** Calculates the geodesic distance distribution of the symmetrized ‘z_network‘.

Usage:

```
iglm.data_generator$geodesic_distances_distribution(
  value_range = NULL,
  prob = TRUE,
  plot = TRUE
)
```

Arguments:

`value_range` (numeric vector) A vector ‘c(min, max)‘ specifying the range of distances to tabulate. If ‘NULL‘ (default), the range is inferred from the data.

`prob` (logical) If ‘TRUE‘ (default), returns a probability distribution (proportions). If ‘FALSE‘, returns raw counts.

`plot` (logical) If ‘TRUE‘, plots the distribution.

Returns: A named vector (a ‘table‘ object) with the distribution of geodesic distances. Includes ‘Inf‘ for unreachable pairs.

Method `geodesic_distances()`**:** Calculates the all-pairs geodesic distance matrix for the symmetrized ‘z_network‘ using a matrix-based BFS algorithm.

Usage:

```
iglm.data_generator$geodesic_distances()
```

Returns: A sparse matrix (‘dgCMatrix‘) where ‘D[i, j]‘ is the shortest path distance from i to j. ‘Inf‘ indicates no path.

Method `edgewise_shared_partner_distribution()`**:** Calculates the distribution of edgewise shared partners.

Usage:

```
iglm.data_generator$edgewise_shared_partner_distribution(
  type = "ALL",
  value_range = NULL,
  prob = TRUE,
  plot = TRUE
)
```

Arguments:

`type` (character) The type of shared partner matrix to use. See ‘edgewise_shared_partner‘ for details. Default is ‘"ALL"‘.

`value_range` (numeric vector) A vector ‘c(min, max)‘ specifying the range of counts to tabulate. If ‘NULL‘ (default), the range is inferred from the data.

`prob` (logical) If ‘TRUE‘ (default), returns a probability distribution (proportions). If ‘FALSE‘, returns raw counts.

`plot` (logical) If ‘TRUE‘, plots the distribution.

Returns: A named vector (a ‘table‘ object) with the distribution of shared partner counts.

Method `dyadwise_shared_partner_distribution()`**:** Calculates the distribution of dyadwise shared partners.

Usage:

```
iglm.data_generator$dyadwise_shared_partner_distribution(
  type = "ALL",
  value_range = NULL,
  prob = TRUE,
  plot = TRUE
)
```

Arguments:

`type` (character) The type of shared partner matrix to use. See 'dyadwise_shared_partner' for details. Default is '"ALL"'.

`value_range` (numeric vector) A vector 'c(min, max)' specifying the range of counts to tabulate. If 'NULL' (default), the range is inferred from the data.

`prob` (logical) If 'TRUE' (default), returns a probability distribution (proportions). If 'FALSE', returns raw counts.

`plot` (logical) If 'TRUE', plots the distribution.

Returns: A named vector (a 'table' object) with the distribution of shared partner counts.

Method `degree_distribution()`**:** Calculates the degree distribution of the 'z_network'.

Usage:

```
iglm.data_generator$degree_distribution(
  value_range = NULL,
  prob = TRUE,
  plot = TRUE
)
```

Arguments:

`value_range` (numeric vector) A vector 'c(min, max)' specifying the range of degrees to tabulate. If 'NULL' (default), the range is inferred from the data.

`prob` (logical) If 'TRUE' (default), returns a probability distribution (proportions). If 'FALSE', returns raw counts.

`plot` (logical) If 'TRUE', plots the degree distribution.

Returns: If the network is directed, a list containing two 'table' objects: 'in_degree' and 'out_degree'. If undirected, a single 'table' object with the degree distribution.

Method `degree()`**:** Calculates the degree sequence(s) of the 'z_network'.

Usage:

```
iglm.data_generator$degree()
```

Returns: If the network is directed, a list containing two vectors: 'in_degree_seq' and 'out_degree_seq'. If undirected, a single list containing the vector 'degree_seq'.

Method `spillover_degree_distribution()`**:** Calculates the spillover degree distribution between actors with 'x_attribute == 1' and actors with 'y_attribute == 1'.

Usage:

```
iglm.data_generator$spillover_degree_distribution(
  prob = TRUE,
  value_range = NULL,
  plot = TRUE
)
```

Arguments:

`prob` (logical) If 'TRUE' (default), returns a probability distribution (proportions). If 'FALSE', returns raw counts.

`value_range` (numeric vector) A vector 'c(min, max)' specifying the range of degrees to tabulate. If 'NULL' (default), the range is inferred from the data.

`plot` (logical) If 'TRUE', plots the distributions.

Returns: A list containing two 'table' objects: 'out_spillover_degree' (from x_i=1 to y_j=1) and 'in_spillover_degree' (from y_i=1 to x_j=1).

Method `plot()`**:** Plot the network using 'igraph'.

Visualizes the 'z_network' using the 'igraph' package. Nodes can be colored by 'x_attribute' and sized by 'y_attribute'. 'neighborhood' edges can be plotted as a background layer.

Usage:

```
iglm.data_generator$plot(
  node_color = "x",
  node_size = "y",
  show_overlap = TRUE,
  layout = igraph::layout_with_fr,
  network_edges_col = "grey60",
  neighborhood_edges_col = "orange",
  main = "",
  legend_col_n_levels = NULL,
  legend_size_n_levels = NULL,
  legend_pos = "right",
  alpha_neighborhood = 0.2,
  edge.width = 1,
  edge.arrow.size = 1,
  vertex.frame.width = 0.5,
  coords = NULL,
  legend_size = 0.5,
  ...
)
```

Arguments:

`node_color` (character) Attribute to map to node color. One of '"x"' (default), '"y"', or '"none"'.

`node_size` (character) Attribute to map to node size. One of '"y"' (default), '"x"', or '"constant"'.

`show_overlap` (logical) If 'TRUE' (default), plot the 'neighborhood' edges as a background layer.

`layout` An 'igraph' layout function (e.g., 'igraph::layout_with_fr').

`network_edges_col` (character) Color for the 'z_network' edges.

`neighborhood_edges_col` (character) Color for the 'neighborhood' edges.

`main` (character) The main title for the plot.

`legend_col_n_levels` (integer) Number of levels for the color legend.

`legend_size_n_levels` (integer) Number of levels for the size legend.

`legend_pos` (character) Position of the legend (e.g., '"right"').

`alpha_neighborhood` (numeric) Alpha transparency for neighborhood edges.

`edge.width` (numeric) Width of the network edges.

`edge.arrow.size` (numeric) Size of the arrowheads for directed edges.

`vertex.frame.width` (numeric) Width of the vertex frame.

`coords` (matrix) Optional matrix of x-y coordinates for node layout.

`legend_size` (numeric) Scaling factor for the size legend.

`...` Additional arguments passed to 'plot.igraph'.

Returns: A list containing the 'igraph' object ('graph') and the layout coordinates ('coords'), invisibly.

Method `print()`**:** Print a summary of the 'iglm.data' object to the console.

Usage:

```
iglm.data_generator$print(digits = 3, ...)
```

Arguments:

`digits` (integer) Number of digits to round numeric output to.

`...` Additional arguments (not used).

Returns: The object's private environment, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
iglm.data_generator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`iglm.object.generator` *iglm Objects (R6 Class)*

Description

The 'iglm.object' class encapsulates all components required to define, estimate, and simulate from a generalized linear model under interference. This includes the model formula, coefficients, the underlying network and attribute data (via a 'iglm.data' object), sampler controls, estimation controls, and storage for results.

Active bindings

`formula` ('formula') Read-only. The model formula specifying terms and data object.

`coef` ('numeric') Read-only. The current vector of non-degrees coefficient estimates or initial values.

`coef_degrees` ('numeric' or 'NULL') Read-only. The current vector of degrees coefficient estimates or initial values, or 'NULL' if not applicable.

`results` ('results') Read-only. The `results` R6 object containing all estimation and simulation outputs.

`iglm.data` ('iglm.data') The associated `iglm.data` R6 object containing the network and attribute data.

`control` ('control.iglm') The `control.iglm` object specifying estimation parameters.

`sampler` ('sampler.iglm') The `sampler.iglm` object specifying MCMC sampling parameters.

`name` ('character') The name of the model.

`sufficient_statistics` ('numeric') Read-only. A named vector of the observed network statistics corresponding to the model terms, calculated on the current 'iglm.data' data.

Methods

Public methods:

- `iglm.object.generator$new()`
- `iglm.object.generator$is_equivalent()`
- `iglm.object.generator$assess()`
- `iglm.object.generator$print()`
- `iglm.object.generator$plot()`
- `iglm.object.generator$gather()`
- `iglm.object.generator$set_name()`
- `iglm.object.generator$set_control()`
- `iglm.object.generator$save()`
- `iglm.object.generator$estimate()`
- `iglm.object.generator$summary()`
- `iglm.object.generator$simulate()`
- `iglm.object.generator$predict()`
- `iglm.object.generator$set_coefficients()`
- `iglm.object.generator$get_samples()`
- `iglm.object.generator$set_sampler()`
- `iglm.object.generator$set_target()`
- `iglm.object.generator$clone()`

Method `new()`**:** Internal method to calculate the observed count statistics based on the model formula and the data in the 'iglm.data' object. Populates the 'private$.sufficient_statistics' field.

Internal validation method. Checks the consistency and validity of all components of the 'iglm.object'. Stops with an error if any check fails.

Creates a new 'iglm.object'. This involves parsing the formula, linking the data object, initializing coefficients, setting up sampler and control objects, calculating initial statistics, and validating.

Usage:

```
iglm.object.generator$new(
  formula = NULL,
  coef = NULL,
  coef_degrees = NULL,
  sampler = NULL,
  control = NULL,
  name = NULL,
  file = NULL
)
```

Arguments:

`formula` A model 'formula' object. The left-hand side should be the name of a `iglm.data` object available in the calling environment. See `iglm-terms` for details on specifying the right-hand side terms.

`coef` A numeric vector of initial coefficients for the terms in the formula (excluding degree coefficeints). If 'NULL', coefficients are initialized to zero.

`coef_degrees` An optional numeric vector of initial degree coefficients. Should be 'NULL' if the formula does not include degree-correcting terms.

`sampler` A `sampler.iglm` object specifying the MCMC sampler settings. If 'NULL', default settings are used.

`control` A `control.iglm` object specifying estimation control parameters. If 'NULL', default settings are used.

`name` An optional character string specifying a name for the model, would be used in plots and model assessment.

`file` (character or 'NULL') If provided, loads the sampler state from the specified .rds file instead of initializing from parameters.

Returns: A new 'iglm.object'.

Method `is_equivalent()`**:** Check if this iglm object is equivalent to another iglm object by comparing their defining features, data, and parameters.

Usage:

```
iglm.object.generator$is_equivalent(other, tol = 1e-05, check_results = FALSE)
```

Arguments:

`other` Another object to compare against.

`tol` Tolerance for numeric comparisons (default is 1e-5).

`check_results` (logical) If 'TRUE', also requires the estimation results and MCMC samples to match exactly. Default is 'FALSE' (only compares model specification, input data, and initial coefficients).

Returns: 'TRUE' if the objects are equivalent, otherwise 'FALSE'.

Method `assess()`**:** Performs model assessment by calculating specified network statistics on the observed network and comparing their distribution to the distribution obtained from simulated networks based on the current model parameters. Requires simulations to have been run first (via `iglm.object$simulate` or `iglm.object_generator$estimate`).

Usage:

```
iglm.object.generator$assess(formula, plot = TRUE)
```

Arguments:

`formula` A formula specifying the network statistics to assess (e.g., '~ degree_distribution() + geodesic_distances_distribution()'). The terms should correspond to methods available in the `iglm.data` object that end with 'distributions'. If the term mcmc_diagnostics is included, MCMC diagnostics will also be computed.

`plot` (logical) If 'TRUE', generates plots comparing observed and simulated statistics. Default is 'TRUE'.

Returns: An object of class 'iglm_model_assessment' containing the observed statistics and the distribution of simulated statistics. The result is also stored internally.

Method `print()`**:** Print a summary of the 'iglm.object'. If estimation results are available, they are printed in a standard coefficient table format.

Usage:

```
iglm.object.generator$print(
  digits = 3,
  rows = c(1, 2),
  signif.stars = getOption("show.signif.stars"),
  eps.Pvalue = 1e-04,
  print.formula = TRUE,
  print.fitinfo = TRUE,
  print.coefmat = TRUE,
  print.call = TRUE,
  ...
)
```

Arguments:

`digits` (integer) Number of digits for rounding numeric output.

`rows` If a numeric vector with values between 1 and 5 is provided, only the corresponding columns are printed (1: Estimate, 2: S.E., 3: t-value, 4: Pr(>|t|), 5: Global Count of Sufficient Statistic). Default is 'c(1, 2)' to show only estimates and standard errors.

`signif.stars` (logical) If 'TRUE', prints significance stars for the coefficients. Default is 'getOption("show.signif.stars")'.

`eps.Pvalue` (numeric) Tolerance for small p-values. Default is '0.0001'.

`print.formula` (logical) If 'TRUE' (default), prints the model formula.

`print.fitinfo` (logical) If 'TRUE' (default), prints information about the estimation results.

`print.coefmat` (logical) If 'TRUE' (default), prints the coefficient table.

`print.call` (logical) If 'TRUE' (default), prints the call that generated the object.

`...` Additional arguments passed to `printCoefmat`.

Method `plot()`**:** Plot the estimation results, including coefficient convergence paths and model assessment diagnostics if available.

Usage:

```
iglm.object.generator$plot(
  stats = FALSE,
  trace = FALSE,
  model_assessment = FALSE
)
```

Arguments:

`stats` (logical) If 'TRUE', plot the observed vs. simulated statistics from model assessment. Default is 'FALSE'.

`trace` (logical) If 'TRUE', plot the coefficient convergence paths. Default is 'FALSE'.

`model_assessment` (logical) If 'TRUE', plot diagnostics from the model assessment (if already carried out). Default is 'FALSE'.

Method `gather()`**:** Gathers all components of the `iglm.object` into a single list for easy saving or inspection.

Usage:

```
iglm.object.generator$gather()
```

Returns: A list containing all key components of the `iglm.object`. This includes the formula, coefficients, sampler, control settings, preprocessing info, time taken for estimation, count statistics, results, and the underlying `iglm.data` data object.

Method `set_name()`**:** Set the name of the `iglm.object`.

Usage:

```
iglm.object.generator$set_name(name)
```

Arguments:

`name` (character) The name to assign to the object.

Returns: The name of the object as a character string.

Method `set_control()`**:** Set control parameters for model estimation.

Usage:

```
iglm.object.generator$set_control(control)
```

Arguments:

`control` A `control.iglm` object specifying new control settings.

Returns: Invisibly returns 'NULL'.

Method `save()`**:** Save the `iglm.object` to a file in RDS format.

Usage:

```
iglm.object.generator$save(file = NULL)
```

Arguments:

`file` (character) File path to save the object to (has to be a RDS object).

Returns: Invisibly returns 'NULL'.

Method `estimate()`**:** Estimate the model parameters using the specified control settings. Stores the results internally and updates the coefficient fields.

Usage:

```
iglm.object.generator$estimate()
```

Returns: If no preprocessing should be returned (as per control settings), this function returns a list containing detailed estimation results, invisibly. Includes final coefficients, variance-covariance matrix, convergence path, Fisher information, score vector, log-likelihood, and any simulations performed during estimation. Else, the function returns a list of the desired preprocessed data (as a data.frame) and needed time.

Method `summary()`**:** Provides a summary of the estimation results with the following columns: Estimate, SE, t-value, and Pr(>|t|). Requires the model to have been estimated first.

Usage:

```
iglm.object.generator$summary(digits = 2, ...)
```

Arguments:

`digits` (integer) Number of digits for rounding numeric output.

`...` Additional arguments passed to `printCoefmat`.

Returns: Prints the summary to the console and returns 'NULL' invisibly.

Method `simulate()`**:** Simulate networks from the fitted model or a specified model. Stores the simulations and/or summary statistics internally. The simulation is carried out using the internal MCMC sampler described in `simulate_iglm`.

Usage:

```
iglm.object.generator$simulate(
  only_stats = FALSE,
  display_progress = TRUE,
  offset_nonoverlap = 0
)
```

Arguments:

`only_stats` (logical) If 'TRUE', only calculate and store summary statistics for each simulation, discarding the network object itself. Default is 'FALSE'.

`display_progress` (logical) If 'TRUE' (default), display a progress bar during simulation.

`offset_nonoverlap` (numeric) Offset to apply for non-overlapping dyads during simulation (if applicable to the sampler). This option is useful if the sparsity of edges of units with non-overlapping neighborhoods is known. Default is 0.

Returns: A list containing the simulated networks ('samples', as a 'iglm.data.list' if 'only_stats = FALSE') and/or their summary statistics ('stats'), invisibly.

Method `predict()`**:** Calculates predicted values for the nodal covariates (`x`), the outcome variable (`y`), and the network structure (`z`). The function supports two prediction modes: *marginal* (based on Monte Carlo integration over simulated samples) and *conditional* (based on the analytical linear predictor and point estimates).

Usage:

```
iglm.object.generator$predict(
  variant = c("conditional", "marginal"),
  type = c("x", "y", "z")
)
```

Arguments:

`variant` A character string specifying the type of prediction to generate. Must be one of:

- `"marginal"`: Computes predictions by aggregating over the MCMC samples stored in the internal results. If samples do not exist, `self$simulate()` is triggered automatically.
- `"conditional"`: Computes predictions using the systematic component of the Generalized Linear Model (GLM). It calculates the linear predictor $\eta = X\beta$ (plus offset and degrees terms for the network) and applies the inverse link function $\mu = g^{-1}(\eta)$.

Defaults to `c("conditional", "marginal")`.

`type` A character vector indicating which components to predict. Options are:

- `"x"`: Nodal covariates.
- `"y"`: Nodal outcome variable.
- `"z"`: Dyadic network structure (interaction probabilities).

Defaults to `c("x", "y", "z")`.

Details: **Marginal Predictions:** When `variant = "marginal"`, the function approximates the expected value via Monte Carlo integration:

$$\hat{\mu} = \frac{1}{S} \sum_{s=1}^{S} h^{(s)}$$

where $h^{(s)}$ are the realized values from the s-th simulation sample (being either attribute x, y or the connections z). For the network z, this results in a marginal edge probability matrix averaged over all sampled networks.
Conditional Predictions: When `variant = "conditional"`, the function calculates the theoretical mean μ based on the estimated coefficients $\hat{\theta}$:

- For **Binomial** families: $\mu = (1 + \exp(-\eta))^{-1}$ (Logistic).
- For **Poisson** families: $\mu = \exp(\eta)$ (Exponential).
- For **Gaussian** families: $\mu = \eta$ (Identity).

For the network component z, the linear predictor includes dyadic covariates, degrees effects (sender/receiver variances), and structural offsets.

Returns: A list containing the requested predictions:

`x, y` A matrix or data frame where the first column is the actor ID and subsequent columns represent the predicted mean values.

`z` A data frame containing the edgelist with columns: `sender`, `receiver`, and `prediction` (probability or intensity).

The results are also invisibly stored in the internal state `private$.results`.

Method `set_coefficients()`**:** Manually set the model coefficients to new values. This is useful for sensitivity analyses or applying the model to different scenarios.

Usage:

```
iglm.object.generator$set_coefficients(coef, coef_degrees = NULL)
```

Arguments:

`coef` A numeric vector of new coefficient values for the non-degrees terms.

`coef_degrees` A numeric vector of new coefficient values for the degrees terms, if applicable. Must be provided if the model includes degrees effects.

Returns: The `iglm.object` itself, invisibly.

Method `get_samples()`**:** Retrieve the simulated networks stored in the object. Requires `simulate` or `estimate` to have been run first.

Usage:

```
iglm.object.generator$get_samples()
```

Returns: A list of `iglm.data` objects representing the simulated networks, invisibly. Returns an error if no samples are available.

Method `set_sampler()`**:** Replace the internal MCMC sampler with a new one. This is useful for changing the sampling scheme without redefining the entire model.

Usage:

```
iglm.object.generator$set_sampler(sampler)
```

Arguments:

`sampler` A `sampler.iglm` object. @return The `iglm.object` itself, invisibly.

Method `set_target()`**:** Replace the internal 'iglm.data' data object with a new one. This is useful for applying a fitted model to new observed data. Recalculates count statistics and re-validates the object.

Usage:

```
iglm.object.generator$set_target(x)
```

Arguments:

`x` A `iglm.data` " object containing the new observed data.

Returns: The `iglm.object` itself, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
iglm.object.generator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Fritz, C., Schweinberger, M. , Bhadra S., and D. R. Hunter (2025). A Regression Framework for Studying Relationships among Attributes under Network Interference. Journal of the American Statistical Association, to appear.

Stewart, J. R. and M. Schweinberger (2025). Pseudo-Likelihood-Based M-Estimation of Random Graphs with Dependent Edges and Parameter Vectors of Increasing Dimension. Annals of Statistics, to appear.

Schweinberger, M. and M. S. Handcock (2015). Local dependence in random graph models: characterization, properties, and statistical inference. Journal of the Royal Statistical Society, Series B (Statistical Methodology), 7, 647-676.

results *Constructor for the results R6 Object*

Description

Creates a new instance of the ‘results‘ R6 class. This class is designed to store various outputs from ‘iglm‘ model estimation and simulation. Users typically do not need to call this constructor directly; it is used internally by the ‘iglm_object‘.

Usage

```
results(size_coef, size_coef_degrees, file = NULL)
```

Arguments

size_coef	(integer) The number of non-degrees coefficients the object should be initialized to accommodate.
size_coef_degrees	(integer) The number of degrees coefficients the object should be initialized to accommodate.
file	(character or NULL) Optional file path to load a previously saved ‘results‘ object. If provided, the object will be initialized by loading from this file.

Value

An object of class ‘results‘ (and ‘R6‘), initialized with empty or NA structures appropriately sized based on the input dimensions.

results.generator *iglm Estimation and Simulation Results (R6 Class)*

Description

The ‘results‘ class stores estimation (‘$estimate()‘) and simulation (‘$simulate()‘) results.

This class is primarily intended for internal use within the ‘iglm‘ framework but provides structured access to the results via the active bindings of the main ‘iglm_object‘.

Active bindings

coefficients_path (‘matrix‘ or ‘NULL‘) Read-only. The path of all estimated coefficients across iterations.

samples (‘list‘ or ‘NULL‘) Read-only. A list of simulated ‘iglm.data‘ objects (class ‘iglm.data.list‘).

stats (‘matrix‘ or ‘NULL‘) Read-only. Matrix of summary statistics for simulated samples, which are an ‘mcmc‘ obect from ‘coda‘.

`var` ('matrix' or 'NULL') Read-only. Estimated variance-covariance matrix for non-degrees coefficients.

`fisher_degrees` ('matrix' or 'NULL') Read-only. Fisher information matrix for degrees coefficients.

`fisher_nondegrees` ('matrix' or 'NULL') Read-only. Fisher information matrix for non-degrees coefficients.

`score_degrees` ('numeric' or 'NULL') Read-only. Score vector for degrees coefficients.

`score_nondegrees` ('numeric' or 'NULL') Read-only. Score vector for non-degrees coefficients.

`llh` ('numeric' or 'NULL') Read-only. Vector of log-likelihood values recorded during estimation.

`model_assessment` ('list' or 'NULL') Read-only. Results from model assessment (goodness-of-fit).

`estimated` ('logical') Read-only. Flag indicating if estimation has been completed.

Methods

Public methods:

- `results.generator$new()`
- `results.generator$set_model_assessment()`
- `results.generator$set_prediction()`
- `results.generator$gather()`
- `results.generator$save()`
- `results.generator$resize()`
- `results.generator$update()`
- `results.generator$remove_samples()`
- `results.generator$plot()`
- `results.generator$print()`
- `results.generator$clone()`

Method `new()`**:** Creates a new 'results' object. Initializes internal fields, primarily setting up an empty matrix for the 'coefficients_path' based on the expected number of coefficients.

Usage:

```
results.generator$new(size_coef, size_coef_degrees, file)
```

Arguments:

`size_coef` (integer) The number of non-degrees (structural) coefficients in the model.

`size_coef_degrees` (integer) The number of degrees coefficients in the model (0 if none).

`file` (character or 'NULL') If provided, loads the sampler state from the specified .rds file instead of initializing from parameters.

Returns: A new 'results' object, initialized to hold results for a model with the specified dimensions.

Method `set_model_assessment()`**:** Stores the results object generated by a model assessment (goodness-of-fit) procedure within this 'results' container.

Usage:

```
results.generator$set_model_assessment(res)
```

Arguments:

`res` An object containing the model assessment results, expected to have the class 'iglm_model_assessment'.

Returns: The 'results' object itself ('self'), invisibly. Called for its side effect of storing the assessment results.

Method `set_prediction()`**:** Stores prediction results.

Usage:

```
results.generator$set_prediction(prediction)
```

Arguments:

`prediction` An object containing the prediction results (is a list of class 'iglm.prediction'.

Method `gather()`**:** Gathers the current state of the 'results' object into a list for saving or inspection. This includes all internal fields such as coefficient paths, samples, statistics, variance-covariance matrix, Fisher information, score vectors, log-likelihood values, model assessment results, and estimation status.

Usage:

```
results.generator$gather()
```

Returns: A list containing all the internal fields of the 'results' object.

Method `save()`**:** Saves the current state of the 'results' object to a specified file path in RDS format. This allows for persisting the results for later retrieval and analysis.

Usage:

```
results.generator$save(file)
```

Arguments:

`file` (character) The file path where the results state should be saved. Must be a valid character string. The file will be saved in RDS format, so it should end with '.rds'.

Returns: The 'results' object itself ('self'), invisibly.

Method `resize()`**:** Resizes the internal storage for the coefficient paths to accommodate a different number of coefficients. This is useful if the model structure changes and the results object needs to be reset.

Usage:

```
results.generator$resize(size_coef, size_coef_degrees)
```

Arguments:

`size_coef` (integer) The new number of non-degrees coefficients.

`size_coef_degrees` (integer) The new number of degrees coefficients. @return The 'results' object itself ('self'), invisibly.

Method `update()`**:** Updates the internal fields of the 'results' object with new outputs, typically after an estimation run ('$estimate()') or simulation run ('$simulate()'). Allows selectively updating components. Appends to 'coefficients_path' and 'llh' if called multiple times after estimation. Replaces 'samples' and 'stats'.

Usage:

```
results.generator$update(
  coefficients_path = NULL,
  samples = NULL,
  var = NULL,
  fisher_degrees = NULL,
  fisher_nondegrees = NULL,
  score_degrees = NULL,
  score_nondegrees = NULL,
  llh = NULL,
  stats = NULL,
  estimated = FALSE
)
```

Arguments:

`coefficients_path` (matrix) A matrix where rows represent iterations and columns represent all coefficients (non-degrees then degrees), showing their values during estimation. If provided, appends to any existing path.

`samples` (list) A list of simulated 'iglm.data' objects (class 'iglm.data.list'). If provided, replaces any existing samples.

`var` (matrix) The estimated variance-covariance matrix for the non-degrees coefficients. Replaces existing matrix.

`fisher_degrees` (matrix) The Fisher information matrix for degrees coefficients. Replaces existing matrix.

`fisher_nondegrees` (matrix) The Fisher information matrix for non-degrees coefficients. Replaces existing matrix.

`score_degrees` (numeric) The score vector for degrees coefficients. Replaces existing vector.

`score_nondegrees` (numeric) The score vector for non-degrees coefficients. Replaces existing vector.

`llh` (numeric) Log-likelihood value(s). If provided, appends to the existing vector of log-likelihoods.

`stats` (matrix) A matrix of summary statistics from simulations, where rows correspond to simulations and columns to statistics. Replaces or extends the existing matrix and will be turned into a mcmc object from the 'coda' package.

`estimated` (logical) A flag indicating whether these results come from a completed estimation run. Updates the internal status.

Returns: The 'results' object itself ('self'), invisibly. Called for its side effects.

Method `remove_samples()`**:** Clears the stored simulation samples ('.samples') and statistics ('.stats') from the object, resetting it to an empty list. This might be used to save memory or before running new simulations.

Usage:

```
results.generator$remove_samples()
```

Returns: The 'results' object itself ('self'), invisibly.

Method `plot()`**:** Generates diagnostic plots for the estimation results. Currently plots:

- The log-likelihood path across iterations.

- The convergence paths for degrees coefficients (if present).
- The convergence paths for non-degrees coefficients.

Optionally, can also trigger plotting of model assessment results if available.

Usage:

```
results.generator$plot(
  trace = FALSE,
  stats = FALSE,
  model_assessment = FALSE,
  ...
)
```

Arguments:

`trace` (logical) If 'TRUE' (default), plot the trace plots of the estimation (log-likelihood and coefficient paths). Requires model to be estimated.

`stats` (logical) If 'TRUE', plots the normalized statistics from simulations. Default is 'FALSE'.

`model_assessment` (logical) If 'TRUE', attempts to plot the results stored in the '.model_assessment' field. Requires model assessment to have been run and a suitable 'plot' method for 'iglm_model_assessment' objects to exist. Default is 'FALSE'.

`...` Additional fits with identical model_assessment terms are currently identified from this argument. The names of the arguments are shown as the legend in the model assessment plots.

Details: Requires estimation results ('private$.estimated == TRUE') to plot convergence diagnostics. Requires model assessment results for the model assessment plots.

Method `print()`**:** Prints a concise summary of the contents of the 'results' object, indicating whether various components (coefficients path, variance matrix, Fisher info, score, samples, stats, etc.) are available.

Usage:

```
results.generator$print(...)
```

Arguments:

`...` Additional arguments (currently ignored).

Returns: The 'results' object itself ('self'), invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
results.generator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`sampler.iglm` *Constructor for a iglm Sampler*

Description

Creates an object of class ‘sampler.iglm‘ (and ‘R6‘) which holds all parameters controlling the MCMC sampling process for ‘iglm‘ models. This includes global settings like the number of simulations and burn-in, as well as references to specific samplers for the network (‘z‘) and attribute (‘x‘, ‘y‘) components.

This function provides a convenient way to specify these settings before passing them to the ‘iglm‘ constructor or simulation functions.

Usage

```
sampler.iglm(
  sampler_x = NULL,
  sampler_y = NULL,
  sampler_z = NULL,
  n_simulation = 100,
  n_burn_in = 10,
  init_empty = TRUE,
  seed = NA,
  cluster = NULL,
  file = NULL
)
```

Arguments

`sampler_x`	An object of class ‘sampler.net.attr‘ (created by ‘sampler.net.attr()‘) specifying how to sample the ‘x_attribute‘. If ‘NULL‘ (default), default ‘sampler.net.attr()‘ settings are used.
`sampler_y`	An object of class ‘sampler.net.attr‘ specifying how to sample the ‘y_attribute‘. If ‘NULL‘ (default), default settings are used.
`sampler_z`	An object of class ‘sampler.net.attr‘ specifying how to sample the ‘z_network‘ ties *within* the defined neighborhood/overlap region. If ‘NULL‘ (default), default settings are used.
`n_simulation`	(integer) The number of independent samples to generate after the burn-in period. Default: 100. Must be non-negative.
`n_burn_in`	(integer) The number of MCMC iterations to discard at the start for burn-in. Default: 10. Must be non-negative.
`init_empty`	(logical) If ‘TRUE‘ (default), initialize the MCMC chain from an empty state.
`seed`	(integer or ‘NA‘) A single integer seed set once before sampling begins to ensure reproducibility. If ‘NA‘ (default), a random seed is generated automatically.
`cluster`	A parallel cluster object (e.g., from ‘parallel::makeCluster()‘) for parallel simulations. If ‘NULL‘ (default), simulations run sequentially.

`file` (character or 'NULL') If provided, loads the sampler state from the specified .rds file instead of initializing from parameters.

Value

An object of class 'sampler.iglm' (and 'R6').

See Also

'sampler.net.attr', 'iglm', 'control.iglm'

Examples

```
n_actor <- 50
sampler_new <- sampler.iglm(
  n_burn_in = 100, n_simulation = 10,
  seed = 42,
  sampler_x = sampler.net.attr(n_proposals = n_actor * 10),
  sampler_y = sampler.net.attr(n_proposals = n_actor * 10),
  sampler_z = sampler.net.attr(n_proposals = n_actor^2, tnt = TRUE),
  init_empty = FALSE
)
sampler_new
sampler_new$seed
sampler_new$set_n_simulation(100)
sampler_new$n_simulation
```

---

sampler.iglm.generator

iglm Sampler Settings (R6 Class)

Description

The 'sampler.iglm' class is an R6 container for specifying and storing the parameters that control the MCMC (Markov Chain Monte Carlo) sampling process used in `iglm` simulations and potentially during estimation. It includes settings for the number of simulations, burn-in period, initialization, and parallelization options. It also holds references to component samplers (`sampler.net.attr` objects) responsible for sampling individual parts (attributes x, y, network z).

Active bindings

`sampler_x` ('sampler_net_attr') The sampler configuration object for the x attribute.

`sampler_y` ('sampler_net_attr') The sampler configuration object for the y attribute.

`sampler_z` ('sampler_net_attr') The sampler configuration object for the z network (overlap region).

`n_simulation` ('integer') The number of configurations to simulate.

`n_burn_in` ('integer') The number of initial MCMC iterations to discard.

`init_empty` (‘logical‘) Whether to initialize simulations from an empty state.

`seed` (‘integer‘) Read-only. The random seed used for sampling.

`cluster` (‘cluster‘ object or ‘NULL‘) The parallel cluster object being used, or ‘NULL‘.

Methods

Public methods:

- `sampler.iglm.generator$new()`
- `sampler.iglm.generator$set_cluster()`
- `sampler.iglm.generator$deactive_cluster()`
- `sampler.iglm.generator$set_n_simulation()`
- `sampler.iglm.generator$set_n_burn_in()`
- `sampler.iglm.generator$set_init_empty()`
- `sampler.iglm.generator$set_x_sampler()`
- `sampler.iglm.generator$set_y_sampler()`
- `sampler.iglm.generator$set_z_sampler()`
- `sampler.iglm.generator$set_seed()`
- `sampler.iglm.generator$print()`
- `sampler.iglm.generator$gather()`
- `sampler.iglm.generator$save()`
- `sampler.iglm.generator$clone()`

Method `new()`**:** Create a new ‘sampler.iglm‘ object. Initializes all sampler settings, using defaults for component samplers (‘sampler.net.attr‘) if not provided, and validates inputs.

Usage:

```
sampler.iglm.generator$new(
  sampler_x = NULL,
  sampler_y = NULL,
  sampler_z = NULL,
  n_simulation = 100,
  n_burn_in = 10,
  init_empty = TRUE,
  seed = NA,
  cluster = NULL,
  file = NULL
)
```

Arguments:

`sampler_x` An object of class ‘sampler.net.attr‘ controlling sampling for the x attribute. If ‘NULL‘, defaults from ‘sampler.net.attr()‘ are used.

`sampler_y` An object of class ‘sampler.net.attr‘ controlling sampling for the y attribute. If ‘NULL‘, defaults from ‘sampler.net.attr()‘ are used.

`sampler_z` An object of class ‘sampler.net.attr‘ controlling sampling for the z network (within the defined neighborhood/overlap). If ‘NULL‘, defaults from ‘sampler.net.attr()‘ are used.

`n_simulation` (integer) The number of network/attribute configurations to simulate and store after the burn-in period. Default is 100. Must be non-negative.

`n_burn_in` (integer) The number of initial MCMC iterations to discard (burn-in) before starting to collect simulations. Default is 10. Must be non-negative.

`init_empty` (logical) If 'TRUE' (default), the MCMC chain is initialized from an empty state (e.g., empty network, attributes at mean). If 'FALSE', initialization might depend on the specific sampler implementation (e.g., starting from observed data).

`seed` (integer or 'NA') A single integer seed for the random number generator, set once before sampling begins. If 'NA' (default), a random seed is generated automatically.

`cluster` A parallel cluster object (e.g., from the 'parallel' package) to use for running simulations in parallel. If 'NULL' (default), simulations are run sequentially.

`file` (character or 'NULL') If provided, loads the sampler state from the specified .rds file instead of initializing from parameters.

Returns: A new 'sampler.iglm' object.

Method `set_cluster()`**:** Sets the parallel cluster object to be used for simulations.

Usage:

```
sampler.iglm.generator$set_cluster(cluster)
```

Arguments:

`cluster` A parallel cluster object from the 'parallel' package.

Method `deactive_cluster()`**:** Deactivates parallel processing for this sampler instance by setting the internal cluster object reference to 'NULL'.

Usage:

```
sampler.iglm.generator$deactive_cluster()
```

Returns: The 'sampler.iglm' object itself ('self'), invisibly.

Method `set_n_simulation()`**:** Sets the number of simulations to generate after burn-in.

Usage:

```
sampler.iglm.generator$set_n_simulation(n_simulation)
```

Arguments:

`n_simulation` (integer) The number of simulations to set.

Returns: None.

Method `set_n_burn_in()`**:** Sets the number of burn-in iterations.

Usage:

```
sampler.iglm.generator$set_n_burn_in(n_burn_in)
```

Arguments:

`n_burn_in` (integer) The number of burn-in iterations to set.

Returns: None.

Method `set_init_empty()`**:** Sets whether to initialize simulations from an empty state.

Usage:

```
sampler.iglm.generator$set_init_empty(init_empty)
```

Arguments:

`init_empty` (logical) 'TRUE' to initialize from empty, 'FALSE' otherwise.

Returns: None.

Method `set_x_sampler()`**:** Sets the sampler configuration for the x attribute.

Usage:

```
sampler.iglm.generator$set_x_sampler(sampler_x)
```

Arguments:

`sampler_x` An object of class 'sampler_net_attr'.

Returns: None.

Method `set_y_sampler()`**:** Sets the sampler configuration for the y attribute.

Usage:

```
sampler.iglm.generator$set_y_sampler(sampler_y)
```

Arguments:

`sampler_y` An object of class 'sampler_net_attr'.

Returns: None.

Method `set_z_sampler()`**:** Sets the sampler configuration for the z attribute.

Usage:

```
sampler.iglm.generator$set_z_sampler(sampler_z)
```

Arguments:

`sampler_z` An object of class 'sampler_net_attr'.

Returns: None.

Method `set_seed()`**:** Sets the random seed for this sampler.

Usage:

```
sampler.iglm.generator$set_seed(seed)
```

Arguments:

`seed` (integer) The random seed to set.

Returns: None.

Method `print()`**:** Prints a formatted summary of the sampler configuration to the console.

Usage:

```
sampler.iglm.generator$print(digits = 3, ...)
```

Arguments:

`digits` (integer) Number of digits for formatting numeric values. Default: 3.

`...` Additional arguments (currently ignored).

Returns: The 'sampler.iglm' object itself ('self'), invisibly.

Method `gather()`**:** Gathers all data from private fields into a list.

Usage:

```
sampler.iglm.generator$gather()
```

Returns: A list containing all information of the sampler.

Method `save()`**:** Save the object's complete state to a directory. This will save the main sampler's settings to a file named 'sampler.iglm_state.rds' within the specified directory, and will also call the 'save()' method for each nested sampler (.x, .y, .z), saving them into the same directory.

Usage:

```
sampler.iglm.generator$save(file)
```

Arguments:

`file` (character) The file to a directory where the state files will be saved. The directory will be created if it does not exist.

Returns: The object itself, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
sampler.iglm.generator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`sampler.net.attr` *Constructor for Single Component Sampler Settings*

Description

Creates an object of class 'sampler_net_attr' (and 'R6'). Specifies MCMC sampling parameters for one component (attribute or network) within the 'iglm' simulation framework. Used as input to 'sampler.iglm()'.

Usage

```
sampler.net.attr(n_proposals = 10000, file = NULL, tnt = TRUE)
```

Arguments

`n_proposals`	(integer) Number of MCMC proposals per sampling update. Default: 10000.
`file`	(character or 'NULL') If provided, loads state from an .rds file.
`tnt`	(logical) If 'TRUE' (default), use Tie-No-Tie sampling.

Value

An object of class 'sampler_net_attr' (and 'R6').

See Also

'sampler.iglm'

Examples

```
sampler_comp_default <- sampler.net.attr()
sampler_comp_default

sampler_comp_custom <- sampler.net.attr(n_proposals = 50000, tnt = FALSE)
sampler_comp_custom
```

`sampler.net.attr.generator`
Single Component Sampler Settings (R6 Class)

Description

The 'sampler_net_attr' class is a simple R6 container used within the 'sampler.iglm' class. It holds the MCMC sampling parameters for a single component of the 'iglm' model, such as one attribute (e.g., 'x_attribute') or a part of the network (e.g., 'z_network' within the overlap). It stores the number of proposals and the TNT flag. The random seed is managed centrally by the parent 'sampler.iglm' object.

Active bindings

`n_proposals` ('integer') Read-only. Number of MCMC proposals per step.

`tnt` ('logical') Read-only. Whether TNT sampling is used.

Methods

Public methods:

- `sampler.net.attr.generator$new()`
- `sampler.net.attr.generator$print()`
- `sampler.net.attr.generator$gather()`
- `sampler.net.attr.generator$set_n_proposals()`
- `sampler.net.attr.generator$set_tnt()`
- `sampler.net.attr.generator$save()`
- `sampler.net.attr.generator$clone()`

Method `new()`**:** Create a new 'sampler_net_attr' object.

Usage:

```
sampler.net.attr.generator$new(n_proposals = 10000, file = NULL, tnt = TRUE)
```

Arguments:

`n_proposals` (integer) The number of MCMC proposals (iterations) to perform for this specific component during each sampling step. Default is 10000. Must be a non-negative integer.

`file` (character or 'NULL') If provided, loads the sampler state from the specified .rds file instead of initializing from parameters.

`tnt` (logical) If 'TRUE' (default), use Tie-No-Tie sampling (only if used for networks).

Returns: A new ‘sampler_net_attr‘ object.

Method `print()`**:** Print a summary of the sampler settings for this component.

Usage:

```
sampler.net.attr.generator$print(indent = "  ")
```

Arguments:

`indent` (character) Indentation string. Default is " ".

Returns: The object itself, invisibly.

Method `gather()`**:** Gathers all data into a list.

Usage:

```
sampler.net.attr.generator$gather()
```

Returns: A list with ‘n_proposals‘ and ‘tnt‘.

Method `set_n_proposals()`**:** Sets the number of MCMC proposals.

Usage:

```
sampler.net.attr.generator$set_n_proposals(n_proposals)
```

Arguments:

`n_proposals` (integer) Number of proposals.

Method `set_tnt()`**:** Sets whether to use TNT sampling.

Usage:

```
sampler.net.attr.generator$set_tnt(tnt)
```

Arguments:

`tnt` (logical) ‘TRUE‘ to use TNT sampling.

Method `save()`**:** Save state to an .rds file.

Usage:

```
sampler.net.attr.generator$save(file)
```

Arguments:

`file` (character) File path.

Returns: The object itself, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
sampler.net.attr.generator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`simulate_iglm` *Simulate Responses and Connections*

Description

Simulate responses and connections.

Usage

```
simulate_iglm(
  formula,
  basis = NULL,
  coef,
  coef_degrees = NULL,
  sampler = NULL,
  only_stats = TRUE,
  display_progress = FALSE,
  offset_nonoverlap = 0,
  cluster = NULL,
  fix_x = FALSE,
  fix_z = FALSE
)
```

Arguments

`formula`	A model ‘formula‘ object. The left-hand side should be the name of a ‘iglm.data‘ object available in the calling environment. See `iglm-terms` for details on specifying the right-hand side terms.
`basis`	An optional ‘iglm.data‘ object to serve as the basis for the simulation. If provided, the simulation starts from the state defined in this object. If ‘NULL‘ (default), the initial state is taken from the ‘iglm.data‘ object referenced in the ‘formula‘.
`coef`	Numeric vector containing the coefficient values for the structural (non-degrees) terms defined in the ‘formula‘.
`coef_degrees`	Numeric vector specifying the degrees coefficient values (expansiveness/attractiveness). This is required **only if** the ‘formula‘ includes degrees terms. Its length must be ‘n_actor‘ (for undirected networks) or ‘2 * n_actor‘ (for directed networks), where ‘n_actor‘ is determined from the ‘iglm.data‘ object in the formula.
`sampler`	An object of class ‘sampler.iglm‘ (created by ‘sampler.iglm()‘) specifying the MCMC sampling parameters. This includes the number of simulations (‘n_simulation‘), burn-in iterations (‘n_burn_in‘), initialization settings (‘init_empty‘), and component sampler settings (‘sampler_x‘, ‘sampler_y‘, etc.). If ‘NULL‘ (default), default settings from ‘sampler.iglm()‘ are used.
`only_stats`	(logical). If `TRUE` (default, consistent with the usage signature), the function returns only the matrix of features calculated for each simulation. The full simulated `iglm.data` objects are discarded to minimize memory usage. If `FALSE`,

the complete simulated `iglm.data` objects are created and returned within the `samples` component of the output list.

`display_progress`
: Logical. If 'TRUE', progress messages or a progress bar (depending on the backend implementation) are displayed during simulation. Default is 'FALSE'.

`offset_nonoverlap`
: Numeric scalar value passed to the C++ simulator. This value is typically added to the linear predictor for dyads that are **not** part of the 'overlap' set defined in the 'iglm.data' object, potentially modifying tie probabilities outside the primary neighborhood. Default is '0'.

`cluster`
: Optional parallel cluster object created, for example, by "parallel::makeCluster". If provided and valid, the function performs a single burn-in simulation on the main R process, then distributes the remaining 'n_simulation' tasks across the cluster workers using "parallel::parLapply". The master seed is offset for each worker to ensure different random streams. If 'NULL' (default), all simulations are run sequentially in the main R process.

`fix_x`
: Logical. If 'TRUE', the simulation holds the 'x_attribute' fixed at its initial state (from the `iglm.data` object) and only simulates the 'y_attribute' and 'z_network'. If 'FALSE' (default), all components (x, y, z) are simulated according to the model and sampler settings.

`fix_z`
: Logical. If 'TRUE', the simulation holds the 'z_network' fixed at its initial state (from the `iglm.data` object). If 'FALSE' (default), the network component is simulated according to the model and sampler settings.

Details

Parallel Execution: When a 'cluster' object is provided, the simulation process is adapted:

1. A single simulation run (including burn-in specified by 'sampler$n_burn_in') is performed on the master node to obtain a starting state for the parallel chains.
2. The total number of requested simulations ('sampler$n_simulation') is divided among the cluster workers.
3. "parallel::parLapply" is used to run simulations on each worker. Each worker starts from the state obtained after the initial burn-in, performs **zero** additional burn-in ('n_burn_in = 0' passed to workers), and generates its assigned share of the simulations. Component sampler seeds are offset based on the worker ID to ensure pseudo-independent random number streams.
4. Results (simulated objects or statistics) from all workers are collected and combined.

This approach ensures that the initial burn-in phase happens only once, saving time.

Value

A list containing one or two components (depending on 'only_stats'):

'samples' If 'only_stats = FALSE', this is a list of length 'sampler$n_simulation' where each element is a 'iglm.data' object representing one simulated draw from the model. The list has the S3 class '"iglm.data.list"'. If 'only_stats = TRUE', this component is omitted.

'stats' A numeric matrix with 'sampler$n_simulation' rows and 'length(coef)' columns. Each row contains the features (corresponding to the model terms in 'formula') calculated for one simulation draw. Column names are set to match the term names.

Errors

The function stops with an error if:

- The length of 'coef' does not match the number of terms derived from 'formula'.
- 'formula_preprocess' fails.
- The 'sampler' object is not of class 'sampler.iglm'.
- The C++ backend 'xyz_simulate_cpp' encounters an error.
- Helper functions like 'XYZ_to_R' or 'is_cluster_active' are not found.

Warnings may be issued if default sampler settings are used.

See Also

`iglm` for creating the model object, `sampler.iglm` for creating the sampler object, `iglm.data` for the data object structure.

`state_twitter` *Twitter (X) data list for U.S. state legislators (10-state subset)*

Description

This data object is data derived from the Twitter (X) interactions between U.S. state legislators, which is a subset of the data analyzed in Fritz et al. (2025).' The data is filtered to include only legislators from 10 states (NY, CA, TX, FL, IL, PA, OH, GA, NC, MI) and is further subset to the largest connected component based on mention or retweet activity.

This object contains the main `iglm.data` object and 5 pre-computed dyadic covariates.

Usage

```
data(state_twitter)
```

Format

A `list` object containing 6 components. Let N be the number of legislators in the filtered 10-state subset.

iglm.data A `iglm.data` object (which is also a `list`) parameterized as follows:

- `x_attribute`: A binary numeric vector of length N. Value is `1` if the legislator's party is 'Republican', `0` otherwise.
- `y_attribute`: A Poisson numeric vector of length N. Represents the count of hatespeech incidents (`actors_data$number_hatespeech`) for each legislator.

- `z_network`: A directed edgelist (2-column matrix) of size `n_edges` x 2. A tie `(i, j)` exists if legislator `i` either mentioned or retweeted legislator `j`.
- `neighborhood`: A directed edgelist (2-column matrix). Represents the follower network, where a tie `(i, j)` exists if legislator `i` follows legislator `j`. Self-loops (diagonal) are included.

match_gender An N x N `matrix`. `matrix[i, j]` = 1 if legislator `i` and legislator `j` have the same gender, `0` otherwise.

match_race An N x N `matrix`. `matrix[i, j]` = 1 if legislator `i` and legislator `j` have the same race, `0` otherwise.

match_state An N x N `matrix`. `matrix[i, j]` = 1 if legislator `i` and legislator `j` are from the same state, `0` otherwise.

white_attribute A 1 x N `matrix` (a row vector). `matrix[1, i]` = 1 if legislator `i` is 'White', `0` otherwise.

gender_attribute A 1 x N `matrix` (a row vector). `matrix[1, i]` = 1 if legislator `i` is 'female', `0` otherwise.

References

Gopal, Kim, Nakka, Boehmke, Harden, Desmarais. The National Network of U.S. State Legislators on Twitter. Political Science Research & Methods, Forthcoming.

Kim, Nakka, Gopal, Desmarais,Mancinelli, Harden, Ko, and Boehmke (2022). Attention to the COVID-19 pandemic on Twitter: Partisan differences among U.S. state legislators. Legislative Studies Quarterly 47, 1023–1041.

Fritz, C., Schweinberger, M. , Bhadra S., and D. R. Hunter (2025). A Regression Framework for Studying Relationships among Attributes under Network Interference. Journal of the American Statistical Association, to appear.

`statistics` *Compute Statistics*

Description

Computes statistics.

Usage

```
statistics(formula)
```

Arguments

`formula`	A model 'formula' object. The left-hand side should be the name of a `iglm.data` object available in the calling environment. Alternatively, the left-hand side can be a `iglm.data.list` object to compute statistics for multiple `iglm.data` objects at once (is, e.g., the normal outcome of all simulations). See `iglm-terms` for details on specifying the right-hand side terms.

Value

A named numeric vector. Each element corresponds to a term in the 'formula', and its value is the calculated observed feature for that term based on the data in the `iglm.data` object. The names of the vector match the coefficient names derived from the formula terms.

Examples

```
# Create a iglm.data object
n_actor <- 10
neighborhood <- matrix(1, nrow = n_actor, ncol = n_actor)
type_x <- "binomial"
type_y <- "binomial"
x_attr_data <- rbinom(n_actor, 1, 0.5)
y_attr_data <- rbinom(n_actor, 1, 0.5)
z_net_data <- matrix(0, nrow = n_actor, ncol = n_actor)
object <- iglm.data(
  z_network = z_net_data, x_attribute = x_attr_data,
  y_attribute = y_attr_data, neighborhood = neighborhood,
  directed = FALSE, type_x = type_x, type_y = type_y
)
statistics(object ~ edges(mode = "local") + attribute_y + attribute_x)
```

Index