Package ‘bayesdfa’

October 12, 2022

Type Package

Title Bayesian Dynamic Factor Analysis (DFA) with 'Stan'

Version 1.2.0

Description Implements Bayesian dynamic factor analysis with 'Stan'. Dynamic factor analysis is a dimension reduction tool for multivariate time series. 'bayesdfa' extends conventional dynamic factor models in several ways. First, extreme events may be estimated in the latent trend by modeling process error with a student-t distribution. Second, alternative constraints (including proportions are allowed). Third, the estimated dynamic factors can be analyzed with hidden Markov models to evaluate support for latent regimes.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.5.0)

Imports dplyr, ggplot2, loo (>= 2.0.0), methods, mgcv (>= 1.8.13), Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), reshape2, rlang, rstan (>= 2.18.1), rstantools (>= 2.1.1), viridisLite

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), StanHeaders (>= 2.18.1)

Suggests testthat, parallel, knitr, rmarkdown

URL https://fate-ewi.github.io/bayesdfa/

BugReports https://github.com/fate-ewi/bayesdfa/issues

RoxygenNote 7.1.1

VignetteBuilder knitr

SystemRequirements GNU make

Biarch true

NeedsCompilation yes
Description

A DESCRIPTION OF THE PACKAGE

References


dfa_cv  
Apply cross validation to DFA model

Description

Apply cross validation to DFA model

Usage

dfa_cv(
  stanfit,
  cv_method = c("loocv", "lfov")
   fold_ids = NULL,
   n_folds = 10,
   iter = 2000,
   chains = 4,
   thin = 1,
   ...)

Arguments

stanfit  A stanfit object, to preserve the model structure from a call to fit_dfa()

cv_method  The method used for cross validation. The options are 'loocv', where time is ignored and each data point is assigned randomly to a fold. The method 'lfov' is leave time out cross validation, and time slices are iteratively held out. Finally the method 'lfocv' implements leave future out cross validation to do one-step ahead predictions.

fold_ids  A vector whose length is the same as the number of total data points. Elements are the fold id of each data point. If not all data points are used (e.g. the lfovcv or lfoov approach might only use 10 time steps) the value can be something other than a number, e.g. NA
### Examples

```r
## Not run:
set.seed(42)
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
obs <- c(s$y_sim[1, ], s$y_sim[2, ], s$y_sim[3, ])
long <- data.frame("obs" = obs, "ts" = sort(rep(1:3, 20)),
                   "time" = rep(1:20, 3))
m <- fit_dfa(y = long, iter = 50, chains = 1, data_shape = "long",
             sample = FALSE)
# random folds
fit_cv <- dfa_cv(m, cv_method = "loocv", n_folds = 5, iter = 50, chains = 1)

# folds can also be passed in
fold_ids <- sample(1:5, size = nrow(long), replace = TRUE)
m <- fit_dfa(y = long, iter = 50, chains = 1, data_shape = "long",
             sample = FALSE)
fit_cv <- dfa_cv(m, cv_method = "loocv", n_folds = 5, iter = 50, chains = 1,
                 fold_ids = fold_ids)

# do an example of leave-time-out cross validation where years are dropped
fold_ids <- long$time
m <- fit_dfa(y = long, iter = 50, chains = 1, data_shape = "long",
             sample = FALSE)
fit_cv <- dfa_cv(m, cv_method = "loocv", iter = 100, chains = 1,
                 fold_ids = fold_ids)

# example with covariates and long format data
obs_covar <- expand.grid("time" = 1:20, "timeseries" = 1:3,
                        "covariate" = 1:2)
obs_covar$value <- rnorm(nrow(obs_covar), 0, 0.1)
obs <- c(s$s$y_sim[1, ], s$s$y_sim[2, ], s$s$y_sim[3, ])
m <- fit_dfa(y = long, iter = 50, chains = 1, obs_covar = obs_covar,
             data_shape = "long", sample = FALSE)
fit_cv <- dfa_cv(m, cv_method = "loocv", n_folds = 5, iter = 50, chains = 1)
## End(Not run)
```

dfa_fitted

Get the fitted values from a DFA as a data frame
**Description**

Get the fitted values from a DFA as a data frame

**Usage**

```r
dfa_fitted(modelfit, conf_level = 0.95, names = NULL)
```

**Arguments**

- `modelfit`: Output from `fit_dfa`.
- `conf_level`: Probability level for CI.
- `names`: Optional vector of names for time series labels. Should be same length as the number of time series.

**Value**

A data frame with the following columns: `ID` is an identifier for each time series, `time` is the time step, `y` is the observed values standardized to mean 0 and unit variance, `estimate` is the mean fitted value, `lower` is the lower CI, and `upper` is the upper CI.

**See Also**

`predicted`, `plot_fitted`, `fit_dfa`

**Examples**

```r
y <- sim_dfa(num_trends = 2, num_years = 20, num_ts = 4)
m <- fit_dfa(y = y$y_sim, num_trends = 2, iter = 50, chains = 1)
fitted <- dfa_fitted(m)
```

---

**Description**

Get the loadings from a DFA as a data frame

**Usage**

```r
dfa_loadings(rotated_modelfit, names = NULL, summary = TRUE, conf_level = 0.95)
```
Arguments

rotated_modelfit
Output from \texttt{rotate_trends}.

names
An optional vector of names for plotting the loadings.

summary
Logical. Should the full posterior densities be returned? Defaults to \texttt{TRUE}.

conf_level
Confidence level for credible intervals. Defaults to 0.95.

Value

A data frame with the following columns: name is an identifier for each loading, trend is the trend for the loading, median is the posterior median loading, lower is the lower CI, upper is the upper CI, and prob_diff0 is the probability the loading is different than 0. When \texttt{summary = FALSE}, there is no lower or upper columns and instead there are columns chain and draw.

See Also

\texttt{plot_loadings fit_dfa rotate_trends}

Examples

```r
set.seed(42)
s <- sim_dfa(num_trends = 2, num_ts = 4, num_years = 10)
# only 1 chain and 180 iterations used so example runs quickly:
m <- fit_dfa(y = s$y_sim, num_trends = 2, iter = 50, chains = 1)
r <- rotate_trends(m)
loadings <- dfa_loadings(r, summary = TRUE)
loadings <- dfa_loadings(r, summary = FALSE)
```

---

\texttt{dfa_trends} \hspace{1cm} \textit{Get the trends from a DFA as a data frame}

Description

Get the trends from a DFA as a data frame

Usage

\texttt{dfa_trends(rotated_modelfit, years = NULL)}

Arguments

rotated_modelfit
Output from \texttt{rotate_trends}.

years
Optional numeric vector of years.
**find_dfa_trends**

**Value**

A data frame with the following columns: `time` is the time step, `trend_number` is an identifier for each trend, `estimate` is the trend mean, `lower` is the lower CI, and `upper` is the upper CI.

**See Also**

plot_trends fit_dfa rotate_trends

**Examples**

```r
set.seed(1)
s <- sim_dfa(num_trends = 1)
m <- fit_dfa(y = s$y_sim, num_trends = 1, iter = 50, chains = 1)
r <- rotate_trends(m)
trends <- dfa_trends(r)
```

---

**Description**

Fit a DFA with different number of trends and return the leave one out (LOO) value as calculated by the `loo` package.

**Usage**

```r
find_dfa_trends(
  y = y,
  kmin = 1,
  kmax = 5,
  iter = 2000,
  thin = 1,
  compare_normal = FALSE,
  convergence_threshold = 1.05,
  variance = c("equal", "unequal"),
  ...
)
```

**Arguments**

- **y**
  A matrix of data to fit. Columns represent time element.
- **kmin**
  Minimum number of trends, defaults to 1.
- **kmax**
  Maximum number of trends, defaults to 5.
- **iter**
  Iterations when sampling from each Stan model, defaults to 2000.
- **thin**
  Thinning rate when sampling from each Stan model, defaults to 1.
- **compare_normal**
  If TRUE, does model selection comparison of Normal vs. Student-t errors
find_inverted_chains

convergence_threshold
   The maximum allowed value of Rhat to determine convergence of parameters

variance
   Vector of variance arguments for searching over large groups of models. Can be
   either or both of ("equal","unequal")

... Other arguments to pass to fit_dfa()

Examples

set.seed(42)
s <- sim_dfa(num_trends = 2, num_years = 20, num_ts = 3)
# only 1 chain and 100 iterations used so example runs quickly:
m <- find_dfa_trends(
   y = s$y_sim, iter = 50,
   kmin = 1, kmax = 2, chains = 1, compare_normal = FALSE,
   variance = "equal", convergence_threshold = 1.1,
   control = list(adapt_delta = 0.95, max_treedepth = 20)
)
m$summary
m$best_model

find_inverted_chains Find which chains to invert

Description

Find which chains to invert by checking the sum of the squared deviations between the first chain
and each other chain.

Usage

find_inverted_chains(model, trend = 1, plot = FALSE)

Arguments

model A Stan model, rstanfit object
trend Which trend to check
plot Logical: should a plot of the trend for each chain be made? Defaults to FALSE

See Also

invert_chains
find_regimes

Examples

set.seed(2)
s <- sim_dfa(num_trends = 2)
set.seed(1)
m <- fit_dfa(y = s$y_sim, num_trends = 1, iter = 30, chains = 2)
# chains were already inverted, but we can redo that, as an example, with:
find_inverted_chains(m$model, plot = TRUE)

find_regimes

Fit multiple models with differing numbers of regimes to trend data

Description

Fit multiple models with differing numbers of regimes to trend data

Usage

find_regimes(
y, sds = NULL, min_regimes = 1, max_regimes = 3, iter = 2000, thin = 1, chains = 1, ...
)

Arguments

y Data, time series or trend from fitted DFA model.
sds Optional time series of standard deviations of estimates. If passed in, residual variance not estimated.
min_regimes Smallest of regimes to evaluate, defaults to 1.
max_regimes Biggest of regimes to evaluate, defaults to 3.
iter MCMC iterations, defaults to 2000.
thin MCMC thinning rate, defaults to 1.
chains MCMC chains; defaults to 1 (note that running multiple chains may result in a "label switching" problem where the regimes are identified with different IDs across chains).
...

Other parameters to pass to rstan::sampling().

Examples

data(Nile)
find_regimes(log(Nile), iter = 50, chains = 1, max_regimes = 2)
find_swans

Find outlying "black swan" jumps in trends

Description

Find outlying "black swan" jumps in trends

Usage

find_swans(rotated_modelfit, threshold = 0.01, plot = FALSE)

Arguments

rotated_modelfit

Output from rotate_trends().

threshold

A probability threshold below which to flag trend events as extreme

plot

Logical: should a plot be made?

Value

Prints a ggplot2 plot if plot = TRUE; returns a data frame indicating the probability that any given point in time represents a "black swan" event invisibly.

References


Examples

set.seed(1)
s <- sim_dfa(num_trends = 1, num_ts = 3, num_years = 30)
s$y_sim[1, 15] <- s$y_sim[1, 15] - 6
plot(s$y_sim[1, ], type = "o")
abline(v = 15, col = "red")
# only 1 chain and 250 iterations used so example runs quickly:
m <- fit_dfa(y = s$y_sim, num_trends = 1, iter = 50, chains = 1, nu_fixed = 2)
r <- rotate_trends(m)
p <- plot_trends(r) #+ geom_vline(xintercept = 15, colour = "red")
print(p)
# a 1 in 1000 probability if was from a normal distribution:
find_swans(r, plot = TRUE, threshold = 0.001)
Description

Fit a Bayesian DFA

Usage

```r
fit_dfa(
  y = y,
  num_trends = 1,
  varIndx = NULL,
  scale = c("zscore", "center", "none"),
  iter = 2000,
  chains = 4,
  thin = 1,
  control = list(adapt_delta = 0.99, max_treedepth = 20),
  nu_fixed = 101,
  est_correlation = FALSE,
  estimate_nu = FALSE,
  estimate_trend_ar = FALSE,
  estimate_trend_ma = FALSE,
  estimate_process_sigma = FALSE,
  equal_process_sigma = TRUE,
  estimation = c("sampling", "optimizing", "vb", "none"),
  data_shape = c("wide", "long"),
  obs_covar = NULL,
  pro_covar = NULL,
  z_bound = NULL,
  z_model = c("dfa", "proportion"),
  trend_model = c("rw", "bs", "ps", "gp"),
  n_knots = NULL,
  knot_locs = NULL,
  par_list = NULL,
  family = "gaussian",
  verbose = FALSE,
  gp_theta_prior = c(3, 1),
  expansion_prior = FALSE,
  ...
)
```

Arguments

- `y`: A matrix of data to fit. See `data_shape` option to specify whether this is long or wide format data. Wide format data (default) is a matrix with time across columns and unique time series across rows, and can only contain 1 observation.
per time series - time combination. In contrast, long format data is a data frame that includes observations ("obs"), time ("time") and time series ("ts") identifiers – the benefit of long format is that multiple observations per time series can be included. Correlation matrix currently not estimated if data shape is long.

- `num_trends`: Number of trends to fit.
- `varIndx`: Indices indicating which timeseries should have shared variances.
- `scale`: Character string, used to standardized data. Can be "zscore" to center and standardize data, "center" to just standardize data, or "none". Defaults to "zscore".
- `iter`: Number of iterations in Stan sampling, defaults to 2000. Used for both `rstan::sampling()` and `rstan::vb()`.
- `chains`: Number of chains in Stan sampling, defaults to 4.
- `thin`: Thinning rate in Stan sampling, defaults to 1.
- `control`: A list of options to pass to Stan sampling. Defaults to `list(adapt_delta = 0.99, max_treedepth = 20)`.
- `nu_fixed`: Student t degrees of freedom parameter. If specified as greater than 100, a normal random walk is used instead of a random walk with a t-distribution. Defaults to 101.
- `est_correlation`: Boolean, whether to estimate correlation of observation error matrix $R$. Defaults to `FALSE`. Currently can’t be estimated if data are in long format.
- `estimate_nu`: Logical. Estimate the student t degrees of freedom parameter? Defaults to `FALSE`.
- `estimate_trend_ar`: Logical. Estimate AR(1) parameters on DFA trends? Defaults to "FALSE", in which case AR(1) parameters are set to 1.
- `estimate_trend_ma`: Logical. Estimate MA(1) parameters on DFA trends? Defaults to "FALSE", in which case MA(1) parameters are set to 0.
- `estimate_process_sigma`: Logical. Defaults FALSE, whether or not to estimate process error sigma. If not estimated, sigma is fixed at 1, like conventional DFAs.
- `equal_process_sigma`: Logical. If process sigma is estimated, whether or not to estimate a single shared value across trends (default) or estimate equal values for each trend.
- `estimation`: Character string. Should the model be sampled using `rstan::sampling()` ("sampling", default), `rstan::optimizing()` ("optimizing"), variational inference `rstan::vb()` ("vb"), or no estimation done ("none"). No estimation may be useful for debugging and simulation.
- `data_shape`: If `wide` (the current default) then the input data should have rows representing the various timeseries and columns representing the values through time. This matches the MARSS input data format. If `long` then the long format data is a data frame that includes observations ("obs"), time ("time") and time series ("ts") identifiers – the benefit of long format is that multiple observations per time series can be included.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obs_cvar</td>
<td>Optional dataframe of data with 4 named columns (&quot;time&quot;,&quot;timeseries&quot;,&quot;covariate&quot;,&quot;value&quot;), representing: (1) time, (2) the time series affected, (3) the covariate number for models with more than one covariate affecting each trend, and (4) the value of the covariate</td>
</tr>
<tr>
<td>pro_cvar</td>
<td>Optional dataframe of data with 4 named columns (&quot;time&quot;,&quot;trend&quot;,&quot;covariate&quot;,&quot;value&quot;), representing: (1) time, (2) the trend affected, (3) the covariate number for models with more than one covariate affecting each trend, and (4) the value of the covariate</td>
</tr>
<tr>
<td>z_bound</td>
<td>Optional hard constraints for estimated factor loadings – really only applies to model with 1 trend. Passed in as a 2-element vector representing the lower and upper bound, e.g. (0, 100) to constrain positive</td>
</tr>
<tr>
<td>z_model</td>
<td>Optional argument allowing for elements of Z to be constrained to be proportions (each time series modeled as a mixture of trends). Arguments can be &quot;dfa&quot; (default) or &quot;proportion&quot;</td>
</tr>
<tr>
<td>trend_model</td>
<td>Optional argument to change the model of the underlying latent trend. By default this is set to 'rw', where the trend is modeled as a random walk - as in conventional DFA. Alternative options are 'bs', where B-splines are used to model the trends, &quot;ps&quot; where P-splines are used to model the trends, or ‘gp’, where gaussian predictive processes are used. If models other than 'rw' are used, there are some key points. First, the MA and AR parameters on these models will be turned off. Second, for B-splines and P-splines, the process_sigma becomes an optional scalar on the spline coefficients, and is turned off by default. Third, the number of knots can be specified (more knots = more wiggliness, and n_knots &lt; N). For models with &gt; 2 trends, each trend has their own spline coefficients estimated though the knot locations are assumed shared. If knots aren’t specified, the default is N/3. By default both the B-spline and P-spline models use 3rd degree functions for smoothing, and include an intercept term. The P-spline model uses a difference penalty of 2.</td>
</tr>
<tr>
<td>n_knots</td>
<td>The number of knots for the B-spline, P-spline, or Gaussian predictive process models. Optional, defaults to round(N/3)</td>
</tr>
<tr>
<td>knot_locs</td>
<td>Locations of knots (optional), defaults to uniform spacing between 1 and N</td>
</tr>
<tr>
<td>par_list</td>
<td>A vector of parameter names of variables to be estimated by Stan. If NULL, this will default to c(&quot;x&quot;, &quot;Z&quot;, &quot;sigma&quot;, &quot;log_lik&quot;, &quot;psi&quot;,&quot;xstar&quot;) for most models – though if AR / MA, or Student-t models are used additional parameters will be monitored. If you want to use diagnostic tools in rstan, including moment_matching, you will need to pass in a larger list. Setting this argument to &quot;all&quot; will monitor all parameters, enabling the use of diagnostic functions – but making the models a lot larger for storage. Finally, this argument may be a custom string of parameters to monitor, e.g. c(&quot;x&quot;,&quot;sigma&quot;)</td>
</tr>
<tr>
<td>family</td>
<td>String describing the observation model. Default is &quot;gaussian&quot;, but included options are &quot;gamma&quot;, &quot;lognormal&quot;, negative binomial (&quot;nbinom2&quot;), &quot;poisson&quot;, or &quot;binomial&quot;. The binomial family is assumed to have logit link, gaussian family is assumed to be identity, and the rest are log-link.</td>
</tr>
<tr>
<td>verbose</td>
<td>Whether to print iterations and information from Stan, defaults to FALSE.</td>
</tr>
</tbody>
</table>
| gp_theta_prior | A 2-element vector controlling the prior on the Gaussian process parameter in cov_exp_quad. This prior is a half-Student t prior, with the first argument of
gp_theta_prior being the degrees of freedom (nu), and the second element being
the standard deviation

expansion_prior

Defaults to FALSE, if TRUE uses the parameter expansion prior of Ghosh &
Dunson 2009

Any other arguments to pass to rstan::sampling().

Details

Note that there is nothing restricting the loadings and trends from being inverted (i.e. multiplied by
-1) for a given chain. Therefore, if you fit multiple chains, the package will attempt to determine
which chains need to be inverted using the function find_inverted_chains().

See Also

plot_loadings plot_trends rotate_trends find_swans

Examples

set.seed(42)
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
# only 1 chain and 250 iterations used so example runs quickly:
m <- fit_dfa(y = s$y_sim, iter = 50, chains = 1)
## Not run:
# example of observation error covariates
set.seed(42)
obs_covar <- expand.grid("time" = 1:20, "timeseries" = 1:3, "covariate" = 1)
obs_covar$value <- rnorm(nrow(obs_covar), 0, 0.1)
m <- fit_dfa(y = s$y_sim, iter = 50, chains = 1, obs_covar = obs_covar)

# example of process error covariates
pro_covar <- expand.grid("time" = 1:20, "trend" = 1:2, "covariate" = 1)
pro_covar$value <- rnorm(nrow(pro_covar), 0, 0.1)
m <- fit_dfa(y = s$y_sim, iter = 50, chains = 1, num_trends = 2, pro_covar = pro_covar)

# example of long format data
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
obs <- c(s$y_sim[1, ], s$y_sim[2, ], s$y_sim[3, ])
long <- data.frame("obs" = obs, "ts" = sort(rep(1:3, 20)), "time" = rep(1:20, 3))
m <- fit_dfa(y = long, data_shape = "long", iter = 50, chains = 1)

# example of long format data with obs covariates
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
obs <- c(s$y_sim[1, ], s$y_sim[2, ], s$y_sim[3, ])
long <- data.frame("obs" = obs, "ts" = sort(rep(1:3, 20)), "time" = rep(1:20, 3))
obs_covar <- expand.grid("time" = 1:20, "timeseries" = 1:3, "covariate" = 1:2)
obs_covar$value <- rnorm(nrow(obs_covar), 0, 0.1)
m <- fit_dfa(y = long, data_shape = "long", iter = 50, chains = 1, obs_covar = obs_covar)

# example of model with Z constrained to be proportions and wide format data
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
m <- fit_dfa(y = s$y_sim, z_model = "proportion", iter = 50, chains = 1)
# example of model with Z constrained to be proportions and long format data
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
obs <- c(s$y_sim[1,], s$y_sim[2,], s$y_sim[3,])
long <- data.frame("obs" = obs, "ts" = sort(rep(1:3, 20)), "time" = rep(1:20, 3))
m <- fit_dfa(y = long, data_shape = "long", z_model = "proportion", iter = 50, chains = 1)

# example of B-spline model with wide format data
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
m <- fit_dfa(y = s$y_sim, iter = 50, chains = 1, trend_model = "bs", n_knots = 10)

# example of P-spline model with wide format data
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
m <- fit_dfa(y = s$y_sim, iter = 50, chains = 1, trend_model = "ps", n_knots = 10)

# example of Gaussian process model with wide format data
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
m <- fit_dfa(y = s$y_sim, iter = 50, chains = 1, trend_model = "gp", n_knots = 5)

## End(Not run)

**fit_regimes**

Fit models with differing numbers of regimes to trend data

### Description

Fit models with differing numbers of regimes to trend data

### Usage

```r
fit_regimes(
  y,
  sds = NULL,
  n_regimes = 2,
  iter = 2000,
  thin = 1,
  chains = 1,
  ...
)
```

### Arguments

- **y**: Data, time series or trend from fitted DFA model.
- **sds**: Optional time series of standard deviations of estimates. If passed in, residual variance not estimated. Defaults to `NULL`.
- **n_regimes**: Number of regimes to evaluate, defaults 2
- **iter**: MCMC iterations, defaults to 2000.
- **thin**: MCMC thinning rate, defaults to 1.
invert_chains

chains        MCMC chains, defaults to 1 (note that running multiple chains may result in a label switching problem where the regimes are identified with different IDs across chains).
...
Other parameters to pass to rstan::sampling().

Examples

data(Nile)
fit_regimes(log(Nile), iter = 50, n_regimes = 1)

hmm_init Create initial values for the HMM model.

Description
Create initial values for the HMM model.

Usage
hmm_init(K, x_t)

Arguments
K            The number of regimes or clusters to fit. Called by rstan::sampling().
x_t           A matrix of values. Called by rstan::sampling().

Value
list of initial values (mu, sigma)

invert_chains Invert chains

Description
Invert chains

Usage
invert_chains(model, trends = 1, print = FALSE, ...)

Arguments
model        A Stan model, rstanfit object
trends        The number of trends in the DFA, defaults to 1
print        Logical indicating whether the summary should be printed. Defaults to FALSE.
...            Other arguments to pass to find_inverted_chains().
**is_converged**

*Summarize Rhat convergence statistics across parameters*

**Description**

Pass in `rstanfit` model object, and a threshold Rhat value for convergence. Returns boolean.

**Usage**

```r
is_converged(fitted_model, threshold = 1.05, parameters = c("sigma", "x", "Z"))
```

**Arguments**

- `fitted_model`: Samples extracted (with `permuted = FALSE`) from a Stan model. E.g. output from `invert_chains()`.
- `threshold`: Threshold for maximum Rhat.
- `parameters`: Vector of parameters to be included in convergence determination. Defaults = c("sigma", "x", "Z"). Other elements can be added including "pred", "log_lik", or "lp__".

**loo.bayesdfa**

*LOO information criteria*

**Description**

Extract the LOOIC (leave-one-out information criterion) using `loo::loo()`. Note that we’ve implemented slightly different variants of loo, based on whether the DFA observation model includes correlation between time series or not (default is no correlation). Importantly, these different versions are not directly comparable to evaluate data support for including correlation or not in a DFA. If time series are not correlated, the point-wise log-likelihood for each observation is calculated and used in the loo calculations. However if time series are correlated, then each time slice is assumed to be a joint observation of all variables, and the point-wise log-likelihood is calculated as the joint likelihood of all variables under the multivariate normal distribution.

**Usage**

```r
## S3 method for class 'bayesdfa'
loo(x, ...)
```

**Arguments**

- `x`: Output from `fit_dfa()`.
- `...`: Arguments for `loo::relative_eff()` and `loo::loo.array()`.
Examples

```
set.seed(1)
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
m <- fit_dfa(y = s$y_sim, iter = 50, chains = 1, num_trends = 1)
loo(m)
```

```
y <- sim_dfa(num_trends = 2, num_years = 20, num_ts = 4)
m <- fit_dfa(y = y$y_sim, num_trends = 2, iter = 50, chains = 1)
p <- plot_fitted(m)
print(p)
```

---

**plot_fitted**  
*Plot the fitted values from a DFA*

### Description

Plot the fitted values from a DFA

### Usage

```
plot_fitted(
  modelfit,  
  conf_level = 0.95,
  names = NULL,  
  spaghetti = FALSE,
  time_labels = NULL
)
```

### Arguments

- **modelfit**: Output from `fit_dfa`, a rstanfit object
- **conf_level**: Probability level for CI.
- **names**: Optional vector of names for plotting labels TODO. Should be same length as the number of time series
- **spaghetti**: Defaults to FALSE, but if TRUE puts all raw time series (grey) and fitted values on a single plot
- **time_labels**: Optional vector of time labels for plotting, same length as number of time steps

### See Also

plot_loadings fit_dfa rotate_trends dfa_fitted

### Examples

```
y <- sim_dfa(num_trends = 2, num_years = 20, num_ts = 4)
m <- fit_dfa(y = y$y_sim, num_trends = 2, iter = 50, chains = 1)
p <- plot_fitted(m)
print(p)
```
p <- plot_fitted(m, spaghetti = TRUE)
print(p)

plot_loadings

Plot the loadings from a DFA

Description

Plot the loadings from a DFA

Usage

plot_loadings(
  rotated_modelfit,
  names = NULL,
  facet = TRUE,
  violin = TRUE,
  conf_level = 0.95,
  threshold = NULL
)

Arguments

rotated_modelfit
  Output from rotate_trends().

names
  An optional vector of names for plotting the loadings.

facet
  Logical. Should there be a separate facet for each trend? Defaults to TRUE.

violin
  Logical. Should the full posterior densities be shown as a violin plot? Defaults to TRUE.

conf_level
  Confidence level for credible intervals. Defaults to 0.95.

threshold
  Numeric (0-1). Optional for plots, if included, only plot loadings who have Pr(<0) or Pr(>0) > threshold. For example threshold = 0.8 would only display estimates where 80% of posterior density was above/below zero. Defaults to NULL (not used).

See Also

plot_trends fit_dfa rotate_trends
Examples

```r
set.seed(42)
s <- sim_dfa(num_trends = 2, num_ts = 4, num_years = 10)
# only 1 chain and 180 iterations used so example runs quickly:
m <- fit_dfa(y = s$y_sim, num_trends = 2, iter = 50, chains = 1)
r <- rotate_trends(m)
plot_loadings(r, violin = FALSE, facet = TRUE)
plot_loadings(r, violin = FALSE, facet = FALSE)
plot_loadings(r, violin = TRUE, facet = FALSE)
plot_loadings(r, violin = TRUE, facet = TRUE)
```

---

plot_regime_model  
Plot the state probabilities from `find_regimes()`

Description

Plot the state probabilities from `find_regimes()`

Usage

```r
plot_regime_model(
  model,
  probs = c(0.05, 0.95),
  type = c("probability", "means"),
  regime_prob_threshold = 0.9,
  plot_prob_indices = NULL,
  flip_regimes = FALSE
)
```

Arguments

- `model`: A model returned by `find_regimes()`.
- `probs`: A numeric vector of quantiles to plot the credible intervals at. Defaults to `c(0.05, 0.95)`.
- `type`: Whether to plot the probabilities (default) or means.
- `regime_prob_threshold`: The probability density that must be above 0.5. Defaults to 0.9 before we classify a regime (only affects "means" plot).
- `plot_prob_indices`: Optional indices of probability plots to plot. Defaults to showing all.
- `flip_regimes`: Optional whether to flip regimes in plots, defaults to FALSE

Details

Note that the original timeseries data (dots) are shown scaled between 0 and 1.
Examples

```r
data(Nile)
m <- fit_regimes(log(Nile), n_regimes = 2, chains = 1, iter = 50)
plot_regime_model(m)
plot_regime_model(m, plot_prob_indices = c(2))
plot_regime_model(m, type = "means")
```

---

**plot_trends**

*Plot the trends from a DFA*

**Description**

Plot the trends from a DFA

**Usage**

```r
plot_trends(
  rotated_modelfit,
  years = NULL,
  highlight_outliers = FALSE,
  threshold = 0.01
)
```

**Arguments**

- `rotated_modelfit`
  - Output from `rotate_trends`
- `years`
  - Optional numeric vector of years for the plot
- `highlight_outliers`
  - Logical. Should trend events that exceed the probability of occurring with a normal distribution as defined by `threshold` be highlighted? Defaults to FALSE
- `threshold`
  - A probability threshold below which to flag trend events as extreme. Defaults to 0.01

**See Also**

dfa_trends plot_loadings fit_dfa rotate_trends

**Examples**

```r
set.seed(1)
s <- sim_dfa(num_trends = 1)
m <- fit_dfa(y = s$y_sim, num_trends = 1, iter = 50, chains = 1)
r <- rotate_trends(m)
p <- plot_trends(r)
print(p)
```
predicted

*Calculate predicted value from DFA object*

**Description**

Pass in rstanfit model object. Returns array of predictions, dimensioned number of MCMC draws x number of MCMC chains x time series length x number of time series

**Usage**

`predicted(fitted_model)`

**Arguments**

- **fitted_model**: Samples extracted (with permuted = FALSE) from a Stan model. E.g. output from `invert_chains()`.

**Examples**

```r
## Not run:
set.seed(42)
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
# only 1 chain and 1000 iterations used so example runs quickly:
m <- fit_dfa(y = s$y_sim, iter = 2000, chains = 3, num_trends = 1)
pred <- predicted(m)
## End(Not run)
```

rotate_trends

*Rotate the trends from a DFA*

**Description**

Rotate the trends from a DFA

**Usage**

`rotate_trends(fitted_model, conf_level = 0.95, invert = FALSE)`

**Arguments**

- **fitted_model**: Output from `fit_dfa()`.
- **conf_level**: Probability level for CI.
- **invert**: Whether to invert the trends and loadings for plotting purposes
Examples

```r
set.seed(42)
s <- sim_dfa(num_trends = 1, num_years = 20, num_ts = 3)
# only 1 chain and 800 iterations used so example runs quickly:
m <- fit_dfa(y = s$y_sim, iter = 50, chains = 1)
r <- rotate_trends(m)
plot_trends(r)
```

---

### sim_dfa

Simulate from a DFA

#### Description

Simulate from a DFA

#### Usage

```r
sim_dfa(
    num_trends = 1,
    num_years = 20,
    num_ts = 4,
    loadings_matrix = matrix(nrow = num_ts, ncol = num_trends, rnorm(num_ts * num_trends, 0, 1)),
    sigma = rlnorm(1, meanlog = log(0.2), 0.1),
    varIndex = rep(1, num_ts),
    extreme_value = NULL,
    extreme_loc = NULL,
    nu_fixed = 100,
    user_supplied_deviations = NULL
)
```

#### Arguments

- **num_trends**: The number of trends.
- **num_years**: The number of years.
- **num_ts**: The number of timeseries.
- **loadings_matrix**: A loadings matrix. The number of rows should match the number of timeseries and the number of columns should match the number of trends. Note that this loadings matrix will be internally manipulated by setting some elements to 0 and constraining some elements to 1 so that the model can be fitted. See `fit_dfa()`.
- **sigma**: A vector of standard deviations on the observation error. Should be of the same length as the number of trends. If not specified, random numbers are used `rlnorm(1, meanlog = log(0.2), 0.1)`. 

---
trend_cor

<table>
<thead>
<tr>
<th>varIndx</th>
<th>Indices of unique observation variances. Defaults to c(1, 1, 1, 1). Unique observation error variances would be specified as c(1, 2, 3, 4) in the case of 4 time series.</th>
</tr>
</thead>
<tbody>
<tr>
<td>extreme_value</td>
<td>Value added to the random walk in the extreme time step. Defaults to not included.</td>
</tr>
<tr>
<td>extreme_loc</td>
<td>Location of single extreme event in the process. The same for all processes, and defaults to round(n_t/2) where n_t is the time series length.</td>
</tr>
<tr>
<td>nu_fixed</td>
<td>Nu is the degrees of freedom parameter for the t-distribution, defaults to 100, which is effectively normal.</td>
</tr>
<tr>
<td>user_supplied_deviations</td>
<td>An optional matrix of deviations for the trend random walks. Columns are for trends and rows are for each time step.</td>
</tr>
</tbody>
</table>

Value

A list with the following elements: y_sim is the simulated data, pred is the true underlying data without observation error added, x is the underlying trends, Z is the manipulated loadings matrix that is fed to the model.

Examples

```r
x <- sim_dfa(num_trends = 2)
names(x)
matplot(t(x$y_sim), type = "l")
matplot(t(x$x), type = "l")

set.seed(42)
x <- sim_dfa(extreme_value = -4, extreme_loc = 10)
matplot(t(x$x), type = "l")
abline(v = 10)
matplot(t(x$pred), type = "l")
abline(v = 10)

set.seed(42)
x <- sim_dfa()
matplot(t(x$x), type = "l")
abline(v = 10)
matplot(t(x$pred), type = "l")
abline(v = 10)
```

trend_cor  

Estimate the correlation between a DFA trend and some other time-series

Description

Fully incorporates the uncertainty from the posterior of the DFA trend
trend_cor

Usage

trend_cor(
  rotated_modelfit, 
  y, 
  trend = 1, 
  time_window = seq_len(length(y)), 
  trend_samples = 100, 
  stan_iter = 300, 
  stan_chains = 1, 
  ... 
)

Arguments

rotated_modelfit
  Output from rotate_trends().

y
  A numeric vector to correlate with the DFA trend. Must be the same length as the DFA trend.

trend
  A number corresponding to which trend to use, defaults to 1.

time_window
  Indices indicating a time window slice to use in the correlation. Defaults to using the entire time window. Can be used to walk through the timeseries and test the cross correlations.

trend_samples
  The number of samples from the trend posterior to use. A model will be run for each trend sample so this value shouldn’t be too large. Defaults to 100.

stan_iter
  The number of samples from the posterior with each Stan model run, defaults to 300.

stan_chains
  The number of chains for each Stan model run, defaults to 1.

... Other arguments to pass to sampling

Details

Uses a $\sigma \sim \text{half}_t(3, 0, 2)$ prior on the residual standard deviation and a $\text{uniform}(-1, 1)$ prior on the correlation coefficient. Fitted as a linear regression of $y \sim x$, where $y$ represents the $y$ argument to trend_cor() and $x$ represents the DFA trend, and both $y$ and $x$ have been scaled by subtracting their means and dividing by their standard deviations. Samples are drawn from the posterior of the trend and repeatedly fed through the Stan regression to come up with a combined posterior of the correlation.

Value

A numeric vector of samples from the correlation coefficient posterior.

Examples

set.seed(1)
s <- sim_dfa(num_trends = 1, num_years = 15)
m <- fit_dfa(y = s$y_sim, num_trends = 1, iter = 50, chains = 1)
```r
r <- rotate_trends(m)
n_years <- ncol(r$trends[, 1, ])
fake_dat <- rnorm(n_years, 0, 1)
correlation <- trend_cor(r, fake_dat, trend_samples = 25)
hist(correlation)
correlation <- trend_cor(r,
  y = fake_dat, time_window = 5:15,
  trend_samples = 25
)
hist(correlation)
```
Index

bayesdfa (bayesdfa-package), 3
bayesdfa-package, 3

dfa_cv, 3
dfa_fitted, 4
dfa_loadings, 5
dfa_trends, 6

find_dfa_trends, 7
find_inverted_chains, 8
find_inverted_chains(), 14, 16
find_regimes, 9
find_regimes(), 20
find_swans, 10
fit_dfa, 5, 11, 18
fit_dfa(), 17, 22, 23
fit_regimes, 15

hmm_init, 16

invert_chains, 16
invert_chains(), 17, 22
is_converged, 17

loo, 7
loo (loo.bayesdfa), 17
loo.bayesdfa, 17
loo::loo(), 17
loo::loo.array(), 17
loo::relative_eff(), 17

plot_fitted, 18
plot_loadings, 19
plot_regime_model, 20
plot_trends, 21
predicted, 22

rotate_trends, 6, 21, 22
rotate_trends(), 10, 19, 25
rstan::optimizing(), 12
rstan::sampling(), 4, 9, 12, 14, 16