Package ‘CausalQueries’

April 26, 2024

Type Package

Title Make, Update, and Query Binary Causal Models

Version 1.1.1

Description Users can declare binary causal models, update beliefs about causal types given data and calculate arbitrary estimands. Model definition makes use of ‘dagitty’ functionality. Updating is implemented in ‘stan’. The approach used in ‘CausalQueries’ is a generalization of the ‘biqq’ models described in ```Mixing Methods: A Bayesian Approach``` (Humphreys and Jacobs, 2015, <DOI:10.1017/S0003055415000453>). The conceptual extension makes use of work on probabilistic causal models described in Pearl's Causality (Pearl, 2009, <DOI:10.1017/CBO9780511803161>).

BugReports https://github.com/integrated-inferences/CausalQueries/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Depends dplyr, methods, R (>= 3.4.0), Rcpp (>= 0.12.0)

Imports dagitty (>= 0.3-1), dirmult (>= 0.1.3-4), stats (>= 4.1.1), rlang (>= 0.2.0), rstan (>= 2.26.0), rstantools (>= 2.0.0), stringr (>= 1.4.0), ggdag (>= 0.2.4), latex2exp (>= 0.9.4), knitr (>= 1.45), ggplot2 (>= 3.3.5), lifecycle (>= 1.0.1)

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppArmadillo, RcppEigen (>= 0.3.3.3.0), rstan (>= 2.26.0), StanHeaders (>= 2.26.0)

Suggests testthat, rmarkdown, DeclareDesign, fabricatr, bayesplot, covr

SystemRequirements GNU make

Biarch true

VignetteBuilder knitr

NeedsCompilation yes
Author  Clara Bicalho [ctb],
Jasper Cooper [ctb],
Macartan Humphreys [aut] (<https://orcid.org/0000-0001-7029-2326>),
Till Tietz [aut, cre] (<https://orcid.org/0000-0002-2916-9059>),
Alan Jacobs [aut],
Merlin Heidemanns [ctb],
Lily Medina [aut] (<https://orcid.org/0009-0004-2423-524X>),
Julio Solis [ctb],
Georgiy Syunyaev [aut] (<https://orcid.org/0000-0002-4391-6313>)

Maintainer  Till Tietz <ttietz2014@gmail.com>

Repository  CRAN

Date/Publication  2024-04-26 09:00:07 UTC

R topics documented:

CausalQueries-package .................................................. 4
collapse_data .............................................................. 4
complements ..................................................................... 5
data_type_names ............................................................. 6
decreasing ......................................................................... 7
democracy_data ................................................................. 8
draw_causal_type .............................................................. 8
expand_data ..................................................................... 9
expand wildcard ................................................................ 10
find_rounding_threshold ..................................................... 10
get_all_data_types ............................................................ 11
get_ambiguities_matrix ...................................................... 12
get_event_probabilities ...................................................... 12
get_parameter_names ......................................................... 13
get_parents ....................................................................... 14
get_parmap ...................................................................... 14
get_query_types ............................................................... 15
get_type_prob ................................................................. 16
get_type_prob_multiple_c ................................................... 17
get_type_prob_multiple_c ................................................... 18
grab .................................................................................. 18
increasing ......................................................................... 20
institutions_data ............................................................... 21
interacts ........................................................................... 22
interpret_type ................................................................. 22
lipids_data ....................................................................... 23
make_data ....................................................................... 24
make_events ..................................................................... 26
make_model ..................................................................... 28
make_parameter_matrix ..................................................... 30
make_parmap .................................................................... 30
make_prior_distribution ..................................................... 31
R topics documented:

- non_decreasing .......................................................... 31
- non_increasing .......................................................... 32
- observe_data ............................................................... 33
- parameter_setting ......................................................... 34
- print.causal_model ....................................................... 36
- print.causal_types ....................................................... 37
- print.dag ................................................................. 37
- print.event_probabilities ................................................ 38
- print.model_query ........................................................ 38
- print.nodal_types ......................................................... 39
- print.nodes ............................................................... 39
- print.parameters ......................................................... 40
- print.parameters_df ...................................................... 40
- print.parameters_posterior ............................................. 41
- print.parameters_prior .................................................. 41
- print.parameter_mapping ................................................ 42
- print.parents_df .......................................................... 42
- print.posterior_event_probabilities ................................... 43
- print.stan_summary ....................................................... 43
- print.statement ........................................................... 44
- print.type_distribution .................................................. 44
- print.type_prior ........................................................... 45
- prior_setting ............................................................. 45
- query_distribution ....................................................... 48
- query_model .............................................................. 51
- realise_outcomes ........................................................ 53
- set_ambiguities_matrix .................................................. 54
- set_confound .............................................................. 55
- set_parameter_matrix .................................................... 56
- set_parmap ................................................................. 57
- set_prior_distribution ................................................... 58
- set_restrictions ........................................................... 58
- simulate_data ............................................................. 61
- substitutes ............................................................... 62
- summarise_distribution .................................................. 63
- summary.causal_model ................................................... 63
- te ................................................................. 64
- update_model ............................................................. 65

Index 67
CausalQueries-package 'CausalQueries'

Description

'CausalQueries' is a package that lets you generate binary causal models, update over models given data and calculate arbitrary causal queries. Model definition makes use of dagitty syntax. Updating is implemented in 'stan'.

collapse_data

Make compact data with data strategies

Description

Take a 'data.frame' and return compact 'data.frame' of event types and strategies.

Usage

collapse_data(
  data,  
  model, 
  drop_NA = TRUE, 
  drop_family = FALSE, 
  summary = FALSE
)

Arguments

data A data.frame. Data of nodes that can take three values: 0, 1, and NA. In long form as generated by make_events

model A causal_model. A model object generated by make_model.

drop_NA Logical. Whether to exclude strategy families that contain no observed data. Exceptionally if no data is provided, minimal data on data on first node is returned. Defaults to 'TRUE'.

drop_family Logical. Whether to remove column strategy from the output. Defaults to 'FALSE'.

summary Logical. Whether to return summary of the data. See details. Defaults to 'FALSE'.
complements

Value

A vector of data events

If `summary = TRUE` `collapse_data` returns a list containing the following components:

- **data_events**: A compact data.frame of event types and strategies.
- **observed_events**: A vector of character strings specifying the events observed in the data
- **unobserved_events**: A vector of character strings specifying the events not observed in the data

Examples

```r
model <- make_model('X -> Y')
df <- data.frame(X = c(0,1,NA), Y = c(0,0,1))
df %>% collapse_data(model)

collapse_data(df, model, drop_NA = FALSE)
collapse_data(df, model, drop_family = TRUE)
collapse_data(df, model, summary = TRUE)
data <- make_data(model, n = 0)
collapse_data(data, model)

model <- make_model('X -> Y') %>% set_restrictions('X[1]==1')
df <- make_data(model, n = 10)
df[1,1] <- ''
collapse_data(df, model)
data <- data.frame(X= 0:1)
collapse_data(data, model)
```

Complements

Make statement for complements

Description

Generate a statement for X1, X1 complement each other in the production of Y
Usage

complements(X1, X2, Y)

Arguments

X1 A character. The quoted name of the input node 1.
X2 A character. The quoted name of the input node 2.
Y A character. The quoted name of the outcome node.

Value

A character statement of class statement

See Also

Other statements: decreasing(), increasing(), interacts(), non_decreasing(), non_increasing(), substitutes(), te()

Examples

complements('A', 'B', 'W')

data_type_names

Data type names

Description

Provides names to data types

Usage

data_type_names(model, data)

Arguments

model A causal_model. A model object generated by make_model.
data A data.frame. Data of nodes that can take three values: 0, 1, and NA. In long form as generated by make_events

Value

A vector of strings of data types
Examples

```r
model <- make_model('X -> Y')
data <- make_data(model, n = 2)
data_type_names(model, data)
```

---

**decreasing**  
Make monotonicity statement (negative)

---

**Description**

Generate a statement for Y monotonic (decreasing) in X

**Usage**

`decreasing(X, Y)`

**Arguments**

- *X*: A character. The quoted name of the input node
- *Y*: A character. The quoted name of the outcome node

**Value**

A character statement of class `statement`

**See Also**

Other statements: `complements()`, `increasing()`, `interacts()`, `non_decreasing()`, `non_increasing()`, `substitutes()`, `te()`

**Examples**

```r
decreasing('A', 'B')
```
democracy_data

*Development and Democratization: Data for replication of analysis in *Integrated Inferences*

**Description**

A dataset containing information on inequality, democracy, mobilization, and international pressure. Made by devtools::use_data(democracy_data, CausalQueries)

**Usage**

democracy_data

**Format**

A data frame with 84 rows and 5 nodes:

- **Case**
- **D** Democracy
- **I** Inequality
- **P** International Pressure
- **M** Mobilization

**Source**


draw_causal_type

*Draw a single causal type given a parameter vector*

**Description**

Output is a parameter dataframe recording both parameters (case level priors) and the case level causal type.

**Usage**

draw_causal_type(model, ...)

**Arguments**

- **model** A causal_model. A model object generated by make_model.
- **...** Arguments passed to `set_parameters`
### Examples

```r
# Simple draw using model's parameter vector
make_model("X -> M -> Y") %>%
draw_causal_type(.)

# Draw parameters from priors and draw type from parameters
make_model("X -> M -> Y") %>%
draw_causal_type(. , param_type = "prior_draw")

# Draw type given specified parameters
make_model("X -> M -> Y") %>%
draw_causal_type(. , parameters = 1:10)

# Define a causal type and reveal data
model <- make_model("X -> Y; X <-> Y")
type <- model %>% draw_causal_type()
make_data(model, parameters = type$causal_type)
```

---

**expand_data**

Expand compact data object to data frame

**Description**

Expand compact data object to data frame

**Usage**

```r
expand_data(data_events = NULL, model)
```

**Arguments**

- `data_events` A `data.frame`. It must be compatible with nodes in `model`. The default columns are `event`, `strategy` and `count`.
- `model` A `causal_model`. A model object generated by `make_model`.

**Value**

A `data.frame` with rows as data observation

**Examples**

```r
model <- make_model('X->M->Y')
make_events(model, n = 5) %>%
  expand_data(model)
make_events(model, n = 0) %>%
  expand_data(model)
```
**find_rounding_threshold**

**Description**

helper to find rounding thresholds for print methods

**Example**

```r
exp = 'Y[X=1]' + 'M=' + '.

expand_wildcard(exp)
```

**Description**

helper to find rounding thresholds for print methods

**Example**

```r
exp = 'Y[X=1]' + 'M=' + '.

expand_wildcard(exp)
```
get_all_data_types

Usage

find_rounding_threshold(x)

Arguments

x

An object for rounding

get_all_data_types

Get all data types

Description

Creates dataframe with all data types (including NA types) that are possible from a model.

Usage

get_all_data_types(
  model,
  complete_data = FALSE,
  possible_data = FALSE,
  given = NULL
)

Arguments

model
A causal_model. A model object generated by make_model.

complete_data
Logical. If ‘TRUE’ returns only complete data types (no NAs). Defaults to ‘FALSE’.

possible_data
Logical. If ‘TRUE’ returns only complete data types (no NAs) that are *possible* given model restrictions. Note that in principle an intervention could make observationally impossible data types arise. Defaults to ‘FALSE’.

given
A character. A quoted statement that evaluates to logical. Data conditional on specific values.

Value

A data.frame with all data types (including NA types) that are possible from a model.

Examples

make_model(‘X -> Y’) |> get_all_data_types()
model <- make_model(‘X -> Y’) %>%
  set_restrictions(labels = list(Y = ‘00’), keep = TRUE)
get_all_data_types(model)
get_all_data_types(model, complete_data = TRUE)
get_all_data_types(model, possible_data = TRUE)
get_all_data_types(model, given = 'X==1')
get_all_data_types(model, given = 'X==1 & Y==1')

get_ambiguities_matrix

Get ambiguities matrix

Description

Return ambiguities matrix if it exists; otherwise calculate it assuming no confounding. The ambiguities matrix maps from causal types into data types.

Usage

get_ambiguities_matrix(model)

Arguments

model A causal_model. A model object generated by make_model.

Value

A data.frame. Causal types (rows) corresponding to possible data realizations (columns).

get_event_probabilities

Draw event probabilities

Description

'get_event_probabilities' draws event probability vector 'w' given a single realization of parameters

Usage

get_event_probabilities(
    model,
    parameters = NULL,
    A = NULL,
    P = NULL,
    given = NULL
)
get_parameter_names

Arguments

model A causal_model. A model object generated by make_model.
parameters A vector of real numbers in [0,1]. Values of parameters to specify (optional). By default, parameters is drawn from model$parameters_df.
A A data.frame. Ambiguity matrix. Not required but may be provided to avoid repeated computation for simulations.
P A data.frame. Parameter matrix. Not required but may be provided to avoid repeated computation for simulations.
given A string specifying known values on nodes, e.g. "X==1 & Y==1"

Value

An array of event probabilities

Examples

model <- make_model('X -> Y')
get_event_probabilities(model = model)
get_event_probabilities(model = model, given = "X==1")
get_event_probabilities(model = model, parameters = rep(1, 6))
get_event_probabilities(model = model, parameters = 1:6)

Description

Parameter names taken from P matrix or model if no P matrix provided

Usage

get_parameter_names(model, include_paramset = TRUE)

Arguments

model A causal_model. A model object generated by make_model.
include_paramset Logical. Whether to include the param set prefix as part of the name.

Value

A character vector with the names of the parameters in the model
**get_parents**

*Get list of parents of all nodes in a model*

**Description**

Get list of parents of all nodes in a model.

**Usage**

```r
get_parents(model)
```

**Arguments**

- `model`: A `causal_model`. A model object generated by `make_model`.

**Value**

A list of parents in a DAG.

---

**get_parmap**

*Get parmap: a matrix mapping from parameters to data types*

**Description**

Gets parmap from a model, or generates if not available.

**Usage**

```r
get_parmap(model, A = NULL, P = NULL)
```

**Arguments**

- `model`: A `causal_model`. A model object generated by `make_model`.
- `A`: A `data.frame`. Ambiguity matrix. Not required but may be provided to avoid repeated computation for simulations.
- `P`: A `data.frame`. Parameter matrix. Not required but may be provided to avoid repeated computation for simulations.

**Value**

A matrix.
get_query_types

Look up query types

Description

Find which nodal or causal types are satisfied by a query.

Usage

get_query_types(model, query, map = "causal_type", join_by = "|")

Arguments

model A causal_model. A model object generated by make_model.
query A character string. An expression defining nodal types to interrogate realise_outcomes. An expression of the form "Y[X=1]" asks for the value of Y when X is set to 1
map Types in query. Either nodal_type or causal_type. Default is causal_type.
join_by A logical operator. Used to connect causal statements: AND ('&') or OR ('|'). Defaults to '|'.

Value

A list containing some of the following elements

types A named vector with logical values indicating whether a nodal_type or a causal_type satisfy 'query'
query A character string as specified by the user
expanded_query A character string with the expanded query. Only differs from 'query' if this contains wildcard '.'
evaluated_nodes Value that the nodes take given a query
node A character string of the node whose nodal types are being queried
type_list List of causal types satisfied by a query

Examples

model <- make_model('X -> M -> Y; X->Y')
query <- '(Y[X=0] > Y[X=1])'

get_query_types(model, query, map="nodal_type")
get_query_types(model, query, map="causal_type")
get_query_types(model, query)

# Examples with map = "nodal_type"
query <- '(Y[X=0, M = .] > Y[X=1, M = 0])'
get_query_types(model, query, map="nodal_type")

query <- '(Y[] == 1)'
get_query_types(model, query, map="nodal_type")
get_query_types(model, query, map="nodal_type", join_by = '&')

# Root nodes specified with []
get_query_types(model, '(X[] == 1)', map="nodal_type")

query <- '(M[X=1] == M[X=0])'
get_query_types(model, query, map="nodal_type")

# Nested do operations
get_query_types(
  model = make_model('A -> B -> C -> D'),
  query = '(D[C=C[B=B[A=1]], A=0] > D[C=C[B=B[A=0]], A=0])'
)

# Helpers
model <- make_model('M->Y; X->Y')
query <- complements('X', 'M', 'Y')
get_query_types(model, query, map="nodal_type")

# Examples with map = "causal_type"
model <- make_model('X -> M -> Y; X->Y')
query <- 'Y[M=M[X=0], X=1]==1'
get_query_types(model, query, map="causal_type")

query <- '(Y[X = 1, M = 1] > Y[X = 0, M = 1]) &
          (Y[X = 1, M = 0] > Y[X = 0, M = 0])'
get_query_types(model, query, "causal_type")

query <- 'Y[X=1] == Y[X=0]'
get_query_types(model, query, "causal_type")

query <- '(X == 1) & (M==1) & (Y ==1) & (Y[X=0] ==1)'
get_query_types(model, query, "causal_type")

query <- '(Y[X = .]==1)'
get_query_types(model, query, "causal_type")

---

**get_type_prob**

*Get type probabilities*

**Description**

Gets probability of vector of causal types given a single realization of parameters, possibly drawn from model priors.
get_type_prob_c

Usage

get_type_prob(model, P = NULL, parameters = NULL)

Arguments

model A causal_model. A model object generated by make_model.

P A data.frame. Parameter matrix. Not required but may be provided to avoid repeated computation for simulations.

parameters A vector of real numbers in [0,1]. Values of parameters to specify (optional). By default, parameters is drawn from model$parameters_df.

Details

By default, parameters is drawn from ‘using’ argument (either from priors, posteriors, or from model$parameters)

Value

A vector with probabilities of vector of causal types

generate one draw from type probability distribution for each type in P

get_type_prob_c(P, parameters)

Arguments

P parameter_matrix of parameters and causal types

parameters, priors or posteriors

Value

draw from type distribution for each type in P
get_type_prob_multiple_c

generates n draws from type probability distribution for each type in P

Description

generates n draws from type probability distribution for each type in P

Usage

get_type_prob_multiple_c(params, P)

Arguments

params parameters, priors or posteriors
P parameter_matrix of parameters and causal types

Value
draws from type distribution for each type in P

grab

Grab

Description

Returns specified elements from a causal_model. Users can use grab to extract model’s components or objects implied by the model structure including nodal types, causal types, parameter priors, parameter posteriors, type priors, type posteriors, and other relevant elements. See argument object for other options.

Usage

grab(model, object = NULL, ...)

Arguments

model A causal_model. A model object generated by make_model.
object A character string specifying the component to retrieve. Available options are:

- "causal_statement" a character. Statement describing causal relations using dagitty syntax.
- "dag" A data frame with columns 'parent' and 'children' indicating how nodes relate to each other.
- "nodes" A list containing the nodes in the model,
• "parents_df" a table listing nodes, whether they are root nodes or not, and the number and names of parents they have,
• "parameters_df" a data frame containing parameter information,
• "causal_types" a data frame listing causal types and the nodal types that produce them,
• "causal_types_interpretation" a key to interpreting types; see "$interpret_type" for options,
• "nodal_types" a list with the nodal types of the model,
• "data_types" a list with the all data types consistent with the model; for options see "?get_all_data_types",
• "event_probabilities" a vector of data (event) probabilities given a parameter vector; for options see "?get_event_probabilities",
• "ambiguities_matrix" a matrix mapping from causal types into data types,
• "parameters" a vector of 'true' parameters,
• "parameter_names" a vector of names of parameters,
• "parameter_mapping" a matrix mapping from parameters into data types,
• "parameter_matrix" a matrix mapping from parameters into causal types,
• "prior_hyperparameters" a vector of alpha values used to parameterize Dirichlet prior distributions; optionally provide node names to reduce output "grab(prior_hyperparameters, c('M', 'Y'))"
• "prior_distribution" a data frame of the parameter prior distribution,
• "posterior_distribution" a data frame of the parameter posterior distribution,
• "posterior_event_probabilities" a sample of data (event) probabilities from the posterior,
• "stan_objects" stan_objects is a list of Stan outputs that can include the stanfit object, the data that was used, and distributions over causal types and event probabilities.
• "data" the data that was provided to update the model,
• "stan_fit" the stanfit object generated by Stan,
• "stan_summary" a summary of the stanfit object generated by Stan,
• "type_prior" a matrix of type probabilities using priors,
• "type_distribution" a matrix of type probabilities using posteriors,

Other arguments passed to helper "get_*" functions.

Value

Objects from a causal_model as specified.

Examples

```
model <-
  make_model('X -> Y') |>
  update_model(
    keep_event_probabilities = TRUE,
```
increasing

Make monotonicity statement (positive)

Description

Generate a statement for Y monotonic (increasing) in X

Usage

increasing(X, Y)
Arguments

- **X**: A character. The quoted name of the input node
- **Y**: A character. The quoted name of the outcome node

Value

- A character statement of class `statement`

See Also

- Other statements: `complements()`, `decreasing()`, `interacts()`, `non_decreasing()`, `non_increasing()`, `substitutes()`, `te()`

Examples

```r
increasing('A', 'B')
```

---

**institutions_data**

*Institutions and growth: Data for replication of analysis in *Integrated Inferences***

Description


Usage

```r
institutions_data
```

Format

A data frame with 79 rows and 5 columns:

- **Y**: Income (GDP PPP 1995), dichotomized
- **R**: Institutions, (based on Kaufmann, Kraay, and Zoido-Lobaton (2002)) dichotomized
- **D**: Distance from the equator (in degrees), dichotomized
- **M**: Settler mortality (from Acemoglu, Johnson, and Robinson), dichotomized
- **country**: Country

Source

https://drodrik.scholar.harvard.edu/publications/institutions-rule-primacy-institutions-over-geography-and-integration
interacts

*Make statement for any interaction*

**Description**

Generate a statement for X1, X1 interact in the production of Y

**Usage**

`interacts(X1, X2, Y)`

**Arguments**

- **X1**
  - A character. The quoted name of the input node 1.
- **X2**
  - A character. The quoted name of the input node 2.
- **Y**
  - A character. The quoted name of the outcome node.

**Value**

A character statement of class `statement`

**See Also**

Other statements: `complements()`, `decreasing()`, `increasing()`, `non_decreasing()`, `non_increasing()`, `substitutes()`, `te()`

**Examples**

```r
teracts('A', 'B', 'W')
get_query_types(model = make_model('X-> Y <- W'),
  query = interacts('X', 'W', 'Y'), map = "causal_type")
```

interpret_type

*Interpret or find position in nodal type*

**Description**

Interprets the position of one or more digits (specified by `position`) in a nodal type. Alternatively returns nodal type digit positions that correspond to one or more given condition.

**Usage**

`interpret_type(model, condition = NULL, position = NULL, nodes = NULL)`
Arguments

model A causal_model. A model object generated by make_model.
condition A vector of characters. Strings specifying the child node, followed by '|' (given) and the values of its parent nodes in model.
position A named list of integers. The name is the name of the child node in model, and its value a vector of digit positions in that node’s nodal type to be interpreted. See ‘Details’.
nodes A vector of names of nodes. Can be used to limit interpretation to selected nodes.

Details

A node for a child node X with k parents has a nodal type represented by X followed by \(2^k\) digits. Argument position allows user to interpret the meaning of one or more digit positions in any nodal type. For example position = list(X = 1:3) will return the interpretation of the first three digits in causal types for X. Argument condition allows users to query the digit position in the nodal type by providing instead the values of the parent nodes of a given child. For example, condition = \('X \mid Z=0 & R=1'\) returns the digit position that corresponds to values X takes when Z = 0 and R = 1.

Value

A named list with interpretation of positions of the digits in a nodal type

Examples

model <- make_model('R -> X; Z -> X; X -> Y')
#Return interpretation of all digit positions of all nodes
interpret_type(model)
#Example using digit position
interpret_type(model, position = list(X = c(3,4), Y = 1))
interpret_type(model, position = list(R = 1))
#Example using condition
interpret_type(model, condition = c('X \mid Z=0 & R=1', 'X \mid Z=0 & R=0'))
# Example using node names
interpret_type(model, nodes = c("Y", "R"))

lipids_data Lipids: Data for Chickering and Pearl replication

Description

Usage
lipids_data

Format
A data frame with 8 rows and 3 columns:

- **event**  The data type
- **strategy**  For which nodes is data available
- **count**  Number of units with this data type

Source

---

**make_data**  Make data

Description
Make data

Usage
make_data(
  model,
  n = NULL,
  parameters = NULL,
  param_type = NULL,
  nodes = NULL,
  n_steps = NULL,
  probs = NULL,
  subsets = TRUE,
  complete_data = NULL,
  given = NULL,
  verbose = TRUE,
  ...
)

Arguments
- **model**  A `causal_model`. A model object generated by `make_model`.
- **n**  Non negative integer. Number of observations. If not provided it is inferred from the largest `n_step`.
- **parameters**  A vector of real numbers in [0,1]. Values of parameters to specify (optional). By default, parameters is drawn from `model$parameters_df`. 
param_type A character. String specifying type of parameters to make ("flat", "prior_mean", "posterior_mean", "prior_draw", "posterior_draw", "define"). With param_type set to define use arguments to be passed to make_priors; otherwise flat sets equal probabilities on each nodal type in each parameter set; prior_mean, prior_draw, posterior_mean, posterior_draw take parameters as the means or as draws from the prior or posterior.

nodes A list. Which nodes to be observed at each step. If NULL all nodes are observed.

n_steps A list. Number of observations to be observed at each step

probs A list. Observation probabilities at each step

subsets A list. Strata within which observations are to be observed at each step. TRUE for all, otherwise an expression that evaluates to a logical condition.

complete_data A data.frame. Dataset with complete observations. Optional.

given A string specifying known values on nodes, e.g. "X==1 & Y==1"

verbose Logical. If TRUE prints step schedule.

... additional arguments that can be passed to link{make_parameters}

Details

Note that default behavior is not to take account of whether a node has already been observed when determining whether to select or not. One can however specifically request observation of nodes that have not been previously observed.

Value

A data.frame with simulated data.

Examples

# Simple draws
model <- make_model("X -> M -> Y")
make_data(model)
make_data(model, n = 3, nodes = c("X","Y"))
make_data(model, n = 3, param_type = "prior_draw")
make_data(model, n = 10, param_type = "define", parameters = 0:9)

# Data Strategies
# A strategy in which X, Y are observed for sure and M is observed
# with 50% probability for X=1, Y=0 cases

model <- make_model("X -> M -> Y")
make_data(
  model,
  n = 8,
  nodes = list(c("X", "Y"), "M"),
  probs = list(1, .5),
  subsets = list(TRUE, "X==1 & Y==0"))
# n not provided but inferred from largest n_step (not from sum of n_steps)
make_data(
  model,
  nodes = list(c("X", "Y"), "M"),
  n_steps = list(5, 2))

# Wide then deep
make_data(
  model,
  n = 8,
  nodes = list(c("X", "Y"), "M"),
  subsets = list(TRUE, !is.na(X) & !is.na(Y)),
  n_steps = list(6, 2))

make_data(
  model,
  n = 8,
  nodes = list(c("X", "Y"), c("X", "M")),
  subsets = list(TRUE, is.na(X)),
  n_steps = list(3, 2))

# Example with probabilities at each step
make_data(
  model,
  n = 8,
  nodes = list(c("X", "Y"), c("X", "M")),
  subsets = list(TRUE, is.na(X)),
  probs = list(.5, .2))

# Example with given data
make_data(model, given = "X==1 & Y==1", n = 5)

---

**make_events**  
Make data in compact form

**Description**

Draw n events given event probabilities. Draws full data only. For incomplete data see `make_data`.

**Usage**

```r
make_events(
  model,
  n = 1,
  w = NULL,
  P = NULL,
  A = NULL,
```
make_events

parameters = NULL,
param_type = NULL,
include_strategy = FALSE,
...
)

Arguments

model A causal_model. A model object generated by make_model.

n An integer. Number of observations.

w A numeric matrix. A ‘n_parameters x 1’ matrix of event probabilities with named rows.

P A data.frame. Parameter matrix. Not required but may be provided to avoid repeated computation for simulations.

A A data.frame. Ambiguity matrix. Not required but may be provided to avoid repeated computation for simulations.

parameters A vector of real numbers in [0,1]. Values of parameters to specify (optional). By default, parameters is drawn from model$parameters df.

param_type A character. String specifying type of parameters to make 'flat', 'prior_mean', 'posterior_mean', 'prior_draw', 'posterior_draw', 'define. With param_type set to define use arguments to be passed to make_priors; otherwise flat sets equal probabilities on each nodal type in each parameter set; prior_mean, prior_draw, posterior_mean, posterior_draw take parameters as the means or as draws from the prior or posterior.

include_strategy Logical. Whether to include a 'strategy' vector. Defaults to FALSE. Strategy vector does not vary with full data but expected by some functions.

... Arguments to be passed to make_priors if param_type == define

Value

A data.frame of events

Examples

model <- make_model('X -> Y')
make_events(model = model)
make_events(model = model, param_type = 'prior_draw')
make_events(model = model, include_strategy = TRUE)
Description

`make_model` uses `dagitty` syntax and functionality to specify nodes and edges of a graph. Implied causal types are calculated and default priors are provided under the assumption of no confounding. Models can be updated with specification of a parameter matrix, $P$, by providing restrictions on causal types, and/or by providing informative priors on parameters. The default setting for a causal model have flat (uniform) priors and parameters putting equal weight on each parameter within each parameter set. These can be adjust with `set_priors` and `set_parameters`.

Usage

```r
make_model(statement, add_causal_types = TRUE, nodal_types = NULL)
```

Arguments

- `statement`: A character. Statement describing causal relations using `dagitty` syntax. Only directed relations are permitted. For instance "X -> Y" or "X1 -> Y <- X2; X1 -> X2".
- `add_causal_types`: Logical. Whether to create and attach causal types to model. Defaults to `TRUE`.
- `nodal_types`: List of nodal types associated with model nodes.

Value

An object of class `causal_model`.

An object of class "causal_model" is a list containing at least the following components:

- `statement`: A character vector of the statement that defines the model.
- `dag`: A `data.frame` with columns 'parent' and 'children' indicating how nodes relate to each other.
- `nodes`: A named list with the nodes in the model.
- `parents_df`: A `data.frame` listing nodes, whether they are root nodes or not, and the number of parents they have.
- `nodal_types`: Optional: A named list with the nodal types in the model. List should be ordered according to the causal ordering of nodes. If NULL nodal types are generated. If FALSE, a parameters data frame is not generated.
- `parameters_df`: A `data.frame` with descriptive information of the parameters in the model.
- `causal_types`: A `data.frame` listing causal types and the nodal types that produce them.

See Also

`summary.causal_model` provides summary method for output objects of class `causal_model`.
Examples

```r
make_model(statement = "X -> Y")
modelXKY <- make_model("X -> K -> Y; X -> Y")

# Example where cyclicaly dag attempted
## Not run:
modelXKX <- make_model("X -> K -> X")

## End(Not run)

# Examples with confounding
model <- make_model("X->Y; X <-> Y")
dim(model$P)
model <- make_model("Y2 <- X -> Y1; X <-> Y1; X <-> Y2")
dim(model$P)
model <- make_model("X1 -> Y <- X2; X1 <-> Y; X2 <-> Y")
dim(model$P)
model$parameters_df

# A single node graph is also possible
model <- make_model("X")

# Unconnected nodes not allowed
## Not run:
model <- make_model("X <-> Y")
## End(Not run)

nodal_types <-
list(
  A = c("0", "1"),
  B = c("0", "1"),
  C = c("0", "1"),
  D = c("0", "1"),
  E = c("0", "1"),
  Y = c(
    "00000000000000000000000000000000",
    "01010101010101010101010101010101",
    "00110011001100110011001100110011",
    "00001111000011110000111100001111",
    "00000000111111110000000011111111",
    "00000000000000000111111111111111",
    "11111111111111111111111111111111"
  )
)
make_model("A -> Y; B ->Y; C->Y; D->Y; E->Y", 
nodal_types = nodal_types)$parameters_df

nodal_types = list(Y = c("01", "10"), Z = c("0", "1"))
make_model("Z -> Y", nodal_types = nodal_types)$parameters_df
make_model("Z -> Y", nodal_types = FALSE)$parents_df
```
### make_parameter_matrix

*Make parameter matrix*

**Description**

Calculate parameter matrix assuming no confounding. The parameter matrix maps from parameters into causal types. In models without confounding parameters correspond to nodal types.

**Usage**

```r
make_parameter_matrix(model)
```

**Arguments**

- `model`: A `causal_model`. A model object generated by `make_model`.

**Value**

A `data.frame`, the parameter matrix, mapping from parameters to causal types.

### make_parmap

*Make parmap: a matrix mapping from parameters to data types*

**Description**

Generates a matrix with a row per parameter and a column per data type.

**Usage**

```r
make_parmap(model, A = NULL, P = NULL)
```

**Arguments**

- `model`: A `causal_model`. A model object generated by `make_model`.
- `A`: A `data.frame`. Ambiguity matrix. Not required but may be provided to avoid repeated computation for simulations.
- `P`: A `data.frame`. Parameter matrix. Not required but may be provided to avoid repeated computation for simulations.

**Value**

A matrix
make_prior_distribution

*Make a prior distribution from priors*

**Description**
Create a `n_param`x `n_draws` database of possible lambda draws to be attached to the model.

**Usage**
```r
code_block
make_prior_distribution(model, n_draws = 4000)
```

**Arguments**
- `model` : A causal_model. A model object generated by `make_model`.
- `n_draws` : A scalar. Number of draws.

**Value**
A `data.frame` with dimension `n_param`x `n_draws` of possible lambda draws

**See Also**
Other prior_distribution: `get_prior_distribution()`, `set_prior_distribution()`

**Examples**
```r
code_block
make_model('X -> Y') %>% make_prior_distribution(n_draws = 5)
```

---

**non_decreasing**

*Make monotonicity statement (non negative)*

**Description**
Generate a statement for Y weakly monotonic (increasing) in X

**Usage**
```r
code_block
non_decreasing(X, Y)
```

**Arguments**
- `X` : A character. The quoted name of the input node
- `Y` : A character. The quoted name of the outcome node
non_increasing

Value
A character statement of class statement

See Also
Other statements: `complements()`, `decreasing()`, `increasing()`, `interacts()`, `non_increasing()`, `substitutes()`.

Examples
```r
non_increasing('A', 'B')
```

Description
Generate a statement for Y weakly monotonic (not increasing) in X

Usage
```r
non_increasing(X, Y)
```

Arguments
- **X**: A character. The quoted name of the input node
- **Y**: A character. The quoted name of the outcome node

Value
A character statement of class statement

See Also
Other statements: `complements()`, `decreasing()`, `increasing()`, `interacts()`, `non_decreasing()`, `substitutes()`, `te()`

Examples
```r
non_increasing('A', 'B')
```
**observe_data**  
*Observe data, given a strategy*

**Description**
Observe data, given a strategy

**Usage**
```r
observe_data(
  complete_data,
  observed = NULL,
  nodes_to_observe = NULL,
  prob = 1,
  m = NULL,
  subset = TRUE
)
```

**Arguments**
- `complete_data` A `data.frame`. Data observed and unobserved.
- `observed` A `data.frame`. Data observed.
- `nodes_to_observe` A list. Nodes to observe.
- `prob` A scalar. Observation probability.
- `m` A integer. Number of units to observe; if specified, m overrides prob.
- `subset` A character. Logical statement that can be applied to rows of complete data. For instance observation for some nodes might depend on observed values of other nodes; or observation may only be sought if data not already observed!

**Value**
A `data.frame` with logical values indicating which nodes to observe in each row of `complete_data`.

**Examples**
```r
model <- make_model("X -> Y")
df <- make_data(model, n = 8)
# Observe X values only
observe_data(complete_data = df, nodes_to_observe = "X")
# Observe half the Y values for cases with observed X = 1
observe_data(complete_data = df,
  observed = observe_data(complete_data = df, nodes_to_observe = "X"),
  nodes_to_observe = "Y", prob = .5,
  subset = "X==1")
```
Setting parameters

Description

Functionality for altering parameters:

- A vector of 'true' parameters; possibly drawn from prior or posterior.

Add a true parameter vector to a model. Parameters can be created using arguments passed to `make_parameters` and `make_priors`.

Extracts parameters as a named vector.

Usage

```r
make_parameters(
  model,  
  parameters = NULL,  
  param_type = NULL,  
  warning = TRUE,  
  normalize = TRUE,  
  ...
)
```

```r
set_parameters(
  model,  
  parameters = NULL,  
  param_type = NULL,  
  warning = FALSE,  
  ...  
)
```

```r
get_parameters(model, param_type = NULL)
```

Arguments

- `model`: A causal_model. A model object generated by `make_model`.
- `parameters`: A vector of real numbers in [0,1]. Values of parameters to specify (optional). By default, parameters is drawn from `model$parameters_df`.
- `param_type`: A character. String specifying type of parameters to make "flat", "prior_mean", "posterior_mean", "prior_draw", "posterior_draw", "define". With param_type set to define use arguments to be passed to `make_priors`; otherwise flat sets equal probabilities on each nodal type in each parameter set; prior_mean, prior_draw, posterior_mean, posterior_draw take parameters as the means or as draws from the prior or posterior.
- `warning`: Logical. Whether to warn about parameter renormalization.
**normalize** Logical. If parameter given for a subset of a family the residual elements are normalized so that parameters in param_set sum to 1 and provided params are unaltered.

... Options passed onto `make_priors`.

**Value**

A vector of draws from the prior or distribution of parameters

An object of class `causal_model`. It essentially returns a list containing the elements comprising a model (e.g. 'statement', 'nodal_types' and 'DAG') with true vector of parameters attached to it.

A vector of draws from the prior or distribution of parameters

**Examples**

```r
# make_parameters examples:

# Simple examples
model <- make_model('X -> Y')
data <- make_data(model, n = 2)
model <- update_model(model, data)
make_parameters(model, parameters = c(.25, .75, 1.25, .25, .25, .25))
make_parameters(model, param_type = 'flat')
make_parameters(model, param_type = 'prior_draw')
make_parameters(model, param_type = 'prior_mean')
make_parameters(model, param_type = 'posterior_draw')
make_parameters(model, param_type = 'posterior_mean')

#altering values using `alter_at`
make_model('X -> Y') %>% make_parameters(parameters = c(0.5, 0.25),
alter_at = "node == 'Y' & nodal_type %in% c('00', '01')")

#altering values using `param_names`
make_model('X -> Y') %>% make_parameters(parameters = c(0.5, 0.25),
param_names = c("Y.10", "Y.01"))

#altering values using `statement`
make_model('X -> Y') %>% make_parameters(parameters = c(0.5),
statement = "Y[X=1] > Y[X=0]"

#altering values using a combination of other arguments
make_model('X -> Y') %>% make_parameters(parameters = c(0.5, 0.25),
node = "Y", nodal_type = c("00", "01"))

# Normalize renormalizes values not set so that value set is not renomalized
make_parameters(make_model('X -> Y'),
statement = 'Y[X=1]>Y[X=0]', parameters = .5)
make_parameters(make_model('X -> Y'),
parameters = c(0.5),
statement = 'Y[X=1]>Y[X=0]',
node = "Y", nodal_type = c("00", "01"))
```

---

The code above demonstrates how to use the `make_parameters` function in R, showing examples of setting parameters, altering values, and normalizing parameters. It includes a mix of simple and more complex examples, illustrating the flexibility of the function in handling different types of parameter specifications and manipulations.
# set_parameters examples:

```r
make_model('X -> Y') %>% set_parameters(1:6) %>% grab("parameters")
```

# Simple examples

```r
model <- make_model('X -> Y')
data <- make_data(model, n = 2)
model <- update_model(model, data)
set_parameters(model, parameters = c(.25, .75, 1.25, .25, .25, .25))
set_parameters(model, param_type = 'flat')
set_parameters(model, param_type = 'prior_draw')
set_parameters(model, param_type = 'prior_mean')
set_parameters(model, param_type = 'posterior_draw')
set_parameters(model, param_type = 'posterior_mean')
```

# altering values using `alter_at`

```r
make_model("X -> Y") %>% set_parameters(parameters = c(0.5, 0.25),
alter_at = "node == 'Y' & nodal_type %in% c('00', '01')")
```

# altering values using `param_names`

```r
make_model("X -> Y") %>% set_parameters(parameters = c(0.5, 0.25),
param_names = c("Y.10", "Y.01"))
```

# altering values using `statement`

```r
make_model("X -> Y") %>% set_parameters(parameters = c(0.5),
statement = "Y[X=1] > Y[X=0]"
```

# altering values using a combination of other arguments

```r
make_model("X -> Y") %>% set_parameters(parameters = c(0.5, 0.25),
node = "Y", nodal_type = c("00", "01"))
```

---

**print.causal_model**

Print a short summary for a causal model

---

**Description**

print method for class causal_model.
print.causal_types

Usage

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

Arguments

x 
An object of causal_model class, usually a result of a call to make_model or update_model.

... 
Further arguments passed to or from other methods.

Details

The information regarding the causal model includes the statement describing causal relations using dagitty syntax, number of nodal types per parent in a DAG, and number of causal types.

----

print.causal_types  
Print a short summary for causal_model causal-types

Description

print method for class causal_types.

Usage

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

Arguments

x 
An object of causal_types class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.

... 
Further arguments passed to or from other methods.

----

print.dag  
Print a short summary for a causal_model DAG

Description

print method for class dag.

Usage

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

...
Arguments

x  An object of dag class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.

...  Further arguments passed to or from other methods.

print.event_probabilities

Print a short summary for event probabilities

Description

print method for class event_probabilities.

Usage

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

Arguments

x  An object of event_probabilities class, which is a sub-object of an object of the causal_model class produced using update_model.

...  Further arguments passed to or from other methods.

print.model_query  Print a tightened summary of model queries

Description

print method for class model_query.

Usage

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

Arguments

x  An object of model_query class.

...  Further arguments passed to or from other methods.
print.nodal_types

Description

print method for class nodal_types.

Usage

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

Arguments

- `x`: An object of nodal_types class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.
- `...`: Further arguments passed to or from other methods.

print.nodes

Description

print method for class nodes.

Usage

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

Arguments

- `x`: An object of nodes class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.
- `...`: Further arguments passed to or from other methods.
print.parameters

Print a short summary for causal_model parameters

Description

print method for class parameters.

Usage

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

Arguments

x An object of parameters class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.

... Further arguments passed to or from other methods.

print.parameters_df

Print a short summary for a causal_model parameters data-frame

Description

print method for class parameters_df.

Usage

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

Arguments

x An object of parameters_df class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.

... Further arguments passed to or from other methods.
print.parameters_posterior

Print a short summary for causal_model parameter posterior distributions

Description

print method for class parameters_posterior.

Usage

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

Arguments

x An object of parameters_posterior class, which is a sub-object of an object of the causal_model class produced using update_model.

... Further arguments passed to or from other methods.

print.parameters_prior

Print a short summary for causal_model parameter prior distributions

Description

print method for class parameters_prior.

Usage

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

Arguments

x An object of parameters_prior class, which is a sub-object of an object of the causal_model class produced using set_prior_distribution.

... Further arguments passed to or from other methods.
print.parameter_mapping

Print a short summary for parameter mapping matrix

Description

print method for class parameter_mapping.

Usage

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

Arguments

x An object of parameter_mapping class.
...
Further arguments passed to or from other methods.

print.parents_df

Print a short summary for causal_model parents data-frame

Description

print method for class parents_df.

Usage

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

Arguments

x An object of parents_df class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.
...
Further arguments passed to or from other methods.
print.posterior_event_probabilities

Print a short summary of posterior_event_probabilities

Description

print method for class posterior_event_probabilities.

Usage

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

Arguments

x  An object of posterior_event_probabilities class.
...
Further arguments passed to or from other methods.

print.stan_summary

Print a short summary for stan fit

Description

print method for class stan_summary.

Usage

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

Arguments

x  An object of stan_summary class, which is a sub-object of an object of the causal_model class produced using update_model.
...
Further arguments passed to or from other methods.
### print.statement

**Print a short summary for a causal_model statement**

**Description**

print method for class statement.

**Usage**

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

**Arguments**

- `x` An object of statement class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.
- `...` Further arguments passed to or from other methods.

### print.type_distribution

**Print a short summary for causal-type posterior distributions**

**Description**

print method for class type_distribution.

**Usage**

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

**Arguments**

- `x` An object of type_distribution class, which is a sub-object of an object of the causal_model class produced using get_type_prob_multiple.
- `...` Further arguments passed to or from other methods.
print.type_prior

Print a short summary for causal-type prior distributions

Description

print method for class type_prior.

Usage

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

Arguments

x          An object of type_prior class, which is a sub-object of an object of the causal_model class produced using make_model or update_model.
...        Further arguments passed to or from other methods.

prior_setting  Setting priors

Description

Functionality for altering priors:
make_priors Generates priors for a model.
set_priors Adds priors to a model.
Extracts priors as a named vector

Usage

make_priors(
    model,
    alphas = NA,
    distribution = NA,
    alter_at = NA,
    node = NA,
    nodal_type = NA,
    label = NA,
    param_set = NA,
    given = NA,
    statement = NA,
    join_by = "|",
    param_names = NA
)


set_priors(
    model,
    alphas = NA,
    distribution = NA,
    alter_at = NA,
    node = NA,
    nodal_type = NA,
    label = NA,
    param_set = NA,
    given = NA,
    statement = NA,
    join_by = "|",
    param_names = NA
)

get_priors(model, nodes = NULL)

Arguments

model        A model object generated by make_model().
alphas       Real positive numbers giving hyperparameters of the Dirichlet distribution
distribution  string indicating a common prior distribution (uniform, jeffreys or certainty)
alter_at     string specifying filtering operations to be applied to parameters_df, yielding a logical vector indicating parameters for which values should be altered. (see examples)
node         string indicating nodes which are to be altered
odal_type    string. Label for nodal type indicating nodal types for which values are to be altered
label        string. Label for nodal type indicating nodal types for which values are to be altered. Equivalent to nodal_type.
param_set    string indicating the name of the set of parameters to be altered
given        string indicates the node on which the parameter to be altered depends
statement    causal query that determines nodal types for which values are to be altered
join_by      string specifying the logical operator joining expanded types when statement contains wildcards. Can take values '& (logical AND) or ' | ' (logical OR).
param_names  vector of strings. The name of specific parameter in the form of, for example, 'X.1', 'Y.01'

Details

Seven arguments govern which parameters should be altered. The default is 'all' but this can be reduced by specifying
*alter_at* String specifying filtering operations to be applied to parameters\_df, yielding a logical vector indicating parameters for which values should be altered. “node == ‘X’ & nodal\_type

*node*, which restricts for example to parameters associated with node ‘X’

*nodal\_type* or *label* The label of a particular nodal type, written either in the form Y0000 or Y.Y0000

*param\_set* The param\_set of a parameter.

*given* Given parameter set of a parameter.

*statement*, which restricts for example to nodal types that satisfy the statement ‘Y[X=1] > Y[X=0]’

*param\_set, given*, which are useful when setting confound statements that produce several sets of parameters

Two arguments govern what values to apply:

*alphas* is one or more non-negative numbers and

*distribution* indicates one of a common class: uniform, Jeffreys, or ’certain’

Forbidden statements include:

• Setting distribution and values at the same time.
• Setting a distribution other than uniform, Jeffreys, or certainty.
• Setting negative values.
• specifying alter\_at with any of node, nodal\_type, param\_set, given, statement, or param\_names
• specifying param\_names with any of node, nodal\_type, param\_set, given, statement, or alter\_at
• specifying statement with any of node or nodal\_type

Value

A vector indicating the parameters of the prior distribution of the nodal types ("hyperparameters"). An object of class causal\_model. It essentially returns a list containing the elements comprising a model (e.g. ’statement’, ’nodal\_types’ and ’DAG’) with the ‘priors’ attached to it.

A vector indicating the hyperparameters of the prior distribution of the nodal types.

Examples

```r
# make_priors examples:

# Pass all nodal types
model <- make_model("Y <- X")
make_priors(model, alphas = .4)
make_priors(model, distribution = "jeffreys")

model <- CausalQueries::make_model("X -> M -> Y; X <-> Y")

#altering values using \code{alter_at}
```
query_distribution

Description

Calculated distribution of a query from a prior or posterior distribution of parameters

Usage

query_distribution(

make_priors(model = model, alphas = c(0.5,0.25),
alter_at = "node == 'Y' & nodal_type %in% c('00','01') & given == 'X.0'")

#altering values using \code{param_names}
make_priors(model = model, alphas = c(0.5,0.25),
param_names = c("Y.10_X.0","Y.10_X.1"))

#altering values using \code{statement}
make_priors(model = model, alphas = c(0.5,0.25),
statement = "Y[M=1] > Y[M=0]"

#altering values using a combination of other arguments
make_priors(model = model, alphas = c(0.5,0.25),
node = "Y", nodal_type = c("00","01"), given = "X.0")

# set_priors examples:

# Pass all nodal types
model <- make_model("Y <- X")
set_priors(model, alphas = .4)
set_priors(model, distribution = "jeffreys")

model <- CausalQueries::make_model("X -> M -> Y; X <-> Y")

#altering values using \code{alter_at}
set_priors(model = model, alphas = c(0.5,0.25),
alter_at = "node == 'Y' & nodal_type %in% c('00','01') & given == 'X.0'"

#altering values using \code{param_names}
set_priors(model = model, alphas = c(0.5,0.25),
param_names = c("Y.10_X.0","Y.10_X.1"))

#altering values using \code{statement}
set_priors(model = model, alphas = c(0.5,0.25),
statement = "Y[M=1] > Y[M=0]"

#altering values using a combination of other arguments
set_priors(model = model, alphas = c(0.5,0.25), node = "Y",
nodal_type = c("00","01"), given = "X.0")

query_distribution   Calculate query distribution
query_distribution

query_distribution(model, queries, given = NULL, using = "parameters", parameters = NULL, n_draws = 4000, join_by = "|", case_level = FALSE, query = NULL)

Arguments

model  A causal_model. A model object generated by make_model.
queries  A character vector or list of character vectors specifying queries on potential outcomes such as "Y[X=1] - Y[X=0]"
given  A character vector specifying givens for each query. A given is a quoted expression that evaluates to logical statement. given allows the query to be conditioned on *observational* distribution. A value of TRUE is interpreted as no conditioning.
using  A character. Whether to use priors, posteriors or parameters
parameters  A vector or list of vectors of real numbers in [0,1]. A true parameter vector to be used instead of parameters attached to the model in case using specifies parameters
n_draws  An integer. Number of draws
join_by  A character. The logical operator joining expanded types when query contains wildcard (.). Can take values "&" (logical AND) or "|" (logical OR). When restriction contains wildcard (.) and join_by is not specified, it defaults to "|", otherwise it defaults to NULL.
case_level  Logical. If TRUE estimates the probability of the query for a case.
query  alias for queries

Value

A DataFrame where columns contain draws from the distribution of the potential outcomes specified in query

Examples

model <- make_model("X -> Y") %>%
  set_parameters(c(.5, .5, .1, .2, .3, .4))

# simple queries
query_distribution(model, query = "(Y[X=1] > Y[X=0])", using = "priors") |> head()
# multiple queries
query_distribution(model, 
    query = list("(Y[X=1] > Y[X=0])", 
                  "(Y[X=1] < Y[X=0])"), 
    using = "priors") |> head()

# multiple queries and givens
query_distribution(model, 
    query = list("(Y[X=1] > Y[X=0])", "(Y[X=1] < Y[X=0])"), 
    given = list("Y==1", "(Y[X=1] <= Y[X=0])"), 
    using = "priors") |> head()

# linear queries
query_distribution(model, query = "(Y[X=1] - Y[X=0])")

# queries conditional on observables
query_distribution(model, query = "(Y[X=1] > Y[X=0])", 
                    given = "X==1 & Y ==1")

# Linear query conditional on potential outcomes
query_distribution(model, query = "(Y[X=1] - Y[X=0])", 
                    given = "Y[X=1]==0")

# Use join_by to amend query interpretation
query_distribution(model, query = "(Y[.] == 1)", join_by = "/")

# Probability of causation query
query_distribution(model, 
    query = "(Y[X=1] > Y[X=0])", 
    given = "X==1 & Y ==1", 
    using = "priors") |> head()

# Case level probability of causation query
query_distribution(model, 
    query = "(Y[X=1] > Y[X=0])", 
    given = "X==1 & Y==1", 
    case_level = TRUE, 
    using = "priors")

# Query posterior
update_model(model, make_data(model, n = 3)) |> 
query_distribution(query = "(Y[X=1] - Y[X=0])", using = "posteriors") |> head()

# Case level queries provide the inference for a case, which is a scalar
# The case level query *updates* on the given information
# For instance, here we have a model for which we are quite sure that X
# causes Y but we do not know whether it works through two positive effects
# or two negative effects. Thus we do not know if M=0 would suggest an
# effect or no effect
```r
set.seed(1)
model <-
  make_model("X --> M --> Y") |>
  update_model(data.frame(X = rep(0:1, 8), Y = rep(0:1, 8)), iter = 10000)
Q <- "Y[X=1] > Y[X=0]"
G <- "X==1 & Y==1 & M==1"
QG <- "(Y[X=1] > Y[X=0]) & (X==1 & Y==1 & M==1)"

# In this case these are very different:
query_distribution(model, Q, given = G, using = "posteriors")[[1]] |> mean()
query_distribution(model, Q, given = G, using = "posteriors",
  case_level = TRUE)

# These are equivalent:
# 1. Case level query via function
query_distribution(model, Q, given = G,
  using = "posteriors", case_level = TRUE)

# 2. Case level query by hand using Bayes
distribution <- query_distribution(
  model, list(QG = QG, G = G), using = "posteriors")

mean(distribution$QG)/mean(distribution$G)
```

---

**query_model**

*Generate estimands dataframe*

**Description**

Calculated from a parameter vector, from a prior or from a posterior distribution.

**Usage**

```r
query_model(
  model,
  queries = NULL,
  given = NULL,
  using = list("parameters"),
  parameters = NULL,
  stats = NULL,
  n_draws = 4000,
  expand_grid = FALSE,
  case_level = FALSE,
  query = NULL,
  cred = 95
)
```
query_model

**Arguments**

- **model**: A causal_model. A model object generated by `make_model`.
- **queries**: A vector of strings or list of strings specifying queries on potential outcomes such as "Y[ X=1 ] - Y[ X=0 ]".
- **given**: A vector or list of strings specifying givens. A given is a quoted expression that evaluates to a logical statement. Allows estimand to be conditioned on *observational* (or counterfactual) distribution.
- **using**: A vector or list of strings. Whether to use priors, posteriors or parameters.
- **parameters**: A vector of real numbers in [0,1]. Values of parameters to specify (optional). By default, parameters is drawn from `model$parameters_df`.
- **stats**: Functions to be applied to estimand distribution. If NULL, defaults to mean, standard deviation, and 95% confidence interval. Functions should return a single numeric value.
- **n_draws**: An integer. Number of draws.
- **expand_grid**: Logical. If TRUE then all combinations of provided lists are examined. If not then each list is cycled through separately. Defaults to FALSE.
- **case_level**: Logical. If TRUE estimates the probability of the query for a case.
- **query**: alias for queries
- **cred**: size of the credible interval ranging between 0 and 100

**Details**

Queries can condition on observed or counterfactual quantities. Nested or "complex" counterfactual queries of the form Y[ X=1, M[X=0] ] are allowed.

**Value**

A DataFrame with columns Model, Query, Given and Using defined by corresponding input values. Further columns are generated as specified in stats.

**Examples**

```r
model <- make_model("X -> Y")
query_model(model, "Y[ X=1 ] - Y[ X = 0 ]", using = "priors")
query_model(model, "Y[ X=1 ] > Y[ X = 0 ]", using = "parameters")
query_model(model, "Y[ X=1 ] > Y[ X = 0 ]", using = c("priors", "parameters"))

# `expand_grid= TRUE` requests the Cartesian product of arguments

models <- list(
  M1 = make_model("X -> Y"),
  M2 = make_model("X -> Y") |> set_restrictions("Y[ X=1 ] < Y[ X=0 ]")
)
```


### realise_outcomes

Realise outcomes for all causal types. Calculated by sequentially calculating endogenous nodes. If a do operator is applied to any node then it takes the given value and all its descendants are generated accordingly.

#### Usage

```r
realise_outcomes(model, dos = NULL, node = NULL, add_rownames = TRUE)
```

#### Arguments

- **model**: A causal_model. A model object generated by `make_model`.
- **dos**: A named list. Do actions defining node values, e.g., `list(X = 0, M = 1)`.
- **node**: A character. An optional quoted name of the node whose outcome should be revealed. If specified all values of parents need to be specified via `dos`.
- **add_rownames**: logical indicating whether to add causal types as rownames to the output.
Details

If a node is not specified all outcomes are realised for all possible causal types consistent with the model. If a node is specified then outcomes of Y are returned conditional on different values of parents, whether or not these values of the parents obtain given restrictions under the model.

`realise_outcomes` starts off by creating types (via `get_nodal_types`). It then takes types of endogenous and reveals their outcome based on the value that their parents took. Exogenous nodes outcomes correspond to their type.

Value

A `data.frame` object of revealed data for each node (columns) given causal / nodal type (rows).

Examples

```r
make_model("X -> Y") |> realise_outcomes()
make_model("X -> Y <- W") |> set_restrictions(labels = list(X = "1", Y="0010"), keep = TRUE) |> realise_outcomes()
make_model("X1->Y; X2->M; M->Y") |> realise_outcomes(dos = list(X1 = 1, M = 0))

# With node specified
make_model("X->M->Y") |> realise_outcomes(node = "Y")
make_model("X->M->Y") |> realise_outcomes(dos = list(M = 1), node = "Y")
```

### set_ambiguities_matrix

*Set ambiguity matrix*

**Description**

Add an ambiguities matrix to a model

**Usage**

```r
set_ambiguities_matrix(model, A = NULL)
```
**set_confound**

**Arguments**

- **model**: A causal_model. A model object generated by `make_model`.
- **confound**: A data.frame. Ambiguity matrix. Not required but may be provided to avoid repeated computation for simulations.

**Value**

An object of type causal_model with the ambiguities matrix attached.

**Description**

Adjust parameter matrix to allow confounding.

**Usage**

```r
set_confound(model, confound = NULL)
```

**Arguments**

- **model**: A causal_model. A model object generated by `make_model`.
- **confound**: A list of statements indicating pairs of nodes whose types are jointly distributed (e.g. list("A <-> B", "C <-> D")).

**Details**

Confounding between X and Y arises when the nodal types for X and Y are not independently distributed. In the X -> Y graph, for instance, there are 2 nodal types for X and 4 for Y. There are thus 8 joint nodal types:

```
|  | t^X |  |  | t^Y | 00 | Pr(t^X=0 & t^Y=00) | Pr(t^X=1 & t^Y=00) | Pr(t^Y=00) |
|  | 01 | . |  | 10 | . | . | . |
|  | 11 | . |  | Sum | Pr(t^X=0) | Pr(t^X=1) | 1 |
```

This table has 8 interior elements and so an unconstrained joint distribution would have 7 degrees of freedom. A no confounding assumption means that Pr(t^X | t^Y) = Pr(t^X), or Pr(t^X, t^Y) = Pr(t^X)Pr(t^Y). In this case there would be 3 degrees of freedom for Y and 1 for X, totaling 4 rather than 7.
set_confound lets you relax this assumption by increasing the number of parameters characterizing the joint distribution. Using the fact that \( P(A,B) = P(A)P(B|A) \) new parameters are introduced to capture \( P(B|A=a) \) rather than simply \( P(B) \). For instance here two parameters (and one degree of freedom) govern the distribution of types X and four parameters (with 3 degrees of freedom) govern the types for Y given the type of X for a total of 1+3+3 = 7 degrees of freedom.

Value

An object of class `causal_model` with updated parameters_df and parameter matrix.

Examples

```r
make_model('X -> Y; X <-> Y') |>
grab("parameters")
make_model('X -> M -> Y; X <-> Y') |>
grab("parameters")
model <- make_model('X -> M -> Y; X <-> Y; M <-> Y')
model$parameters_df

# Example where set_confound is implemented after restrictions
make_model("A -> B -> C") |>
set_restrictions(increasing("A", "B")) |>
set_confound("B <-> C") |>
grab("parameters")

# Example where two parents are confounded
make_model('A -> B <- C; A <-> C') |>
set_parameters(node = "C", c(0.05, .95, .95, 0.05)) |>
make_data(n = 50) |>
cor()

# Example with two confounds, added sequentially
model <- make_model('A -> B -> C') |>
set_confound(list("A <-> B", "B <-> C"))
model$statement
# plot(model)
```

`set_parameter_matrix`  

**Description**

Add a parameter matrix to a model

**Usage**

`set_parameter_matrix(model, P = NULL)`
Arguments

model A causal_model. A model object generated by make_model.

P A data.frame. Parameter matrix. Not required but may be provided to avoid repeated computation for simulations.

Value

An object of class causal_model. It essentially returns a list containing the elements comprising a model (e.g. 'statement', 'nodal_types' and 'DAG') with the parameter matrix attached to it.

Examples

model <- make_model("X -> Y")
P <- diag(8)
colnames(P) <- rownames(model$causal_types)
model <- set_parameter_matrix(model, P = P)
set_prior_distribution

Add prior distribution draws

Description

Add \( n \_param \times n \_draws \) database of possible parameter draws to the model.

Usage

\[
\text{set_prior_distribution}(\text{model}, n\_draws = 4000)
\]

Arguments

- model: A causal_model. A model object generated by \text{make_model}.
- n_draws: A scalar. Number of draws.

Value

An object of class causal_model with the ‘prior_distribution’ attached to it.

See Also

Other prior_distribution: \text{get_prior_distribution()}, \text{make_prior_distribution}()

Examples

\[
\text{make_model}('X -> Y') %>% \\
\text{set_prior_distribution}(n\_draws = 5) %>% \\
\text{grab}("prior\_distribution")
\]

set_restrictions

Restrict a model

Description

Restrict a model’s parameter space. This reduces the number of nodal types and in consequence the number of unit causal types.
set_restrictions

Usage

```r
set_restrictions(
  model, 
  statement = NULL, 
  join_by = "|", 
  labels = NULL, 
  param_names = NULL, 
  given = NULL, 
  keep = FALSE
)
```

Arguments

- **model**: A causal_model. A model object generated by `make_model`.
- **statement**: A quoted expressions defining the restriction. If values for some parents are not specified, statements should be surrounded by parentheses, for instance `(Y[A = 1] > Y[A=0])` will be interpreted for all combinations of other parents of Y set at possible levels they might take.
- **join_by**: A string. The logical operator joining expanded types when `statement` contains wildcard (`.`). Can take values ' & ' (logical AND) or '|' (logical OR). When restriction contains wildcard (`.`) and `join_by` is not specified, it defaults to '|', otherwise it defaults to NULL. Note that `join_by` joins within statements, not across statements.
- **labels**: A list of character vectors specifying nodal types to be kept or removed from the model. Use `get_nodal_types` to see syntax. Note that `labels` gets overwritten by `statement` if `statement` is not NULL.
- **param_names**: A character vector of names of parameters to restrict on.
- **given**: A character vector or list of character vectors specifying nodes on which the parameter set to be restricted depends. When restricting by `statement`, `given` must either be NULL or of the same length as `statement`. When mixing statements that are further restricted by `given` and ones that are not, statements without `given` restrictions should have `given` specified as one of `NULL`, `NA`, `''` or `"`````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````
Statements with implicitly controlled nodes should be surrounded by parentheses, as in these examples.
Note that prior probabilities are redistributed over remaining types.

Value
An object of class model. The causal types and nodal types in the model are reduced according to the stated restriction.

See Also
Other restrictions: restrict_by_labels(), restrict_by_query()

Examples

```r
# 1. Restrict parameter space using statements
model <- make_model('X->Y') %>%
  set_restrictions(statement = c('X[] == 0'))
model <- make_model('X->Y') %>%
  set_restrictions(non_increasing('X', 'Y'))
model <- make_model('X -> Y <- W') %>%
  set_restrictions(c(decreasing('X', 'Y'), substitutes('X', 'W', 'Y')))  
model$parameters_df
model <- make_model('X-> Y <- W') %>%
  set_restrictions(statement = decreasing('X', 'Y'))
model$parameters_df
model <- make_model('X->Y') %>%
  set_restrictions(decreasing('X', 'Y'))
model$parameters_df
model <- make_model('X->Y') %>%
  set_restrictions(c(increasing('X', 'Y'), decreasing('X', 'Y')))  
model$parameters_df
# Restrict to define a model with monotonicity
model <- make_model('X->Y') %>%
  set_restrictions(statement = c('Y[X=1] < Y[X=0]'))  
grab(model, "parameter_matrix")
# Restrict to a single type in endogenous node
model <- make_model('X->Y') %>%
  set_restrictions(statement = 'Y[X = 1] == 1', join_by = '&', keep = TRUE)  
grab(model, "parameter_matrix")
# Use of | and &
# Keep node if *for some value of B* Y[A = 1] == 1
```
model <- make_model('A -> Y <- B') %>%
set_restrictions(statement = 'Y[A == 1] == 1', join_by = '|', keep = TRUE)
dim(grab(model, "parameter_matrix"))

# Keep node if *for all values of B*: Y[A == 1] == 1
model <- make_model('A -> Y <-> B') %>%
set_restrictions(statement = 'Y[A == 1] == 1', join_by = '&', keep = TRUE)
dim(grab(model, "parameter_matrix"))

# Restrict multiple nodes
model <- make_model('X -> Y <- M; X -> M') %>%
set_restrictions(statement = c('Y[X == 1] == 1', 'M[X == 1] == 1'),
join_by = '&', keep = TRUE)
grab(model, "parameter_matrix")

# Restrict using statements and given:
model <- make_model("X -> Y -> Z; X <-> Z") %>%
set_restrictions(list(decreasing('X', 'Y'),
                      decreasing('Y', 'Z'),
given = c(NA,'X.0')))
grab(model, "parameter_matrix")

# Restrictions on levels for endogenous nodes aren't allowed
## Not run:
model <- make_model('X -> Y') %>%
set_restrictions(statement = '(Y == 1)')
## End(Not run)

# 2. Restrict parameter space Using labels:
model <- make_model('X -> Y') %>%
set_restrictions(labels = list(X = '0', Y = '00'))

# Restrictions can be with wildcards
model <- make_model('X -> Y') %>%
set_restrictions(labels = list(Y = '?0'))
grab(model, "parameter_matrix")

# Deterministic model
model <- make_model('S -> C -> Y <- R <- X; X -> C -> R') %>%
set_restrictions(labels = list(C = '1000', R = '0001', Y = '0001'),
keep = TRUE)
grab(model, "parameter_matrix")

# Restrict using labels and given:
model <- make_model("X -> Y -> Z; X <-> Z") %>%
set_restrictions(labels = list(X = '0', Z = '00'), given = c(NA,'X.0'))
grab(model, "parameter_matrix")

```
simulate_data

simulate_data is an alias for make_data
```
### simulate_data

**Description**

simulate_data is an alias for make_data

**Usage**

```r
simulate_data(...)```

**Arguments**

... arguments for `make_model`

**Value**

A `data.frame` with simulated data.

**Examples**

```r
simulate_data(make_model("X->Y"))```

### substitutes

**Description**

Generate a statement for X1, X1 substitute for each other in the production of Y

**Usage**

```r
substitutes(X1, X2, Y)```

**Arguments**

- **X1**: A character. The quoted name of the input node 1.
- **X2**: A character. The quoted name of the input node 2.
- **Y**: A character. The quoted name of the outcome node.

**Value**

A character statement of class `statement`

**See Also**

Other statements: `complements()`, `decreasing()`, `increasing()`, `interacts()`, `non_decreasing()`, `non_increasing()`, `te()`
Examples

```r
get_query_types(model = make_model('A -> B <- C'),
query = substitutes('A', 'C', 'B'), map = "causal_type")

query_model(model = make_model('A -> B <- C'),
queries = substitutes('A', 'C', 'B'),
using = 'parameters')
```

summarise_distribution

`helper to compute mean and sd of a distribution data.frame`

Description

helper to compute mean and sd of a distribution data.frame

Usage

`summarise_distribution(x)`

Arguments

- `x` An object for summarizing

summarise_distribution

`helper to compute mean and sd of a distribution data.frame`

Description

summary method for class `causal_model`.

Usage

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

## S3 method for class 'summary.causal_model'
print(x, stanfit = FALSE, ...)
```

Arguments

- `object` An object of causal_model class produced using `make_model` or `update_model`.
- `...` Further arguments passed to or from other methods.
- `x` An object of summary.causal_model class, usually a result of a call to `summary.causal_model`.
- `stanfit` Logical. Whether to include readable summary of stanfit produced when updating a model via `update_model`. Defaults to `FALSE`.
Details

print.summary.causal_model reports DAG data frame, full specification of nodal types and summary of model restrictions in addition to standard print.causal_model output.

---

`te`  
*Make treatment effect statement (positive)*

Description

Generate a statement for `(Y(1) - Y(0))`. This statement when applied to a model returns an element in `(1,0,-1)` and not a set of cases. This is useful for some purposes such as querying a model, but not for uses that require a list of types, such as `set_restrictions`.

Usage

```
te(X, Y)
```

Arguments

- `X`  
  A character. The quoted name of the input node

- `Y`  
  A character. The quoted name of the outcome node

Value

A character statement of class `statement`

See Also

Other statements: `complements()`, `decreasing()`, `increasing()`, `interacts()`, `non_decreasing()`, `non_increasing()`, `substitutes()`

Examples

```
te('A', 'B')
```

```
model <- make_model('X->Y') %>% set_restrictions(increasing('X', 'Y'))
query_model(model, list(ate = te('X', 'Y'), using = 'parameters'))
```

# set_restrictions breaks with te because it requires a listing  
# of causal types, not numeric output.

## Not run:
```
model <- make_model('X->Y') %>% set_restrictions(te('X', 'Y'))
```

## End(Not run)
**update_model**

*Fit causal model using ‘stan’*

**Description**

Takes a model and data and returns a model object with data attached and a posterior model.

**Usage**

```r
update_model(
  model,  
data = NULL,  
data_type = NULL,  
keep_type_distribution = TRUE,  
keep_event_probabilities = FALSE,  
keep_fit = FALSE,  
censored_types = NULL,
  ...
)
```

**Arguments**

- `model`: A `causal_model`. A model object generated by `make_model`.
- `data`: A `data.frame`. Data of nodes that can take three values: 0, 1, and NA. In long form as generated by `make_events`.
- `data_type`: Either 'long' (as made by `make_data`) or 'compact' (as made by `collapse_data`). Compact data must have entries for each member of each strategy family to produce a valid simplex. When long form data is provided with missingness, missing data is assumed to be missing at random.
- `keep_type_distribution`: Logical. Whether to keep the (transformed) distribution of the causal types. Defaults to 'TRUE'.
- `keep_event_probabilities`: Logical. Whether to keep the (transformed) distribution of event probabilities. Defaults to 'FALSE'.
- `keep_fit`: Logical. Whether to keep the `stanfit` object produced by `sampling` for further inspection. See ?stanfit for more details. Defaults to 'FALSE'. Note the `stanfit` object has internal names for parameters (lambda), event probabilities (w), and the type distribution (types).
- `censored_types`: vector of data types that are selected out of the data, e.g. c("X0Y0")
- `...`: Options passed onto `sampling` call. For details see ?rstan::sampling

**Value**

An object of class `causal_model`. The returned model is a list containing the elements comprising a model (e.g. ‘statement’, ‘nodal_types’ and ‘DAG’) with the posterior_distribution returned by `stan` attached to it.
See Also

`make_model` allows to create new model, `summary.causal_model` provides summary method for output objects of class `causal_model`

Examples

```r
model <- make_model('X->Y')
data_long <- simulate_data(model, n = 4)
data_short <- collapse_data(data_long, model)

model <- update_model(model, data_long)
model <- update_model(model, data_short)

## Not run:
# It is possible to implement updating without data, in which
# case the posterior is a stan object that reflects the prior
update_model(model)

data <- data.frame(X=rep(0:1, 10), Y=rep(0:1,10))

# Censored data types
# We update less than we might because we are aware of filtered data
uncensored <-
  make_model("X->Y") |> update_model(data) |> query_model(te("X", "Y"), using = "posteriors")
censored <-
  make_model("X->Y") |> update_model(
    data,
    censored_types = c("X1Y0")) |> query_model(te("X", "Y"), using = "posteriors")

# Censored data: We learn nothing because the data
# we see is the only data we could ever see
make_model("X->Y") |> update_model(
  data,
  censored_types = c("X1Y0", "X0Y0", "X0Y1")) |> query_model(te("X", "Y"), using = "posteriors")

## End(Not run)
```
Index

* **datasets**
  - democracy_data, 8
  - institutions_data, 21
  - lipids_data, 23
* **parameters**
  - parameter_setting, 34
* **prior_distribution**
  - make_prior_distribution, 31
  - set_prior_distribution, 58
* **priors**
  - prior_setting, 45
* **restrictions**
  - set_restrictions, 58
* **statements**
  - complements, 5
  - decreasing, 7
  - increasing, 20
  - interacts, 22
  - non_decreasing, 31
  - non_increasing, 32
  - substitutes, 62
tele, 64
CausalQueries-package, 4
collapse_data, 4, 65
complements, 5, 7, 21, 22, 32, 62, 64
dagitty, 28, 37
data_type_names, 6
increasing, 6, 7, 21, 22, 32, 62, 64
democracy_data, 8
draw_causal_type, 8

expand_data, 9
expand_wildcard, 10
find_rounding_threshold, 10
get_all_data_types, 11
get_ambiguities_matrix, 12
get_event_probabilities, 12
get_nodal_types, 54
get_parameter_names, 13
get_parameters (parameter_setting), 34
get_parents, 14
get_parmap, 14
get_prior_distribution, 31, 58
get_priors (prior_setting), 45
get_query_types, 15
get_type_prob, 16
get_type_prob_c, 17
get_type_prob_multiple_c, 18
grab, 18
increasing, 6, 7, 20, 21, 22, 32, 62, 64
institutions_data, 21
interacts, 6, 7, 21, 22, 32, 64
interpret_type, 22
lipids_data, 23
make_data, 24, 26, 65
make_events, 4, 6, 26, 65
make_model, 4, 6, 8, 9, 11–15, 17, 18, 23, 24,
27, 28, 30, 31, 34, 49, 52, 53, 55, 57–59, 62, 65, 66
make_parameter_matrix, 30
make_parameters, 34
make_parameters (parameter_setting), 34
make_parmap, 30
make_prior_distribution, 31, 58
make_priors, 34, 35
make_priors (prior_setting), 45
non_decreasing, 6, 7, 21, 22, 31, 32, 62, 64
non_increasing, 6, 7, 21, 22, 32, 62, 64
observe_data, 33
parameter_setting, 34
print.causal_model, 36
print.causal_types, 37
print.dag, 37
print.event_probabilities, 38
print.model_query, 38
print.nodal_types, 39
print.nodes, 39
print.parameter_mapping, 42
print.parameters, 40
print.parameters_df, 40
print.parameters_posterior, 41
print.parameters_prior, 41
print.parents_df, 42
print.posterior_event_probabilities, 43
print.stan_summary, 43
print.statement, 44
print.summary.causal_model
  (summary.causal_model), 63
print.type_distribution, 44
print.type_prior, 45
prior_setting, 45
query_distribution, 48
query_model, 51
realise_outcomes, 15, 53
restrict_by_labels, 60
restrict_by_query, 60
sampling, 65
set_ambiguities_matrix, 54
set_confound, 55
set_parameter_matrix, 56
set_parameters (parameter_setting), 34
set_parmap, 57
set_prior_distribution, 31, 58
set_priors (prior_setting), 45
set_restrictions, 58
simulate_data, 61
stan, 65
substitutes, 6, 7, 21, 22, 32, 62, 64
summarise_distribution, 63
summary.causal_model, 28, 63, 66
te, 6, 7, 21, 22, 32, 62, 64
update_model, 65