Package: fHMM (via r-universe)

October 20, 2026

Type Package

Title Fitting Hidden Markov Models to Financial Data

Version 1.4.3

Description Fitting (hierarchical) hidden Markov models to financial data via maximum likelihood estimation. See Oelschläger, L. and Adam, T. ``Detecting Bearish and Bullish Markets in Financial Time Series Using Hierarchical Hidden Markov Models" (2021, Statistical Modelling) <doi:10.1177/1471082X211034048> for a reference on the method. A user guide is provided by the accompanying software paper ``fHMM: Hidden Markov Models for Financial Time Series in R", Oelschläger, L., Adam, T., and Michels, R. (2024, Journal of Statistical Software) <doi:10.18637/jss.v109.i09>.

Language en-US

URL https://loelschlaeger.de/fHMM/

BugReports https://github.com/loelschlaeger/fHMM/issues

License GPL-3

Encoding UTF-8

Depends R (>= 4.0.0)

Imports checkmate, cli, curl, foreach, graphics, grDevices, httr, jsonlite, MASS, oeli (>= 0.7.6), padr, pracma, progress, Rcpp, stats, utils

LinkingTo Rcpp, RcppArmadillo

Suggests covr, devtools, doSNOW, knitr, parallel, rmarkdown, testthat (>= 3.0.0), tseries

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

LazyDataCompression xz

Config/pak/sysreqs make libicu-dev libssl-dev

Repository https://loelschlaeger.r-universe.dev

Date/Publication 2026-05-05 12:47:15 UTC

RemoteUrl https://github.com/loelschlaeger/fhmm

RemoteRef HEAD

RemoteSha d5ca69a8bd42fc9eea2d3844a0d339e3fd828520

Contents

compare_models . . . 3
compute_residuals . . . 3
dax . . . 4
dax_model_2n . . . 5
dax_model_3t . . . 6
dax_vw_model . . . 7
decode_states . . . 8
download_data . . . 9
fHMM_data . . . 11
fHMM_events . . . 12
fHMM_model . . . 13
fHMM_parameters . . . 15
fit_model . . . 18
ll_hmm . . . 21
plot.fHMM_data . . . 24
plot.fHMM_model . . . 25
prepare_data . . . 26
reorder_states . . . 27
set_controls . . . 28
sim_model_2gamma . . . 33
simulate_hmm . . . 34
spx . . . 37
unemp . . . 37
unemp_spx_model_3_2 . . . 38
vw . . . 39

Index **40**

compare_models *Compare multiple models*

Description

This function performs model comparison by comparing multiple `fHMM_model` objects with respect to

- the number of model parameters,
- the log-likelihood value,
- the AIC value,
- the BIC value.

Usage

```
compare_models(...)
```

Arguments

`...`	A list of one or more objects of class `fHMM_model`.

Value

A `data.frame` with models in rows and comparison criteria in columns.

Examples

```
### 3-state HMM with t-distributions is preferred over 2-state HMM with
### normal distributions for the DAX data based on AIC and BIC
compare_models(dax_model_2n, dax_model_3t)
```

compute_residuals *Compute (pseudo-) residuals*

Description

This function computes (pseudo-) residuals of an `fHMM_model` object.

Usage

```
compute_residuals(x, verbose = TRUE)
```

Arguments

x	[fHMM_model] An object of class fHMM_model.
verbose	[logical(1)] Set to TRUE (default) to print progress messages.

Value

An object of class fHMM_model with residuals included.

Examples

```
compute_residuals(dax_model_3t)
summary(residuals(dax_model_3t))
```

---

dax *Deutscher Aktienindex (DAX) index data*

Description

Deutscher Aktienindex (DAX) index data from 1988 to 2022 from Yahoo Finance.

Usage

```
dax
```

Format

A data.frame with 9012 rows and the following 7 columns:

- Date: The date.
- Open: Opening price.
- High: Highest price.
- Low: Lowest price.
- Close: Close price adjusted for splits.
- Adj.Close: Close price adjusted for dividends and splits.
- Volume: Trade volume.

Details

The data was obtained via:

```
dax <- download_data(
  symbol = "^GDAXI",   # DAX identifier on Yahoo Finance
  from = "1988-01-01", # first observation
  to = "2022-12-31"    # last observation
)
```

dax_model_2n *DAX 2-state HMM with normal distributions*

Description

A pre-computed HMM on closing prices of the DAX from 2000 to 2022 with two hidden states and normal state-dependent distributions for demonstration purposes.

Usage

```
data("dax_model_2n")
```

Format

An object of class `fHMM_model`.

Details

The model was estimated via:

```
controls <- set_controls(
  states = 2,
  sdds   = "normal",
  data   = list(
    file        = dax,
    date_column = "Date",
    data_column = "Close",
    logreturns  = TRUE,
    from        = "2000-01-03",
    to          = "2022-12-31"
  ),
  fit    = list("runs" = 10, "gradtol" = 1e-6, "steptol" = 1e-6)
)
dax_data <- prepare_data(controls)
dax_model_2n <- fit_model(dax_data, seed = 1)
dax_model_2n <- decode_states(dax_model_2n)
dax_model_2n <- compute_residuals(dax_model_2n)
summary(dax_model_2n)
```

dax_model_3t *DAX 3-state HMM with t-distributions*

Description

A pre-computed HMM on closing prices of the DAX from 2000 to 2022 with three hidden states and state-dependent t-distributions for demonstration purposes.

Usage

```
data("dax_model_3t")
```

Format

An object of class `fHMM_model`.

Details

The model was estimated via:

```
controls <- set_controls(
  states = 3,
  sdds   = "t",
  data   = list(
    file        = dax,
    date_column = "Date",
    data_column = "Close",
    logreturns  = TRUE,
    from        = "2000-01-03",
    to          = "2022-12-31"
  ),
  fit    = list(
    runs        = 100,
    iterlim     = 300,
    gradtol     = 1e-6,
    steptol     = 1e-6
  )
)
dax_data <- prepare_data(controls)
dax_model_3t <- fit_model(dax_data, seed = 1, ncluster = 10)
dax_model_3t <- decode_states(dax_model_3t)
dax_model_3t <- compute_residuals(dax_model_3t)
summary(dax_model_3t)
```

dax_vw_model *DAX/VW hierarchical HMM with t-distributions*

Description

A pre-computed HHMM with monthly averaged closing prices of the DAX from 2010 to 2022 on the coarse scale, Volkswagen AG stock data on the fine scale, two hidden fine-scale and coarse-scale states, respectively, and state-dependent t-distributions for demonstration purposes.

Usage

```
data("dax_vw_model")
```

Format

An object of class fHMM_model.

Details

The model was estimated via:

```
controls <- set_controls(
  hierarchy = TRUE,
  states    = c(2, 2),
  sdds      = c("t", "t"),
  period    = "m",
  data      = list(
    file       = list(dax, vw),
    from       = "2010-01-01",
    to         = "2022-12-31",
    logreturns = c(TRUE, TRUE)
  ),
  fit       = list(
    runs       = 200,
    iterlim    = 300,
    gradtol    = 1e-6,
    steptol    = 1e-6
  )
)
dax_vw_data <- prepare_data(controls)
dax_vw_model <- fit_model(dax_vw_data, seed = 1, ncluster = 10)
dax_vw_model <- decode_states(dax_vw_model)
dax_vw_model <- compute_residuals(dax_vw_model)
summary(dax_vw_model)
```

`decode_states` *Decode the underlying hidden state sequence*

Description

This function decodes the (most likely) underlying hidden state sequence by applying the Viterbi algorithm for global decoding.

Usage

```
decode_states(x, verbose = TRUE)

viterbi(observations, nstates, sdd, Gamma, mu, sigma = NULL, df = NULL)
```

Arguments

`x`	[`fHMM_model`] An object of class `fHMM_model`.
`verbose`	[`logical(1)`] Set to `TRUE` to print progress messages.
`observations`	[`numeric()`] A `numeric vector` of state-dependent observations.
`nstates`	[`integer(1)`] The number of states.
`sdd`	[`character(1)`] A `character`, specifying the state-dependent distribution. One of • `"normal"` (the normal distribution), • `"lognormal"` (the log-normal distribution), • `"t"` (the t-distribution), • `"gamma"` (the gamma distribution), • `"poisson"` (the Poisson distribution).
`Gamma`	[`matrix()`] A transition probability `matrix` of dimension `nstates`.
`mu`	[`numeric(nstates)`] A `numeric` vector of expected values for the state-dependent distribution in the different states of length `nstates`. For the gamma- or Poisson-distribution, `mu` must be positive.
`sigma`	[`NULL` | `numeric(nstates)`] A positive `numeric` vector of standard deviations for the state-dependent distribution in the different states of length `nstates`. Not relevant in case of a state-dependent Poisson distribution.
`df`	[`NULL` | `numeric(nstates)`] A positive `numeric` vector of degrees of freedom for the state-dependent distribution in the different states of length `nstates`. Only relevant in case of a state-dependent t-distribution.

Value

An object of class fHMM_model with decoded state sequence included.

References

https://en.wikipedia.org/wiki/Viterbi_algorithm

Examples

```
decode_states(dax_model_3t)
plot(dax_model_3t, type = "ts")
viterbi(
  observations = c(1, 1, 1, 10, 10, 10),
  nstates      = 2,
  sdd          = "poisson",
  Gamma        = matrix(0.5, 2, 2),
  mu           = c(1, 10)
)
```

---

download_data *Download financial data from Yahoo Finance*

Description

This function downloads financial data from https://finance.yahoo.com/ and returns it as a data.frame.

Usage

```
download_data(
  symbol,
  from = "1902-01-01",
  to = Sys.Date(),
  fill_dates = FALSE,
  columns = c("Date", "Open", "High", "Low", "Close", "Adj.Close", "Volume")
)
```

Arguments

symbol	[character(1)] A character, the stock's symbol. It must match the identifier on https://finance.yahoo.com/.
from	[character(1)] A character in the format "YYYY-MM-DD", setting the lower data bound. Must not be earlier than "1902-01-01" (default).

`to` [character(1)]
A character in the format "YYYY-MM-DD", setting the upper data bound.
Default is the current date Sys.date().

`fill_dates` [logical(1)]
Set to TRUE to fill missing dates (e.g., days at which the stock market is closed) with NA's.
By default, fill_dates = FALSE.

`columns` [character()]
A character of requested data columns, see the details.
By default, all columns are returned.

Details

Yahoo Finance provides historical daily data for stocks or indices. The following data columns are available:

- Date: The date.
- Open: Opening price.
- High: Highest price.
- Low: Lowest price.
- Close: Close price adjusted for splits.
- Adj.Close: Close price adjusted for dividends and splits.
- Volume: Trade volume.

Value

A data.frame.

Examples

```
### 21st century DAX closing prices
data <- download_data(
  symbol = "^GDAXI", from = "2000-01-01", columns = c("Date", "Close"),
  fill_dates = TRUE
)
head(data)
```

`fHMM_data` *Constructor of an* `fHMM_data` *object*

Description

This function constructs an object of class `fHMM_data`, which contains the financial data for modeling.

Usage

```
fHMM_data(
  dates,
  time_points,
  markov_chain,
  data,
  time_series,
  T_star,
  controls,
  true_parameters
)

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

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

Arguments

`dates`	[any] The dates in the empirical case.
`time_points`	[any] The time points in the simulated case.
`markov_chain`	[any] The states in the simulated case.
`data`	[any] The data for modeling.
`time_series`	[any] The data before transformation.
`T_star`	[`NULL` | `integer()`] The fine-scale chunk sizes.
`controls`	[`fHMM_controls`] The `fHMM_controls` object.
`true_parameters`	[`NULL` | `fHMM_parameters`] The `fHMM_parameters` object in the simulated case.

x	[fHMM_data] An object of class fHMM_data.
...	Currently not used.
object	[fHMM_data] An object of class fHMM_data.

Value

An object of class fHMM_data, which is a list containing the following elements:

- The matrix of the dates if simulated = FALSE and controls$data$data_column is specified,
- the matrix of the time_points if simulated = TRUE or controls$data$data_column is not specified,
- the matrix of the simulated markov_chain if simulated = TRUE,
- the matrix of the simulated or empirical data used for estimation,
- the matrix time_series of empirical data before the transformation to log-returns if simulated = FALSE,
- the vector of fine-scale chunk sizes T_star if controls$hierarchy = TRUE,
- the input controls,
- the true_parameters.

---

fHMM_events *Checking events*

Description

This function checks the input events.

Usage

```
fHMM_events(events)

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

Arguments

events	[list()] A list of two elements. • The first element is named "dates" and contains a character vector in format "YYYY-MM-DD". • The second element is named "labels" and is a character vector of the same length as "dates".
x	[fHMM_events] An object of class fHMM_events.
...	Currently not used.

Value

An object of class fHMM_events.

Examples

```
events <- list(
  dates = c("2001-09-11", "2008-09-15", "2020-01-27"),
  labels = c(
    "9/11 terrorist attack", "Bankruptcy Lehman Brothers",
    "First COVID-19 case Germany"
  )
)
events <- fHMM_events(events)
```

---

fHMM_model *Constructor of a model object*

Description

This function constructs an object of class fHMM_model, which contains details about the fitted (hierarchical) Hidden Markov model.

Usage

```
fHMM_model(
  data,
  estimate,
  nlm_output,
  estimation_time,
  ll,
  lls,
  gradient,
  inverse_fisher,
  decoding
)

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

## S3 method for class 'fHMM_model'
residuals(object, ...)

## S3 method for class 'fHMM_model'
summary(object, alpha = 0.05, ...)

## S3 method for class 'fHMM_model'
```

```
coef(object, alpha = 0.05, digits = 2, ...)

## S3 method for class 'fHMM_model'
AIC(object, ..., k = 2)

## S3 method for class 'fHMM_model'
BIC(object, ...)

## S3 method for class 'fHMM_model'
nobs(object, ...)

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

npar(object, ...)

## S3 method for class 'fHMM_model'
npar(object, ...)

## S3 method for class 'fHMM_model'
predict(object, ahead = 5, alpha = 0.05, ...)
```

Arguments

`data`	[`fHMM_data`] An object of class `fHMM_data`.
`estimate`	[`numeric()`] A `numeric` vector of unconstrained model estimates.
`nlm_output`	[`list()`] The output of `nlm` for the selected optimization run.
`estimation_time`	[`difftime`] A `diff.time` object, the total estimation time.
`ll`	[`numeric(1)`] A `numeric`, the model log-likelihood.
`lls`	[`numeric()`] A `numeric` vector, the model log-likelihoods in all optimization runs.
`gradient`	[`numeric()`] A `numeric` vector, the gradient at the optimum.
`inverse_fisher`	[`numeric()`] A `numeric` vector, the inverse Fisher information for each parameter.
`decoding`	[`NULL` | `numeric()` | `matrix()`] A `numeric` vector, the decoded time series.
`x`, `object`	[`fHMM_model`] An object of class `fHMM_model`.
`...`	Currently not used.

alpha [numeric(1)]
A numeric between 0 and 1, the confidence level.

digits [integer(1)]
The number of decimal places.

k [numeric(1)]
Passed on to AIC.

ahead [integer(1)]
The number of time points to predict ahead.

Value

An object of class fHMM_model.

---

fHMM_parameters *Set and check model parameters*

Description

This function sets and checks model parameters. Unspecified parameters are sampled.

Usage

```
fHMM_parameters(
  controls = list(),
  hierarchy = FALSE,
  states = if (!hierarchy) 2 else c(2, 2),
  sdds = if (!hierarchy) "normal" else c("normal", "normal"),
  Gamma = NULL,
  mu = NULL,
  sigma = NULL,
  df = NULL,
  Gamma_star = NULL,
  mu_star = NULL,
  sigma_star = NULL,
  df_star = NULL,
  scale_par = c(1, 1),
  seed = NULL,
  check_controls = TRUE
)

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

Arguments

`controls` [list() | fHMM_controls]
Either a list or an object of class fHMM_controls.
The list can contain the following elements, which are described in more detail below:

- hierarchy, defines a hierarchical HMM,
- states, defines the number of states,
- sdds, defines the state-dependent distributions,
- horizon, defines the time horizon,
- period, defines a flexible, periodic fine-scale time horizon,
- data, a list of controls that define the data,
- fit, a list of controls that define the model fitting

Either none, all, or selected elements can be specified.
Unspecified parameters are set to their default values.
Important: Specifications in controls always override individual specifications.

`hierarchy` [logical(1)]
A logical, set to TRUE for a hierarchical HMM.
If hierarchy = TRUE, some of the other controls must be specified for the coarse-scale and the fine-scale layer.
By default, hierarchy = FALSE.

`states` [integer(1) | integer(2)]
An integer, the number of states of the underlying Markov chain.
If hierarchy = TRUE, states must be a vector of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.
By default, states = 2 if hierarchy = FALSE and states = c(2, 2) if hierarchy = TRUE.

`sdds` [character(1) | character(2)]
A character, specifying the state-dependent distribution. One of

- "normal" (the normal distribution),
- "lognormal" (the log-normal distribution),
- "t" (the t-distribution),
- "gamma" (the gamma distribution),
- "poisson" (the Poisson distribution).

The distribution parameters, i.e. the

- mean mu,
- standard deviation sigma (not for the Poisson distribution),
- degrees of freedom df (only for the t-distribution),

can be fixed via, e.g., "t(df = 1)" or "gamma(mu = 0, sigma = 1)". To fix different values of a parameter for different states, separate by "|", e.g. "poisson(mu = 1|2|3)".

If hierarchy = TRUE, sdds must be a vector of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, sdds = "normal" if hierarchy = FALSE and sdds = c("normal", "normal") if hierarchy = TRUE.

Gamma, Gamma_star

[NULL | matrix() | list()]
A transition probability matrix.

It should have dimension states[1].

Gamma_star is a list of fine-scale transition probability matrices. The list must be of length states[1]. Each transition probability matrix must be of dimension states[2].

mu, mu_star

[NULL | numeric() | list()]
A numeric vector of expected values for the state-dependent distribution in the different states.

For the gamma- or Poisson-distribution, mu must be positive.

It should have length states[1].

mu_star is a list of vectors with fine-scale expectations. The list must be of length states[1]. Each vector must be of length states[2].

sigma, sigma_star

[NULL | numeric() | list()]
A positive numeric vector of standard deviations for the state-dependent distribution in the different states.

It should have length states[1].

sigma_star is a list of vectors with fine-scale standard deviations. The list must be of length states[1]. Each vector must be of length states[2].

df, df_star

[NULL | numeric() | list()]
A positive numeric vector of degrees of freedom for the state-dependent distribution in the different states.

It should have length states[1].

Only relevant in case of a state-dependent t-distribution.

df_star is a list of vectors with fine-scale degrees of freedom. The list must be of length states[1]. Each vector must be of length states[2]. Only relevant in case of a fine-scale state-dependent t-distribution.

scale_par

[numeric(2)]
A positive numeric vector of length two, containing scales for sampled expectations and standard deviations.

The first entry is the scale for mu and sigma, the second entry is the scale for mu_star and sigma_star (if any).

seed

[NULL | integer(1)]
Sets a seed for the sampling of parameters.

check_controls

[logical(1)]
Either TRUE to check the defined controls or FALSE to not check them (which saves computation time), else.

x	[fHMM_parameters] An object of class fHMM_parameters.
...	Currently not used.

Details

See the package vignettes at https://loelschlaeger.de/fHMM/articles/ for details.

Value

An object of class fHMM_parameters.

Examples

```
parameters <- fHMM_parameters(states = 2, sdds = "normal")
parameters$Gamma
```

---

fit_model *Model fitting*

Description

This function fits a hidden Markov model via numerical likelihood maximization.

Usage

```
fit_model(
  data,
  controls = data[["controls"]],
  fit = list(),
  runs = 10,
  origin = FALSE,
  accept = 1:3,
  gradtol = 0.01,
  iterlim = 100,
  print.level = 0,
  steptol = 0.01,
  ncluster = 1,
  seed = NULL,
  verbose = TRUE,
  initial_estimate = NULL
)
```

Arguments

`data` [fHMM_data]
An object of class fHMM_data.

`controls` [list() | fHMM_controls]
Either a list or an object of class fHMM_controls.

The list can contain the following elements, which are described in more detail below:

- hierarchy, defines a hierarchical HMM,
- states, defines the number of states,
- sdds, defines the state-dependent distributions,
- horizon, defines the time horizon,
- period, defines a flexible, periodic fine-scale time horizon,
- data, a list of controls that define the data,
- fit, a list of controls that define the model fitting

Either none, all, or selected elements can be specified.

Unspecified parameters are set to their default values.

Important: Specifications in controls always override individual specifications.

`fit` [list()]
A list of controls specifying the model fitting.

The list can contain the following elements, which are described in more detail below:

- runs, defines the number of numerical optimization runs,
- origin, defines initialization at the true parameters,
- accept, defines the set of accepted optimization runs,
- gradtol, defines the gradient tolerance,
- iterlim, defines the iteration limit,
- print.level, defines the level of printing,
- steptol, defines the minimum allowable relative step length.

Either none, all, or selected elements can be specified.

Unspecified parameters are set to their default values, see below.

Specifications in fit override individual specifications.

`runs` [integer(1)]
An integer, setting the number of randomly initialized optimization runs of the model likelihood from which the best one is selected as the final model.

By default, runs = 10.

`origin` [logical(1)]
Only relevant for simulated data, i.e., if the data control is NA.

In this case, a logical. If origin = TRUE the optimization is initialized at the true parameter values. This sets run = 1 and accept = 1:5.

By default, origin = FALSE.

accept [integer() | "all"]
An integer (vector), specifying which optimization runs are accepted based on the output code of nlm.
By default, accept = 1:3.

gradtol [numeric(1)]
A positive numeric value, specifying the gradient tolerance, passed on to nlm.
By default, gradtol = 0.01.

iterlim [integer(1)]
A positive integer value, specifying the iteration limit, passed on to nlm.
By default, iterlim = 100.

print.level [0 | 1 | 2]
One of 0, 1, and 2 to control the verbosity of the numerical likelihood optimization, passed on to nlm.
By default, print.level = 0.

steptol [numeric(1)]
A positive numeric value, specifying the step tolerance, passed on to nlm.
By default, steptol = 0.01.

ncluster [integer(1)]
Set the number of clusters for parallel optimization runs to reduce optimization time. By default, ncluster = 1 (no clustering).

seed [NULL | integer(1)]
Set a seed for the generation of initial values. No seed by default.

verbose [logical(1)]
Set to TRUE to print progress messages.

initial_estimate
[NULL | parUncon]
Optionally defines an initial estimate for the numerical likelihood optimization. Good initial estimates can improve the optimization process. Can be:

- NULL (the default), in this case
  - applies a heuristic to calculate a good initial estimate
  - or uses the true parameter values (if available and data$controls$origin is TRUE)
- or an object of class parUncon (i.e., a numeric of unconstrained model parameters), for example the estimate of a previously fitted model (i.e. the element model$estimate).

Details

Multiple optimization runs starting from different initial values are computed in parallel if ncluster > 1.

Value

An object of class fHMM_model.

Examples

```
### 2-state HMM with normal distributions

# set specifications
controls <- set_controls(
  states = 2, sdds = "normal", horizon = 100, runs = 10
)

# define parameters
parameters <- fHMM_parameters(controls, mu = c(-1, 1), seed = 1)

# sample data
data <- prepare_data(controls, true_parameter = parameters, seed = 1)

# fit model
model <- fit_model(data, seed = 1)

# inspect fit
summary(model)
plot(model, "sdds")

# decode states
model <- decode_states(model)
plot(model, "ts")

# predict
predict(model, ahead = 5)
```

`ll_hmm` *Log-likelihood function of an (H)HMM*

Description

This function computes the log-likelihood value of a (hierarchical) hidden Markov model for given observations and parameter values.

Usage

```
ll_hmm(
  parUncon,
  observations,
  controls = list(),
  hierarchy = FALSE,
  states = if (!hierarchy) 2 else c(2, 2),
  sdds = if (!hierarchy) "normal" else c("normal", "normal"),
  negative = FALSE,
  check_controls = TRUE
)
```

Arguments

parUncon	[parUncon] An object of class parUncon, which is a numeric vector with identified and unconstrained model parameters in the following order: 1. non-diagonal transition probabilities gammasUncon 2. expectations muUncon 3. standard deviations sigmaUncon (if any) 4. degrees of freedom dfUncon (if any) 5. fine-scale parameters for each coarse-scale state, in the same order (if any)
observations	[numeric() | matrix()] A numeric vector of time-series data. In the hierarchical case (hierarchy = TRUE), a matrix with coarse-scale data in the first column and corresponding fine-scale data in the rows.
controls	[list() | fHMM_controls] Either a list or an object of class fHMM_controls. The list can contain the following elements, which are described in more detail below: • hierarchy, defines a hierarchical HMM, • states, defines the number of states, • sdds, defines the state-dependent distributions, • horizon, defines the time horizon, • period, defines a flexible, periodic fine-scale time horizon, • data, a list of controls that define the data, • fit, a list of controls that define the model fitting Either none, all, or selected elements can be specified. Unspecified parameters are set to their default values. Important: Specifications in controls always override individual specifications.
hierarchy	[logical(1)] A logical, set to TRUE for a hierarchical HMM. If hierarchy = TRUE, some of the other controls must be specified for the coarse-scale and the fine-scale layer. By default, hierarchy = FALSE.
states	[integer(1) | integer(2)] An integer, the number of states of the underlying Markov chain. If hierarchy = TRUE, states must be a vector of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer. By default, states = 2 if hierarchy = FALSE and states = c(2, 2) if hierarchy = TRUE.
sdds	[character(1) | character(2)] A character, specifying the state-dependent distribution. One of • "normal" (the normal distribution),

- "lognormal" (the log-normal distribution),
- "t" (the t-distribution),
- "gamma" (the gamma distribution),
- "poisson" (the Poisson distribution).

The distribution parameters, i.e. the

- mean mu,
- standard deviation sigma (not for the Poisson distribution),
- degrees of freedom df (only for the t-distribution),

can be fixed via, e.g., "t(df = 1)" or "gamma(mu = 0, sigma = 1)". To fix different values of a parameter for different states, separate by "|", e.g. "poisson(mu = 1|2|3)".

If hierarchy = TRUE, sdds must be a vector of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, sdds = "normal" if hierarchy = FALSE and sdds = c("normal", "normal") if hierarchy = TRUE.

negative [logical(1)]
Either TRUE to return the negative log-likelihood value (useful for optimization) or FALSE (default), else.

check_controls [logical(1)]
Either TRUE to check the defined controls or FALSE to not check them (which saves computation time), else.

Value

The (negative) log-likelihood value.

Examples

```
### HMM log-likelihood
controls <- set_controls(states = 2, sdds = "normal")
parameters <- fHMM_parameters(controls)
parUncon <- par2parUncon(parameters, controls)
observations <- 1:10
ll_hmm(parUncon, observations, controls)

### HHMM log-likelihood
controls <- set_controls(
  hierarchy = TRUE, states = c(2, 2), sdds = c("normal", "normal")
)
parameters <- fHMM_parameters(controls)
parUncon <- par2parUncon(parameters, controls)
observations <- matrix(dnorm(110), ncol = 11, nrow = 10)
ll_hmm(parUncon, observations, controls)
```

`plot.fHMM_data` *Plot method for an object of class* `fHMM_data`

Description

This function is the plot method for an object of class `fHMM_data`.

Usage

```
## S3 method for class 'fHMM_data'
plot(x, events = NULL, title = NULL, from = NULL, to = NULL, ...)
```

Arguments

`x`	[`fHMM_data`] An object of class `fHMM_data`.
`events`	[`NULL` | `fHMM_events`] An object of class `fHMM_events`.
`title`	[`NULL` | `character(1)`] Optionally a `character` for a custom title.
`from`	[`NULL` | `character(1)`] Optionally a `character`, a date in format `"YYYY-MM-DD"`, setting the lower date bound for plotting. By default, `from = NULL`, i.e. no lower bound.
`to`	[`NULL` | `character(1)`] Optionally a `character`, a date in format `"YYYY-MM-DD"`, setting the upper date bound for plotting. By default, `to = NULL`, i.e. no upper bound.
`...`	Currently not used.

Value

No return value. Draws a plot to the current device.

Examples

```
plot(dax_model_3t$data, title = "DAX time series")
```

`plot.fHMM_model` *Plot method for an object of class* `fHMM_model`

Description

This function is the plot method for an object of class `fHMM_model`.

Usage

```
## S3 method for class 'fHMM_model'
plot(
  x,
  plot_type = "ts",
  events = NULL,
  colors = NULL,
  ll_relative = TRUE,
  title = NULL,
  from = NULL,
  to = NULL,
  ...
)
```

Arguments

`x`	[`fHMM_model`] An object of class `fHMM_model`.
`plot_type`	[`character()`] A character (vector), specifying the type of plot and can be one (or more) of • `"ll"` for a visualization of the likelihood values in the different optimization runs, • `"sdds"` for a visualization of the estimated state-dependent distributions, • `"pr"` for a visualization of the model's (pseudo-) residuals, • `"ts"` for a visualization of the financial time series.
`events`	[`NULL` | `fHMM_events`] An object of class `fHMM_events`.
`colors`	[`NULL` | `character()`] Either `NULL` (default) or a `character` vector of color names or hexadecimal RGB triplets.
`ll_relative`	[`logical(1)`] A `logical`, set to `TRUE` (default) to plot the differences from the best log-likelihood value. Set to `FALSE` to plot the absolute values.
`title`	[`NULL` | `character(1)`] Optionally a `character` for a custom title.

from	[NULL | character(1)] Optionally a character, a date in format "YYYY-MM-DD", setting the lower date bound for plotting. By default, from = NULL, i.e. no lower bound.
to	[NULL | character(1)] Optionally a character, a date in format "YYYY-MM-DD", setting the upper date bound for plotting. By default, to = NULL, i.e. no upper bound.
...	Currently not used.

Value

No return value. Draws a plot to the current device.

`prepare_data` *Prepare data*

Description

This function simulates or reads financial data for the {fHMM} package.

Usage

```
prepare_data(controls, true_parameters = NULL, seed = NULL)
```

Arguments

controls	[fHMM_controls] An object of class fHMM_controls.
true_parameters	[NULL | fHMM_parameters] An object of class fHMM_parameters, used as simulation parameters. By default, true_parameters = NULL, i.e., sampled true parameters.
seed	[NULL | integer(1)] Set a seed for the data simulation. No seed by default.

Value

An object of class fHMM_data.

Examples

```
controls <- set_controls()
data <- prepare_data(controls)
class(data)
summary(data)
```

`reorder_states` *Reorder estimated states*

Description

This function reorders the estimated states, which can be useful for a comparison to true parameters or the interpretation of states.

Usage

```
reorder_states(x, state_order = "mean")
```

Arguments

`x`	[`fHMM_model`] An object of class `fHMM_model`.
`state_order`	[`character(1)` | `numeric()` | `matrix()`] Either • `"mean"`, in which case the states are ordered according to the means of the state-dependent distributions, • or a vector (or a matrix) which determines the new ordering: – If `x$data$controls$hierarchy = FALSE`, `state_order` must be a vector of length `x$data$controls$states` with integer values from 1 to `x$data$controls$states`. If the old state number `x` should be the new state number `y`, put the value `x` at the position `y` of `state_order`. E.g. for a 2-state HMM, specifying `state_order = c(2, 1)` swaps the states. – If `x$data$controls$hierarchy = TRUE`, `state_order` must be a matrix of dimension `x$data$controls$states[1]` x `x$data$controls$states[2] + 1`. The first column orders the coarse-scale states with the logic as described above. For each row, the elements from second to last position order the fine-scale states of the coarse-scale state specified by the first element. E.g. for an HHMM with 2 coarse-scale and 2 fine-scale states, specifying `state_order = matrix(c(2, 1, 2, 1, 1, 2), 2, 3)` swaps the coarse-scale states and the fine-scale states connected to coarse-scale state 2.

Value

An object of class `fHMM_model`, in which states are reordered.

Examples

```
dax_model_3t_reordered <- reorder_states(dax_model_3t, state_order = 3:1)
```

`set_controls` *Define and validate model specifications*

Description

This function defines and validates specifications for model estimation.

Usage

```
set_controls(
  controls = list(),
  hierarchy = FALSE,
  states = if (!hierarchy) 2 else c(2, 2),
  sdds = if (!hierarchy) "normal" else c("normal", "normal"),
  horizon = if (!hierarchy) 100 else c(100, 30),
  period = NA,
  data = NA,
  file = NA,
  date_column = if (!hierarchy) "Date" else c("Date", "Date"),
  data_column = if (!hierarchy) "Close" else c("Close", "Close"),
  from = NA,
  to = NA,
  logreturns = if (!hierarchy) FALSE else c(FALSE, FALSE),
  merge = function(x) mean(x),
  fit = list(),
  runs = 10,
  origin = FALSE,
  accept = 1:3,
  gradtol = 0.01,
  iterlim = 100,
  print.level = 0,
  steptol = 0.01
)

validate_controls(controls)

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

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

Arguments

`controls`	[`list()` | `fHMM_controls`] Either a `list` or an object of class `fHMM_controls`.

The `list` can contain the following elements, which are described in more detail below:

- `hierarchy`, defines a hierarchical HMM,
- `states`, defines the number of states,
- `sdds`, defines the state-dependent distributions,
- `horizon`, defines the time horizon,
- `period`, defines a flexible, periodic fine-scale time horizon,
- `data`, a `list` of controls that define the data,
- `fit`, a `list` of controls that define the model fitting

Either none, all, or selected elements can be specified.

Unspecified parameters are set to their default values.

Important: Specifications in `controls` always override individual specifications.

`hierarchy` [`logical(1)`]
A `logical`, set to `TRUE` for a hierarchical HMM.

If `hierarchy = TRUE`, some of the other controls must be specified for the coarse-scale and the fine-scale layer.

By default, `hierarchy = FALSE`.

`states` [`integer(1) | integer(2)`]
An `integer`, the number of states of the underlying Markov chain.

If `hierarchy = TRUE`, `states` must be a `vector` of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, `states = 2` if `hierarchy = FALSE` and `states = c(2, 2)` if `hierarchy = TRUE`.

`sdds` [`character(1) | character(2)`]
A `character`, specifying the state-dependent distribution. One of

- `"normal"` (the normal distribution),
- `"lognormal"` (the log-normal distribution),
- `"t"` (the t-distribution),
- `"gamma"` (the gamma distribution),
- `"poisson"` (the Poisson distribution).

The distribution parameters, i.e. the

- mean `mu`,
- standard deviation `sigma` (not for the Poisson distribution),
- degrees of freedom `df` (only for the t-distribution),

can be fixed via, e.g., `"t(df = 1)"` or `"gamma(mu = 0, sigma = 1)"`. To fix different values of a parameter for different states, separate by "|", e.g. `"poisson(mu = 1|2|3)"`.

If `hierarchy = TRUE`, `sdds` must be a `vector` of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, `sdds = "normal"` if `hierarchy = FALSE` and `sdds = c("normal", "normal")` if `hierarchy = TRUE`.

horizon [integer(1) | integer(2)]
A numeric, specifying the length of the time horizon.

If hierarchy = TRUE, horizon must be a vector of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, horizon = 100 if hierarchy = FALSE and horizon = c(100, 30) if hierarchy = TRUE.

If data is specified (i.e., not NA), the first entry of horizon is ignored and the (coarse-scale) time horizon is defined by available data.

period [NA | character(1)]
Only relevant if hierarchy = TRUE.

In this case, a character which specifies a flexible, periodic fine-scale time horizon and can be one of

- "w" for a week,
- "m" for a month,
- "q" for a quarter,
- "y" for a year.

By default, period = NA. If period is not NA, it overrules horizon[2].

data [NA | list()]
Either NA, in which case data is simulated (the default), or a list of controls specifying the empirical data set.

The list can contain the following elements, which are described in more detail below:

- file, defines the data set,
- date_column, defines the date column,
- data_column, defines the data column,
- from, defines a lower date limit,
- to, defines an upper date limit,
- logreturns, defines a data transformation to log-returns,
- merge, defines the merging for coarse-scale observations.

Either none, all, or selected elements can be specified.

Unspecified parameters are set to their default values, see below.

Specifications in data override individual specifications.

file [data.frame | character() | list()]
A data.frame with data and dates for modeling.

If hierarchy = TRUE, file can be a list of length 2. The first entry is a data.frame and provides the data for the coarse-scale layer, while the second entry corresponds to the fine-scale layer. If file is a single data.frame, then the same data.frame is used for both layers.

Alternatively, it can be a character (of length two), the path to a .csv-file with financial data.

date_column [character(1) | character(2)]
A character, the name of the column in file with dates.

If `hierarchy = TRUE` and `file` is a `list` of two `data.frames`, `data_column` must be a `vector` of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, `date_column = "Date"`.

`data_column` [`character(1) | character(2)`]
A `character`, the name of the column in `file` with observations.

If `hierarchy = TRUE`, `data_column` must be a `vector` of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, `data_column = "Close"` if `hierarchy = FALSE` and `data_column = c("Close", "Close")` if `hierarchy = TRUE`.

`from` [`NA | character(1)`]
A `character` of the format `"YYYY-MM-DD"`, setting a lower date limit. No lower limit if `from = NA` (default).

`to` [`NA | character(1)`]
A `character` of the format `"YYYY-MM-DD"`, setting an upper date limit. No lower limit if `to = NA` (default).

`logreturns` [`logical(1) | logical(2)`]
A `logical`, if `TRUE` the data is transformed to log-returns.

If `hierarchy = TRUE`, `logreturns` must be a `vector` of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, `logreturns = FALSE` if `hierarchy = FALSE` and `logreturns = c(FALSE, FALSE)` if `hierarchy = TRUE`.

`merge` [`function`]
Only relevant if `hierarchy = TRUE`.

In this case, a `function` which merges an input numeric vector of fine-scale data `x` into one coarse-scale observation. For example,

- `merge = function(x) mean(x)` (default) defines the mean of the fine-scale data as the coarse-scale observation,
- `merge = function(x) mean(abs(x))` for the mean of the absolute values,
- `merge = function(x) sum(abs(x))` for the sum of the absolute values,
- `merge = function(x) (tail(x, 1) - head(x, 1)) / head(x, 1)` for the relative change of the first to the last fine-scale observation.

`fit` [`list()`]
A `list` of controls specifying the model fitting.

The `list` can contain the following elements, which are described in more detail below:

- `runs`, defines the number of numerical optimization runs,
- `origin`, defines initialization at the true parameters,
- `accept`, defines the set of accepted optimization runs,
- `gradtol`, defines the gradient tolerance,
- `iterlim`, defines the iteration limit,
- `print.level`, defines the level of printing,

- `steptol`, defines the minimum allowable relative step length.

Either none, all, or selected elements can be specified.

Unspecified parameters are set to their default values, see below.

Specifications in `fit` override individual specifications.

`runs` [`integer(1)`]
An `integer`, setting the number of randomly initialized optimization runs of the model likelihood from which the best one is selected as the final model.

By default, `runs = 10`.

`origin` [`logical(1)`]
Only relevant for simulated data, i.e., if the `data` control is `NA`.

In this case, a `logical`. If `origin = TRUE` the optimization is initialized at the true parameter values. This sets `run = 1` and `accept = 1:5`.

By default, `origin = FALSE`.

`accept` [`integer() | "all"`]
An `integer` (vector), specifying which optimization runs are accepted based on the output code of `nlm`.

By default, `accept = 1:3`.

`gradtol` [`numeric(1)`]
A positive `numeric` value, specifying the gradient tolerance, passed on to `nlm`.

By default, `gradtol = 0.01`.

`iterlim` [`integer(1)`]
A positive `integer` value, specifying the iteration limit, passed on to `nlm`.

By default, `iterlim = 100`.

`print.level` [`0 | 1 | 2`]
One of `0`, `1`, and `2` to control the verbosity of the numerical likelihood optimization, passed on to `nlm`.

By default, `print.level = 0`.

`steptol` [`numeric(1)`]
A positive `numeric` value, specifying the step tolerance, passed on to `nlm`.

By default, `steptol = 0.01`.

`x, object` [`fHMM_controls`]
An object of class `fHMM_controls`.

`...` Currently not used.

Details

See the vignette on controls for more details.

Value

An object of class `fHMM_controls`, which is a `list` that contains model and estimation specifications.

Examples

```
# 2-state HMM with t-distributions for simulated data
set_controls(
  states = 2,   # the number of states
  sdds   = "t", # the state-dependent distribution
  runs   = 50   # the number of optimization runs
)

# 3-state HMM with normal distributions for the DAX closing prices
set_controls(
  states      = 3,
  sdds        = "normal",
  file        = download_data("^GDAXI"), # the data set
  date_column = "Date",                  # the column with the dates
  data_column = "Close"                  # the column with the data
)

# hierarchical HMM with Gamma and Poisson state distributions
set_controls(
  hierarchy = TRUE,                 # defines a hierarchy
  states    = c(3, 2),              # coarse scale and fine scale states
  sdds      = c("gamma", "poisson"), # distributions for both layers
  horizon   = c(100, NA),           # 100 coarse-scale data points
  period    = "m"                   # monthly simulated fine-scale data
)

# hierarchical HMM with data from .csv-file
set_controls(
  hierarchy = TRUE,
  states    = c(3, 2),
  sdds      = c("t", "t"),
  file      = c(
    system.file("extdata", "dax.csv", package = "fHMM"),
    system.file("extdata", "dax.csv", package = "fHMM")
  ),
  date_column = c("Date", "Date"),
  data_column = c("Close", "Close"),
  logreturns  = c(TRUE, TRUE)
)
```

`sim_model_2gamma` *Simulated 2-state HMM with gamma distributions*

Description

A pre-computed 2-state HMM with state-dependent gamma distributions with means fixed to `0.5` and `2` on `500` simulated observations.

Usage

```
data("sim_model_2gamma")
```

Format

An object of class fHMM_model.

Details

The model was estimated via:

```
controls <- set_controls(
  states  = 2,
  sdds    = "gamma(mu = 1|2)",
  horizon = 200,
  runs    = 10
)
pars <- fHMM_parameters(
  controls = controls,
  Gamma = matrix(c(0.9, 0.2, 0.1, 0.8), nrow = 2),
  sigma = c(0.5, 1),
  seed = 1
)
data_sim <- prepare_data(controls, true_parameters = pars, seed = 1)
sim_model_2gamma <- fit_model(data_sim, seed = 1)
sim_model_2gamma <- decode_states(sim_model_2gamma)
sim_model_2gamma <- compute_residuals(sim_model_2gamma)
summary(sim_model_2gamma)
```

---

simulate_hmm *Simulate data*

Description

This helper function simulates HMM data.

Usage

```
simulate_hmm(
  controls = list(),
  hierarchy = FALSE,
  states = if (!hierarchy) 2 else c(2, 2),
  sdds = if (!hierarchy) "normal" else c("normal", "normal"),
  horizon = if (!hierarchy) 100 else c(100, 30),
  period = NA,
 true_parameters = fHMM_parameters(controls = controls, hierarchy = hierarchy, states =
    states, sdds = sdds),
  seed = NULL
)
```

Arguments

`controls` [list() | fHMM_controls]
Either a list or an object of class fHMM_controls.
The list can contain the following elements, which are described in more detail below:

- hierarchy, defines a hierarchical HMM,
- states, defines the number of states,
- sdds, defines the state-dependent distributions,
- horizon, defines the time horizon,
- period, defines a flexible, periodic fine-scale time horizon,
- data, a list of controls that define the data,
- fit, a list of controls that define the model fitting

Either none, all, or selected elements can be specified.
Unspecified parameters are set to their default values.
Important: Specifications in controls always override individual specifications.

`hierarchy` [logical(1)]
A logical, set to TRUE for a hierarchical HMM.
If hierarchy = TRUE, some of the other controls must be specified for the coarse-scale and the fine-scale layer.
By default, hierarchy = FALSE.

`states` [integer(1) | integer(2)]
An integer, the number of states of the underlying Markov chain.
If hierarchy = TRUE, states must be a vector of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.
By default, states = 2 if hierarchy = FALSE and states = c(2, 2) if hierarchy = TRUE.

`sdds` [character(1) | character(2)]
A character, specifying the state-dependent distribution. One of

- "normal" (the normal distribution),
- "lognormal" (the log-normal distribution),
- "t" (the t-distribution),
- "gamma" (the gamma distribution),
- "poisson" (the Poisson distribution).

The distribution parameters, i.e. the

- mean mu,
- standard deviation sigma (not for the Poisson distribution),
- degrees of freedom df (only for the t-distribution),

can be fixed via, e.g., "t(df = 1)" or "gamma(mu = 0, sigma = 1)". To fix different values of a parameter for different states, separate by "|", e.g. "poisson(mu = 1|2|3)".

If `hierarchy = TRUE`, `sdds` must be a `vector` of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, `sdds = "normal"` if `hierarchy = FALSE` and `sdds = c("normal", "normal")` if `hierarchy = TRUE`.

`horizon` [`integer(1)` | `integer(2)`]
A `numeric`, specifying the length of the time horizon.

If `hierarchy = TRUE`, `horizon` must be a `vector` of length 2. The first entry corresponds to the coarse-scale layer, while the second entry corresponds to the fine-scale layer.

By default, `horizon = 100` if `hierarchy = FALSE` and `horizon = c(100, 30)` if `hierarchy = TRUE`.

If `data` is specified (i.e., not `NA`), the first entry of `horizon` is ignored and the (coarse-scale) time horizon is defined by available data.

`period` [`NA` | `character(1)`]
Only relevant if `hierarchy = TRUE`.

In this case, a `character` which specifies a flexible, periodic fine-scale time horizon and can be one of

- `"w"` for a week,
- `"m"` for a month,
- `"q"` for a quarter,
- `"y"` for a year.

By default, `period = NA`. If `period` is not `NA`, it overrules `horizon[2]`.

`true_parameters` [`NULL` | `fHMM_parameters`]
An object of class `fHMM_parameters`, used as simulation parameters. By default, `true_parameters = NULL`, i.e., sampled true parameters.

`seed` [`NULL` | `integer(1)`]
Set a seed for the data simulation. No seed by default.

Value

A `list` containing the following elements:

- `time_points`, the `vector` (or `matrix` in the hierarchical case) of time points,
- `markov_chain`, the `vector` (or `matrix` in the hierarchical case) of the simulated states,
- `data`, the `vector` (or `matrix` in the hierarchical case) of the simulated state-dependent observations,
- `T_star`, the `numeric` vector of fine-scale chunk sizes in the hierarchical case

Examples

```
simulate_hmm(states = 2, sdds = "normal", horizon = 10)
```

spx *Standard & Poor's 500 (S&P 500) index data*

Description

Standard & Poor's 500 (S&P 500) index data from 1928 to 2022 from Yahoo Finance.

Usage

```
spx
```

Format

A `data.frame` with 23864 rows and the following 7 columns:

- `Date`: The date.
- `Open`: Opening price.
- `High`: Highest price.
- `Low`: Lowest price.
- `Close`: Close price adjusted for splits.
- `Adj.Close`: Close price adjusted for dividends and splits.
- `Volume`: Trade volume.

Details

The data was obtained via:

```
spx <- download_data(
  symbol = "^GSPC",    # S&P 500 identifier on Yahoo Finance
  from = "1928-01-01", # first observation
  to = "2022-12-31"    # last observation
)
```

unemp *Unemployment rate data USA*

Description

The monthly unemployment rate in the USA from 1955 to 2022 on a daily observation basis.

Usage

```
unemp
```

Format

A `data.frame` with 24806 rows and the following 3 columns:

- `date`: The date.
- `rate`: The unemployment rate.
- `rate_diff`: The difference rate to previous month.

Source

OECD (2023), Unemployment rate (indicator). doi: 10.1787/52570002-en (Accessed on 18 January 2023) https://www.oecd.org/en/data/indicators/unemployment-rate.html

---

unemp_spx_model_3_2 *Unemployment rate and S&P 500 hierarchical HMM*

Description

A pre-computed HHMM with monthly unemployment rate in the US on the coarse scale using 3 states and S&P 500 index data on the fine scale using 2 states from 1970 to 2020 for demonstration purposes.

Usage

```
data("unemp_spx_model_3_2")
```

Format

An object of class `fHMM_model`.

Details

The model was estimated via:

```
controls <- list(
  hierarchy = TRUE, states = c(3, 2),
  sdds = c("t", "t"), period = "m",
  data = list(
    file = list(unemp, spx),
    data_column = c("rate_diff", "Close"),
    date_column = c("date", "Date"),
    from = "1970-01-01", to = "2020-01-01",
    logreturns = c(FALSE, TRUE)
  ),
  fit = list(runs = 50, iterlim = 1000, gradtol = 1e-6, steptol = 1e-6)
)
controls <- set_controls(controls)
unemp_spx_data <- prepare_data(controls)
```

```
unemp_spx_model_3_2 <- fit_model(unemp_spx_data, seed = 1, ncluster = 10)
unemp_spx_model_3_2 <- decode_states(unemp_spx_model_3_2)
unemp_spx_model_3_2 <- compute_residuals(unemp_spx_model_3_2)
summary(unemp_spx_model_3_2)
state_order <- matrix(c(3, 2, 1, 2, 2, 2, 1, 1, 1), 3, 3)
unemp_spx_model_3_2 <- reorder_states(unemp_spx_model_3_2, state_order)
```

---

vw *Volkswagen AG (VW) stock data*

Description

Volkswagen AG (VW) stock data from 1998 to 2022 from Yahoo Finance.

Usage

```
vw
```

Format

A `data.frame` with 6260 rows and the following 7 columns:

- `Date`: The date.
- `Open`: Opening price.
- `High`: Highest price.
- `Low`: Lowest price.
- `Close`: Close price adjusted for splits.
- `Adj.Close`: Close price adjusted for dividends and splits.
- `Volume`: Trade volume.

Details

The data was obtained via:

```
vw <- download_data(
  symbol = "VOW3.DE",  # Volkswagen AG identifier on Yahoo Finance
  from = "1988-07-22", # first observation
  to = "2022-12-31"    # last observation
)
```

Index

∗ **data**
  dax, 4
  spx, 37
  unemp, 37
  vw, 39
∗ **model**
  dax_model_2n, 5
  dax_model_3t, 6
  dax_vw_model, 7
  sim_model_2gamma, 33
  unemp_spx_model_3_2, 38

AIC, *15*
AIC.fHMM_model *(*fHMM_model*)*, 13

BIC.fHMM_model *(*fHMM_model*)*, 13

coef.fHMM_model *(*fHMM_model*)*, 13
compare_models, 3
compute_residuals, 3

dax, 4
dax_model_2n, 5
dax_model_3t, 6
dax_vw_model, 7
decode_states, 8
download_data, 9

fHMM_data, 11, *14*, *19*, *26*
fHMM_events, 12, *24, 25*
fHMM_model, *3–9*, 13, *13–15*, *20*, *25*, *27*, *34*, *38*
fHMM_parameters, 15
fit_model, 18

ll_hmm, 21
logLik.fHMM_model *(*fHMM_model*)*, 13

nlm, *14*, *20*, *32*
nobs.fHMM_model *(*fHMM_model*)*, 13
npar *(*fHMM_model*)*, 13

plot.fHMM_data, 24
plot.fHMM_model, 25
predict.fHMM_model *(*fHMM_model*)*, 13
prepare_data, 26
print.fHMM_controls *(*set_controls*)*, 28
print.fHMM_data *(*fHMM_data*)*, 11
print.fHMM_events *(*fHMM_events*)*, 12
print.fHMM_model *(*fHMM_model*)*, 13
print.fHMM_parameters
        *(*fHMM_parameters*)*, 15

reorder_states, 27
residuals.fHMM_model *(*fHMM_model*)*, 13

set_controls, 28
sim_model_2gamma, 33
simulate_hmm, 34
spx, 37
summary.fHMM_controls *(*set_controls*)*, 28
summary.fHMM_data *(*fHMM_data*)*, 11
summary.fHMM_model *(*fHMM_model*)*, 13

unemp, 37
unemp_spx_model_3_2, 38

validate_controls *(*set_controls*)*, 28
viterbi *(*decode_states*)*, 8
vw, 39