Package 'hmmTMB'

January 13, 2023

Type Package

Title Fit Hidden Markov Models using Template Model Builder

Version 1.0.1

Maintainer Theo Michelot <theo.michelot@dal.ca>

Description Fitting hidden Markov models using automatic differentiation and Laplace approximation, allowing for fast inference and flexible covariate effects (including random effects and smoothing splines) on model parameters. The package is described by Michelot (2022) <arXiv:2211.14139>.

URL https://github.com/TheoMichelot/hmmTMB

License GPL-3

Depends R6, mgcv, TMB, ggplot2

Imports Matrix, stringr, optimx, CircStats, MASS, tmbstan, methods

LinkingTo TMB, RcppEigen

Suggests rstan, testthat, knitr, moveHMM, scico, MSwM, unmarked

Encoding UTF-8

RoxygenNote 7.2.2

VignetteBuilder knitr

NeedsCompilation yes

Author Theo Michelot [aut, cre], Richard Glennie [aut, ctb]

Repository CRAN

Date/Publication 2023-01-13 09:10:02 UTC

R topics documented:

`as_character_formula` *Read formula with as.character without splitting*

Description

Read formula with as.character without splitting

Usage

```
as_character_formula(x, ...)
```

Arguments

x	R formula
...	Unused

Details

Citation: this function was taken from the R package formula.tools: Christopher Brown (2018). formula.tools: Programmatic Utilities for Manipulating Formulas, Expressions, Calls, Assignments and Other R Objects. R package version 1.7.1. https://CRAN.R-project.org/package=formula.tools

as_sparse *Transforms matrix to dgTMatrix*

Description

Transforms matrix to dgTMatrix

Usage

```
as_sparse(x)
```

Arguments

x	Matrix or vector. If this is a vector, it is formatted into a single-column matrix.

Value

Sparse matrix of class dgTMatrix

bdiag_check *Create block diagonal matrix (safe version)*

Description

This version of bdiag checks whether the matrices passed as arguments are NULL. This avoids errors that would arise if using bdiag directly.

Usage

```
bdiag_check(...)
```

Arguments

...	Matrix or list of matrices (only the first argument is used)

Value

Block diagonal matrix

cov_grid *Grid of covariates*

Description

Grid of covariates

Usage

```
cov_grid(var, data = NULL, obj = NULL, covs = NULL, formulas, n_grid = 1000)
```

Arguments

var	Name of variable
data	Data frame containing the covariates. If not provided, data are extracted from obj
obj	HMM model object containing data and formulas
covs	Optional named list for values of covariates (other than 'var') that should be used in the plot (or dataframe with single row). If this is not specified, the mean value is used for numeric variables, and the first level for factor variables.
formulas	List of formulas used in the model
n_grid	Grid size (number of points). Default: 1000.

Value

Data frame of covariates, with 'var' defined over a grid, and other covariates fixed to their mean (numeric) or first level (factor).

Dist *R6 class for probability distribution*

Description

R6 class for probability distribution

R6 class for probability distribution

Details

Contains the probability density/mass function, and the link and inverse link functions for a probability distribution.

Methods

Public methods:

- Dist$new()
- Dist$name()
- Dist$pdf()
- Dist$rng()
- Dist$link()
- Dist$invlink()
- Dist$npar()
- Dist$parnames()
- Dist$parapprox()
- Dist$fixed()
- Dist$code()
- Dist$name_long()
- Dist$set_npar()
- Dist$set_parnames()
- Dist$set_code()
- Dist$pdf_apply()
- Dist$rng_apply()
- Dist$n2w()
- Dist$w2n()
- Dist$clone()

Method `new()`**:** Create a Dist object

Usage:

```
Dist$new(
  name,
  pdf,
  rng,
  link,
  invlink,
  npar,
  parnames,
  parapprox = NULL,
  fixed = NULL,
  name_long = name
)
```

Arguments:

`name` Name of distribution

`pdf` Probability density/mass function of the distribution (e.g. `dnorm` for normal distribution).

`rng` Random generator function of the distribution (e.g. `rnorm` for normal distribution).

`link` Named list of link functions for distribution parameters

`invlink` Named list of inverse link functions for distribution parameters

`npar` Number of parameters of the distribution

`parnames` Character vector with name of each parameter

`parapprox` Function that takes a sample and produces approximate values for the unknown parameters

`fixed` Vector with element for each parameter which is TRUE if parameter is fixed

`name_long` Long version of the name of the distribution, possibly more user-readable than name.

Returns: A new Dist object

Method `name()`**:** Return name of Dist object

Usage:

```
Dist$name()
```

Method `pdf()`**:** Return pdf of Dist object

Usage:

```
Dist$pdf()
```

Method `rng()`**:** Return random generator function of Dist object

Usage:

```
Dist$rng()
```

Method `link()`**:** Return link function of Dist object

Usage:

```
Dist$link()
```

Method `invlink()`**:** Return inverse link function of Dist object

Usage:

```
Dist$invlink()
```

Method `npar()`**:** Return number of parameters of Dist object

Usage:

```
Dist$npar()
```

Method `parnames()`**:** Return names of parameters

Usage:

```
Dist$parnames()
```

Method `parapprox()`**:** Return function that approximates parameters

Usage:

```
Dist$parapprox()
```

Method `fixed()`**:** Return which parameters are fixed

Usage:

```
Dist$fixed()
```

Method `code()`**:** Return code of Dist object

Usage:

```
Dist$code()
```

Method `name_long()`**:** Human-readable name of Dist object

Usage:

```
Dist$name_long()
```

Method `set_npar()`**:** Set number of parameters this distribution has

Usage:

```
Dist$set_npar(new_npar)
```

Arguments:

`new_npar` Number of parameters

Method `set_parnames()`**:** Set parameter names

Usage:

```
Dist$set_parnames(new_parnames)
```

Arguments:

`new_parnames` Parameter names

Method `set_code()`**:** Set distribution code

Usage:

```
Dist$set_code(new_code)
```

Arguments:

`new_code` Distribution code

Method `pdf_apply()`**:** Evaluate probability density/mass function

This method is used in the Dist$obs_probs() method. It is a wrapper around Dist$pdf(), which prepares the parameters and passes them to the function.

Usage:

```
Dist$pdf_apply(x, par, log = FALSE)
```

Arguments:

`x` Value at which the function should be evaluated

`par` Vector of parameters. The entries should be named if they are not in the same order as expected by the R function. (E.g. shape/scale rather than shape/rate for gamma distribution.)

`log` Logical. If TRUE, the log-density is returned. Default: FALSE.

Returns: Probability density/mass function evaluated at x for parameters par

Method `rng_apply()`**:** Random number generator

This method is a wrapper around Dist$rng(), which prepares the parameters and passes them to the function.

Usage:

```
Dist$rng_apply(n, par)
```

Arguments:

`n` Number of realisations to generate

`par` Vector of parameters. The entries should be named if they are not in the same order as expected by the R function. (E.g. shape/scale rather than shape/rate for gamma distribution.)

Returns: Vector of `n` realisations of this distribution

Method `n2w()`**:** Natural to working parameter transformation

This method transforms parameters from the natural scale (i.e., their domain of definition) to the "working" or "linear predictor" scale (i.e., the real line). It is a wrapper for Dist$link().

Usage:

```
Dist$n2w(par)
```

Arguments:

`par` List of parameters

Returns: Vector of parameters on the working scale

Method `w2n()`**:** Working to natural parameter transformation

This method transforms parameters from the "working" or "linear predictor" scale (i.e., the real line) to the natural scale (i.e., their domain of definition). It is a wrapper for Dist$invlink().

Usage:

```
Dist$w2n(wpar, as_matrix = FALSE)
```

Arguments:

`wpar` Vector of working parameters

`as_matrix` Logical. If TRUE, the natural parameters are returned as a matrix with one row for each state and one column for each parameter. If FALSE, the natural parameters are returned as a list (default).

Returns: List or matrix of parameters on natural scale

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

Usage:

```
Dist$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`find_re` *Find s(, bs = "re") terms in formula*

Description

This function is used to identify the variables "x" which are included as s(x, bs = "re") in the formula, in particular to check that they are factors.

Usage

```
find_re(form)
```

Arguments

form	Model formula

Value

Vector of names of variables for which a random effect term is included in the model.

HMM *R6 class for hidden Markov model*

Description

R6 class for hidden Markov model

R6 class for hidden Markov model

Details

Encapsulates the observation and hidden state models for a hidden Markov model.

Methods

Public methods:

- HMM$new()
- HMM$obs()
- HMM$hid()
- HMM$out()
- HMM$tmb_obj()
- HMM$tmb_obj_joint()
- HMM$tmb_rep()
- HMM$states()
- HMM$coeff_fe()
- HMM$coeff_re()
- HMM$coeff_list()
- HMM$fixpar()
- HMM$coeff_array()
- HMM$lambda()
- HMM$update_par()
- HMM$sd_re()
- HMM$par()
- HMM$set_priors()

- HMM$priors()
- HMM$iters()
- HMM$out_stan()
- HMM$llk()
- HMM$edf()
- HMM$setup()
- HMM$fit_stan()
- HMM$fit()
- HMM$mle()
- HMM$forward_backward()
- HMM$cond()
- HMM$pseudores()
- HMM$viterbi()
- HMM$sample_states()
- HMM$state_probs()
- HMM$post_coeff()
- HMM$post_linpred()
- HMM$post_fn()
- HMM$predict()
- HMM$confint()
- HMM$simulate()
- HMM$check()
- HMM$plot_ts()
- HMM$plot_dist()
- HMM$plot()
- HMM$AIC_marginal()
- HMM$AIC_conditional()
- HMM$print_obspar()
- HMM$print_tpm()
- HMM$formulation()
- HMM$print()
- HMM$clone()

Method `new()`**:** Create new HMM object

Usage:

```
HMM$new(obs = NULL, hid = NULL, file = NULL, init = NULL, fixpar = NULL)
```

Arguments:

`obs` Observation object, created with `Observation$new()`. This contains the formulation for the observation model.

`hid` MarkovChain object, created with `MarkovChain$new()`. This contains the formulation for the state process model.

`file` Path to specification file for HMM. If this argument is used, then `obs` and `hid` are unnecessary.

`init` HMM object, used to initialise the parameters for this model. If `init` is passed, then all parameters that are included in init and in the present model are copied. This may be useful when fitting increasingly complex models: start from a simple model, then pass it as init to create a more complex model, and so on.

`fixpar` Named list, with optional elements: 'hid', 'obs', 'delta0', 'lambda_obs', and 'lambda_hid'. Each element is a named vector of parameters in coeff_fe that should either be fixed (if the corresponding element is set to NA) or estimated to a common value (using integers or factor levels).don See examples in the vignettes, and check the TMB documentation to understand the inner workings (argument `map` of `TMB::MakeADFun()`).

Returns: A new HMM object

Examples:

```
# Load data set (included with R)
data(nottem)
data <- data.frame(temp = as.vector(t(nottem)))

# Create hidden state and observation models
hid <- MarkovChain$new(data = data, n_states = 2)
par0 <- list(temp = list(mean = c(40, 60), sd = c(5, 5)))
obs <- Observation$new(data = data, n_states = 2,
                       dists = list(temp = "norm"),
                       par = par0)

# Create HMM
hmm <- HMM$new(hid = hid, obs = obs)
```

Method `obs()`**:** Observation object for this model

Usage:

```
HMM$obs()
```

Method `hid()`**:** MarkovChain object for this model

Usage:

```
HMM$hid()
```

Method `out()`**:** Output of optimiser after model fitting

Usage:

```
HMM$out()
```

Method `tmb_obj()`**:** Model object created by TMB. This is the output of the TMB function `MakeADFun`, and it is a list including elements

- `fn`Objective function
- `gr`Gradient function of fn
- `par`Vector of initial parameters on working scale

Usage:

```
HMM$tmb_obj()
```

Method `tmb_obj_joint()`: Model object created by TMB for the joint likelihood of the fixed and random effects. This is the output of the TMB function `MakeADFun`, and it is a list including elements

- `fn`Objective function
- `gr`Gradient function of fn
- `par`Vector of initial parameters on working scale

Usage:

```
HMM$tmb_obj_joint()
```

Method `tmb_rep()`: Output of the TMB function `sdreport`, which includes estimates and standard errors for all model parameters.

Usage:

```
HMM$tmb_rep()
```

Method `states()`: Vector of estimated states, after `viterbi` has been run

Usage:

```
HMM$states()
```

Method `coeff_fe()`: Coefficients for fixed effect parameters

Usage:

```
HMM$coeff_fe()
```

Method `coeff_re()`: Coefficients for random effect parameters

Usage:

```
HMM$coeff_re()
```

Method `coeff_list()`: List of all model coefficients

These are the parameters estimated by the model, including fixed and random effect parameters for the observation parameters and the transition probabilities, (transformed) initial probabilities, and smoothness parameters.

Usage:

```
HMM$coeff_list()
```

Method `fixpar()`: Fixed parameters

Usage:

```
HMM$fixpar(all = FALSE)
```

Arguments:

`all` Logical. If FALSE, only user-specified fixed parameters are returned, but not parameters that are fixed by definition (e.g., size of binomial distribution).

Method `coeff_array()`: Array of working parameters

Usage:

```
HMM$coeff_array()
```

Method `lambda()`: Smoothness parameters

Usage:

```
HMM$lambda()
```

Method `update_par()`**:** Update parameters stored inside model object

Usage:

```
HMM$update_par(par_list = NULL, iter = NULL)
```

Arguments:

`par_list` List with elements for coeff_fe_obs, coeff_fe_hid, coeff_re_obs, coeff_re_hid, log_delta0, log_lambda_hid, and log_lambda_obs

`iter` Optional argument to update model parameters based on MCMC iterations (if using rstan). Either the index of the iteration to use, or "mean" if the posterior mean should be used.

Method `sd_re()`**:** Standard deviation of smooth terms (or random effects)

This function transforms the smoothness parameter of each smooth term into a standard deviation, given by SD = 1/sqrt(lambda). It is particularly helpful to get the standard deviations of independent normal random effects.

Usage:

```
HMM$sd_re()
```

Returns: List of standard deviations for observation model and hidden state model.

Method `par()`**:** Model parameters

Usage:

```
HMM$par(t = 1)
```

Arguments:

`t` returns parameters at time t, default is t = 1

Returns: A list with elements:

- `obspar`Parameters of observation model
- `tpm`Transition probability matrix of hidden state model

Method `set_priors()`**:** Set priors for coefficients

Usage:

```
HMM$set_priors(new_priors = NULL)
```

Arguments:

`new_priors` is a list of matrices for optionally coeff_fe_obs, coeff_fe_hid, log_lambda_obs log_lambda_hid each matrix has two rows (first row = mean, second row = sd) specifying parameters for Normal priors

Method `priors()`**:** Extract stored priors

Usage:

```
HMM$priors()
```

Method `iters()`**:** Iterations from stan MCMC fit

Usage:

```
HMM$iters(type = "response")
```

Arguments:

`type` Either "response" for parameters on the response (natural) scale, or "raw" for parameters on the linear predictor scale.

Returns: see output of as.matrix in stan

Method `out_stan()`**:** fitted stan object from MCMC fit

Usage:

```
HMM$out_stan()
```

Returns: the stanfit object

Method `llk()`**:** Log-likelihood at current parameters

Usage:

```
HMM$llk()
```

Returns: Log-likelihood

Method `edf()`**:** Effective degrees of freedom

Usage:

```
HMM$edf()
```

Returns: Number of effective degrees of freedom (accounting for flexibility in non-parametric terms implied by smoothing)

Method `setup()`**:** TMB setup

This creates an attribute `tmb_obj`, which can be used to evaluate the negative log-likelihood function.

Usage:

```
HMM$setup(silent = TRUE)
```

Arguments:

`silent` Logical. If TRUE, all tracing outputs are hidden (default).

Method `fit_stan()`**:** Fit model using tmbstan

Consult documentation of the tmbstan package for more information. After this method has been called, the Stan output can be accessed using the method `out_stan()`. This Stan output can for example be visualised using functions from the rstan package. The parameters stored in this HMM object are automatically updated to the mean posterior estimate, although this can be changed using `update_par()`.

Usage:

```
HMM$fit_stan(..., silent = FALSE)
```

Arguments:

`...` Arguments passed to tmbstan

`silent` Logical. If FALSE, all tracing outputs are shown (default).

Method `fit()`**:** Model fitting

The negative log-likelihood of the model is minimised using the function `optimx`. TMB uses the Laplace approximation to integrate the random effects out of the likelihood.

After the model has been fitted, the output of `optim` can be accessed using the method `out`.

Usage:

```
HMM$fit(silent = FALSE, ...)
```

Arguments:

`silent` Logical. If FALSE, all tracing outputs are shown (default).

`...` Other arguments to optimx which is used to optimise likelihood, see ?optimx

Examples:

```
# Load data set (included with R)
data(nottem)
data <- data.frame(temp = as.vector(t(nottem)))

# Create hidden state and observation models
hid <- MarkovChain$new(data = data, n_states = 2)
par0 <- list(temp = list(mean = c(40, 60), sd = c(5, 5)))
obs <- Observation$new(data = data, n_states = 2,
                       dists = list(temp = "norm"),
                       par = par0)

# Create HMM
hmm <- HMM$new(hid = hid, obs = obs)

# Fit HMM
hmm$fit(silent = TRUE)
```

Method `mle()`**:** Get maximum likelihood estimates once model fitted

Usage:

```
HMM$mle()
```

Returns: list of maximum likelihood estimates as described as input for the function update_par()

Method `forward_backward()`**:** Forward-backward algorithm

The forward probability for time step t and state j is the joint pdf/pmf of observations up to time t and of being in state j at time t, p(Z[1], Z[2], ..., Z[t], S[t] = j). The backward probability for time t and state j is the conditional pdf/pmf of observations between time t + 1 and n, given state j at time t, p(Z[t+1], Z[t+2], ..., Z[n] | S[t] = j). This function returns their logarithm, for use in other methods `state_probs`, and `sample_states`.

Usage:

```
HMM$forward_backward()
```

Returns: Log-forward and log-backward probabilities

Method `cond()`**:** Compute conditional cumulative distribution functions

Usage:

```
HMM$cond(ngrid = 1000, silent = FALSE)
```

Arguments:

`ngrid` how many cells on the grid that CDF is computed on

`silent` if TRUE then no messages are printed

Returns: cdfs on grid for each variable

Method `pseudores()`**:** Pseudo-residuals

Compute pseudo-residuals for the fitted model. If the fitted model is the "true" model, the pseudo-residuals follow a standard normal distribution. Deviations from normality suggest lack of fit.

Usage:

```
HMM$pseudores()
```

Returns: Matrix of pseudo-residuals, with one row for each response variable and one column for each observation

Method `viterbi()`**:** Viterbi algorithm

Usage:

```
HMM$viterbi()
```

Returns: Most likely state sequence

Method `sample_states()`**:** Sample posterior state sequences using forward-filtering backward-sampling

The forward-filtering backward-sampling algorithm returns a sequence of states, similarly to the Viterbi algorithm, but it generates it from the posterior distribution of state sequences, i.e., accounting for uncertainty in the state classification. Multiple generated sequences will therefore generally not be the same.

Usage:

```
HMM$sample_states(nsamp = 1, full = FALSE)
```

Arguments:

`nsamp` Number of samples to produce

`full` If TRUE and model fit by fit_stan then parameter estimates are sampled from the posterior samples before simulating each sequence

Returns: Matrix where each column is a different sample of state sequences, and each row is a time of observation

Method `state_probs()`**:** Compute posterior probability of being in each state

Usage:

```
HMM$state_probs()
```

Returns: matrix with a row for each observation and a column for each state

Method `post_coeff()`**:** Posterior sampling for model coefficients

Usage:

```
HMM$post_coeff(n_post)
```

Arguments:

`n_post` Number of posterior samples

Returns: Matrix with one column for each coefficient and one row for each posterior draw

Method `post_linpred()`**:** Posterior sampling for linear predictor

Usage:

```
HMM$post_linpred(n_post)
```

Arguments:

`n_post` Number of posterior samples

Returns: List with elements obs and hid, where each is a matrix with one column for each predictor and one row for each posterior draw

Method `post_fn()`**:** Create posterior simulations of a function of a model component

Usage:

```
HMM$post_fn(fn, n_post, comp = NULL, ..., level = 0, return_post = FALSE)
```

Arguments:

`fn` Function which takes a vector of linear predictors as input and produces either a scalar or vector output

`n_post` Number of posterior simulations

`comp` Either "obs" for observation model linear predictor, or "hid" for hidden model linear predictor

`...` Arguments passed to fn

`level` Confidence interval level if required (e.g., 0.95 for 95 confidence intervals). Default is 0, i.e., confidence intervals are not returned.

`return_post` Logical indicating whether to return the posterior samples. If FALSE (default), only mean estimates and confidence intervals are returned

Returns: A list with elements:

- postIf return_post = TRUE, this is a vector (for scalar outputs of fn) or matrix (for vector outputs) with a column for each simulation
- meanMean over posterior samples
- lclLower confidence interval bound (if level !=0)
- uclUpper confidence interval bound (if level !=0)

Method `predict()`**:** Predict estimates from a fitted model

By default, this returns point estimates of the HMM parameters for a new data frame of covariates. See the argument 'n_post' to also get confidence intervals.

Usage:

```
HMM$predict(
  what,
  t = 1,
  newdata = NULL,
  n_post = 0,
  level = 0.95,
  return_post = FALSE
)
```

Arguments:

`what` Which estimates to predict? Options include transition probability matrices "tpm", stationary distributions "delta", or observation distribution parameters "obspar"

`t` Time points to predict at

`newdata` New dataframe to use for prediction

`n_post` If greater than zero then n_post posterior samples are produced, and used to create confidence intervals.

`level` Level of the confidence intervals, e.g. CI = 0.95 will produce 95% confidence intervals (default)

`return_post` Logical. If TRUE, a list of posterior samples is returned.

`...` Other arguments to the respective functions for hidtpm, hiddelta, obs$par

Returns: Named array of predictions and confidence intervals, if requested

Examples:

```
# Load data set (included with R)
data(nottem)
data <- data.frame(temp = as.vector(t(nottem)))

# Create hidden state and observation models
hid <- MarkovChain$new(data = data, n_states = 2)
par0 <- list(temp = list(mean = c(40, 60), sd = c(5, 5)))
obs <- Observation$new(data = data, n_states = 2,
                       dists = list(temp = "norm"),
                       par = par0)

# Create HMM
hmm <- HMM$new(hid = hid, obs = obs)

# Fit HMM
hmm$fit(silent = TRUE)

# Get transition probability matrix with confidence intervals
hmm$predict(what = "tpm", n_post = 1000)
```

Method `confint()`**:** Confidence intervals for working parameters

This function computes standard errors for all fixed effect model parameters based on the diagonal of the inverse of the Hessian matrix, and then derives Wald-type confidence intervals.

Usage:

```
HMM$confint(level = 0.95)
```

Arguments:

`level` Level of confidence intervals. Defaults to 0.95, i.e., 95% confidence intervals.

Returns: List of matrices with three columns: mle (maximum likelihood estimate), lcl (lower confidence limit), and ucl (upper confidence limit). One such matrix is produced for the working parameters of the observation model, the working parameters of the hidden state model, the smoothness parameters of the observation model, and the smoothness parameters of the hidden state model.

Method `simulate()`**:** Simulate from hidden Markov model

Usage:

```
HMM$simulate(n, data = NULL, silent = FALSE)
```

Arguments:

`n` Number of time steps to simulate

`data` Optional data frame including covariates

`silent` if TRUE then no messages are printed

Returns: Data frame including columns of data (if provided), and simulated data variables

Method `check()`**:** Compute goodness-of-fit statistics using simulation

Many time series are simulated from the fitted model, and the statistic(s) of interest are calculated for each. A histogram of those values can for example be used to compare to the observed value of the statistic. An observation far in the tails of the distribution of simulated statistics suggests lack of fit.

Usage:

```
HMM$check(check_fn, nsims = 100, full = FALSE, silent = FALSE)
```

Arguments:

`check_fn` Goodness-of-fit function which accepts "data" as input and returns a statistic (either a vector or a single number) to be compared between observed data and simulations.

`nsims` Number of simulations to perform

`full` If model fitted with 'fit_stan', then full = TRUE will sample from posterior for each simulation

`silent` Logical. If FALSE, simulation progress is shown. (Default: TRUE)

Returns: List with elements:

- obs_stat: Vector of values of goodness-of-fit statistics for the observed data
- stats: Matrix of values of goodness-of-fit statistics for the simulated data sets (one row for each statistic, and one column for each simulation)
- plot: ggplot object

Method `plot_ts()`**:** Time series plot coloured by states

Creates a plot of the data coloured by the most likely state sequence, as estimated by the Viterbi algorithm. If one variable name is passed as input, it is plotted against time. If two variables are passed, they are plotted against each other.

Usage:

```
HMM$plot_ts(var, var2 = NULL)
```

Arguments:

`var` Name of the variable to plot.

`var2` Optional name of a second variable, for 2-d plot.

Returns: A ggplot object

Method `plot_dist()`**:** Plot observation distributions weighted by frequency in Viterbi

This is a wrapper around Observation$plot_dist, where the distribution for each state is weighted by the proportion of time spent in that state (according to the Viterbi state sequence).

Usage:

```
HMM$plot_dist(var)
```

Arguments:

`var` Name of data variable

Returns: Plot of distribution with data histogram

Method `plot()`**:** Plot a model component

Usage:

```
HMM$plot(
  what,
  var = NULL,
  covs = NULL,
  i = NULL,
  j = NULL,
  n_grid = 50,
  n_post = 1000
)
```

Arguments:

`what` Name of model component to plot: should be one of "tpm" (transition probabilities), "delta" (stationary state probabilities), or "obspar" (state-dependent observation parameters)

`var` Name of covariate to plot on x-axis

`covs` Optional named list for values of covariates (other than 'var') that should be used in the plot (or dataframe with single row). If this is not specified, the mean value is used for numeric variables, and the first level for factor variables.

`i` If plotting tpm then rows of tpm; if plotting delta then indices of states to plot; if plotting obspar then full names of parameters to plot (e.g., obsvar.mean)

`j` If plotting tpm then columnss of tpm to plot; if plotting delta then this is ignored,; if plotting obspar then indices of states to plot

`n_grid` Number of points in grid over x-axis (default: 50)

`n_post` Number of posterior simulations to use when computing confidence intervals; default: 1000. See `predict` function for more detail.

Returns: A ggplot object

Method `AIC_marginal()`**:** Marginal Akaike Information Criterion

The marginal AIC is for example defined by Wood (2017), as AIC = - 2L + 2k where L is the maximum marginal log-likelihood (of fixed effects), and k is the number of degrees of freedom of the fixed effect component of the model

Usage:

```
HMM$AIC_marginal()
```

Returns: Marginal AIC

Method `AIC_conditional()`**:** Conditional Akaike Information Criterion

The conditional AIC is for example defined by Wood (2017), as AIC = - 2L + 2k where L is the maximum joint log-likelihood (of fixed and random effects), and k is the number of effective degrees of freedom of the model (accounting for flexibility in non-parametric terms implied by smoothing)

Usage:

```
HMM$AIC_conditional()
```

Returns: Conditional AIC

Method `print_obspar()`**:** Print observation parameters at t = 1

Usage:

```
HMM$print_obspar()
```

Method `print_tpm()`**:** Print observation parameters at t = 1

Usage:

```
HMM$print_tpm()
```

Method `formulation()`**:** Print model formulation

Usage:

```
HMM$formulation()
```

Method `print()`**:** Print HMM object

Usage:

```
HMM$print()
```

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

Usage:

```
HMM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `HMM$new`
## ------------------------------------------------

# Load data set (included with R)
data(nottem)
data <- data.frame(temp = as.vector(t(nottem)))

# Create hidden state and observation models
hid <- MarkovChain$new(data = data, n_states = 2)
par0 <- list(temp = list(mean = c(40, 60), sd = c(5, 5)))
obs <- Observation$new(data = data, n_states = 2,
                       dists = list(temp = "norm"),
                       par = par0)

# Create HMM
hmm <- HMM$new(hid = hid, obs = obs)

## ------------------------------------------------
```

```
## Method `HMM$fit`
## ------------------------------------------------

# Load data set (included with R)
data(nottem)
data <- data.frame(temp = as.vector(t(nottem)))

# Create hidden state and observation models
hid <- MarkovChain$new(data = data, n_states = 2)
par0 <- list(temp = list(mean = c(40, 60), sd = c(5, 5)))
obs <- Observation$new(data = data, n_states = 2,
                       dists = list(temp = "norm"),
                       par = par0)

# Create HMM
hmm <- HMM$new(hid = hid, obs = obs)

# Fit HMM
hmm$fit(silent = TRUE)

## ------------------------------------------------
## Method `HMM$predict`
## ------------------------------------------------

# Load data set (included with R)
data(nottem)
data <- data.frame(temp = as.vector(t(nottem)))

# Create hidden state and observation models
hid <- MarkovChain$new(data = data, n_states = 2)
par0 <- list(temp = list(mean = c(40, 60), sd = c(5, 5)))
obs <- Observation$new(data = data, n_states = 2,
                       dists = list(temp = "norm"),
                       par = par0)

# Create HMM
hmm <- HMM$new(hid = hid, obs = obs)

# Fit HMM
hmm$fit(silent = TRUE)

# Get transition probability matrix with confidence intervals
hmm$predict(what = "tpm", n_post = 1000)
```

`hmmTMB_cols` *hmmTMB colour palette*

Description

hmmTMB colour palette

Usage

```
hmmTMB_cols
```

Format

An object of class `character` of length 6.

---

invmlogit *Multivarite inverse logit function*

Description

Multivarite inverse logit function

Usage

```
invmlogit(x)
```

Arguments

`x`	Numeric vector

---

is_whole_number *Check if number of whole number*

Description

Check if number of whole number

Usage

```
is_whole_number(x, tol = 1e-10)
```

Arguments

`x`	number to check or vector of numbers
`tol`	how far away from whole number is ok?

Value

TRUE if it is a whole number within tolerance

logLik.HMM *logLik function for SDE objects*

Description

This function makes it possible to call generic R methods such as AIC and BIC on HMM objects. It is based on the number of degrees of freedom of the *conditional* AIC (rather than marginal AIC), i.e., including degrees of freedom from the smooth/random effect components of the model.

Usage

```
## S3 method for class 'HMM'
logLik(object, ...)
```

Arguments

object	SDE model object
...	For compatibility with S3 method

Value

Maximum log-likelihood value for the model, with attributes df (degrees of freedom) and nobs (number of observations)

logsumexp *Log of sum of exponentials*

Description

Log of sum of exponentials

Usage

```
logsumexp(x)
```

Arguments

x	Numeric vector

`make_formulas` *Process formulas and store in nested list*

Description

Process formulas and store in nested list

Usage

```
make_formulas(input_forms, var_names, par_names, n_states)
```

Arguments

`input_forms`	Nested list of formulas, with two levels: observed variable, and parameter of the observation distribution. The formulas can contain state-specific terms, e.g. "~ state1(x1) + x2".
`var_names`	character vector name of each observation variable
`par_names`	list with element for each observation variable that contains character vector of name of each parameter in its distribution
`n_states`	Number of states

Details

Formulas for the observation parameters can be different for the different states, using special functions of the form "state1", "state2", etc. This method processes the list of formulas passed by the user to extract the state-specific formulas. Missing formulas are assumed to be intercept-only ~1.

Value

Nested list of formulas, with three levels: observed variable, parameter of the observation distribution, and state.

Examples

```
input_forms <- list(step = list(shape = ~ state1(x1) + x2,
                                scale = ~ x1),
                    count = list(lambda = ~ state1(x1) + state2(s(x2, bs = "cs"))))

make_formulas(input_forms = input_forms,
              var_names = names(input_forms),
              par_names = lapply(input_forms, names),
              n_states = 2)
```

make_matrices *Create model matrices*

Description

Create model matrices

Usage

```
make_matrices(formulas, data, new_data = NULL)
```

Arguments

formulas	List of formulas (possibly nested, e.g. for use within Observation)
data	Data frame including covariates
new_data	Optional new data set, including covariates for which the design matrices should be created. This needs to be passed in addition to the argument 'data', for cases where smooth terms or factor covariates are included, and the original data set is needed to determine the full range of covariate values.

Value

A list of

- X_fe Design matrix for fixed effects
- X_re Design matrix for random effects
- S Smoothness matrix
- ncol_fe Number of columns of X_fe for each parameter
- ncol_re Number of columns of X_re and S for each random effect

MarkovChain *R6 class for HMM hidden process model*

Description

R6 class for HMM hidden process model

R6 class for HMM hidden process model

Details

Contains the parameters and model formulas for the hidden process model.

Methods

Public methods:

- MarkovChain$new()
- MarkovChain$formula()
- MarkovChain$formulas()
- MarkovChain$tpm()
- MarkovChain$ref()
- MarkovChain$ref_mat()
- MarkovChain$ref_delta0()
- MarkovChain$coeff_fe()
- MarkovChain$delta()
- MarkovChain$delta0()
- MarkovChain$stationary()
- MarkovChain$fixpar()
- MarkovChain$coeff_re()
- MarkovChain$X_fe()
- MarkovChain$X_re()
- MarkovChain$lambda()
- MarkovChain$sd_re()
- MarkovChain$nstates()
- MarkovChain$terms()
- MarkovChain$unique_ID()
- MarkovChain$initial_state()
- MarkovChain$update_tpm()
- MarkovChain$update_coeff_fe()
- MarkovChain$update_coeff_re()
- MarkovChain$update_X_fe()
- MarkovChain$update_X_re()
- MarkovChain$update_delta0()
- MarkovChain$update_lambda()
- MarkovChain$update_fixpar()
- MarkovChain$make_mat()
- MarkovChain$make_mat_grid()
- MarkovChain$tpm2par()
- MarkovChain$par2tpm()
- MarkovChain$linpred()
- MarkovChain$simulate()
- MarkovChain$formulation()
- MarkovChain$print()
- MarkovChain$clone()

Method `new()`**:** Create new MarkovChain object

Usage:

```
MarkovChain$new(
  data,
  formula = NULL,
  n_states,
  tpm = NULL,
  initial_state = "estimated",
  fixpar = NULL,
  ref = 1:n_states
)
```

Arguments:

`data` Data frame, needed to create model matrices, and to identify the number of time series (which each have a separate initial distribution)

`formula` Either (1) R formula, used for all transition probabilities, or (2) matrix of character strings giving the formula for each transition probability, with "." along the diagonal (or for reference elements; see `ref` argument). (Default: no covariate dependence.)

`n_states` Number of states. If not specified, then `formula` needs to be provided as a matrix, and n_states is deduced from its dimensions.

`tpm` Optional transition probability matrix, to initialise the model parameters (intercepts in model with covariates). If not provided, the default is a matrix with 0.9 on the diagonal.

`initial_state` Specify model for initial state distribution. There are five different options:

- "estimated": a separate initial distribution is estimated for each ID (default)
- "stationary": the initial distribution is fixed to the stationary distribution of the transition probability matrix for the first time point of each ID
- "shared": a common initial distribution is estimated for all IDs
- integer value between 1 and n_states: used as the known initial state for all IDs
- vector of integers between 1 and n_states (of length the number of IDs): each element is used as the known initial state for the corresponding ID

`fixpar` List with optional elements "hid" (fixed parameters for transition probabilities), "lambda_hid" (fixed smoothness parameters), and "delta0" (fixed parameters for initial distribution). Each element is a named vector of coefficients that should either be fixed (if the corresponding element is set to NA) or estimated to a common value (using integers or factor levels).

`ref` Vector of indices for reference transition probabilities, of length `n_states`. The i-th element is the index for the reference in the i-th row of the transition probability matrix. For example, ref = c(1, 1) means that the first element of the first row Pr(1>1) and the first element of the second row Pr(2>1) are used as reference elements and are not estimated. If this is not provided, the diagonal transition probabilities are used as references.

Returns: A new MarkovChain object

Examples:

```
# Load data set from MSwM package
data(energy, package = "MSwM")

# Create 2-state covariate-free model and initialise transition
# probability matrix
hid <- MarkovChain$new(data = energy, n_states = 2,
```

```
                        tpm = matrix(c(0.8, 0.3, 0.2, 0.7), 2, 2))

# Create 2-state model with non-linear effect of Oil on all transition
# probabilities
hid <- MarkovChain$new(data = energy, n_states = 2,
                        formula = ~ s(Oil, k = 5, bs = "cs"))

# Create 2-state model with quadratic effect of Oil on Pr(1 > 2)
structure <- matrix(c(".", "~poly(Oil, 2)",
                      "~1", "."),
                    ncol = 2, byrow = TRUE)
hid <- MarkovChain$new(data = energy, n_states = 2,
                        formula = structure)
```

Method `formula()`**:** Formula of MarkovChain model

Usage:

```
MarkovChain$formula()
```

Method `formulas()`**:** List of formulas for MarkovChain model

Usage:

```
MarkovChain$formulas()
```

Method `tpm()`**:** Get transition probability matrices

Usage:

```
MarkovChain$tpm(t = 1, linpred = NULL)
```

Arguments:

`t` Time index or vector of time indices; default = 1. If t = "all" then all transition probability matrices are returned.

`linpred` Optional custom linear predictor

Returns: Array with one slice for each transition probability matrix

Method `ref()`**:** Indices of reference elements in transition probability matrix

Usage:

```
MarkovChain$ref()
```

Method `ref_mat()`**:** Matrix of reference elements in transition probability matrix

Usage:

```
MarkovChain$ref_mat()
```

Method `ref_delta0()`**:** Indices of reference elements in initial distribution

Usage:

```
MarkovChain$ref_delta0()
```

Method `coeff_fe()`**:** Current parameter estimates (fixed effects)

Usage:

```
MarkovChain$coeff_fe()
```

Method `delta()`**:** Stationary distribution

Usage:

```
MarkovChain$delta(t = NULL, linpred = NULL)
```

Arguments:

`t` Time point(s) for which stationary distribution should be returned. If t = "all", all deltas are returned; else this should be a vector of time indices. If NULL (default), the stationary distribution for the first time step is returned.

`linpred` Optional custom linear predictor

Returns: Matrix of stationary distributions. Each row corresponds to a row of the design matrices, and each column corresponds to a state.

Method `delta0()`**:** Initial distribution

Usage:

```
MarkovChain$delta0(log = FALSE, as_matrix = TRUE)
```

Arguments:

`log` Logical indicating whether to return the log of the initial probabilities (default: FALSE). If TRUE, then the last element is excluded, as it is not estimated.

`as_matrix` Logical indicating whether the output should be formatted as a matrix (default). If as_matrix is FALSE and log is TRUE, the result is formatted as a column vector.

Returns: Matrix with one row for each time series ID, and one column for each state. For each ID, the i-th element of the corresponding row is the probability Pr(S[1] = i)

Method `stationary()`**:** Use stationary distribution as initial distribution?

Usage:

```
MarkovChain$stationary()
```

Method `fixpar()`**:** Fixed parameters

Usage:

```
MarkovChain$fixpar(all = FALSE)
```

Arguments:

`all` Logical. If FALSE, only user-specified fixed parameters are returned, but not parameters that are fixed for some other reason (e.g., from '.' in formula)

Method `coeff_re()`**:** Current parameter estimates (random effects)

Usage:

```
MarkovChain$coeff_re()
```

Method `X_fe()`**:** Fixed effect design matrix

Usage:

```
MarkovChain$X_fe()
```

Method `X_re()`**:** Random effect design matrix

Usage:

```
MarkovChain$X_re()
```

Method `lambda()`: Smoothness parameters

Usage:

```
MarkovChain$lambda()
```

Method `sd_re()`: Standard deviation of smooth terms

This function transforms the smoothness parameter of each smooth term into a standard deviation, given by SD = 1/sqrt(lambda). It is particularly helpful to get the standard deviations of independent normal random effects.

Usage:

```
MarkovChain$sd_re()
```

Method `nstates()`: Number of states

Usage:

```
MarkovChain$nstates()
```

Method `terms()`: Terms of model formulas

Usage:

```
MarkovChain$terms()
```

Method `unique_ID()`: Number of time series

Usage:

```
MarkovChain$unique_ID()
```

Method `initial_state()`: Initial state (see constructor argument)

Usage:

```
MarkovChain$initial_state()
```

Method `update_tpm()`: Update transition probability matrix

Usage:

```
MarkovChain$update_tpm(tpm)
```

Arguments:

`tpm` New transition probability matrix

Method `update_coeff_fe()`: Update coefficients for fixed effect parameters

Usage:

```
MarkovChain$update_coeff_fe(coeff_fe)
```

Arguments:

`coeff_fe` Vector of coefficients for fixed effect parameters

Method `update_coeff_re()`: Update coefficients for random effect parameters

Usage:

```
MarkovChain$update_coeff_re(coeff_re)
```

Arguments:

`coeff_re` Vector of coefficients for random effect parameters

Method `update_X_fe()`: Update design matrix for fixed effects

Usage:

```
MarkovChain$update_X_fe(X_fe)
```

Arguments:

`X_fe` new design matrix for fixed effects

Method `update_X_re()`: Update design matrix for random effects

Usage:

```
MarkovChain$update_X_re(X_re)
```

Arguments:

`X_re` new design matrix for random effects

Method `update_delta0()`: Update initial distribution

Usage:

```
MarkovChain$update_delta0(delta0)
```

Arguments:

`delta0` Either a matrix where the i-th row is the initial distribution for the i-th time series in the data, or a vector which is then used for all time series. Entries of each row of delta0 should sum to one.

Method `update_lambda()`: Update smoothness parameters

Usage:

```
MarkovChain$update_lambda(lambda)
```

Arguments:

`lambda` New smoothness parameter vector

Method `update_fixpar()`: Update information about fixed parameters

Usage:

```
MarkovChain$update_fixpar(fixpar)
```

Arguments:

`fixpar` New list of fixed parameters, in the same format expected by MarkovChain$new()

Method `make_mat()`: Make model matrices

Usage:

```
MarkovChain$make_mat(data, new_data = NULL)
```

Arguments:

`data` Data frame containing all needed covariates

`new_data` Optional new data set, including covariates for which the design matrices should be created. This needs to be passed in addition to the argument '`data`', for cases where smooth terms or factor covariates are included, and the original data set is needed to determine the full range of covariate values.

Returns: A list with elements:

- X_feDesign matrix for fixed effects
- X_reDesign matrix for random effects
- SSmoothness matrix for random effects
- ncol_feNumber of columns of X_fe for each parameter
- ncol_reNumber of columns of X_re and S for each random effect

Method `make_mat_grid()`**:** Design matrices for grid of covariates

Used in plotting functions such as HMM$plot_tpm and HMM$plot_stat_dist

Usage:

```
MarkovChain$make_mat_grid(var, data, covs = NULL, n_grid = 1000)
```

Arguments:

`var` Name of variable

`data` Data frame containing the covariates

`covs` Optional named list for values of covariates (other than 'var') that should be used in the plot (or dataframe with single row). If this is not specified, the mean value is used for numeric variables, and the first level for factor variables.

`n_grid` Grid size (number of points). Default: 1000.

Returns: A list with the same elements as the output of make_mat, plus a data frame of covariates values.

Method `tpm2par()`**:** Transform transition probabilities to working scale

Apply the multinomial logit link function to get the corresponding parameters on the working scale (i.e., linear predictor scale).

Usage:

```
MarkovChain$tpm2par(tpm)
```

Arguments:

`tpm` Transition probability matrix

Returns: Vector of parameters on linear predictor scale

Method `par2tpm()`**:** Transform working parameters to transition probabilities

Apply the inverse multinomial logit link function to transform the parameters on the working scale (i.e., linear predictor scale) into the transition probabilities.

Usage:

```
MarkovChain$par2tpm(par)
```

Arguments:

`par` Vector of parameters on working scale

Returns: Transition probability matrix

Method `linpred()`**:** Linear predictor for transition probabilities

Usage:

```
MarkovChain$linpred()
```

Method `simulate()`**:** Simulate from Markov chain

Usage:

```
MarkovChain$simulate(n, data = NULL, new_data = NULL, silent = FALSE)
```

Arguments:

`n` Number of time steps to simulate

`data` Optional data frame containing all needed covariates

`new_data` Optional new data set, including covariates for which the design matrices should be created. This needs to be passed in addition to the argument '`data`', for cases where smooth terms or factor covariates are included, and the original data set is needed to determine the full range of covariate values.

`silent` if TRUE then no messages are printed

Returns: Sequence of states of simulated chain

Method `formulation()`**:** Print model formulation

Usage:

```
MarkovChain$formulation()
```

Method `print()`**:** Print MarkovChain object

Usage:

```
MarkovChain$print()
```

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

Usage:

```
MarkovChain$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `MarkovChain$new`
## ------------------------------------------------

# Load data set from MSwM package
data(energy, package = "MSwM")

# Create 2-state covariate-free model and initialise transition
# probability matrix
hid <- MarkovChain$new(data = energy, n_states = 2,
                       tpm = matrix(c(0.8, 0.3, 0.2, 0.7), 2, 2))

# Create 2-state model with non-linear effect of Oil on all transition
# probabilities
hid <- MarkovChain$new(data = energy, n_states = 2,
                       formula = ~ s(Oil, k = 5, bs = "cs"))

# Create 2-state model with quadratic effect of Oil on Pr(1 > 2)
```

```
structure <- matrix(c(".", "~poly(Oil, 2)",
                      "~1", "."),
                    ncol = 2, byrow = TRUE)
hid <- MarkovChain$new(data = energy, n_states = 2,
                       formula = structure)
```

`mlogit` *Multivariate logit function*

Description

Multivariate logit function

Usage

```
mlogit(x)
```

Arguments

`x`	Numeric vector

`mvnorm_invlink` *Multivariate Normal inverse link function*

Description

Multivariate Normal inverse link function

Usage

```
mvnorm_invlink(x)
```

Arguments

`x`	Vector of parameters on linear predictor scale (in the order: means, SDs, correlations)

mvnorm_link *Multivariate Normal link function*

Description

Multivariate Normal link function

Usage

```
mvnorm_link(x)
```

Arguments

x	Vector of parameters on natural scale (in the order: means, SDs, correlations)

na_fill *Fill in NAs*

Description

Replace NA entries in a vector by the last non-NA value. If the first entry of the vector is NA, it is replaced by the first non-NA value. If the vector passed as input doesn't contain NAs, it is returned as is.

Usage

```
na_fill(x)
```

Arguments

x	Vector in which NAs should be removed

Value

Copy of x in which NAs have been replaced by nearest available value.

`Observation` *R6 class for HMM observation model*

Description

R6 class for HMM observation model

R6 class for HMM observation model

Details

Contains the data, distributions, parameters, and formulas for the observation model from a hidden Markov model.

Methods

Public methods:

- `Observation$new()`
- `Observation$data()`
- `Observation$dists()`
- `Observation$nstates()`
- `Observation$par()`
- `Observation$inipar()`
- `Observation$coeff_fe()`
- `Observation$coeff_re()`
- `Observation$X_fe()`
- `Observation$X_re()`
- `Observation$lambda()`
- `Observation$sd_re()`
- `Observation$formulas()`
- `Observation$terms()`
- `Observation$obs_var()`
- `Observation$known_states()`
- `Observation$fixpar()`
- `Observation$update_par()`
- `Observation$update_coeff_fe()`
- `Observation$update_coeff_re()`
- `Observation$update_X_fe()`
- `Observation$update_X_re()`
- `Observation$update_lambda()`
- `Observation$update_data()`
- `Observation$update_fixpar()`
- `Observation$make_mat()`

- Observation$make_newdata_grid()
- Observation$n2w()
- Observation$w2n()
- Observation$linpred()
- Observation$obs_probs()
- Observation$suggest_initial()
- Observation$plot_dist()
- Observation$formulation()
- Observation$print()
- Observation$clone()

Method `new()`**:** Create new Observation object

Usage:

```
Observation$new(data, dists, formulas = NULL, n_states, par, fixpar = NULL)
```

Arguments:

`data` Data frame containing response variables (named in dists and par) and covariates (named in formulas)

`dists` Named list of distribution names for each data stream, with the following options: beta, binom, cat, dir, exp, foldednorm, gamma, gamma2, lnorm, mvnorm, nbinom, norm, pois, t, truncnorm, tweedie, vm, weibull, wrpcauchy, zibinom, zigamma, zigamma2, zinbinom, zipois, ztnbinom, ztpois. See vignette about list of distributions for more detail, e.g., list of parameters for each distribution.

`formulas` List of formulas for observation parameters. This should be a nested list, where the outer list has one element for each observed variable, and the inner lists have one element for each parameter. Any parameter that is not included is assumed to have the formula ~1. By default, all parameters have the formula ~1 (i.e., no covariate effects).

`n_states` Number of states (needed to construct model formulas)

`par` List of initial observation parameters. This should be a nested list, where the outer list has one element for each observed variable, and the inner lists have one element for each parameter. The choice of good initial values can be important, especially for complex models; the package vignettes discuss approaches to selecting them (e.g., see `Observation$suggest_initial()`).

`fixpar` List with optional elements "obs" (fixed coefficients for observation parameters), and "lambda_obs" (fixed smoothness parameters), Each element is a named vector of coefficients that should either be fixed (if the corresponding element is set to NA) or estimated to a common value (using integers or factor levels).

Returns: A new Observation object

Examples:

```
# Load data set from MSwM package
data(energy, package = "MSwM")

# Initial observation parameters
par0 <- list(Price = list(mean = c(3, 6), sd = c(2, 2)))

# Model "energy" with normal distributions
```

```
obs <- Observation$new(data = energy,
                       dists = list(Price = "norm"),
                       par = par0,
                       n_states = 2)

# Model "energy" with gamma distributions
obs <- Observation$new(data = energy,
                       dists = list(Price = "gamma2"),
                       par = par0,
                       n_states = 2)

# Model with non-linear effect of EurDol on mean price
f <- list(Price = list(mean = ~ s(EurDol, k = 5, bs = "cs")))
obs <- Observation$new(data = energy,
                       dists = list(Price = "norm"),
                       par = par0,
                       n_states = 2,
                       formula = f)
```

Method `data()`**:** Data frame

Usage:

```
Observation$data()
```

Method `dists()`**:** List of distributions

Usage:

```
Observation$dists()
```

Method `nstates()`**:** Number of states

Usage:

```
Observation$nstates()
```

Method `par()`**:** Parameters on natural scale

Usage:

```
Observation$par(t = 1, full_names = TRUE, linpred = NULL, as_list = FALSE)
```

Arguments:

`t` Time index or vector of time indices; default t = 1. If t = "all", then return observation parameters for all time points.

`full_names` Logical. If TRUE, the rows of the output are named in the format "variable.parameter" (default). If FALSE, the rows are names in the format "parameter". The latter is used in various internal functions, when the parameters need to be passed on to an R function.

`linpred` Optional custom linear predictor.

`as_list` Logical. If TRUE, the output is a nested list with three levels: (1) time step, (2) observed variable, (3) observation parameter. If FALSE (default), the output is an array with one row for each observation parameter, one column for each state, and one slice for each time step.

Returns: Array of parameters with one row for each observation parameter, one column for each state, and one slice for each time step. (See as_list argument for alternative output format.)

Examples:

```
# Load data set from MSwM package
data(energy, package = "MSwM")

# Initial observation parameters
par0 <- list(Price = list(mean = c(3, 6), sd = c(2, 2)))

# Model with linear effect of EurDol on mean price
f <- list(Price = list(mean = ~ EurDol))
obs <- Observation$new(data = energy,
                       dists = list(Price = "norm"),
                       par = par0,
                       n_states = 2,
                       formula = f)

# Set slope coefficients
obs$update_coeff_fe(coeff_fe = c(3, 2, 6, -2, log(2), log(2)))

# Observation parameter values for given data rows
obs$par(t = c(1, 10, 20))
```

Method `inipar()`**:** Return initial parameter values supplied

Usage:

```
Observation$inipar()
```

Method `coeff_fe()`**:** Fixed effect parameters on working scale

Usage:

```
Observation$coeff_fe()
```

Method `coeff_re()`**:** Random effect parameters

Usage:

```
Observation$coeff_re()
```

Method `X_fe()`**:** Fixed effect design matrix

Usage:

```
Observation$X_fe()
```

Method `X_re()`**:** Random effect design matrix

Usage:

```
Observation$X_re()
```

Method `lambda()`**:** Smoothness parameters

Usage:

```
Observation$lambda()
```

Method `sd_re()`**:** Standard deviation of smooth terms

This function transforms the smoothness parameter of each smooth term into a standard deviation, given by SD = 1/sqrt(lambda). It is particularly helpful to get the standard deviations of independent normal random effects.

Usage:

```
Observation$sd_re()
```

Method `formulas()`**:** List of model formulas for observation model

Usage:

```
Observation$formulas(raw = FALSE)
```

Arguments:

`raw` Logical. If FALSE, returns the nested list created by make_formulas (default). If TRUE, returns formulas passed as input.

Method `terms()`**:** Terms of model formulas

Usage:

```
Observation$terms()
```

Method `obs_var()`**:** Data frame of response variables

Usage:

```
Observation$obs_var(expand = FALSE)
```

Arguments:

`expand` If TRUE, then multivariate variables in observations are expanded to be univariate, creating extra columns.

Returns: Data frame of observation variables

Method `known_states()`**:** Vector of known states

Usage:

```
Observation$known_states(mat = TRUE)
```

Arguments:

`mat` Logical.

Method `fixpar()`**:** Fixed parameters

Usage:

```
Observation$fixpar(all = FALSE)
```

Arguments:

`all` Logical. If FALSE, only user-specified fixed parameters are returned, but not parameters that are fixed for some other reason (e.g., size of binomial distribution)

Method `update_par()`**:** Update parameters

Updates the 'par' attribute to the list passed as input, and updates the intercept elements of 'coeff_fe' using the list passed as input

Usage:

```
Observation$update_par(par)
```

Arguments:

`par` New list of parameters

Method `update_coeff_fe()`: Update coefficients for fixed effect parameters

Usage:

```
Observation$update_coeff_fe(coeff_fe)
```

Arguments:

`coeff_fe` New vector of coefficients for fixed effect parameters

Method `update_coeff_re()`: Update random effect parameters

Usage:

```
Observation$update_coeff_re(coeff_re)
```

Arguments:

`coeff_re` New vector of coefficients for random effect parameters

Method `update_X_fe()`: Update fixed effect design matrix

Usage:

```
Observation$update_X_fe(X_fe)
```

Arguments:

`X_fe` New fixed effect design matrix

Method `update_X_re()`: Update random effect design matrix

Usage:

```
Observation$update_X_re(X_re)
```

Arguments:

`X_re` New random effect design matrix

Method `update_lambda()`: Update smoothness parameters

Usage:

```
Observation$update_lambda(lambda)
```

Arguments:

`lambda` New smoothness parameter vector

Method `update_data()`: Update data

Usage:

```
Observation$update_data(data)
```

Arguments:

`data` New data frame

Method `update_fixpar()`: Update information about fixed parameters

Usage:

```
Observation$update_fixpar(fixpar)
```

Arguments:

`fixpar` New list of fixed parameters, in the same format expected by Observation$new()

Method `make_mat()`**:** Make model matrices

Usage:

```
Observation$make_mat(new_data = NULL)
```

Arguments:

`new_data` Optional new data set, including covariates for which the design matrices should be created. If this argument is not specified, the design matrices are based on the original data frame.

Returns: A list with elements:

- X_feDesign matrix for fixed effects
- X_reDesign matrix for random effects
- SSmoothness matrix for random effects
- ncol_feNumber of columns of X_fe for each parameter
- ncol_reNumber of columns of X_re and S for each random effect

Design matrices for grid of covariates

Method `make_newdata_grid()`**:**

Usage:

```
Observation$make_newdata_grid(var, covs = NULL, n_grid = 1000)
```

Arguments:

`var` Name of variable

`covs` Optional named list for values of covariates (other than 'var') that should be used in the plot (or dataframe with single row). If this is not specified, the mean value is used for numeric variables, and the first level for factor variables.

`n_grid` Grid size (number of points). Default: 1000.

Returns: A list with the same elements as the output of make_mat, plus a data frame of covariates values.

Method `n2w()`**:** Natural to working parameter transformation

This function applies the link functions of the distribution parameters, to transform parameters from their natural scale to the working scale (i.e., linear predictor scale)

Usage:

```
Observation$n2w(par)
```

Arguments:

`par` List of parameters on natural scale

Returns: Vector of parameters on working scale

Method `w2n()`**:** Working to natural parameter transformation

This function applies the inverse link functions of the distribution parameters, to transform parameters from the working scale (i.e., linear predictor scale) to their natural scale.

Usage:

```
Observation$w2n(wpar)
```

Arguments:

`wpar` Vector of parameters on working scale

Returns: List of parameters on natural scale

Method `linpred()`**:** Compute linear predictor

Usage:

```
Observation$linpred()
```

Method `obs_probs()`**:** Observation likelihoods

Usage:

```
Observation$obs_probs(data = NULL)
```

Arguments:

`data` Optional dataframe to include in form of obs_var() output

Returns: Matrix of likelihoods of observations, with one row for each time step, and one column for each state.

Method `suggest_initial()`**:** Suggest initial observation parameters

The K-means algorithm is used to define clusters of observations (supposed to approximate the HMM states). Then, for each cluster, the `parapprox` function of the relevant `Dist` object is used to obtain parameter values.

Usage:

```
Observation$suggest_initial()
```

Returns: List of initial parameters for each observation variable

Examples:

```
# Load data set from MSwM package
data(energy, package = "MSwM")

# Initial observation parameters
par0 <- list(Price = list(mean = c(3, 6), sd = c(2, 2)))

# Model "energy" with normal distributions
obs <- Observation$new(data = energy,
                       dists = list(Price = "norm"),
                       par = par0,
                       n_states = 2)

# Print observation parameters
obs$par()

# Suggest initial parameters
par0_new <- obs$suggest_initial()
par0_new

# Update model parameters to suggested
obs$update_par(par = par0_new)
obs$par()
```

Method `plot_dist()`**:** Plot histogram of data and pdfs

Plot histogram of observations for the variable specified by the argument name, overlaid with the pdf of the specified distribution for that data stream. Helpful to select initial parameter values for model fitting, or to visualise fitted state-dependent distributions.

Usage:

```
Observation$plot_dist(var, weights = NULL, t = 1)
```

Arguments:

`var` Name of response variable for which the histogram and pdfs should be plotted.

`weights` Optional vector of length the number of pdfs that are plotted. Useful to visualise a mixture of distributions weighted by the proportion of time spent in the different states.

`t` Index of time step to use for covariates (default: 1).

Returns: A ggplot object

Method `formulation()`**:** Print model formulation

Usage:

```
Observation$formulation()
```

Method `print()`**:** Print Observation object Check constructor arguments

Usage:

```
Observation$print()
```

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

Usage:

```
Observation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Observation$new`
## ------------------------------------------------

# Load data set from MSwM package
data(energy, package = "MSwM")

# Initial observation parameters
par0 <- list(Price = list(mean = c(3, 6), sd = c(2, 2)))

# Model "energy" with normal distributions
obs <- Observation$new(data = energy,
                       dists = list(Price = "norm"),
                       par = par0,
                       n_states = 2)

# Model "energy" with gamma distributions
```

```
obs <- Observation$new(data = energy,
                       dists = list(Price = "gamma2"),
                       par = par0,
                       n_states = 2)

# Model with non-linear effect of EurDol on mean price
f <- list(Price = list(mean = ~ s(EurDol, k = 5, bs = "cs")))
obs <- Observation$new(data = energy,
                       dists = list(Price = "norm"),
                       par = par0,
                       n_states = 2,
                       formula = f)

## ------------------------------------------------
## Method `Observation$par`
## ------------------------------------------------

# Load data set from MSwM package
data(energy, package = "MSwM")

# Initial observation parameters
par0 <- list(Price = list(mean = c(3, 6), sd = c(2, 2)))

# Model with linear effect of EurDol on mean price
f <- list(Price = list(mean = ~ EurDol))
obs <- Observation$new(data = energy,
                       dists = list(Price = "norm"),
                       par = par0,
                       n_states = 2,
                       formula = f)

# Set slope coefficients
obs$update_coeff_fe(coeff_fe = c(3, 2, 6, -2, log(2), log(2)))

# Observation parameter values for given data rows
obs$par(t = c(1, 10, 20))

## ------------------------------------------------
## Method `Observation$suggest_initial`
## ------------------------------------------------

# Load data set from MSwM package
data(energy, package = "MSwM")

# Initial observation parameters
par0 <- list(Price = list(mean = c(3, 6), sd = c(2, 2)))

# Model "energy" with normal distributions
obs <- Observation$new(data = energy,
                       dists = list(Price = "norm"),
                       par = par0,
                       n_states = 2)
```

```
# Print observation parameters
obs$par()

# Suggest initial parameters
par0_new <- obs$suggest_initial()
par0_new

# Update model parameters to suggested
obs$update_par(par = par0_new)
obs$par()
```

`prec_to_cov` *Get covariance matrix from precision matrix*

Description

The covariance matrix is the inverse of the precision matrix. By default, the function `solve` is used for inversion. If it fails (e.g., singular system), then `MASS::ginv` is used instead, and returns the Moore-Penrose generalised inverse of the precision matrix.

Usage

```
prec_to_cov(prec_mat)
```

Arguments

`prec_mat`	Precision matrix (either of 'matrix' type or sparse matrix on which as.matrix can be used)

Value

Precision matrix

`quad_pos_solve` *Solve for positive root of quadratic ax^2 + bx + c = 0 when it exists*

Description

Solve for positive root of quadratic ax^2 + bx + c = 0 when it exists

Usage

```
quad_pos_solve(a, b, c)
```

Arguments

a	coefficient of x^2
b	coefficient of x
c	scalar coefficient

Value

real positive root if it exists

---

strip_comments *Strip comments marked with a hash from a character vector*

Description

Strip comments marked with a hash from a character vector

Usage

```
strip_comments(str)
```

Arguments

str	the character vector

Value

character vector with comments removed (and lines with only comments completely removed)

---

update.HMM *Update a model to a new model by changing one formula*

Description

Update a model to a new model by changing one formula

Usage

```
## S3 method for class 'HMM'
update(object, type, i, j, change, fit = TRUE, silent = FALSE, ...)
```

Arguments

object	HMM model object
type	Character string for the part of the model that is updated (either "hid" or "obs")
i	If type = "hid" then i is the row of the formula containing the change. If type = "obs" then i is the observation variable name.
j	If type = "hid" then j is the column of the formula containing the change. If type = "obs" then j is the parameter whose formula is to be changed.
change	The change to make to the formula, see ?update.formula for details.
fit	If FALSE then change is made but model is not re-fit.
silent	If TRUE then no model fitting output is given
...	Additional arguments are ignored (for compatibility with generic S3 method)

Examples

```
# Load data set from MSwM package
data(energy, package = "MSwM")

# Create hidden state and observation models
hid <- MarkovChain$new(data = energy, n_states = 2)
par0 <- list(Price = list(mean = c(3, 6), sd = c(2, 3)))
obs <- Observation$new(data = energy, n_states = 2,
                       dists = list(Price = "norm"),
                       par = par0)

# Create HMM (no covariate effects)
hmm <- HMM$new(hid = hid, obs = obs)
hmm$hid()$formula()
hmm$obs()$formulas()

# Update transition probability formulas (one at a time)
hmm <- update(hmm, type = "hid", i = 1, j = 2,
              change = ~ . + Oil, fit = FALSE)
hmm <- update(hmm, type = "hid", i = 2, j = 1,
              change = ~ . + Gas + Coal, fit = FALSE)
hmm$hid()$formula()

# Update observation parameter formulas (one at a time)
hmm <- update(hmm, type = "obs", i = "Price", j = "mean",
              change = ~ . + EurDol, fit = FALSE)
hmm$obs()$formulas()
```

Index