Package ‘dynamite’

February 2, 2024

Title Bayesian Modeling and Causal Inference for Multivariate Longitudinal Data

Version 1.4.9

Description Easy-to-use and efficient interface for Bayesian inference of complex panel (time series) data using dynamic multivariate panel models by Helske and Tikka (2022) <doi:10.31235/osf.io/mdwu5>. The package supports joint modeling of multiple measurements per individual, time-varying and time-invariant effects, and a wide range of discrete and continuous distributions. Estimation of these dynamic multivariate panel models is carried out via 'Stan'. For an in-depth tutorial of the package, see (Tikka and Helske, 2023) <arxiv:2302.01607>.

License GPL (>= 3)


BugReports https://github.com/ropensci/dynamite/issues/

Depends R (>= 3.6.0)

Imports checkmate, cli, data.table, glue, ggplot2, loo, methods, patchwork, posterior, rlang, rstan, stats, tibble (>= 2.0.0), utils

Suggests bookdown, cmdstanr, covr, dplyr, knitr, mockthat, rmarkdown, testthat (>= 3.0.0), tidyR

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

LazyData true

LazyDataCompression xz

Additional_repositories https://mc-stan.org/r-packages/

NeedsCompilation no
Author  Santtu Tikka [aut, cre] (<https://orcid.org/0000-0003-4039-4342>),
       Jouni Helske [aut] (<https://orcid.org/0000-0001-7130-793X>),
       Nicholas Clark [rev],
       Lucy D’Agostino McGowan [rev]
Maintainer Santtu Tikka <santtuth@gmail.com>
Repository  CRAN
Date/Publication  2024-02-02 17:10:02 UTC

R topics documented:

dynamite-package.............................. 3
as.data.frame.dynamitefit.................... 3
as.data.table.dynamitefit................... 7
as_draws_df.dynamitefit...................... 8
categorical_example......................... 10
categorical_example_fit...................... 11
coef.dynamitefit............................. 12
confinf.dynamitefit.......................... 13
dynamite...................................... 14
dynamiteformula.............................. 18
fitted.dynamitefit........................... 22
gaussian_example............................. 24
gaussian_example_fit........................ 24
gaussian_simulation_fit...................... 26
get_code.................................... 27
get_data.................................... 28
get_parameterDims............................ 29
get_parameter_names......................... 31
get_parameter_types......................... 32
get_priors.................................. 32
lags......................................... 34
lfactor...................................... 35
lfo......................................... 36
loo.dynamitefit................................ 38
mcmc_diagnostics............................ 39
multichannel_example....................... 40
multichannel_example_fit................... 40
ndraws.dynamitefit......................... 42
nobs.dynamitefit............................. 42
plot.dynamitefit............................. 43
plot.lfo.................................... 44
plot_betas.................................. 45
plot_deltas.................................. 46
plot_lambdas................................ 47
plot_nus.................................... 48
plot_psis................................... 49
predict.dynamitefit........................ 50
as.data.frame.dynamitefit

Extract Samples From a dynamitefit Object as a Data Frame

Description

Provides a data.frame representation of the posterior samples of the model parameters.

Usage

```r
## S3 method for class 'dynamitefit'
```

Index

<table>
<thead>
<tr>
<th>dynamite-package</th>
<th>The dynamite package.</th>
</tr>
</thead>
<tbody>
<tr>
<td>print.lfo</td>
<td></td>
</tr>
<tr>
<td>random_spec</td>
<td></td>
</tr>
<tr>
<td>splines</td>
<td></td>
</tr>
<tr>
<td>update.dynamitefit</td>
<td></td>
</tr>
</tbody>
</table>

Authors

Santtu Tikka (author) santtuth@gmail.com
Jouni Helske (author) jouni.helske@iki.fi

See Also

- The package vignette.
- `dynamiteformula()` for information on defining models.
- `dynamite()` for information on fitting models.
- [https://github.com/ropensci/dynamite/issues/](https://github.com/ropensci/dynamite/issues/) to submit a bug report or a feature request.
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  parameters = NULL,
  responses = NULL,
  types = NULL,
  summary = FALSE,
  probs = c(0.05, 0.95),
  include_fixed = TRUE,
  ...
)

Arguments

x [dynamitefit]
  The model fit object.

row.names Ignored.

optional Ignored.

parameters [character()]
  Parameter(s) for which the samples should be extracted. Possible options can
  be found with function get_parameter_names(). Default is all parameters of
  specific type for all responses.

responses [character()]
  Response(s) for which the samples should be extracted. Possible options are
  elements of unique(x$priors$response), and the default is this entire vector.
  Ignored if the argument parameters is supplied.

types [character()]
  Type(s) of the parameters for which the samples should be extracted. See details
  of possible values. Default is all values listed in details except spline coefficients
  omega, omega_alpha, and omega_psi. See also get_parameter_types(). Ignored if
  the argument parameters is supplied.

summary [logical(1)]
  If TRUE, returns posterior mean, standard deviation, and posterior quantiles (as
  defined by the probs argument) for all parameters. If FALSE (default), returns
  the posterior samples instead.

probs [numeric()]
  Quantiles of interest. Default is c(0.05, 0.95).

include_fixed [logical(1)]
  If TRUE (default), time-varying parameters for 1:fixed time points are included
  in the output as NA values. If FALSE, fixed time points are omitted completely
  from the output.

... Ignored.
Details

The arguments `responses` and `types` can be used to extract only a subset of the model parameters (i.e., only certain types of parameters related to a certain response variable).

Potential values for the `types` argument are:

- **alpha**
  Intercept terms (time-invariant or time-varying).
- **beta**
  Time-invariant regression coefficients.
- **delta**
  Time-varying regression coefficients.
- **nu**
  Group-level random effects.
- **lambda**
  Factor loadings.
- **psi**
  Latent factors.
- **tau**
  Standard deviations of the spline coefficients of `delta`.
- **tau_alpha**
  Standard deviations of the spline coefficients of time-varying `alpha`.
- **sigma_nu**
  Standard deviations of the random effects `nu`.
- **corr_nu**
  Pairwise within-group correlations of random effects `nu`. Samples of the full correlation matrix can be extracted manually as `rstan::extract(fit$stanfit, pars = "corr_matrix_nu")` if necessary.
- **sigma_lambda**
  Standard deviations of the latent factor loadings `lambda`.
- **tau_psi**
  Standard deviations of the the spline coefficients of `psi`.
- **corr_psi**
  Pairwise correlations of the latent factors. Samples of the full correlation matrix can be extracted manually as `rstan::extract(fit$stanfit, pars = "corr_matrix_psi")` if necessary.
- **sigma**
  Standard deviations of gaussian responses.
- **corr**
  Pairwise correlations of multivariate gaussian responses.
- **phi**
  Describes various distributional parameters, such as:
  - Dispersion parameter of the Negative Binomial distribution.
  - Shape parameter of the Gamma distribution.
- Precision parameter of the Beta distribution.
- Degrees of freedom of the Student t-distribution.

- omega
  Spline coefficients of the regression coefficients delta.

- omega_alpha
  Spline coefficients of time-varying alpha.

- omega_psi
  Spline coefficients of the latent factors psi.

Value

A tibble containing either samples or summary statistics of the model parameters in a long format.
For a wide format, see `as_draws()`.

See Also

Model outputs `as.data.table.dynamitefit()`, `as_draws_df.dynamitefit()`, `coef.dynamitefit()`, `confint.dynamitefit()`, `dynamite()`, `get_code()`, `get_data()`, `get_parameter_dims()`, `get_parameter_names()`, `get_parameter_types()`, `ndraws.dynamitefit()`, `nobs.dynamitefit()`

Examples

data.table::setDTthreads(1) # For CRAN
as.data.frame(
  gaussian_example_fit,
  responses = "y",
  types = "beta"
)

# Basic summaries can be obtained automatically with summary = TRUE
as.data.frame(
  gaussian_example_fit,
  responses = "y",
  types = "beta",
  summary = TRUE
)

# Time-varying coefficients "delta"
as.data.frame(
  gaussian_example_fit,
  responses = "y",
  types = "delta",
  summary = TRUE
)

# Obtain summaries for a specific parameters
as.data.frame(
  gaussian_example_fit,
  parameters = c("tau_y_x", "sigma_y"),
  summary = TRUE
)
as.data.table.dynamitefit

Extract Samples From a dynamitefit Object as a Data Table

Description

Provides a data.table representation of the posterior samples of the model parameters. See as.data.frame.dynamitefit() for details.

Usage

## S3 method for class 'dynamitefit'
as.data.table(
  x,
  keep.rownames = FALSE,
  row.names = NULL,
  optional = FALSE,
  parameters = NULL,
  responses = NULL,
  types = NULL,
  summary = FALSE,
  probs = c(0.05, 0.95),
  include_fixed = TRUE,
  ...
)

Arguments

x [dynamitefit]
The model fit object.

keep.rownames [logical(1)]
Not used.

row.names Ignored.

optional Ignored.

parameters [character()]
Parameter(s) for which the samples should be extracted. Possible options can be found with function get_parameter_names(.). Default is all parameters of specific type for all responses.

responses [character()]
Response(s) for which the samples should be extracted. Possible options are elements of unique(x$priors$response), and the default is this entire vector. Ignored if the argument parameters is supplied.
types [character()]
Type(s) of the parameters for which the samples should be extracted. See details of possible values. Default is all values listed in details except spline coefficients omega, omega_alpha, and omega_psi. See also get_parameter_types(). Ignored if the argument parameters is supplied.

summary [logical(1)]
If TRUE, returns posterior mean, standard deviation, and posterior quantiles (as defined by the probs argument) for all parameters. If FALSE (default), returns the posterior samples instead.

probs [numeric()]
Quantiles of interest. Default is c(0.05, 0.95).

include_fixed [logical(1)]
If TRUE (default), time-varying parameters for 1:fixed time points are included in the output as NA values. If FALSE, fixed time points are omitted completely from the output.

... Ignored.

Value
A data.table containing either samples or summary statistics of the model parameters.

See Also
Model outputs as.data.frame.dynamitefit(), as_draws_df.dynamitefit(), coef.dynamitefit(), confint.dynamitefit(), dynamite(), get_code(), get_data(), get_parameter_dims(), get_parameter_names(), get_parameter_types(), ndraws.dynamitefit(), nobs.dynamitefit()

Examples

data.table::setDTthreads(1) # For CRAN
as.data.table(
  gaussian_example_fit,
  responses = "y",
  types = "beta",
  summary = FALSE
)

as_draws_df.dynamitefit

Convert dynamite Output to draws_df Format

Description
Converts the output from a dynamite() call to a draws_df format of the posterior package, enabling the use of diagnostics and plotting methods of posterior and bayesplot packages. Note that this function returns variables in a wide format, whereas as.data.frame() uses the long format.
Usage

## S3 method for class 'dynamitefit'
as_draws_df(x, parameters = NULL, responses = NULL, types = NULL, ...)

## S3 method for class 'dynamitefit'
as_draws(x, parameters = NULL, responses = NULL, types = NULL, ...)

Arguments

- **x** [dynamitefit]
  - The model fit object.
- **parameters** [character()]
  - Parameter(s) for which the samples should be extracted. Possible options can be found with function `get_parameter_names()`. Default is all parameters of specific type for all responses.
- **responses** [character()]
  - Response(s) for which the samples should be extracted. Possible options are elements of `unique(x$priors$response)`, and the default is this entire vector. Ignored if the argument `parameters` is supplied.
- **types** [character()]
  - Type(s) of the parameters for which the samples should be extracted. See details of possible values. Default is all values listed in details except spline coefficients `omega`, `omega_alpha`, and `omega_psi`. See also `get_parameter_types()`. Ignored if the argument `parameters` is supplied.
- **...** Ignored.

Details

You can use the arguments `parameters`, `responses` and `types` to extract only a subset of the model parameters (i.e., only certain types of parameters related to a certain response variable).

See potential values for the `types` argument in `as.data.frame.dynamitefit()` and `get_parameter_names()` for potential values for `parameters` argument.

Value

A draws_df object.

A draws_df object.

See Also

Model outputs `as.data.frame.dynamitefit()`, `as.data.table.dynamitefit()`, `coef.dynamitefit()`, `confint.dynamitefit()`, `dynamite()`, `get_code()`, `get_data()`, `get_parameter_dims()`, `get_parameter_names()`, `get_parameter_types()`, `ndraws.dynamitefit()`, `nobs.dynamitefit()`
Examples

data.table::setDTthreads(1) # For CRAN
as_draws(gaussian_example_fit, types = c("sigma", "beta"))

# Compute MCMC diagnostics using the posterior package
posterior::summarise_draws(as_draws(gaussian_example_fit))

categorical_example
Simulated Categorical Multivariate Panel Data

Description

A simulated data containing multiple individuals with two categorical response variables.

Usage

categorical_example

Format

A data frame with 2000 rows and 5 variables:

id  Variable defining individuals (1 to 100).
time Variable defining the time point of the measurement (1 to 20).
x  Categorical variable with three levels, A, B, and C.
y  Categorical variable with three levels, a, b, and c.
z  A continuous covariate.

Source

The data was generated according to a script in https://github.com/ropensci/dynamite/blob/main/data-raw/categorical_example.R

See Also

Example models categorical_example_fit, gaussian_example_fit, gaussian_example, gaussian_simulation_fit, multichannel_example_fit, multichannel_example
Model Fit for the Simulated Categorical Multivariate Panel Data

Description

A dynamitefit object obtained by running dynamite on the categorical_example dataset as

```r
set.seed(1)
library(dynamite)
f <- obs(x ~ z + lag(x) + lag(y), family = "categorical") +
    obs(y ~ z + lag(x) + lag(y), family = "categorical")
categorical_example_fit <- dynamite(
    f,
    data = categorical_example,
    time = "time",
    group = "id",
    chains = 1,
    refresh = 0,
    thin = 5,
    save_warmup = FALSE
)
```

Note the small number of samples due to size restrictions on CRAN.

Usage

categorical_example_fit

Format

A dynamitefit object.

Source


See Also

Example models categorical_example, gaussian_example_fit, gaussian_example, gaussian_simulation_fit, multichannel_example_fit, multichannel_example
Extract Regression Coefficients of a Dynamite Model

Description

Extracts either time-varying or time-invariant parameters of the model.

Usage

```r
## S3 method for class 'dynamitefit'
coef(
object,
parameters = NULL,
type = c("beta", "delta", "nu", "lambda", "psi"),
responses = NULL,
summary = TRUE,
probs = c(0.05, 0.95),
include_alpha = TRUE,
...)
```

Arguments

- `object`: [dynamitefit] The model fit object.
- `parameters`: [character()] Parameter(s) for which the samples should be extracted. Possible options can be found with function `get_parameter_names()`. Default is all parameters of specific type for all responses.
- `type`: [character(1)] Either beta (the default) for time-invariant coefficients, delta for time-varying coefficients, nu for random effects, lambda for factor loadings, or psi for latent factor. Ignored if the argument `parameters` is supplied.
- `responses`: [character()] Response(s) for which the samples should be extracted. Possible options are elements of `unique(x$priors$response)`, and the default is this entire vector. Ignored if the argument `parameters` is supplied.
- `summary`: [logical(1)] If TRUE, returns posterior mean, standard deviation, and posterior quantiles (as defined by the `probs` argument) for all parameters. If FALSE (default), returns the posterior samples instead.
- `probs`: [numeric()] Quantiles of interest. Default is `c(0.05, 0.95)`.
include_alpha [logical(1)]
If TRUE (default), extracts also time-invariant intercept term alpha if time-invariant parameters beta are extracted, and time-varying alpha if time-varying delta are extracted. Ignored if the argument parameters is supplied. @param summary
[logical(1)]
If TRUE (default), returns posterior mean, standard deviation, and posterior quantiles (as defined by the probs argument) for all parameters. If FALSE, returns the posterior samples instead.

Value
A tibble containing either samples or summary statistics of the model parameters in a long format.

See Also
Model outputs as.data.frame.dynamitefit(), as.data.table.dynamitefit(), as_draws_df.dynamitefit(),
confint.dynamitefit(), dynamite(), get_code(), get_data(), get_parameter_dims(), get_parameter_names(),
get_parameter_types(), ndraws.dynamitefit(), nobls.dynamitefit()

Examples
data.table::setDTthreads(1) # For CRAN
betas <- coef(gaussian_example_fit, type = "beta")
deltas <- coef(gaussian_example_fit, type = "delta")

confint.dynamitefit  Credible Intervals for Dynamite Model Parameters

Description
Extracts credible intervals from dynamitefit object.

Usage
## S3 method for class 'dynamitefit'
confint(object, parm, level = 0.95, ...)

Arguments
object [dynamitefit]
The model fit object.
parm Ignored.
level [numeric(1)]
Credible interval width.
... Ignored.
Value

The rows of the resulting matrix will be named using the following logic: \{parameter\}_{time}_{category}_{group}
where parameter is the name of the parameter, time is the time index of the parameter, category specifies the level of the response the parameter is related to if the response is categorical, and group determines which group of observations the parameter is related to in the case of random effects and loadings. Non-applicable fields in this syntax are set to NA.

See Also

Model outputs as.data.frame.dynamitefit(), as.data.table.dynamitefit(), as_draws_df.dynamitefit(),
coef.dynamitefit(), dynamite(), get_code(), get_data(), get_parameter_dims(), get_parameter_names(),
get_parameter_types(), ndraws.dynamitefit(), nobs.dynamitefit()

Examples

data.table::setDTthreads(1) # For CRAN
confint(gaussian_example_fit, level = 0.9)

---

dynamite  
Estimate a Bayesian Dynamic Multivariate Panel Model

Description

Fit a Bayesian dynamic multivariate panel model (DMPM) using Stan for Bayesian inference. The dynamite package supports a wide range of distributions and allows the user to flexibly customize the priors for the model parameters. The dynamite model is specified using standard \texttt{R} formula syntax via dynamiteformula(). For more information and examples, see 'Details' and the package vignettes.

The formula method returns the model definition as a quoted expression.

Information on the estimated dynamite model can be obtained via print including the following: The model formula, the data, the smallest effective sample sizes, largest Rhat and summary statistics of the time- and group-invariant model parameters.

The summary method provides statistics of the posterior samples of the model; this is an alias of as.data.frame.dynamitefit() with summary = TRUE.

Usage

dynamite(
  dformula,  
data, 
  time,   
group = NULL,  
priors = NULL,  
backend = "rstan",  
verbose = TRUE,

Arguments

\texttt{dformula} \quad \texttt{[dynamiteformula]}

The model formula. See \texttt{dynamiteformula()} and 'Details'.

\texttt{data} \quad \texttt{[data.frame, tibble::tibble, or data.table::data.table]}

The data that contains the variables in the model in long format. Supported column types are integer, logical, double, and factor. Columns of type character will be converted to factors. Unused factor levels will be dropped. The data can contain missing values which will simply be ignored in the estimation in a case-wise fashion (per time-point and per channel). Input data is converted to channel specific matrix representations via \texttt{stats::model.matrix.lm}.

\texttt{time} \quad \texttt{[character(1)]}

A column name of \texttt{data} that denotes the time index of observations. If this variable is a factor, the integer representation of its levels are used internally for defining the time indexing.

\texttt{group} \quad \texttt{[character(1)]}

A column name of \texttt{data} that denotes the unique groups or NULL corresponding to a scenario without any groups. If \texttt{group} is NULL, a new column \texttt{.group} is created with constant value 1L is created indicating that all observations belong to the same group. In case of name conflicts with \texttt{data}, see the \texttt{group_var} element of the return object to get the column name of the new variable.

\texttt{priors} \quad \texttt{[data.frame]}

An optional data frame with prior definitions. See \texttt{get_priors()} and 'Details'.

\texttt{backend} \quad \texttt{[character(1)]}

Defines the backend interface to Stan, should be either "rstan" (the default) or "cmdstanr". Note that cmdstanr needs to be installed separately as it is not on CRAN. It also needs the actual CmdStan software. See \url{https://mc-stan.org/cmdstanr/} for details.
```r
dynamite

verbose [logical(1)]
All warnings and messages are suppressed if set to FALSE. Defaults to TRUE. Setting this to FALSE will also disable checks for perfect collinearity in the model matrix.

verbose_stan [logical(1)]
This is the verbose argument for rstan::sampling(). Defaults to FALSE.

stanc_options [list()]
This is the stanc_options argument passed to the compile method of a CmdStanModel object via cmdstanr::cmdstan_model() when backend = "cmdstanr". Defaults to list("O0"). To enable level one compiler optimizations, use list("O1").

threads_per_chain [integer(1)]
A Positive integer defining the number of parallel threads to use within each chain. Default is 1. See rstan::rstan_options() and cmdstanr::sample() for details.

grainsize [integer(1)]
A positive integer defining the suggested size of the partial sums when using within-chain parallelization. Default is number of time points divided by threads_per_chain. Setting this to 1 leads the workload division entirely to the internal scheduler. The performance of the within-chain parallelization can be sensitive to the choice of grainsize, see Stan manual on reduce-sum for details.

custom_stan_model [character(1)]
An optional character string that either contains a customized stan model code or a path to a .stan file that contains the code. Using this will override the generated model code. For expert users only.

debug [list()]
A named list of form name = TRUE indicating additional objects in the environment of the dynamite function which are added to the return object. Additionally, values no_compile = TRUE and no_sampling = TRUE can be used to skip the compilation of the Stan code and sampling steps respectively. This can be useful for debugging when combined with model_code = TRUE, which adds the Stan model code to the return object.

... For dynamite(), additional arguments to rstan::sampling() or cmdstanr::sample(), such as chains and cores (chains and parallel_chains in cmdstanr). For summary(), additional arguments to as.data.frame.dynamitefit(). For print(), further arguments to the print method for tibbles (see tibble::formatting). Not used for formula().

x [dynamitefit]
The model fit object.

full_diagnostics
By default, the effective sample size (ESS) and Rhat are computed only for the time- and group-invariant parameters (full_diagnostics = FALSE). Setting this to TRUE computes ESS and Rhat values for all model parameters, which can take some time for complex models.
```
The model fit object.

Details

The best-case scalability of dynamite in terms of data size should be approximately linear in terms of number of time points and and number of groups, but as wall-clock time of the MCMC algorithms provided by Stan can depend on the discrepancy of the data and the model (and the subsequent shape of the posterior), this can vary greatly.

Value

dynamite returns a dynamitefit object which is a list containing the following components:

- stanfit
  A stanfit object, see rstan::sampling() for details.
- dformulas
  A list of dynamiteformula objects for internal use.
- data
  A processed version of the input data.
- data_name
  Name of the input data object.
- stan
  A list containing various elements related to Stan model construction and sampling.
- group_var
  Name of the variable defining the groups.
- time_var
  Name of the variable defining the time index.
- priors
  Data frame containing the used priors.
- backend
  Either "rstan" or "cmdstanr" indicating which package was used in sampling.
- call
  Original function call as an object of class call.

formula returns a quoted expression.
print returns x invisibly.
summary returns a data.frame.

References


See Also

Model fitting get_priors(), update.dynamitefit()
Model formula construction dynamiteformula(), lags(), lfactor(), random_spec(), splines()
Model outputs as.data.frame.dynamitefit(), as.data.table.dynamitefit(), as_draws_df.dynamitefit(),
coef.dynamitefit(), confint.dynamitefit(), get_code(), get_data(), get_parameterDims(),
get_parameter_names(), get_parameter_types(), ndraws.dynamitefit(), nobs.dynamitefit()

Examples

data.table::setDTthreads(1) # For CRAN

# Please update your rstan and StanHeaders installation before running
# on Windows
if (!identical(.Platform$OS.type, "windows")) {
  fit <- dynamite(
    dformula = obs(y ~ -1 + varying(~x), family = "gaussian") +
    lags(type = "varying") +
    splines(df = 20),
    gaussian_example,
    "time",
    "id",
    chains = 1,
    refresh = 0
  )
}


dynamiteformula  Model formula for dynamite

Description

Defines a new observational or a new auxiliary channel for the model using standard R formula
syntax. Formulas of individual response variables can be joined together via +. See 'Details' and the
package vignette for more information. The function obs is a shorthand alias for dynamiteformula,
and aux is a shorthand alias for dynamiteformula(formula, family = "deterministic").
dynamiteformula

Usage

dynamiteformula(formula, family)

obs(formula, family)
aux(formula)

## S3 method for class 'dynamiteformula'
e1 + e2

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

Arguments

formula [formula]
   An R formula describing the model.
family [character(1)]
   The family name. See ‘Details’ for the supported families.
e1 [dynamiteformula]
   A model formula specification.
e2 [dynamiteformula]
   A model formula specification.
x [dynamiteformula]
   The model formula.
...
   Ignored.

Details

Currently the dynamite package supports the following distributions for the observations:

- Categorical: categorical (with a softmax link using the first category as reference). See
  the documentation of the categorical_logit_glm in the Stan function reference manual
  (https://mc-stan.org/users/documentation/).
- Multinomial: multinomial (softmax link, first category is reference).
- Gaussian: gaussian (identity link, parameterized using mean and standard deviation).
- Multivariate Gaussian: mvgaussian (identity link, parameterized using mean vector, standard
  deviation vector and the Cholesky decomposition of the correlation matrix).
- Poisson: poisson (log-link, with an optional known offset variable).
- Negative-binomial: negbin (log-link, using mean and dispersion parameterization, with an
  optional known offset variable). See the documentation on NegBinomial2 in the Stan function
  reference manual.
- Bernoulli: bernoulli (logit-link).
- Binomial: binomial (logit-link).
- Exponential: exponential (log-link).
• Gamma: \( \text{gamma} \) (log-link, using mean and shape parameterization).

• Beta: \( \text{beta} \) (logit-link, using mean and precision parameterization).

• Student t: \( \text{student} \) (identity link, parametrized using degrees of freedom, location and scale)

The models in the \textit{dynamite} package are defined by combining the channel-specific formulas defined via \texttt{R} formula syntax. Each channel is defined via the \texttt{obs} function, and the channels are combined with \texttt{+}. For example a formula \texttt{obs(y ~ lag(x), family = "gaussian") + obs(x ~ z, family = "poisson")} defines a model with two channels; first we declare that \( y \) is a gaussian variable depending on a previous value of \( x \) \( \text{lag}(x) \), and then we add a second channel declaring \( x \) as Poisson distributed depending on some exogenous variable \( z \) (for which we do not define any distribution).

Number of trials for binomial channels should be defined via a \texttt{trials} model component, e.g., \texttt{obs(y ~ x + trials(n), family = "binomial")}, where \( n \) is a data variable defining the number of trials. For multinomial channels, the number of trials is automatically defined to be the sum of the observations over the categories, but can also be defined using the \texttt{trials} component, for example for prediction.

Multivariate channels are defined by providing a single formula for all components or by providing component-specific formulas separated by a \texttt{|}. The response variables that correspond to the components should be joined by \texttt{c()}. For instance, the following would define \( c(y_1, y_2) \) as multivariate gaussian with \( x \) as a predictor for the mean of the first component and \( x \) and \( z \) as predictors for the mean of the second component: \texttt{obs(c(y1, y2) ~ x | x + z, family = "mvgaussian")}. A multinomial channel should only have a single formula.

In addition to declaring response variables via \texttt{obs}, we can also use the function \texttt{aux} to define auxiliary channels which are deterministic functions of other variables. The values of auxiliary variables are computed dynamically during prediction, making the use of lagged values and other transformations possible. The function \texttt{aux} also does not use the \texttt{family} argument, which is automatically set to \texttt{deterministic} and is a special channel type of \texttt{obs}. Note that lagged values of deterministic \texttt{aux} channels do not imply fixed time points. Instead they must be given starting values using a special function \texttt{init} that directly initializes the lags to specified values, or by \texttt{past} which computes the initial values based on an \texttt{R} expression. Both \texttt{init} and \texttt{past} should appear on the right hand side of the model formula, separated from the primary defining expression via \texttt{|}.

The formula within \texttt{obs} can also contain an additional special function \texttt{varying}, which defines the time-varying part of the model equation, in which case we could write for example \texttt{obs(x ~ z + varying(~ -1 + w), family = "poisson")}, which defines a model equation with a constant intercept and time-invariant effect of \( z \), and a time-varying effect of \( w \). We also remove the duplicate intercept with \texttt{-1} in order to avoid identifiability issues in the model estimation (we could also define a time varying intercept, in which case we would write \texttt{obs(x ~ -1 + z + varying(~ w), family = "poisson")}).

The part of the formula not wrapped with \texttt{varying} is assumed to correspond to the fixed part of the model, so \texttt{obs(x ~ z + varying(~ -1 + w), family = "poisson")} is actually identical to \texttt{obs(x ~ -1 + fixed(~ z) + varying(~ -1 + w), family = "poisson")} and \texttt{obs(x ~ fixed(~ z) + varying(~ -1 + w), family = "poisson")}.

When defining \texttt{varying} effects, we also need to define how the these time-varying regression coefficient behave. For this, a \texttt{splines} component should be added to the model, e.g., \texttt{obs(x ~ varying(~ -1 + w), family = "poisson")}, which defines a cubic B-spline with 10 degrees of freedom for the time-varying coefficient corresponding to the \( w \). If the model contains multiple time-varying coefficients, same spline basis is used for all coefficients, with unique spline coefficients and their standard deviation.
If the desired model contains lagged predictors of each response in each channel, these can be quickly added to the model as either time-invariant or time-varying predictors via `lags()` instead of writing them manually for each channel.

It is also possible to define group-specific (random) effects term using the special syntax `random()` similarly as `varying()`. For example, `random(~1)` leads to a model where in addition to the common intercept, each individual/group has their own intercept with zero-mean normal prior and unknown standard deviation analogously with the typical mixed models. An additional model component `random_spec()` can be used to define whether the random effects are allowed to correlate within and across channels and whether to use centered or noncentered parameterization for the random effects.

**Value**

A `dynamiteformula` object.

**See Also**

Model formula construction `dynamite()`, `lags()`, `lfactor()`, `random_spec()`, `splines()`

**Examples**

data.table::setDTthreads(1) # For CRAN
# A single gaussian response channel with a time-varying effect of 'x',
# and a time-varying effect of the lag of 'y' using B-splines with
# 20 degrees of freedom for the coefficients of the time-varying terms.
obs(y ~ -1 + varying(~x), family = "gaussian") +
  lags(type = "varying") +
  splines(df = 20)

# A two-channel categorical model with time-invariant predictors
# here, lag terms are specified manually
obs(x ~ z + lag(x) + lag(y), family = "categorical") +
  obs(y ~ z + lag(x) + lag(y), family = "categorical")

# The same categorical model as above, but with the lag terms
# added using 'lags'
obs(x ~ z, family = "categorical") +
  obs(y ~ z, family = "categorical") +
  lags(type = "fixed")

# A multichannel model with a gaussian, Poisson and a Bernoulli response and
# an auxiliary channel for the logarithm of 'p' plus one
obs(g ~ lag(g) + lag(logp), family = "gaussian") +
  obs(p ~ lag(g) + lag(logp) + lag(b), family = "poisson") +
  obs(b ~ lag(b) * lag(logp) + lag(b) * lag(g), family = "bernoulli") +
  aux(numeric(logp) ~ log(p + 1))

data.table::setDTthreads(1) # For CRAN
obs(y ~ x, family = "gaussian") + obs(z ~ w, family = "exponential")

data.table::setDTthreads(1) # For CRAN
x <- obs(y ~ x + random(~ 1 + lag(d)), family = "gaussian") +
obs(z ~ varying(~w), family = "exponential") +
aux(numeric(d) ~ log(y) | init(c(0, 1))) +
lags(k = 2) +
splines(df = 5) +
random_spec(correlated = FALSE)
print(x)

---

fitted.dynamitefit  
Extract Fitted Values of a Dynamite Model

Description

Fitted values for a dynamitefit object, i.e., $E(y_{y|\text{newdata}, \theta})$ where $\theta$ contains all the model parameters. See also predict.dynamitefit() for multi-step predictions.

Usage

```r
## S3 method for class 'dynamitefit'
fitted(object, newdata = NULL, n_draws = NULL, expand = TRUE, df = TRUE, ...)
```

Arguments

- **object**  
  [dynamitefit]  
  The model fit object.

- **newdata**  
  [data.frame]  
  Data used in predictions. If NULL (default), the data used in model estimation is used for predictions as well. There should be no new time points that were not present in the data that were used to fit the model, and no new group levels can be included.

- **n_draws**  
  [integer(1)]  
  Number of posterior samples to use, default is NULL which uses all samples.

- **expand**  
  [logical(1)]  
  If TRUE (the default), the output is a single data.frame containing the original newdata and the predicted values. Otherwise, a list is returned with two components, simulated and observed, where the first contains only the predicted values, and the second contains the original newdata. Setting expand to FALSE can help conserve memory because newdata is not replicated n_draws times in the output. This argument is ignored if funs are provided.

- **df**  
  [logical(1)]  
  If TRUE (default) the output consists of data.frame objects, and data.table objects otherwise.

- **...**  
  Ignored.
**fitted.dynamitefit**

**Value**

A data.frame containing the fitted values.

**See Also**

Obtaining predictions `predict.dynamitefit()`

**Examples**

```r
data.table::setDTthreads(1)  # For CRAN
fitted(gaussian_example_fit, n_draws = 2L)

set.seed(1)
# Please update your rstan and StanHeaders installation before running
# on Windows
if (!identical(.Platform$OS.type, "windows")) {
  fit <- dynamite(
    dformula = obs(LakeHuron ~ 1, "gaussian") + lags(),
    data = data.frame(LakeHuron, time = seq_len(length(LakeHuron)), id = 1),
    time = "time",
    group = "id",
    chains = 1,
    refresh = 0
  )

  if (requireNamespace("dplyr") &&
      requireNamespace("tidyr") &&
      base::getRversion() >= "4.1.0") {
    # One-step ahead samples (fitted values) from the posterior
    # (first time point is fixed due to lag in the model):
    fitted(fit) |>  # dplyr::filter(time > 2) |
      ggplot2::ggplot(ggplot2::aes(time, LakeHuron_fitted, group = .draw)) +
      ggplot2::geom_line(alpha = 0.5) +  # observed values
      ggplot2::geom_line(ggplot2::aes(y = LakeHuron), colour = "tomato") +
      ggplot2::theme_bw()

    # Posterior predictive distribution given the first time point:
    predict(fit, type = "mean") |>  # dplyr::filter(time > 2) |
      ggplot2::ggplot(ggplot2::aes(time, LakeHuron_mean, group = .draw)) +
      ggplot2::geom_line(alpha = 0.5) +  # observed values
      ggplot2::geom_line(ggplot2::aes(y = LakeHuron), colour = "tomato") +
      ggplot2::theme_bw()
  }
}
```
Simulated Data of Gaussian Responses

Description

Simulated data containing gaussian response variables with two covariates. The dataset was generated from a model with time-varying effects of covariate x and the lagged value of the response variable, time-varying intercept, and time-invariant effect of covariate z. The time-varying coefficients vary according to a spline with 20 degrees of freedom.

Usage

```r
gaussian_example
```

Format

A data frame with 3000 rows and 5 variables:

- `y` The response variable.
- `x` A continuous covariate.
- `z` A binary covariate.
- `id` Variable defining individuals (1 to 50).
- `time` Variable defining the time point of the measurement (1 to 30).

Source

The data was generated according to a script in `https://github.com/ropensci/dynamite/blob/main/data-raw/ gaussian_example.R`

See Also

Example models `categorical_example_fit, categorical_example, gaussian_example_fit, gaussian_simulation_fit, multichannel_example_fit, multichannel_example`
Description

A dynamitefit object obtained by running dynamite on the gaussian_example dataset as

```r
set.seed(1)
library(dynamite)

gaussian_example_fit <- dynamite(
  obs(y ~ -1 + z + varying(~ x + lag(y)) + random(~1), family = "gaussian") +
  random_spec() + splines(df = 20),
  data = gaussian_example,
  time = "time",
  group = "id",
  iter = 2000,
  warmup = 1000,
  thin = 10,
  chains = 2,
  cores = 2,
  refresh = 0,
  save_warmup = FALSE,
  pars = c("omega_alpha_1_y", "omega_raw_alpha_y", "nu_raw", "nu", "L",
            "sigma_nu", "a_y"),
  include = FALSE
)
```

Note the very small number of samples due to size restrictions on CRAN.

Usage

gaussian_example_fit

Format

A dynamitefit object.

Source

The data was generated according to a script in https://github.com/ropensci/dynamite/blob/main/data-raw/ gaussian_example_fit.R

See Also

Example models categorical_example_fit, categorical_example, gaussian_example, gaussian_simulation_fit, multichannel_example_fit, multichannel_example
gaussian_simulation_fit

Model Fit for the time-varying example in the dynamite_simulation
Vignette

Description

A dynamitefit object obtained by running dynamite with the "Fixed_param" algorithm on the specified inits in the example.

```r
set.seed(1)
library(dynamite)
gaussian_simulation_fit <- dynamite(
  dformula = f,
  data = d,
  time = "time",
  group = "id",
  chains = 1,
  iter = 1,
  algorithm = "Fixed_param",
  init = list(init),
)
```

Usage

`gaussian_simulation_fit`

Format

A dynamitefit object.

Source

The data was generated according to a script in https://github.com/ropensci/dynamite/blob/main/data-raw/gaussian_simulation_fit.R

See Also

Example models `categorical_example_fit`, `categorical_example`, `gaussian_example_fit`, `gaussian_example`, `multichannel_example_fit`, `multichannel_example`
get_code

Extract the Stan Code of the Dynamite Model

Description

Returns the Stan code of the model. Mostly useful for debugging or for building a customized version of the model.

Usage

get_code(x, ...)

## S3 method for class 'dynamiteformula'
get_code(x, data, time, group = NULL, blocks = NULL, ...)

## S3 method for class 'dynamitefit'
get_code(x, blocks = NULL, ...)

Arguments

x
[dynamiteformula or dynamitefit]
The model formula or an existing dynamitefit object. See dynamiteformula() and dynamite().

... Ignored.
data
[data.frame, tibble::tibble, or data.table::data.table]
The data that contains the variables in the model in long format. Supported column types are integer, logical, double, and factor. Columns of type character will be converted to factors. Unused factor levels will be dropped. The data can contain missing values which will simply be ignored in the estimation in a case-wise fashion (per time-point and per channel). Input data is converted to channel specific matrix representations via stats::model.matrix.lm().
time
[character(1)]
A column name of data that denotes the time index of observations. If this variable is a factor, the integer representation of its levels are used internally for defining the time indexing.
group
[character(1)]
A column name of data that denotes the unique groups or NULL corresponding to a scenario without any groups. If group is NULL, a new column .group is created with constant value 1L is created indicating that all observations belong to the same group. In case of name conflicts with data, see the group_var element of the return object to get the column name of the new variable.
blocks
[character()]
Stan block names to extract. If NULL, extracts the full model code.

Value

The Stan model blocks as a character string.
get_data

Extract the Model Data of the Dynamite Model

Description

Returns the input data to the Stan model. Mostly useful for debugging.

Usage

get_data(x, ...)

## S3 method for class 'dynamiteformula'
get_data(x, data, time, group = NULL, ...)

## S3 method for class 'dynamitefit'
get_data(x, ...)

Arguments

x [dynamiteformula or dynamitefit] The model formula or an existing dynamitefit object. See dynamiteformula() and dynamite().

... Ignored.

data [data.frame, tibble::tibble, or data.table::data.table] The data that contains the variables in the model in long format. Supported column types are integer, logical, double, and factor. Columns of type character will be converted to factors. Unused factor levels will be dropped.

Examples

data.table::setDTthreads(1) # For CRAN
d <- data.frame(y = rnorm(10), x = 1:10, time = 1:10, id = 1)
cat(get_code(obs(y ~ x, family = "gaussian"),
      data = d, time = "time", group = "id")
)
# same as
cat(dynamite(obs(y ~ x, family = "gaussian"),
      data = d, time = "time", group = "id",
      debug = list(model_code = TRUE, no_compile = TRUE)
)$model_code)
The data can contain missing values which will simply be ignored in the estimation in a case-wise fashion (per time-point and per channel). Input data is converted to channel specific matrix representations via `stats::model.matrix.lm()`.

time [character(1)]
A column name of data that denotes the time index of observations. If this variable is a factor, the integer representation of its levels are used internally for defining the time indexing.

group [character(1)]
A column name of data that denotes the unique groups or NULL corresponding to a scenario without any groups. If group is NULL, a new column .group is created with constant value 1L is created indicating that all observations belong to the same group. In case of name conflicts with data, see the group_var element of the return object to get the column name of the new variable.

Value
A list containing the input data to Stan.

See Also
Model outputs `as.data.frame.dynamitefit()`, `as.data.table.dynamitefit()`, `as_draws_df.dynamitefit()`, `coef.dynamitefit()`, `confint.dynamitefit()`, `dynamite()`, `get_code()`, `get_parameter_dims()`, `get_parameter_names()`, `get_parameter_types()`, `ndraws.dynamitefit()`, `nobs.dynamitefit()`.

Examples
```
data.table::setDTthreads(1)  # For CRAN
d <- data.frame(y = rnorm(10), x = 1:10, time = 1:10, id = 1)
str(get_data(obs(y ~ x, family = "gaussian"),
  data = d, time = "time", group = "id"
))
```
Usage

get_parameter_dims(x, ...)  

## S3 method for class 'dynamiteformula'  
get_parameter_dims(x, data, time, group = NULL, ...)  

## S3 method for class 'dynamitefit'  
get_parameter_dims(x, ...)  

Arguments

- **x** [dynamiteformula or dynamitefit]
  The model formula or an existing dynamitefit object. See dynamiteformula() and dynamite().

- **...** Ignored.

- **data** [data.frame, tibble::tibble, or data.table::data.table]
  The data that contains the variables in the model in long format. Supported column types are integer, logical, double, and factor. Columns of type character will be converted to factors. Unused factor levels will be dropped. The data can contain missing values which will simply be ignored in the estimation in a case-wise fashion (per time-point and per channel). Input data is converted to channel specific matrix representations via stats::model.matrix.lm().

- **time** [character(1)]
  A column name of data that denotes the time index of observations. If this variable is a factor, the integer representation of its levels are used internally for defining the time indexing.

- **group** [character(1)]
  A column name of data that denotes the unique groups or NULL corresponding to a scenario without any groups. If group is NULL, a new column .group is created with constant value 1L is created indicating that all observations belong to the same group. In case of name conflicts with data, see the group_var element of the return object to get the column name of the new variable.

Value

A named list with all parameter dimensions of the input model.

See Also

Model outputs as.data.frame.dynamitefit(), as.data.table.dynamitefit(), as_draws_df.dynamitefit(), coef.dynamitefit(), confint.dynamitefit(), dynamite(), get_code(), get_data(), get_parameter_names(), get_parameter_types(), ndraws.dynamitefit(), nosbs.dynamitefit()

Examples

```
data.table::setDTthreads(1) # For CRAN  
get_parameter_dims(multichannel_example_fit)
```
**get_parameter_names**

*Get Parameter Names of the Dynamite Model*

**Description**

Extracts all parameter names of used in the dynamitefit object.

**Usage**

```r
get_parameter_names(x, types = NULL, ...)

## S3 method for class 'dynamitefit'
get_parameter_names(x, types = NULL, ...)
```

**Arguments**

- `x` [dynamitefit]
  - The model fit object.
- `types` [character()]
  - Extract only names of parameter of a certain type. See `get_parameter_types()`.
- `...`
  - Ignored.

**Details**

The naming of parameters generally follows style where the name starts with the parameter type (e.g. beta for time-invariant regression coefficient), followed by underscore and the name of the response variable, and in case of time-invariant, time-varying or random effect, the name of the predictor. An exception to this is spline coefficients omega, which also contain the number denoting the knot number.

**Value**

A character vector with parameter names of the input model.

**See Also**

Model outputs `as.data.frame.dynamitefit()`, `as.data.table.dynamitefit()`, `as_draws_df.dynamitefit()`, `coef.dynamitefit()`, `confint.dynamitefit()`, `dynamite()`, `get_code()`, `get_data()` , `get_parameter_dims()`, `get_parameter_types()`, `ndraws.dynamitefit()`, `nobs.dynamitefit()`

**Examples**

```r
data.table::setDTthreads(1) # For CRAN
get_parameter_names(multichannel_example_fit)
```
get_parameter_types  Get Parameter Types of the Dynamite Model

Description
Extracts all parameter types of used in the dynamitefit object. See `as.data.frame.dynamitefit()` for explanations of different types.

Usage
get_parameter_types(x, ...)

## S3 method for class 'dynamitefit'
get_parameter_types(x, ...)

Arguments
- `x` [dynamitefit]
  The model fit object.
- `...` Ignored.

Value
A character vector with all parameter types of the input model.

See Also
Model outputs `as.data.frame.dynamitefit()`, `as.data.table.dynamitefit()`, `as_draws_df.dynamitefit()`, `coef.dynamitefit()`, `confint.dynamitefit()`, `dynamite()`, `get_code()`, `get_data()`, `get_parameter dims()`, `get_parameter_names()`, `ndraws.dynamitefit()`, `nobs.dynamitefit()`

Examples
```r
data.table::setDTthreads() # For CRAN
get_parameter_types(multichannel_example_fit)
```

get_priors  Get Prior Definitions of a Dynamite Model

Description
Extracts the priors used in the dynamite model as a data frame. You can then alter the priors by changing the contents of the `prior` column and supplying this data frame to dynamite function using the argument `priors`. See vignettes for details.
get_priors

Usage

get_priors(x, ...)

## S3 method for class 'dynamiteformula'
get_priors(x, data, time, group = NULL, ...)

## S3 method for class 'dynamitefit'
get_priors(x, ...)

Arguments

x [dynamiteformula or dynamitefit]
The model formula or an existing dynamitefit object. See dynamiteformula() and dynamite().

... Ignored.
data [data.frame, tibble::tibble, or data.table::data.table]
The data that contains the variables in the model in long format. Supported column types are integer, logical, double, and factor. Columns of type character will be converted to factors. Unused factor levels will be dropped. The data can contain missing values which will simply be ignored in the estimation in a case-wise fashion (per time-point and per channel). Input data is converted to channel specific matrix representations via stats::model.matrix.lm().
time [character(1)]
A column name of data that denotes the time index of observations. If this variable is a factor, the integer representation of its levels are used internally for defining the time indexing.
group [character(1)]
A column name of data that denotes the unique groups or NULL corresponding to a scenario without any groups. If group is NULL, a new column .group is created with constant value 1L is created indicating that all observations belong to the same group. In case of name conflicts with data, see the group_var element of the return object to get the column name of the new variable.

Value

A data.frame containing the prior definitions.

Note

Only the prior column of the output should be altered when defining the user-defined priors for the dynamite.

See Also

Model fitting dynamite(), update.dynamitefit()
Examples

```r
data.table::setDTthreads(1) # For CRAN
d <- data.frame(y = rnorm(10), x = 1:10, time = 1:10, id = 1)
get_priors(obs(y ~ x, family = "gaussian"),
    data = d, time = "time", group = "id")
```

### lags

| Add Lagged Responses as Predictors to Each Channel of a Dynamite Model |

**Description**

Adds the lagged value of the response of each channel specified via `dynamiteformula()` as a predictor to each channel. The added predictors can be either time-varying or time-invariant.

**Usage**

```r
lags(k = 1L, type = c("fixed", "varying", "random"))
```

**Arguments**

- **k**
  - [integer()]
  - Values lagged by k units of time of each observed response variable will be added as a predictor for each channel. Should be a positive (unrestricted) integer.

- **type**
  - [integer(1)]
  - Either "fixed" or "varying" which indicates whether the coefficients of the added lag terms should vary in time or not.

**Value**

An object of class lags.

**See Also**

Model formula construction `dynamiteformula()`, `dynamite()`, `lfactor()`, `random_spec()`, `splines()`

**Examples**

```r
data.table::setDTthreads(1) # For CRAN
obs(y ~ -1 + varying(~x), family = "gaussian") +
  lags(type = "varying") + splines(df = 20)
```

# A two-channel categorical model with time-invariant predictors
# here, lag terms are specified manually
obs(x ~ z + lag(x) + lag(y), family = "categorical") +
obs(y ~ z + lag(x) + lag(y), family = "categorical")
# The same categorical model as above, but with the lag terms
# added using 'lags'
obs(x ~ z, family = "categorical") +
  obs(y ~ z, family = "categorical") +
  lags(type = "fixed")

\textbf{lfactor} \hspace{1cm} \textit{Define a Common Latent Factor for the Dynamite Model.}

\textbf{Description}

This function can be used as part of \texttt{dynamiteformula()} to define a common latent factor component. The latent factor is modeled as a spline similarly as a time-varying intercept, but instead of having equal effect on each group, there is an additional loading variable for each group so that in the linear predictor we have a term $\lambda_i \psi_t$ for each group $i$. In order to keep the full factor loadings $\lambda$, the latent factor $\psi$ and the full model identifiable, some restrictions are added to the model. Details will be available in an upcoming paper. This component should be treated as experimental feature.

\textbf{Usage}

\begin{verbatim}
lfactor(
  responses = NULL,
  nonzero_lambda = TRUE,
  correlated = TRUE,
  noncentered_psi = FALSE
)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{responses} [character()]
    Names of the responses that the factor should affect. Default is all responses defined with \texttt{obs} except categorical responses, which do not (yet) support the factor component.
  \item \texttt{nonzero_lambda} [logical()]
    If TRUE (the default), assumes that the mean of factor loadings is nonzero or not. Should be a logical vector matching the length of \texttt{responses} or a single logical value in case \texttt{responses} is NULL. See details.
  \item \texttt{correlated} [logical()]
    If TRUE (the default), the latent factors are assumed to be correlated between channels.
  \item \texttt{noncentered_psi} [logical(1)]
    If TRUE, uses a noncentered parametrization for spline coefficients of all the factors. The number of knots is based \texttt{splines()} call.
\end{itemize}
Value

An object of class latent_factor.

See Also

Model formula construction dynamiteformula(), dynamite(), lags(), random_spec(), splines()

Examples

data.table::setDTthreads(1) # For CRAN
# three channel model with common factor affecting for responses x and y
obs(y ~ 1, family = "gaussian") +
obs(x ~ 1, family = "poisson") +
obs(z ~ 1, family = "gaussian") +
lfactor(
  responses = c("y", "x"), nonzero_lambda = c(TRUE, FALSE),
  correlated = TRUE, noncentered_psi = FALSE
)

lfo

Approximate Leave-Future-Out (LFO) Cross-validation

Description

Estimates the leave-future-out (LFO) information criterion for dynamite models using Pareto smoothed importance sampling.

Usage

lfo(x, L, verbose = TRUE, k_threshold = 0.7, ...)

Arguments

x [dynamitefit]
The model fit object.

L [integer(1)]
Positive integer defining how many time points should be used for the initial fit.

verbose [logical(1)]
If TRUE (default), print the progress of the LFO computations to the console.

k_threshold [numeric(1)]
Threshold for the Pareto k estimate triggering refit. Default is 0.7.

... Additional arguments passed to rstan::sampling() or cmdstanr::sample(), such as chains and cores (parallel_chains in cmdstanr).
Details

For multichannel models, the log-likelihoods of all channels are combined. For models with groups, expected log predictive densities (ELPDs) are computed independently for each group, but the re-estimation of the model is triggered if pareto k values of any group exceeds the threshold.

Value

An lfo object which is a list with the following components:

- **ELPD**
  Expected log predictive density estimate.

- **ELPD_SE**
  Standard error of ELPD. This is a crude approximation which does not take into account potential serial correlations.

- **pareto_k**
  Pareto k values.

- **refits**
  Time points where model was re-estimated.

- **L**
  L value used in the LFO estimation.

- **k_threshold**
  Threshold used in the LFO estimation.

References


See Also

Model diagnostics `loo.dynamitefit()`, `mcmc_diagnostics()`

Examples

data.table::.setDTthreads(1) # For CRAN

# Please update your rstan and StanHeaders installation before running # on Windows
if (!identical(.Platform$OS.type, "windows")){
  # this gives warnings due to the small number of iterations
  out <- suppressWarnings(
    lfo(gaussian_example_fit, L = 20, chains = 1, cores = 1)
  )
  out$ELPD
  out$ELPD_SE
}
Approximate Leave-One-Out (LOO) Cross-validation

Description

Estimates the leave-one-out (LOO) information criterion for dynamite models using Pareto smoothed importance sampling with the loo package.

Usage

## S3 method for class 'dynamitefit'
loo(x, separate_channels = FALSE, ...)

Arguments

x
[dynamitefit]
The model fit object.

separate_channels
[logical(1)]
If TRUE, computes LOO separately for each channel. This can be useful in diagnosing where the model fails. Default is FALSE, in which case the likelihoods of different channels are combined, i.e., all channels of are left out.

...
Ignored.

Value

An output from loo::loo() or a list of such outputs (if separate_channels was TRUE).

References


See Also

Model diagnostics lfo(), mcmc_diagnostics()

Examples

data.table::setDTthreads(1) # For CRAN

# Please update your rstan and StanHeaders installation before running # on Windows
if (!identical(.Platform$OS.type, "windows")) {
  # this gives warnings due to the small number of iterations
  suppressWarnings(loo(gaussian_example_fit))
  suppressWarnings(loo(gaussian_example_fit, separate_channels = TRUE))
}
mcmc_diagnostics Description

Prints HMC diagnostics, and lists parameters with smallest effective sample sizes and largest Rhat values. See `rstan::check_hmc_diagnostics()` and `posterior::default_convergence_measures()` for details.

Usage

```
mcmc_diagnostics(x, n)
```

## S3 method for class 'dynamitefit'
mcmc_diagnostics(x, n = 3L)

Arguments

- `x` [dynamitefit]
The model fit object.
- `n` [integer(1)]
  How many rows to print in parameter-specific convergence measures. The default is 3. Should be a positive (unrestricted) integer.

Value

Returns `x` (invisibly).

See Also

Model diagnostics `lfo()`, `loo.dynamitefit()`

Examples

```
data.table::setDTthreads(1) # For CRAN
mcmc_diagnostics(gaussian_example_fit)
```
**multichannel_example**  
*Simulated Multivariate Panel Data*

**Description**

A simulated multichannel data containing multiple individuals with multiple response variables of different distributions.

**Usage**

`multichannel_example`

**Format**

A data frame with 3000 rows and 5 variables:

- **id** Variable defining individuals (1 to 50).
- **time** Variable defining the time point of the measurement (1 to 20).
- **g** Response variable following gaussian distribution.
- **p** Response variable following Poisson distribution.
- **b** Response variable following Bernoulli distribution.

**Source**

The data was generated according to a script in [https://github.com/ropensci/dynamite/blob/main/data-raw/multichannel_example.R](https://github.com/ropensci/dynamite/blob/main/data-raw/multichannel_example.R)

**See Also**

Example models `categorical_example_fit, categorical_example, gaussian_example_fit, gaussian_example, gaussian_simulation_fit, multichannel_example_fit`
**Description**

A dynamitefit object obtained by running dynamite on the multichannel_example dataset as

```r
set.seed(1)
library(dynamite)
f <- obs(g ~ lag(g) + lag(logp), family = "gaussian") +
    obs(p ~ lag(g) + lag(logp) + lag(b), family = "poisson") +
    obs(b ~ lag(b) * lag(logp) + lag(b) * lag(g), family = "bernoulli") +
    aux(numeric(logp) ~ log(p + 1))
multichannel_example_fit <- dynamite(f,
    data = multichannel_example,
    time = "time",
    group = "id",
    chains = 1,
    cores = 1,
    iter = 2000,
    warmup = 1000,
    init = 0,
    refresh = 0,
    thin = 5,
    save_warmup = FALSE)
```

Note the small number of samples due to size restrictions on CRAN.

**Usage**

`multichannel_example_fit`

**Format**

A dynamitefit object.

**Source**


**See Also**

Example models `categorical_example_fit`, `categorical_example`, `gaussian_example_fit`, `gaussian_example`, `gaussian_simulation_fit`, `multichannel_example`
**ndraws.dynamitefit**

Return the Number of Posterior Draws of a `dynamitefit` Object

**Description**

Return the Number of Posterior Draws of a `dynamitefit` Object

**Usage**

```r
## S3 method for class 'dynamitefit'
ndraws(x)
```

**Arguments**

- `x` [dynamitefit]
  
The model fit object.

**Value**

Number of posterior draws as a single integer value.

**See Also**

Model outputs `as.data.frame.dynamitefit()`, `as.data.table.dynamitefit()`, `as_draws_df.dynamitefit()`, `coef.dynamitefit()`, `confint.dynamitefit()`, `dynamite()`, `get_code()`, `get_data()`, `get_parameter_dims()`, `get_parameter_names()`, `get_parameter_types()`, `nobs.dynamitefit()`

**Examples**

```r
data.table::setDTthreads(1) # For CRAN
ndraws(gaussian_example_fit)
```

---

**nobs.dynamitefit**

Extract the Number of Observations Used to Fit a Dynamite Model

**Description**

Extract the Number of Observations Used to Fit a Dynamite Model

**Usage**

```r
## S3 method for class 'dynamitefit'
nobs(object, ...)
```

**Examples**

```r
nobs.dynamitefit(gaussian_example_fit)
```
plot.dynamitefit

Arguments

object [dynamitefit]
The model fit object.
...
Not used.

Value
Total number of non-missing observations as an integer.

See Also
Model outputs as.data.frame.dynamitefit(), as.data.table.dynamitefit(), as_draws_df.dynamitefit(),
coef.dynamitefit(), confint.dynamitefit(), dynamite(), get_code(), get_data(), get_parameter_dims(),
get_parameter_names(), get_parameter_types(), ndraws.dynamitefit()

Examples

data.table::setDTthreads(1) # For CRAN
nobs(gaussian_example_fit)

________________________________________________________________________

plot.dynamitefit Traceplots and Density Plots for a dynamitefit Object

Description
Produces the traceplots and the density plots of the model parameters.

Usage

## S3 method for class 'dynamitefit'
plot(x, parameters = NULL, type = NULL, responses = NULL, ...)

Arguments

x [dynamitefit]
The model fit object.

parameters [character()]\ Parameter name(s) for which the plots should be drawn. Possible options can be found with the function get_parameter_names(). The default is all parameters of a specific type for all responses, which can lead to too crowded a plot.

type [character(1)]
Type of the parameter for which the plots should be drawn. Possible options can be found with the function get_parameter_types(). Ignored if the argument parameters is supplied.
plot.lfo

Diagnostic Plot for Pareto k Values from LFO

Description

Plots Pareto k values per each time point (with one point per group), together with a horizontal line representing the used threshold.

Usage

```r
## S3 method for class 'lfo'
plot(x, ...)
```

Arguments

- `x` [lfo]
  
  Output from the lfo function.

- `...`
  
  Ignored.

Value

A ggplot object.

See Also

Drawing plots `plot_betas`, `plot_deltas`, `plot_lambdas`, `plot_nus`, `plot_psis`.

Examples

```r
data.table::setDTthreads(1) # For CRAN
plot(gaussian_example_fit, type = "beta")
```
Examples

data.table::setDTthreads(1) # For CRAN

# Please update your rstan and StanHeaders installation before running
# on Windows
if (!identical(.Platform$OS.type, "windows")) {
  # This gives warnings due to the small number of iterations
  plot(suppressWarnings(
    lfo(gaussian_example_fit, L = 20, chains = 1, cores = 1)
  ))
}

plot_betas

Plot Time-invariant Regression Coefficients of a Dynamite Model

Description

Plot Time-invariant Regression Coefficients of a Dynamite Model

Usage

plot_betas(
  x,
  parameters = NULL,
  responses = NULL,
  level = 0.05,
  include_alpha = TRUE
)

Arguments

x [dynamitefit]
The model fit object

parameters [character()] Parameter name(s) for which the plots should be drawn. Possible options can be found with function get_parameter_names(types = "beta").

responses [character()]
Response(s) for which the coefficients should be drawn. Possible options are elements of unique(x$priors$response), and the default is this whole vector.

level [numeric(1)]
Level for posterior intervals. Default is 0.05, leading to 90% intervals.

include_alpha [logical(1)]
If TRUE (default), plots also the time-invariant alphas if such parameters exists in the model.
plot_deltas

Value

A ggplot object.

See Also

Drawing plots `plot.dynamitefit()`, `plot_deltas()`, `plot_lambdas()`, `plot_nus()`, `plot_psis()`.

Examples

```r
data.table::setDTthreads(1) # For CRAN
plot_betas(gaussian_example_fit, level = 0.1)
```

---

plot_deltas  
Plot Time-varying Regression Coefficients of a Dynamite Model

Description

Plot Time-varying Regression Coefficients of a Dynamite Model

Usage

```r
plot_deltas(
  x, 
  parameters = NULL, 
  responses = NULL, 
  level = 0.05, 
  alpha = 0.5, 
  scales = c("fixed", "free"), 
  include_alpha = TRUE 
)
```

Arguments

- **x**  
  [dynamitefit]  
  The model fit object

- **parameters**  
  [character()]  
  Parameter name(s) for which the plots should be drawn. Possible options can be found with function `get_parameter_names(types = "delta")`.

- **responses**  
  [character()]  
  Response(s) for which the coefficients should be drawn. Possible options are elements of `unique(x$priors$response)`, and the default is this whole vector.

- **level**  
  [numeric(1)]  
  Level for posterior intervals. Default is 0.05, leading to 90% intervals.

- **alpha**  
  [numeric(1)]  
  Opacity level for `geom_ribbon`. Default is 0.5.
plot_lambdas

scales [character(1)] Should y-axis of the panels be "fixed" (the default) or "free"? See ggplot2::facet_wrap().
include_alpha [logical(1)] If TRUE (default), plots also the time-varying alphas if such parameters exists in the model.

Value
A ggplot object.

See Also
Drawing plots plot.dynamitefit(), plot.betas(), plot.lambdas(), plot.nus(), plot.psis()

Examples

```r
data.table::setDTthreads(1) # For CRAN
plot_deltas(gaussian_example_fit, level = 0.025, scales = "free") +
  ggplot2::theme_minimal()
```

---

plot_lambdas

Plot Factor Loadings of a Dynamite Model

Description
Plot Factor Loadings of a Dynamite Model

Usage

```r
plot_lambdas(x, responses = NULL, level = 0.05)
```

Arguments

- `x` [dynamitefit] The model fit object
- `responses` [character()] Response(s) for which the coefficients should be drawn. Possible options are elements of unique(x$priors$response), and the default is this whole vector.
- `level` [numeric(1)] Level for posterior intervals. Default is 0.05, leading to 90% intervals.

Value
A ggplot object.

See Also
Drawing plots plot.dynamitefit(), plot.betas(), plot.deltas(), plot.nus(), plot.psis()
plot_nus

Plot Random effects of a Dynamite Model

Description

Note that as this function tries to draw a plot containing effects of all groups, the plot will become messy with a large number of groups.

Usage

plot_nus(x, parameters = NULL, responses = NULL, level = 0.05, groups = NULL)

Arguments

- **x** : [dynamitefit]
The model fit object
- **parameters** : [character()]
  Parameter name(s) for which the plots should be drawn. Possible options can be found with function `get_parameter_names(types = "delta")`.
- **responses** : [character()]
  Response(s) for which the coefficients should be drawn. Possible options are elements of `unique(x$priors$response)`, and the default is this whole vector.
- **level** : [numeric(1)]
  Level for posterior intervals. Default is 0.05, leading to 90% intervals.
- **groups** : Group name(s) for which the plots should be drawn. Default is all groups.

Value

A ggplot object.

See Also

Drawing plots `plot_dynamitefit()`, `plot_betas()`, `plot_deltas()`, `plot_lambdas()`, `plot_psis()`

Examples

data.table::setDTthreads(1) # For CRAN
plot_nus(gaussian_example_fit)
Plot Latent Factors of a Dynamite Model

Description

Plot Latent Factors of a Dynamite Model

Usage

plot_psis(
  x,
  responses = NULL,
  level = 0.05,
  alpha = 0.5,
  scales = c("fixed", "free")
)

Arguments

x [dynamitefit]
  The model fit object

responses [character()]
  Response(s) for which the coefficients should be drawn. Possible options are elements of unique(x$priors$response), and the default is this whole vector.

level [numeric(1)]
  Level for posterior intervals. Default is 0.05, leading to 90% intervals.

alpha [numeric(1)]
  Opacity level for geom_ribbon. Default is 0.5.

scales [character(1)] Should y-axis of the panels be "fixed" (the default) or "free"?
  See ggplot2:::facet_wrap().

Value

A ggplot object.

See Also

Drawing plots plot.dynamitefit(), plot_betas(), plot_deltas(), plot_lambdas(), plot_nus()
predict.dynamitefit  Predict Method for a Dynamite Model

Description

Obtain counterfactual predictions for a dynamitefit object.

Usage

```r
## S3 method for class 'dynamitefit'
predict(
  object,
  newdata = NULL,
  type = c("response", "mean", "link"),
  funs = list(),
  impute = c("none", "locf", "nocb"),
  new_levels = c("none", "bootstrap", "gaussian", "original"),
  global_fixed = FALSE,
  n_draws = NULL,
  expand = TRUE,
  df = TRUE,
  ...
)
```

Arguments

- `object` [dynamitefit]
  The model fit object.

- `newdata` [data.frame]
  Data used in predictions. Predictions are computed for missing (NA) values in the response variable columns, and non-missing values are assumed fixed. If NULL (default), the data used in model estimation is used for predictions as well, after all values in the response variable columns after the first fixed time point are converted to NA values. Missing values in predictor columns can be imputed (argument impute). There should be no new time points that were not present in the data that were used to fit the model. New group levels can be included, but if the model contains random effects, an option for the random effects for the new levels must be chosen (argument new_levels). If the grouping variable of the original data is missing, it is assumed that all observations in newdata belong to the first group in the original data. New group levels are not allowed for models using latent factors.

- `type` [character(1)]
  Type of prediction, "response" (default), "mean", or "link".

- `funs` [list()]
  A named list whose names should correspond to the response variables of the model. Each element of funs should be a a named list of functions that will be
applied to the corresponding predicted type of the channel over the individuals for each combination of the posterior draws and time points. In other words, the resulting predictions will be averages over the individuals. The functions should take the corresponding type variable values as their only argument. If funs is empty, the full individual level values are returned instead. Note that this argument can only be used if there are multiple individuals (i.e., group was not NULL in the dynamite call).

**impute**

[character(1)]
Which imputation scheme to use for missing exogenous predictor values. Currently supported options are no imputation: "none" (default), last observation carried forward: "locf", and next observation carried backward: "nocb".

**new_levels**

[character(1)]
Defines if and how to sample the random effects for observations whose group level was not present in the original data. The options are:

- "none" (the default) which will signal an error if new levels are encountered.
- "bootstrap" which will randomly draw from the posterior samples of the random effects across all original levels.
- "gaussian" which will randomly draw from a gaussian distribution using the posterior samples of the random effects standard deviation (and correlation matrix if applicable).
- "original" which will randomly match each new level to one of the original levels. The posterior samples of the random effects of the matched levels will then be used for the new levels.

This argument is ignored if the model does not contain random effects.

**global_fixed**

[logical(1)]
If FALSE (the default), the first non-fixed time point is counted from the the first non-NA observation for each group member separately. Otherwise, the first non-fixed time point is counted from the first time point globally. If there are no groups, then the options are equivalent.

**n_draws**

[integer(1)]
Number of posterior samples to use, default is NULL which uses all samples.

**expand**

[logical(1)]
If TRUE (the default), the output is a single data.frame containing the original newdata and the predicted values. Otherwise, a list is returned with two components, simulated and observed, where the first contains only the predicted values, and the second contains the original newdata. Setting expand to FALSE can help conserve memory because newdata is not replicated n_draws times in the output. This argument is ignored if funs are provided.

**df**

[logical(1)]
If TRUE (default) the output consists of data.frame objects, and data.table objects otherwise.

... Ignored.
Details

Note that forecasting (i.e., predictions for time indices beyond the last time index in the original data) is not supported by the dynamite package. However, such predictions can be obtained by augmenting the original data with NA values before model estimation.

Value

A data.frame containing the predicted values or a list of two data.frames. See the expand argument for details. Note that the .draw column is not the same as .draw from as.data.frame and as_draws methods as predict uses permuted samples. A mapping between these variables can be done using information in object$stanfit@sim$permutation.

See Also

Obtaining predictions fitted.dynamitefit()

Examples

data.table::setDTthreads(1) # For CRAN
out <- predict(gaussian_example_fit, type = "response", n_draws = 2L)
head(out)

# using summary functions
sumr <- predict(multichannel_example_fit, type = "mean",
    funs = list(g = list(m = mean, s = sd), b = list(sum = sum)),
    n_draws = 2L)
head(sumr$simulated)

# Please update your rstan and StanHeaders installation before running
# on Windows
if (!identical(.Platform$OS.type, "windows")) {
    # Simulate from the prior predictive distribution
    f <- obs(y ~ lag(y) + varying(~ -1 + x), "gaussian") +
        splines(df = 10, noncentered = TRUE)

    # Create data with missing observations
    # Note that due to the lagged term in the model,
    # we need to fix the first time point
    d <- data.frame(y = c(0, rep(NA, 49)), x = rnorm(50), time = 1:50)

    # Suppress warnings due to the lack of data
    suppressWarnings(
        priors <- get_priors(f, data = d, time = "time")
    )

    # Modify default priors which can produce exploding behavior when used
    # without data
    priors$prior <- c(
        "normal(0, 1)",
        "normal(0.6, 0.1)",
        "normal(0.4, 0.01)"
    )

    # Add prior to Stan
    priors$stanfit <- stanfit$mod$stanfit
"normal(-0.2, 0.5)",
"normal(0.2, 0.1)",
"normal(0.5, 0.1)"
)

# Samples from the prior conditional on the first time point and x
fit <- dynamite(
  dformula = f,
  data = d,
  time = "time",
  verbose = FALSE,
  priors = priors,
  chains = 1
)

# Simulate new data
pp <- predict(fit)

ggplot2::ggplot(pp, ggplot2::aes(time, y_new, group = .draw)) +
  ggplot2::geom_line(alpha = 0.1) +
  ggplot2::theme_bw()
}

---

**print.lfo**

*Print the results from the LFO*

---

**Description**

Prints the summary of the leave-future-out cross-validation.

**Usage**

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

**Arguments**

- `x`  
  
  `x [lfo]`
  
  Output of the lfo method.

- `...`  
  
  Ignored.

**Value**

Returns `x` invisibly.
Examples

data.table::setDTthreads(1) # For CRAN

# Please update your rstan and StanHeaders installation before running
# on Windows
if (!identical(.Platform$OS.type, "windows")) {
  # This gives warnings due to the small number of iterations
  suppressWarnings(lfo(gaussian_example_fit, L = 20))
}

random_spec

Additional Specifications for the Group-level Random Effects of the DMPM

Description

This function can be used as part of dynamiteformula() to define whether the group-level random
effects should be modeled as correlated or not.

Usage

random_spec(correlated = TRUE, noncentered = TRUE)

Arguments

correlated [logical(1)]
  If TRUE (the default), correlations of random effects are modeled as multivariate normal.
noncentered [logical(1)]
  If TRUE (the default), use a noncentered parameterization for random effects.
  Try changing this if you encounter divergences or other problems in sampling.

Details

With a large number of time points random intercepts can become challenging sample with default
priors. This is because with large group sizes the group-level intercepts tend to be behave similarly
to fixed group-factor variable so the model becomes overparameterized given these and the common
intercept term. Another potential cause for sampling problems is relatively large variation in the
intercepts (large sigma_nu) compared to the sampling variation (sigma) in the Gaussian case.

Value

An object of class random_spec.

See Also

Model formula construction dynamiteformula(), dynamite(), lags(), lfactor(), splines()
Examples

data.table::setDTthreads(1) # For CRAN
# two channel model with correlated random effects for responses x and y
obs(y ~ 1 + random(~1), family = "gaussian") +
obs(x ~ 1 + random(~1 + z), family = "poisson") +
random_spec(correlated = TRUE)

---

splines

**Define the B-splines Used for the Time-varying Coefficients of the Model.**

Description

This function can be used as part of `dynamiteformula()` to define the splines used for the time-varying coefficients \( \delta \).

Usage

```r
splines(
  df = NULL,
  degree = 3L,
  lb_tau = 0,
  noncentered = FALSE,
  override = FALSE
)
```

Arguments

- `df` [integer(1)]
  Degrees of freedom, i.e., the total number of spline coefficients. See `splines::bs()`.
  Note that the knots are always defined as an equidistant sequence on the interval starting from the first non-fixed time point to the last time point in the data. See `dynamiteformula()` for more information on fixed time points. Should be an (unrestricted) positive integer.

- `degree` [integer(1)]
  See `splines::bs()`. Should be an (unrestricted) positive integer.

- `lb_tau` [numeric(1)]
  Hard constraint(s) on the lower bound of the standard deviation parameters \( \tau \) of the random walk priors. Can be useful in avoiding divergences in some cases. See also the noncentered argument. Can be a single positive value, or vector defining the lower bound separately for each channel, even for channels without varying effects. The ordering is based on the order of channel definitions in the `dynamiteformula` object.
noncentered [logical()]
If TRUE, use a noncentered parameterization for the spline coefficients. Default is FALSE. Try changing this if you encounter divergences or other problems in sampling for example when simulating from prior predictive distribution. Can be a single logical value, or vector of logical values, defining the parameterization separately for each channel, even for channels without varying effects.

override [logical(1)]
If FALSE (the default), an existing definition for the splines will not be overridden by another call to splines(). If TRUE, any existing definitions will be replaced.

Value
An object of class splines.

See Also
Model formula construction dynamiteformula(), dynamite(), lags(), 1factor(), random_spec()

Examples

data.table::setDTthreads(1) # For CRAN
# Two channel model with varying effects, with explicit lower bounds for the
# random walk prior standard deviations, with noncentered parameterization
# for the first channel and centered for the second channel.
obs(y ~ 1, family = "gaussian") + obs(x ~ 1, family = "gaussian") +
  lags(type = "varying") +
  splines(
    df = 20, degree = 3, lb_tau = c(0, 0.1),
    noncentered = c(TRUE, FALSE)
  )

Description
Note that using a different backend for the original model fit and when updating can lead to an error due to different naming in cmdstanr and rstan sampling arguments.

Usage

## S3 method for class 'dynamitefit'
update(
  object,
  dformula = NULL,
  data = NULL,
  priors = NULL,
)
Arguments

object [dynamitefit]
    The model fit object.

dformula [dynamiteformula]
    Updated model formula. By default the original formula is used.

data [data.frame, tibble::tibble, or data.table::data.table]
    Data for the updated model. By default original data is used.

priors [data.frame]
    Updated priors. By default the priors of the original model are used.

recompile [logical(1)]
    Should the model be recompiled? If NULL (default), tries to avoid recompilation.
    Recompilation is forced when the model formula or the priors are changed, or if
    the new data contains missing values in a channel which did not contain missing
    values in the original data. Recompilation is also forced in case the backend
    previous or new backend is cmdstanr.

Value

An updated dynamitefit object.

See Also

Model fitting dynamite(), get_priors()

Examples

data.table::setDTthreads(1) # For CRAN
## Not run:
# re-estimate the example fit without thinning:
# As the model is compiled on Windows, this will fail on other platforms
if (identical(.Platform$OS.type, "windows")) {
  fit <- update(gaussian_example_fit, thin = 1)
}
## End(Not run)
Index

* datasets
  categorical_example, 10
categorical_example_fit, 11
gaussian_example, 24
gaussian_example_fit, 24
gaussian_simulation_fit, 26
multichannel_example, 40
multichannel_example_fit, 40

* diagnostics
  lfo, 36
  loo.dynamitefit, 38
  mcmc_diagnostics, 39

* examples
  categorical_example, 10
categorical_example_fit, 11
gaussian_example, 24
gaussian_example_fit, 24
gaussian_simulation_fit, 26
multichannel_example, 40
multichannel_example_fit, 40

* fitting
  dynamite, 14
  get_priors, 32
  update.dynamitefit, 56

* formulas
  dynamite, 14
dynamiteformula, 18
  lags, 34
  lfactor, 35
  random_spec, 54
  splines, 55

* output
  as.data.frame.dynamitefit, 3
  as.data.table.dynamitefit, 7
  as_draws_df.dynamitefit, 8
  coef.dynamitefit, 12
  confint.dynamitefit, 13
  dynamite, 14
  get_code, 27
  get_data, 28
  get_parameter_dims, 29
  get_parameter_names, 31
  get_parameter_types, 32
  ndraws.dynamitefit, 42
  nobs.dynamitefit, 42

* plotting
  plot.dynamitefit, 43
  plot_betas, 45
  plot_deltas, 46
  plot_lambdas, 47
  plot_nus, 48
  plot_psis, 49

* prediction
  fitted.dynamitefit, 22
  predict.dynamitefit, 50
  +.dynamiteformula(dynamiteformula), 18
  as.data.frame(), 8
  as.data.frame.dynamitefit, 3, 8, 9, 13, 14,
  18, 28–32, 42, 43
  as.data.frame.dynamitefit(), 7, 9, 14, 16,
  32
  as.data.table
    (as.data.table.dynamitefit), 7
  as.data.table.dynamitefit, 6, 7, 9, 13, 14,
  18, 28–32, 42, 43
  as_draws(as_draws_df.dynamitefit), 8
  as_draws(), 6
  as_draws_df(as_draws_df.dynamitefit), 8
  as_draws_df.dynamitefit, 6, 8, 8, 13, 14,
  18, 28–32, 42, 43
  aux(dynamiteformula), 18
  categorical_example, 10, 11, 24–26, 40, 41
  categorical_example_fit, 10, 11, 24–26,
  40, 41
  cmdstanr::cmdstan_model(), 16
  cmdstanr::sample(), 16, 36
coef.dynamitefit, 6, 8, 9, 12, 14, 18, 28–32, 42, 43
coint.dynamitefit, 6, 8, 9, 13, 18, 28–32, 42, 43
dynamite, 6, 8, 9, 13, 14, 18, 28–34, 36, 42, 43, 54, 56, 57
dynamite(), 3, 8, 27, 28, 30, 33
dynamite-package, 3
dynamiteformula, 18, 19, 34, 36, 54, 56
dynamiteformula(), 3, 14, 15, 27, 28, 30, 33–35, 54, 55
fitted.dynamitefit, 22, 52
formula.dynamitefit(dynamite), 14
gaussian_example, 10, 11, 24, 25, 26, 40, 41
gaussian_example_fit, 10, 11, 24, 25, 26, 40, 41
gaussian_simulation_fit, 10, 11, 24, 25, 26, 40, 41
get_code, 6, 8, 9, 13, 14, 18, 27, 29–32, 42, 43
get_data, 6, 8, 9, 13, 14, 18, 28, 28, 30–32, 42, 43
get_parameter_dims, 6, 8, 9, 13, 14, 18, 28, 29, 31, 32, 42, 43
get_parameter_names, 6, 8, 9, 13, 14, 18, 28–30, 31, 32, 42, 43
get_parameter_names(), 9, 29, 43
get_parameter_types, 6, 8, 9, 13, 14, 18, 28–31, 32, 42, 43
get_parameter_types(), 4, 8, 9, 29, 31, 43
get_priors, 18, 32, 57
get_priors(), 15
ggplot2::facet_wrap(), 47, 49
lags, 18, 21, 34, 36, 54, 56
lags(), 21
lfactor, 18, 21, 34, 35, 54, 56
lfo, 36, 38, 39
loo (loo.dynamitefit), 38
loo.dynamitefit, 37, 38, 39
loo::loo(), 38
mcmc_diagnostics, 37, 38, 39
multichannel_example, 10, 11, 24–26, 40, 41
multichannel_example_fit, 10, 11, 24–26, 40, 40
ndraws (ndraws.dynamitefit), 42
ndraws.dynamitefit, 6, 8, 9, 13, 14, 18, 28–32, 42, 43
nobs.dynamitefit, 6, 8, 9, 13, 14, 18, 28–32, 42, 42
obs (dynamiteformula), 18
plot.dynamitefit, 43, 46–49
plot.lfo, 44
plot_betas, 44, 45, 47–49
plot_deltas, 44, 46, 47–49
plot_lambdas, 44, 46, 47, 48, 49
plot_nus, 44, 46, 47, 48, 49
plot_psis, 44, 46–49, 49
posterior::default_convergence_measures(), 39
predict.dynamitefit, 23, 50
predict.dynamitefit(), 22
print.dynamitefit(dynamite), 14
print.dynamiteformula
(dynamiteformula), 18
print.lfo, 53
random_spec, 18, 21, 34, 36, 54, 56
random_spec(), 21
rstan::check_hmc_diagnostics(), 39
rstan::rstan_options(), 16
rstan::sampling(), 16, 17, 36
splines, 18, 21, 34, 36, 54, 55
splines::bs(), 55
stats::model.matrix.lm(), 15, 27, 29, 30, 33
summary.dynamitefit(dynamite), 14
tibble::formatting, 16
update.dynamitefit, 18, 33, 56