AdaptiveDirMRW

Description

This class implements an Adaptive Multi-site Metropolis random walk algorithm, constrained so
the parameter vector sums to 1.

Format

Object of R6Class with methods for updating a DirichletNode instance.

Details

An adaptive multivariate Gaussian proposal is used for $d-1$ elements of a $d$-dimensional pa-
parameter vector contained in node, with the $d$th element updated to ensure that the vector sums to
1. This makes the updater useful for Dirichlet distributed random variables.

For details of the adaptive scheme, see Roberts and Rosenthal (2012) Examples of Adaptive MCMC.
Please note that no checks are performed as to the suitability of this algorithm for a particular StochasticNode. It is up to the user to use the correct update algorithm for the appropriate nodes.

Value

Object of AdaptiveDirMRW

Fields

cov  the current covariance
burnin  the number of updates to burn in
tune  the current tuning matrix
naccept  the number of accepted proposals
ncalls  the number of times update has been called
node  the node to which the updater is attached

Methods

new(node, touupdate = function() 1:length(node$getData()), tune = rep(0.1, length(node$getData())), burning = 100)  constructor takes an instance of a StochasticNode node, function to choose the indices of the elements to update (by default all elements), initial tuning vector (diagonal of adaptive tuning matrix), and the number of calls between adaptations.

update() when called, updates node

acceptance() return the acceptance rate

Description

This class implements an Adaptive Multi-site logarithmic Metropolis-Hastings random walk algorithm, constrained so the parameter vector sums to 1.

Format

Object of R6Class with methods for updating a DirichletNode instance.

Details

An adaptive multivariate log-Gaussian proposal is used for $d-1$ elements of a $d$-dimensional parameter vector contained in node, with the $d$th element updated to ensure that the vector sums to 1. This makes the updater useful for Dirichlet distributed random variables, improving on AdaptiveDirMRW by ensuring proposals do not go negative.


Please note that no checks are performed as to the suitability of this algorithm for a particular StochasticNode. It is up to the user to use the correct update algorithm for the appropriate nodes.
Value

Object of AdaptiveLogDirMRW

Fields

cov  the current covariance
burnin  the number of updates to burn in
tune  the current tuning matrix
naccept  the number of accepted proposals
ncalls  the number of times update has been called
node  the node to which the updater is attached

Methods

new(node, toupdate = function() 1:length(node$getData()), tune = rep(0.1, length(node$getData())), burning = 100)
  constructor takes an instance of a StochasticNode node, function to choose the indices of the elements to update (by default all elements), initial tuning vector (diagonal of adaptive tuning matrix), and number of calls between adaptations.

update() when called, updates node

acceptance() return the acceptance rate

Description

This class implements an Adaptive Multi-site Metropolis random walk algorithm.

Format

Object of R6Class with methods for updating a Node instance.

Details


Please note that no checks are performed as to the suitability of this algorithm for a particular StochasticNode. It is up to the user to use the correct update algorithm for the appropriate nodes.

Value

Object of AdaptiveMultiMRW
Alpha

Fields

cov  the current covariance
burnin  the number of updates to burn in
tune  the current tuning matrix
naccept  the number of accepted proposals
ncalls  the number of times update has been called
node  the node to which the updater is attached

Methods

new(node, tune = rep(0.1, length(node$getData())), burning = 100) constructor takes an instance of a StochasticNode node, initial tuning vector (diagonal of adaptive tuning matrix), and number of burnin calls.
update() when called, updates node
acceptance() return the acceptance rate

Alpha  Constructs alpha prior

Description

The Alpha construct function returns an R6 Alpha_ class which feeds sanitised prior or initialisation values for alpha into the model.

Usage

Alpha(data, alpha, source, time = NULL, location = NULL)

Arguments

data  long-format data.frame containing Dirichlet prior hyperparameter, source, time, and location columns
alpha  name of hyperparameter column
source  name of source column
time  name of optional time column
location  name of optional location column

Value

An Alpha_ data structure for use in sourceR models
Alpha prior hyperparameter class

**Description**

Alpha prior hyperparameter class

---

campy

*Human cases of campylobacteriosis and numbers of source samples positive for Campylobacter.*

**Description**

A dataset containing the number of human cases of campylobacteriosis and numbers of source samples positive for *Campylobacter* for each bacterial subtype.

**Usage**

campy

**Format**

A list containing the human cases (‘cases’), source samples (‘sources’), and prevalences (‘prev’).

- **cases:** data frame with 91 rows and 2 variables:
  - **Human** number of human cases of campylobacteriosis between 2005-2008 in the Manawatu region of New Zealand
  - **Type** MLST type id for the samples

- **sources:** data frame with 690 rows and 3 variables:
  - **Count** number of source samples positive for campylobacteriosis
  - **Source** Source id for the samples
  - **Type** MLST type id for the samples

- **prev:** data frame with 6 rows and 4 variables
  - **Value** Prevalence value (number of positive samples divided by total number of samples)
  - **Source** Source id for the samples
  - **n_positive** number of positive source samples for campylobacter (PCR)
  - **n_total** total number of source samples tested for campylobacter
**DataNode**

<table>
<thead>
<tr>
<th>DataNode</th>
<th>DataNode</th>
</tr>
</thead>
</table>

**Description**

Represents a static data node in a DAG.

**Format**

Object of `R6Class` with methods for constructing a DAG.

**Details**

Derived from `Node`, please see base class documentation.

**Value**

Object of `DataNode`

**Fields**

- `data` the data

**Methods**

- `getData()` returns the node’s data.

---

**DirichletNode**

<table>
<thead>
<tr>
<th>DirichletNode</th>
<th>DirichletNode</th>
</tr>
</thead>
</table>

**Description**

Represents a d-dimensional Dirichlet distribution node in a DAG.

**Format**

Object of `R6Class` with methods for constructing a DAG.

**Details**

Derived from `StochasticNode`, please see base class documentation.

**Value**

Object of `DirichletNode`
Methods

new(data, alpha) Create a DirichletNode with data vector data \((\text{length} > 1)\) and parameter vector alpha.

DirichletProcessNode

Description
Represents a Dirichlet process as a single node in a DAG.

Format
Object of R6Class with methods for constructing a DAG.

Details
Derived from StochasticNode, please see base class documentation.

Value
Object of DirichletProcessNode

Methods

new(theta, s, alpha, base, ...) Create a DirichletProcessNode with value vector theta \((\text{length} > 1)\), initial grouping vector s, concentration parameter alpha, and base distribution base. Base should be a distribution function (dnorm, dgamma, etc) whose parameters are specified in ....

DPMModel_impl

Description
Builds the source attribution model. Is not intended to be used by a regular user. Developers only here!

Format
R6Class object.

Value
Object of R6Class.
Fields

- **y** 3D array of [type, time, location] of the number of human cases
- **X** 3D array of the number of positive samples for each type, source and time [type, source, time]
- **R** 3D array of normalised relative prevalences for each timepoint [type, source, time]
- **Time** a character vector of timepoint ids matching time dimension in y and R
- **Location** a character vector of location ids matching location dimension in y
- **Sources** a character vector of source ids matching the source dimension in X
- **Type** a character vector of type ids matching the type dimension in X
- **prev** a 2D array (matrix) of [source, time]
- **a_q** concentration parameter for the DP
- **a_theta** shape parameter for the Gamma base distribution for the DP
- **b_theta** rate parameter for the Gamma base distribution for the DP
- **a_r** 3D array of [type, src, time] for the hyperprior on the relative prevalences R
- **a_alpha** 3D array of [source, time, location] for the prior on the alpha parameters
- **s** vector giving initial group allocation for each type for the DP
- **theta** vector giving initial values for each group in the DP
- **alpha** 3D array of [source, time, location] giving initial values for the alpha parameters

Description

Represents a formula node in a DAG. Inherit from this node to specify some kind of formula within the DAG, e.g. a linear predictor and/or link function. Override the FormulaNode$getData() method to apply your own function.

Format

Object of R6Class with methods for constructing a DAG.

Details

Derived from Node, please see base class documentation.

Value

Object of FormulaNode

Methods

getData() returns the node’s transformed data.
Description

Represents a Gamma distribution node in a DAG. Requires parent nodes for shape and rate respectively as specified in \texttt{dgamma}.

Format

Object of \texttt{R6Class} with methods for constructing a DAG.

Details

Derived from \texttt{StochasticNode}, please see base class documentation.

Value

Object of \texttt{GammaNode}

Methods

\texttt{new(data, shape=1, rate=1)} Create a Gamma node with data \texttt{data} and Nodes shape and rate as specified in \texttt{dgamma}.

HaldDP

\textit{Builds a HaldDP source attribution model}

Description

Builds a HaldDP source attribution model

Usage

\texttt{HaldDP(y, x, k, priors, a_q, inits = NULL)}

Arguments

\begin{itemize}
  \item \texttt{y} a \texttt{Y} object containing case observations
  \item \texttt{x} an \texttt{X} object containing source observations
  \item \texttt{k} a \texttt{Prev} object containing source prevalences
\end{itemize}
priors list with elements named \( a_r, a_{\alpha}, a_{\theta} \) and \( b_{\theta} \), corresponding to the prior parameters for the \( r, \alpha, \) and base distribution for the DP parameters respectively.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Prior Distribution</th>
<th>Prior Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_r )</td>
<td>Dirichlet(concentration)</td>
<td>A single positive number or an ( X ) object containing the prior values for each source, time and type. If a single number is supplied, it will be used for all times, sources and types.</td>
</tr>
<tr>
<td>( a_{\alpha} )</td>
<td>Dirichlet(concentration)</td>
<td>A single positive number or an ( \text{Alpha} ) object containing the prior values for each source, time and location. If a single number is supplied, it will be used for all times, sources and locations.</td>
</tr>
<tr>
<td>Type effects base distribution parameters</td>
<td>Gamma(shape, rate)</td>
<td>Single number for each of the shape (( a_{\theta} )) and rate (( b_{\theta} )) of the Gamma base distribution.</td>
</tr>
</tbody>
</table>

\( a_q \) the Dirichlet Process concentration parameter.

\textit{inits} initial values for the \textit{mcmc} algorithm. This is an optional list that may contain any of the following items: \( \alpha, q, \) and \( r \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r )</td>
<td>An object of type ( X ) giving the initial values for ( SRS ) matrix. If not specified defaults to the element-wise maximum likelihood estimates of ( r ) from the source matrix.</td>
</tr>
<tr>
<td>Source effects (( \alpha ))</td>
<td>An object of type ( \text{Alpha} ) specifying alpha value for each source/time/location. If not specified, default initial values for the source effects are drawn from the prior distribution.</td>
</tr>
<tr>
<td>Type effects (( q ))</td>
<td>An object of type ( Q ) giving the initial clustering and values for ( q ). If not specified, defaults to a single group with a theta value calculated as ( \theta = \frac{\sum y_{itl}}{\sum l} = 1^L (\sum n_i = 1^T (\sum m_j = 1^m (\alpha_{ijl} * r_{ijl} * k_{jtl}))) ). i.e. ( \theta = \frac{\sum y_{itl}}{\sum \lambda_{ijl} / \theta} ).</td>
</tr>
</tbody>
</table>

\textbf{Format}

Object of \texttt{R6Class} with methods for creating a HaldDP model, running the model, and accessing and plotting the results.

\textbf{Value}

Object of \texttt{HaldDP} with methods for creating a HaldDP model, running the model, and accessing and plotting the results.

\textbf{Description}

This function fits a non-parametric Poisson source attribution model for human cases of disease. It supports multiple types, sources, times and locations. The number of human cases for each type, time and location follow a Poisson likelihood.
HaldDP Object Methods

mcmc_params(n_iter = 1000, burn_in = 0, thin = 1, n_r = ceiling(private$nTypes * 0.2), update_schema = c('q', 'alpha', 'r'))

do when called, sets the mcmc parameters.

n_iter sets the number of iterations returned (after removing burn_in and thinning results by thin i.e. a total of (n_iter * thin) + burn_in iterations are run)
n_r is a positive integer that sets the total number of r_{ijtl} parameters to be updated at each time-location-source combination (the default is 20 percent updated per iteration)
update_schema a character vector containing the parameters to update (any of 'q', 'alpha', 'r').

update(n_iter, append = TRUE) when called, updates the HaldDP model by running n_iter iterations.

If missing n_iter, the n_iter last set using mcmc_params() or update() is used.
append is a logical value which determines whether the next n_iter iterations are appended to any previous iterations, or overwrites them. When append = TRUE, the starting values are the last iteration and no burn_in is removed. Running the model for the first time, or changing any model or fitting parameters will set append = FALSE.

get_data returns a list containing the human data y (an array y[types, times, locations]), the source data X (an array X[types, sources, times]), the prevalence data (an array k[sources, times]), the type names, source names, time names, location names and number of different types, sources, times and locations.

get_priors returns a list containing the DP concentration parameter a_q, and the priors (R6 class with members named a_alpha (members are array a_alpha[sources, times, locations]), a_r (an array a_r[types, sources, times]), a_theta and b_theta).

get_inits returns an R6 class holding the initial values (members are alpha (an array alpha[sources, times, locations]), theta (an array theta[types, iters]), s (an array s[types, iters]), and r (an array r[types, sources, times])).

get_mcmc_params returns a list of fitting parameters (n_iter, append, burn_in, thin, update_schema)

(R6 class with members alpha, q, r)).

get_acceptance returns an R6 class containing the acceptance rates for each parameter (members are alpha (an array alpha[sources, times, locations]), and r (an array r[types, sources, times]).

extract(params = c("alpha", "q", "s", "r", "lambda_i", "xi", "xi_prop"), times = NULL, locations = NULL, sources = NULL, types = NULL, iters = NULL, flatten = FALSE, drop = TRUE)

returns a list containing a subset of the parameters (determined by the params vector, times, locations, sources, types and iters).

If flatten is set to TRUE, it returns a dataframe with 1 column per parameter, otherwise it returns a list containing params containing a subset of the following arrays: alpha[Sources,Times,Locations,iters], q[Types,iters], s[Types,iters], r[Types,Sources,Times,iters], lambda_i[Types,Times,Locations,iters], xi[Sources,Times,Locations,iters].

drop determines whether to delete the dimensions of an array which have only one level when flatten = FALSE.

summary(alpha = 0.05, params = c("alpha", "q", "s", "r", "lambda_i", "xi", "xi_prop"), times = NULL, locations = NULL, sources = NULL, types = NULL, iters = NULL, flatten = FALSE)

returns a list containing the median and credible intervals for a subset of the parameters. The default credible interval type are Chen-Shao ("chen-shao") highest posterior density intervals (alternatives are "percentiles" and "spin"). See extract for details on the subsetting.

xi_prop returns the proportion of cases attributed to each source j and is calculated by dividing each iteration of lambda_{ijtl} values by their sum within each time t and location l.
plot_heatmap(iters, cols = c("blue","white"), hclust_method = "complete") Creates a dendrogram and heatmap for the type effect groupings (s parameter in the model). This uses the heatmap.2 function from gplots.

iters is a vector containing the iterations to be used in constructing the graph. Default is all iterations in posterior.

hclust_method allows the user to select the method used by stats::hclust to cluster the type effect groupings s.

cols gives the colours for completely dissimilar (dissimilarity value of 1), and identical (dissimilarity value of 0). All other values will be in between the two chosen colours. See ?colorRampPalette for more details..

Details

This function fits a source attribution model for human cases of disease. It supports multiple types, sources, times and locations. The number of human cases for each type, time and location follows a Poisson or Negative Binomial likelihood. Model

\[ y_{itl} \sim \text{Poisson}(\lambda_{itl}) \]

where

\[ \lambda_{itl} = \sum_{j=1}^{m} \lambda_{ijtl} = q_{s(i)} \sum_{j=1}^{m} (r_{ijt} \cdot k_{j} \cdot alpha_{jtl}) \]

The parameters are defined as follows:

\( a_{jtl} \)

is the unknown source effect for source \( j \), time \( t \), location \( l \)

\( q_{s(i)} \)

is the unknown type effect for type \( i \) in group \( s \).

\( x_{ij} \)

is the known number of positive samples for each source \( j \) type \( i \) combination

\( n_{ij} \)

is the known total number of samples for each source \( j \) type \( i \) combination

\( k_{j} \)

is the fixed prevalence in source (i.e. the number of positive samples divided by the number of negative samples) \( j \)

\( r_{ijt} \)

is the unknown relative occurrence of type \( i \) on source \( j \).

Priors

\[ r_{ijt} \sim \text{Dirichlet}(a_{r_{ijt}}, \ldots, a_{n_{ijt}}) \]

\[ a_{tl} \sim \text{Dirichlet}(a_{\alpha_{tl}}, \ldots, a_{\alpha_{mtl}}) \]

\[ q \sim \text{DP}(a_{q}, \text{Gamma}(a_{\theta}, b_{\theta})) \]
Examples

```r
### Format data using Y, X, and Prev functions

#### Input data must be in long format

```r
y <- Y(
  data = sim_SA$cases,
  y = "Human",
  type = "Type",
  time = "Time",
  location = "Location"
)
```

```r
x <- X(
  data = sim_SA$sources,
  x = "Count",
  type = "Type",
  time = "Time",
  source = "Source"
)
```

```r
k <- Prev(
  data = sim_SA$prev,
  prev = "Value",
  time = "Time",
  source = "Source"
)
```

```r
### Create Dirichlet(1) priors

#### Create alpha prior data frame

```r
prior_alpha_long <- expand.grid(
  Source = unique(sim_SA$sources$Source),
  Time = unique(sim_SA$sources$Time),
  Location = unique(sim_SA$cases$Location),
  Alpha = 1
)
```

```r
# Use the Alpha() constructor to specify alpha prior
prior_alpha <- Alpha(
  data = prior_alpha_long,
  alpha = 'Alpha',
)
HaldDP

source = 'Source',
time = 'Time',
location = 'Location'
)

## Create r prior data frame
prior_r_long <- expand.grid(
  Type = unique(sim_SA$sources$Type),
  Source = unique(sim_SA$sources$Source),
  Time = unique(sim_SA$sources$Time),
  Value = 0.1
)
# Use X() constructor to specify r prior
prior_r <- X(
  data = prior_r_long,
  x = 'Value',
  type = 'Type',
  time = 'Time',
  source = 'Source'
)

## Pack all priors into a list
priors <- list(
  a_theta = 0.01,
  b_theta = 0.00001,
  a_alpha = prior_alpha,
  a_r = prior_r
)

## If all prior values are the same, they can be specified in shorthand
## Equivalent result to the longform priors specified above
priors <- list(
  a_theta = 0.01,
  b_theta = 0.00001,
  a_alpha = 1,
  a_r = 0.1
)

#### Set initial values (optional) ##########################################
types <- unique(sim_SA$cases$Type)
q_long <- data.frame(q=rep(15, length(types)), Type=types)
init_q <- Q(q_long, q = 'q', type = 'Type')
inits <- list(q = init_q) # Pack starting values into a list

#### Construct model ########################################################
my_model <- HaldDP(y = y, x = x, k = k, priors = priors, inits = inits, a_q = 0.1)

#### Set mcmc parameters ####################################################
my_model$mcmc_params(n_iter = 2, burn_in = 2, thin = 1)

#### Update model ###########################################################
my_model$update()
## Add an additional 10 iterations
my_model$update(n_iter = 2, append = TRUE)

### Extract posterior ####################################################################################
## returns the posterior for the r, alpha, q, c, 
## lambda_i, xi and xi_prop parameters,
## for all times, locations, sources and types
## the posterior is returned as a list or arrays
## Not run: str(my_model$extract())
## returns the posterior for the r and alpha parameters, 
## for time 1, location B, sources Source3, and Source4, 
## types 5, 25, and 50, and iterations 200:300 
## the posterior is returned as a list of dataframes
## Not run:
str(my_model$extract(params = c("r", "alpha"),
                     times = "1", location = "B",
                     sources = c("Source3", "Source4"),
                     types = c("5", "25", "50"),
                     iters = 5:15,
                     flatten = TRUE))
## End(Not run)

### Calculate medians and credible intervals ################################################################
## Not run: my_model$summary(alpha = 0.05, CI_type = "chen-shao")
## subsetting is done in the same way as extract()
## Not run: my_model$summary(alpha = 0.05, CI_type = "chen-shao",
                          params = c("r", "alpha"),
                          times = "1", location = "B",
                          sources = c("Source3", "Source4"),
                          types = c("5", "25", "50"),
                          iters = 5:15,
                          flatten = TRUE)
## End(Not run)

### Plot heatmap and dendrogram of the type effect grouping ################################################################################
my_model$plot_heatmap()

### Extract data, initial values, prior values, acceptance rates for the mcmc algorithm and mcmc parameters
my_model$get_data()
my_model$get_inits()
my_model$get_priors()
my_model$get_acceptance()
my_model$get_mcmc_params()
Description

This is a base class representing a node in a DAG. It is not intended to be used by a regular user. Developers only here!

Format

Object of \texttt{R6Class} with methods for constructing a DAG.

Value

Object of \texttt{Node}

Fields

\begin{itemize}
  \item parents a list of parent nodes
  \item children a list of child nodes
  \item name a tag name applied to the node
\end{itemize}

Methods

\begin{itemize}
  \item \texttt{new(parents = list(), children = list(), name)} creates a new \texttt{Node} with parent nodes, child nodes, and a name.
  \item \texttt{logDensity()} calculate the log probability density/mass function evaluated at the current node value.
  \item \texttt{addChild(node)} add node as a child. Returns node.
  \item \texttt{addParent(node)} add node as a parent. Returns node.
  \item \texttt{removeParent(name)} remove the parent node named \texttt{name}. Returns node.
  \item \texttt{removeChild(name)} remove the child node named \texttt{name}. Returns node.
\end{itemize}

Description

This class implements a marginal Gibbs sampler for a Dirichlet process prior on the mean of a Poisson distributed random variable, with a Gamma-distributed base function.

Format

Object of \texttt{R6Class} with methods for updating a \texttt{DirichletProcessNode} instance.

Details

The marginal Gibbs sampler is based on the description in Gelman et al. Bayesian Data Analysis, 3rd edition, Chapter 23.

Please note that no checks are performed as to the suitability of this algorithm for a particular \texttt{StochasticNode}. It is up to the user to use the correct update algorithm for the appropriate nodes.
**Value**

Object of `PoisGammaDPUpdate`

**Methods**

- `new(node)` constructor takes an instance of a `DirichletProcessNode` node
- `update()` when called, updates `node`

---

**PoissonNode**

**Description**

Represents a Poisson distribution node in a DAG

**Format**

Object of `R6Class` with methods for constructing a DAG.

**Details**

Derived from `StochasticNode`, please see base class documentation.

**Value**

Object of `PoissonNode`

**Methods**

- `new(data, lambda, offset)` create a PoissonNode, with mean `Node` lambda, and offset `Node` offset.

---

**Prev**

**Description**

The Prev constructor function returns an R6 Prevalence class which feeds data into sourceR models.

**Usage**

`Prev(data, prev, source, time = NULL)`
Q

Arguments

data long-format data.frame containing prevalence data by source and time.
prev character string giving name of prevalence column in data
source character string giving name of source column in data
time optional column denoting times of prevalence observation

Value

A Prev data structure for use in sourceR models

Q Constructs initial values for q

Description

The Q constructor returns a R6 Q_ class which feeds sanitised initial values for q into the model.

Usage

Q(data, q, type)

Arguments

data long-format data.frame containing type-name and q-value for each observation.
q name of 'q' column
type name of type column

Value

A Q_ data structure for use in sourceR models

sim_SA Simulated data: Human cases of campylobacteriosis and numbers of source samples positive for Campylobacter.

Description

A simulated dataset containing the number of human cases of campylobacteriosis, the numbers of source samples positive for Campylobacter for each bacterial subtype, and the overall source prevalence.

Usage

sim_SA
Format

A list containing the human cases (‘cases’), source samples (‘sources’), prevalences (‘prev’) and true values (‘truevals’).

cases: data frame with 364 rows and 4 variables:

Human  number of human cases of campylobacteriosis
Time   Time id for the samples
Location Location id for the samples
Type   MLST type id for the samples

sources: data frame with 1092 rows and 4 variables

Count  number of source samples positive for campylobacteriosis
Time   Time id for the samples
Source  Source id for the samples
Type   MLST type id for the samples

prev: data frame with 12 rows and 3 variables

Value  Prevalence value (number of positive samples divided by total number of samples)
Time   Time id for the samples
Source  Source id for the samples

ttruevals: list containing a long format data frame for each model parameter giving the true value of that parameter.

alpha  A dataframe with 24 rows and 4 variables: Value contains the true alpha values, Time, Location and Source contain the time, location and source id’s respectively.
q  A dataframe with 91 rows and 2 variables: Value contains the true q values, and Type contains the type id’s.

lambda_i  A dataframe with 364 rows and 4 variables: Value contains the true lambda_i values, Time, Location and Type contain the time, location and type id’s respectively.
xi  A dataframe with 24 rows and 4 variables: Value contains the true xi values, Time, Location and Source contain the time, location and source id’s respectively.
r  A dataframe with 2184 rows and 5 variables: Value contains the true r values, Time, Type, Location and Source contain the time, type, location and source id’s respectively.
sliceTensor

Slices a tensorA::tensor

Description
Slices a tensorA::tensor, preserving the dimnames. This is a workaround for a buggy implementation of [.tensor as of tensorA v0.36.

Usage
sliceTensor(x, ...)

Arguments
x tensor array to be sliced
... arguments used to subset the tensor array

sourceR

sourceR: A package for fitting Bayesian non-parametric source attribution models.

Description
The sourceR package currently provides an S6 class called HaldDP to fit the HaldDP model. Further source attribution models are planned to be added to the package.

sourceR methods
The main sourceR method is HaldDP.

StochasticNode

StochasticNode

Description
Represents a stochastic node in a DAG

Format
Object of R6Class with methods for constructing a DAG.

Details
Derived from Node, please see base class documentation.
Value

Object of StochasticNode

Fields

data contains the node’s data

Methods

logPosterior() return the value of the log posterior distribution of the node.
getData() returns the node’s data.

---

X Constructs source data

Description

The X constructor function returns an R6 X class which feeds source data into sourceR models.

Usage

X(data, x, type, source, time = NULL)

Arguments

data long-format data.frame containing source data
x character string giving name of source counts column in data
type character string giving name of type column in data
source character string giving name of source column in data
time optional column denoting times of source observation

Value

A X source data structure for use in sourceR models
Y

Constructs disease count data

Description
The Y constructor function returns an R6 Y class which feeds disease count data into sourceR models.

Usage
Y(data, y, type, time = NULL, location = NULL)

Arguments
- data: long-format data.frame containing source data
- y: character string giving name of disease counts column in data
- type: character string giving name of type column in data
- time: optional column denoting times of disease count observations
- location: optional column denoting location of disease count observations

Value
A Y disease count data structure for use in sourceR models
## Index

**DAG**
- AdaptiveDirMRW, 2
- AdaptiveLogDirMRW, 3
- AdaptiveMultiMRW, 4
- DataNode, 7
- DirichletNode, 7
- DirichletProcessNode, 8
- FormulaNode, 9
- GammaNode, 10
- Node, 17
- PoisGammaDPUpdate, 17
- PoissonNode, 18
- StochasticNode, 21

**Dirichlet-process**
- PoisGammaDPUpdate, 17

**MCMC**
- AdaptiveDirMRW, 2
- AdaptiveLogDirMRW, 3
- AdaptiveMultiMRW, 4

**datasets**
- campy, 6
- sim_SA, 19

**node**
- AdaptiveDirMRW, 2
- AdaptiveLogDirMRW, 3
- AdaptiveMultiMRW, 4
- DataNode, 7
- DirichletNode, 7
- DirichletProcessNode, 8
- FormulaNode, 9
- GammaNode, 10
- Node, 17
- PoisGammaDPUpdate, 17
- PoissonNode, 18
- StochasticNode, 21

AdaptiveDirMRW, 2, 3
AdaptiveLogDirMRW, 3, 4
AdaptiveMultiMRW, 4, 4
Alpha, 5, 11