Package: farfima (via r-universe)

November 10, 2026

Type Package

Title Order Selection and Forecasting in Long-Memory Functional Time Series

Version 0.1.0

Description Provides tools for modelling, forecasting and visualising long-memory functional time series with the functional autoregressive fractionally integrated moving-average model. The model is fitted in three steps, each of which can also be used on its own. First, the memory parameter is estimated from the leading principal component scores, for example by local Whittle or by bias-corrected local polynomial Whittle with noise. Second, the autoregressive and moving-average orders of the fractionally differenced curves are selected by a functional information criterion, functional Box-Jenkins or cross-validation. Third, the operators are estimated. The point forecasts come with conformal prediction bands that quantify their uncertainty. Visualisation tools include improved rainbow plots and plots of the fitted model and its forecasts.

License GPL-2

Encoding UTF-8

Depends R (>= 4.1.0)

Imports FCVAR, far, ftsa, longmemo, rainbow, sde, stats, utils, graphics, grDevices, vars, withr, wwntests

Suggests demography, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.2

NeedsCompilation no

Author Chang Liu [aut, cre], Han Lin Shang [aut], Garth Tarr [aut], Samuel Müller [aut]

Maintainer Chang Liu <chang.liu45@students.mq.edu.au>

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

Date/Publication 2026-11-10 17:30:02 UTC

RemoteUrl https://github.com/cran/farfima

RemoteRef HEAD

RemoteSha b943781476b7191401ef0f319832e0010e9251f2

Contents

farfima-package 2
as.data.frame.farfima_forecast 3
estimate_memory 4
estimate_operators 6
farfima 8
fitted.farfima 11
frac_diff 12
plot.farfima 13
plot.farfima_forecast 14
plot.farfima_order 15
plot_curves 18
predict.farfima 19
sim_farfima 21
split_curves 22

Index **23**

`farfima-package` *farfima: Order Selection and Forecasting in Long-Memory Functional Time Series*

Description

Fits FARFIMA models to long-memory functional time series, selects their orders and forecasts the curves with conformal prediction bands. Curves are stored as a matrix with grid points in rows and time in columns. `fts` objects from rainbow and ftsa also work.

Workflow

1. `split_curves()` splits the curves into training, validation and test samples, and `plot_curves()` draws them.
2. `farfima()` fits the model. It estimates the memory parameter with `estimate_memory()`, selects the FARMA orders of the fractionally differenced curves as in `select_order()`, estimates their operators as in `estimate_operators()`, and calibrates the conformal prediction bands on the validation sample. `fitted()` and `residuals()` give the in-sample fit, and `plot()` draws it.
3. `predict()` forecasts from the fit, with conformal prediction bands, and evaluates the forecasts on a test sample.
4. `plot()` visualises the forecasts, and `as.data.frame()` puts them in a table.

Other functions

- `estimate_memory()`, `select_order()` and `estimate_operators()`: the steps of `farfima()`, which can also be run on their own. `select_order()` and `estimate_operators()` take short-memory curves, such as fractionally differenced curves.
- `frac_diff()`: fractional differencing.
- `sim_farfima()`: simulated long-memory functional time series.

Author(s)

Chang Liu, Han Lin Shang, Garth Tarr and Samuel Müller. Maintainer: Chang Liu <chang.liu45@students.mq.edu.au>

Examples

```
sim = sim_farfima(n = 200, d = 0.3, p = 1, q = 1, no_grid = 51, seed = 1)
blocks = split_curves(sim, train = 0.6, valid = 0.2)
fit = farfima(blocks$train, blocks$valid, d = "lpwn2", order = "ic_full",
              p = 0:2, q = 0:2, h = 5, trim = 30)
fc = predict(fit, test = blocks$test, band = "AC")
fc
plot(fc, h = 1)
plot(fc, h = 3, target_test_curve = 10)
```

---

as.data.frame.farfima_forecast

Forecasts and conformal prediction bands as a data frame

Description

One row per grid point, horizon, band and level, for saving the results or for your own plots and summaries.

Usage

```
## S3 method for class 'farfima_forecast'
as.data.frame(x, row.names = NULL, optional = FALSE, test = FALSE, ...)
```

Arguments

`x`	A forecast from `predict()`.
`row.names, optional`	Not used; they come from `as.data.frame()`.
`test`	`FALSE` for the forecasts after the last validation or test curve; `TRUE` for the forecasts of the test curves, with the observed values.
`...`	Not used.

Value

A data frame with one row per grid point, horizon, band and level:

- `band`, `level`, `h`: the band type, its nominal level and the horizon;
- `target`: the time being forecast;
- `grid`: the grid point;
- `point_forecast`, `lower`, `upper`: the point forecast and the edges of the conformal prediction band at that grid point.

With `test = TRUE`, the rows cover the forecasts of the test curves, with three more columns:

- `origin`: the time the forecast was made from;
- `target_test_curve`: the position of the target in the test sample;
- `observed`: the observed value at that grid point.

See Also

`predict()`

Examples

```
sim = sim_farfima(n = 200, d = 0.3, p = 1, q = 0, no_grid = 51, seed = 1)
blocks = split_curves(sim)
fit = farfima(blocks$train, blocks$valid, d = 0.3, order = c(1, 0), h = 3,
              trim = 50, verbose = FALSE)
fc = predict(fit, test = blocks$test, verbose = FALSE)
head(as.data.frame(fc))
d = as.data.frame(fc, test = TRUE)
d = subset(d, band == "AC" & level == 0.8)
tapply(d$observed >= d$lower & d$observed <= d$upper, d$h, mean)
```

`estimate_memory` *Estimate the memory parameter*

Description

Estimates the memory parameter d from the leading score process, the centred curves projected onto the leading eigenfunction of a covariance-type operator. If the estimate is near the upper end of its search interval, the score process is integer differenced and d is estimated again, until the estimate falls below the upper end. The number of differences is then added back to the estimate.

Usage

```
estimate_memory(
  x,
  method = c("bc_lpwn2", "bc_lpwn1", "lpwn2", "lpwn1", "lw", "dfa"),
  cov_type = c("sample", "lrc_RS", "lrc_LRS"),
  m_const = 0.75,
  B = 100,
  burn_in = 200,
  R_noise = 0,
  tuning_const = 1,
  cov_band_const = 1,
  dfa_levels = 50,
  dfa_minnpts = 3,
  dfa_cutoff = 10^c(0.7, 2.5),
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`x`	A matrix of curves (grid points by time).
`method`	Estimator: `"lw"` (local Whittle), `"dfa"` (detrended fluctuation analysis), `"lpwn1"` or `"lpwn2"` (local polynomial Whittle with noise, with one or two polynomial correction terms), or `"bc_lpwn1"` or `"bc_lpwn2"` (BC-LPWN1 or BC-LPWN2, the LPWN estimate bias-corrected by the prefiltered sieve bootstrap).
`cov_type`	Operator of the FPCA that gives the leading eigenfunction: `"sample"` (empirical covariance operator), `"lrc_RS"` (long-run covariance operator with an adaptive bandwidth) or `"lrc_LRS"` (long-run covariance operator in sandwich form).
`m_const`	Bandwidth exponent a, so that $\lfloor 1 + T^a \rfloor$ frequencies are used.
`B`, `burn_in`	Number of bootstrap samples and burn-in length (BC-LPWN).
`R_noise`	Number of polynomial terms for the noise in LPWN.
`tuning_const`	Constant c. The score process is differenced while the estimate is at least $1 - c/\log m$.
`cov_band_const`	Bandwidth constant for `cov_type = "lrc_LRS"`.
`dfa_levels`, `dfa_minnpts`, `dfa_cutoff`	Number of block sizes, minimum number of points per block and range of block sizes in the DFA regression.
`seed`	Optional seed for the bootstrap.
`verbose`	Whether to show progress.

Details

The local Whittle and LPWN code is adapted from the LongMemoryTS package (Leschinski, Voges and Wenger, 2019), and the DFA code from the fArma package (Wuertz, Setz and Chalabi, 2017).

Value

An object of class `farfima_memory` with the estimate `d` and the leading score process `scores`. BC-LPWN also returns the uncorrected estimate `d_initial`, the `bias` and the `bootstrap` estimates.

References

Robinson, P. M. (1995). Gaussian semiparametric estimation of long range dependence. *The Annals of Statistics*.

Peng, C.-K., Buldyrev, S. V., Havlin, S., Simons, M., Stanley, H. E. and Goldberger, A. L. (1994). Mosaic organization of DNA nucleotides. *Physical Review E*.

Frederiksen, P., Nielsen, F. S. and Nielsen, M. O. (2012). Local polynomial Whittle estimation of perturbed fractional processes. *Journal of Econometrics*.

Paparoditis, E. (2018). Sieve bootstrap for functional time series. *The Annals of Statistics*.

Li, D., Robinson, P. M. and Shang, H. L. (2020). Long-range dependent curve time series. *Journal of the American Statistical Association*.

Rice, G. and Shang, H. L. (2017). A plug-in bandwidth selection procedure for long-run covariance estimation with stationary functional time series. *Journal of Time Series Analysis*.

Liu, C. and Shang, H. L. (2026). Bias correction of long-memory estimator of functional time series via the prefiltered sieve bootstrap. arXiv:2607.28177.

Leschinski, C., Voges, M. and Wenger, K. (2019). LongMemoryTS: Long Memory Time Series. R package version 0.1.0.

Wuertz, D., Setz, T. and Chalabi, Y. (2017). fArma: Rmetrics - Modelling ARMA Time Series Processes. R package version 3042.81.

See Also

`frac_diff()` to difference the curves with the estimate, and `farfima()` for the whole fit.

Examples

```
x = sim_farfima(n = 300, d = 0.3, p = 1, q = 0, seed = 1)
estimate_memory(x, method = "lw")
estimate_memory(x, method = "lpwn2")
```

---

estimate_operators *Estimate the FARMA operators*

Description

Estimates the operators of a FARMA(p, q) model by the functional Hannan-Rissanen method of Kuenzer (2024). The curves are assumed to be short-memory, so long-memory curves need to be fractionally differenced first, for example with `frac_diff()`. `farfima()` does this on the centred training curves and drops the first `trim` before estimating the operators.

Usage

```
estimate_operators(x, order, Npc = 1)

## S3 method for class 'farfima_operators'
plot(
  x,
  argvals = NULL,
  xlab = "Grid point v",
  ylab = "Grid point u",
  main = NULL,
  ...
)
```

Arguments

`x`	A matrix of short-memory curves (grid points by time) or an `fts` object. For `plot()`, a `farfima_operators` object.
`order`	FARMA orders `c(p, q)`, or a result of `select_order()`.
`Npc`	Number of principal components in the dimension reduction of the Hannan-Rissanen method.
`argvals`	Grid values. Defaults to the row names of the curves if numeric.
`xlab`, `ylab`, `main`	Axis labels and title.
`...`	Other arguments passed to `graphics::image()`.

Details

Each operator is stored as a grid by grid matrix, the kernel times the grid spacing, so that applying the operator to a curve is a matrix product. `plot()` draws the operators as heatmaps, red for positive and blue for negative values.

The estimation code is adapted from code by Thomas Kuenzer.

Value

An object of class `farfima_operators` with `order`, the AR and MA operators `phi` and `theta` (grid by grid by lag, `NULL` when the order is zero), and the one-step-ahead in-sample `fitted` curves and `residuals`. Printing it shows the norm of each operator.

References

Kuenzer, T. (2024). Estimation of functional ARMA models. *Bernoulli*.

See Also

`select_order()`, `farfima()`

Examples

```
x = sim_farfima(n = 300, d = 0, p = 1, q = 1, seed = 1)
op = estimate_operators(x, order = c(1, 1))
op
plot(op)
```

`farfima` *Fit a FARFIMA model*

Description

Fits a functional autoregressive fractionally integrated moving-average (FARFIMA) model to the training sample and calibrates conformal prediction bands on the validation sample.

Usage

```
farfima(
  train,
  valid = NULL,
  d = c("bc_lpwn2", "bc_lpwn1", "lpwn2", "lpwn1", "lw", "dfa"),
  order = c("ic_full", "ic_auto", "bj", "cv"),
  p = 0:5,
  q = 0:5,
  method = c("farma", "far"),
  h = 10,
  level = c(0.8, 0.95),
  band = c("SR", "NS", "AC"),
  window = c("expanding", "fixed"),
  valid_eval = NULL,
  Npc = 1,
  trim = NULL,
  far_pve = 0.95,
  memory_args = list(),
  order_args = list(),
  eta_grid = c(0, 0.005, 0.01, 0.02, 0.05, 0.1),
  zeta_grid = seq(0.1, 50, length.out = 2496),
  symmetric = TRUE,
  integrate = TRUE,
  KI = NULL,
  scorecast = c("none", "naive", "theta"),
  acmcp_regression = TRUE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

train	Training sample: a matrix of curves (grid points by time) or an fts object.
valid	Validation sample, following train. It is needed for the conformal prediction bands and for order = "cv".
d	Memory parameter estimator for estimate_memory() ("lw", "dfa", "lpwn1", "lpwn2", "bc_lpwn1" or "bc_lpwn2"), or a known value.
order	Order selection method for select_order() ("ic_full", "ic_auto", "bj" or "cv"), or known orders c(p, q). With "cv", the forecasts are compared with the validation curves on their original scale.
p, q	Candidate AR and MA orders searched by the selection method. Not used when order gives the orders. (0, 0) is excluded.
method	"farma" for the functional Hannan-Rissanen estimator, or "far" for the FAR(1) benchmark fitted by far::far(). The benchmark ignores order, and with d = 1 it is the FARFIMA(1, 1, 0) benchmark.
h	Maximum forecast horizon of the conformal prediction bands and of order = "cv". valid needs at least h + 1 curves.
level	Nominal coverage levels of the conformal prediction bands.
band	Conformal prediction bands to calibrate: "SR" (scaled residual), "NS" (non-conformity score), "AC" (adaptive conformal), "PID" (conformal PID control, proportional-integral-derivative) and "AcMCP" (autocorrelated multistep conformal). See predict().
window	"expanding" or "fixed" window at each forecast origin. The fixed window has the length of the training sample.
valid_eval	Optional curves to check the validation forecasts against, in place of valid. The forecasts are still made from valid. Use it when the model is fitted to smoothed curves but the bands should be calibrated on the raw curves. It must have the same size as valid.
Npc	Number of principal components in the operator estimator.
trim	Number of initial fractionally differenced curves dropped before order selection and estimation. Defaults to $\min(100, \lfloor T/3 \rfloor)$, where T is the number of training curves.
far_pve	Proportion of variance explained by the components of the FAR(1) benchmark.
memory_args	List of other arguments to estimate_memory(), such as m_const or B.
order_args	List of other arguments to select_order(), such as n_lags or alpha.
eta_grid	Candidate learning rates of the AC, PID and AcMCP conformal prediction bands. One rate is chosen for each band and level, by the mean interval score on the second half of the validation sample.
zeta_grid	Candidate multipliers of the SR band.
symmetric	If TRUE, the AC and PID bands are symmetric about the point forecast. If FALSE, their lower and upper edges are calibrated separately, each at level $\alpha/2$. The SR and NS bands are always symmetric and the AcMCP band is always asymmetric.

integrate	Whether the PID and AcMCP bands include the integral term, which widens the band after repeated under-coverage and narrows it after repeated over-coverage. In the extreme, the band becomes infinite or empty.
KI	Scale of the integral term. Defaults to the largest absolute standardised validation residual at each horizon.
scorecast	Scorecaster of the PID band: "none", "naive" (the latest observed score curve) or "theta" (the theta method: exponential smoothing of the past scores plus half their linear trend).
acmcp_regression	Whether the centre of the AcMCP band also uses the regression on the scores at shorter horizons. If FALSE, only the MA(h - 1) forecast is used.
seed	Optional seed for the BC-LPWN bootstrap and the white-noise test of order = "bj".
verbose	Whether to show progress messages and a progress bar.

Details

The fit has three steps. The memory parameter is estimated by estimate_memory(). The training curves are then centred and fractionally differenced, and the first trim differenced curves are dropped, which gives short-memory curves. Their FARMA orders are selected as in select_order(), and their operators are estimated by the functional Hannan-Rissanen method, as in estimate_operators().

With a validation sample, forecasts up to h steps ahead are made from every validation origin, and their errors calibrate the conformal prediction bands for each horizon and level. A validation origin whose fit fails is skipped with a warning. A warning is also given when the fitted operators are unstable.

Value

An object of class farfima, a list with

- d and order: the memory parameter and c(p = , q =);
- phi, theta: the AR and MA operators (grid by grid by lag);
- fitted, fitted_differenced, residuals: the in-sample fit on the scale of the data and on the fractionally differenced scale, and the residuals, also given by fitted() and residuals() (NULL for the FAR(1) benchmark);
- memory, order_selection: the estimate_memory() and select_order() results, or NULL when d or the orders are given;
- operators: the estimate_operators() result, or NULL for the FAR(1) benchmark;
- calibration: a table of the calibrated conformal prediction bands by level and h: SR_zeta (SR multiplier), NS_quantile (NS quantile), AC_eta, PID_eta and AcMCP_eta (learning rates, for the bands chosen) and n_origin (number of validation origins). NULL without a validation sample.

print() and summary() show the fit, coef() gives d, phi and theta, and plot() draws the fitted or residual curves.

References

Kuenzer, T. (2024). Estimation of functional ARMA models. *Bernoulli*.

Li, D., Robinson, P. M. and Shang, H. L. (2020). Long-range dependent curve time series. *Journal of the American Statistical Association*.

See Also

`predict()` to forecast from the fit; `estimate_memory()`, `select_order()` and `estimate_operators()` for the steps on their own.

Examples

```
sim = sim_farfima(n = 200, d = 0.3, p = 1, q = 0, no_grid = 51, seed = 1)
blocks = split_curves(sim)
fit = farfima(blocks$train, blocks$valid, d = "lpwn2", order = "ic_full",
              p = 0:2, q = 0:2, h = 5, trim = 30)
fit
```

`fitted.farfima` *Fitted curves and residuals of a FARFIMA fit*

Description

The in-sample fit of a model from `farfima()`. Each fitted curve is the one-step-ahead forecast of a training curve from the curves before it, and each residual curve is the training curve minus its fitted curve. They cover the training curves left after trimming; the first `max(p, q)` columns are `NA`, because the model needs that many earlier curves.

Usage

```
## S3 method for class 'farfima'
fitted(object, scale = c("original", "differenced"), ...)

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

Arguments

`object`	A fit from `farfima()` with `method = "farma"`.
`scale`	`"original"` or `"differenced"`.
`...`	Not used.

Details

`fitted(fit)` is on the scale of the data, so `fitted(fit) + residuals(fit)` gives the training curves. `fitted(fit, scale = "differenced")` is the FARMA fit of the fractionally differenced curves. The residuals are the same on both scales.

Value

`fitted()` returns the fitted curves and `residuals()` the residual curves, each as a matrix with grid points in rows and training curves in columns.

See Also

`farfima()`; `plot()` to draw the curves.

Examples

```
sim = sim_farfima(n = 200, d = 0.3, p = 1, q = 0, seed = 1)
fit = farfima(sim, d = 0.3, order = c(1, 0), trim = 50, verbose = FALSE)
dim(fitted(fit))
range(fitted(fit) + residuals(fit) - sim[, 51:200], na.rm = TRUE)
```

---

frac_diff *Fractional differencing*

Description

Applies $(1 - B)^d$ at every grid point, where B is the backshift operator. A negative `d` integrates, so `frac_diff(frac_diff(x, d), -d)` returns `x`.

Usage

```
frac_diff(x, d)
```

Arguments

`x`	A matrix of curves (grid points by time).
`d`	Memory parameter, the order of differencing. A negative `d` fractionally integrates the curves.

Value

The fractionally differenced curves, or the fractionally integrated curves when `d` is negative, as a matrix of the same size as `x`.

See Also

`estimate_memory()`

Examples

```
x = sim_farfima(n = 100, d = 0.3, p = 1, q = 0, seed = 1)
y = frac_diff(x, 0.3)
max(abs(frac_diff(y, -0.3) - x))
```

`plot.farfima` *Plot the fitted or residual curves of a FARFIMA fit*

Description

Rainbow plots of the in-sample fit of a model from `farfima()`, coloured by time as in `plot_curves()`. `type = "fitted"` draws the fitted curves on the scale of the data, and `type = "residuals"` draws the residual curves. The first `max(p, q)` curves have no fit and are left out.

Usage

```
## S3 method for class 'farfima'
plot(
  x,
  type = c("fitted", "residuals"),
  argvals = NULL,
  xlab = "Grid",
  ylab = NULL,
  main = NULL,
  ...
)
```

Arguments

`x`	A fit from `farfima()` with `method = "farma"`.
`type`	`"fitted"` or `"residuals"`.
`argvals`	Grid values. Defaults to the row names of the curves if numeric.
`xlab, ylab, main`	Axis labels and title.
`...`	Other arguments passed to `graphics::matplot()`.

Value

`x`, invisibly.

See Also

`fitted()` and `residuals()` for the curves.

Examples

```
sim = sim_farfima(n = 200, d = 0.3, p = 1, q = 0, no_grid = 51, seed = 1)
fit = farfima(sim, d = 0.3, order = c(1, 0), trim = 50, verbose = FALSE)
plot(fit)
plot(fit, type = "residuals")
```

`plot.farfima_forecast` *Plot forecasts and conformal prediction bands*

Description

- `type = "curve"`: a single plot of the forecast `h` steps beyond the data with its conformal prediction band. With `target_test_curve = k`, the forecast of the k-th test curve made `h` steps earlier, with the observed curve.
- `type = "horizons"`: a grid of plots, one column per band type and one row per horizon, each showing the point forecast with all levels of the band shaded. With a single band type, the horizons are side by side. With a test sample, the forecasts come from the latest test origin that reaches every horizon shown, and the observed curves are added.
- `type = "series"`: a row of plots, one per grid point in `at`, each showing that grid point over time with the forecasts beyond the data.
- `type = "recent"`: a single plot of the last `n_recent` curves in grey and the forecasts beyond the data, coloured by horizon.

Usage

```
## S3 method for class 'farfima_forecast'
plot(
  x,
  type = c("curve", "horizons", "series", "recent"),
  h = 1,
  band = attr(x, "settings")$band[1],
  level = attr(x, "settings")$level[1],
  target_test_curve = NULL,
  at = NULL,
  n_recent = 30,
  argvals = NULL,
  xlab = NULL,
  ylab = "Value",
  main = NULL,
  legend = "bottomright",
  ...
)
```

Arguments

`x`	A forecast from `predict()`.
`type`	`"curve"`, `"horizons"`, `"series"` or `"recent"`.
`h`	Forecast horizon. For `"horizons"` and `"recent"`, one or more horizons: by default 1, `h/2` and `h` for `"horizons"`, and all for `"recent"`.
`band`	Conformal prediction band to draw. For `"horizons"`, one or more bands, by default all.

level	Level of the band for type = "curve".
target_test_curve	Position in the test sample of the curve forecast, between h and the number of test curves (type = "curve").
at	Grid values for type = "series", by default four spread over the grid.
n_recent	Number of curves shown for type = "recent".
argvals	Grid values. Defaults to the row names of the curves if numeric.
xlab, ylab	Axis labels.
main	Title. For type = "series", the text before the grid value in each panel title, such as "Age".
legend	Legend position, or NULL for no legend.
...	Other arguments passed to graphics::plot(), such as xlim.

Value

x, invisibly.

See Also

predict()

Examples

```
sim = sim_farfima(n = 200, d = 0.3, p = 1, q = 0, no_grid = 51, seed = 1)
blocks = split_curves(sim)
fit = farfima(blocks$train, blocks$valid, d = 0.3, order = c(1, 0), h = 5,
              trim = 50, verbose = FALSE)
fc = predict(fit, test = blocks$test, verbose = FALSE)
plot(fc, h = 1)
plot(fc, h = 2, target_test_curve = 10)
plot(fc, type = "horizons")
plot(fc, type = "series")
plot(fc, type = "recent")
```

---

plot.farfima_order *Select the FARMA orders*

Description

Selects the FARMA(p, q) orders of short-memory curves. Long-memory curves need to be fractionally differenced first, for example with frac_diff(). farfima() does this on the centred training curves and drops the first trim before selecting the orders.

Usage

```
## S3 method for class 'farfima_order'
plot(x, main = NULL, ...)

select_order(
  x,
  method = c("ic_full", "ic_auto", "bj", "cv"),
  p = 0:5,
  q = 0:5,
  Npc = 1,
  n_lags = NULL,
  alpha = 0.05,
  iid = FALSE,
  starts = NULL,
  valid = NULL,
  valid_eval = NULL,
  h = 10,
  window = c("expanding", "fixed"),
  cv_score = c("rmse", "mae"),
  seed = NULL,
  verbose = FALSE
)
```

Arguments

x	A matrix of short-memory training curves (grid points by time) or an `fts` object. For `plot()`, a `farfima_order` object.
main	Plot title.
...	Other arguments passed to `graphics::image()`, or to `graphics::barplot()` for `"bj"`.
method	`"ic_full"`, `"ic_auto"`, `"bj"` or `"cv"`.
p, q	Candidate AR and MA orders. (0, 0) is excluded.
Npc	Number of principal components in the operator estimator.
n_lags	Number of residual autocovariance lags. Defaults to $\lfloor \log T \rfloor$ for the criteria and 20 for `"bj"`.
alpha	Significance level of the white-noise test (`"bj"`).
iid	Whether the white-noise test assumes iid noise rather than weak white noise (`"bj"`).
starts	Starting models, a data frame with columns `p` and `q` (`"ic_auto"`). Defaults to (2, 2), (1, 0) and (0, 1).
valid	Validation curves following `x`, also short-memory (`"cv"`).
valid_eval	Optional curves to check the validation forecasts against, in place of `valid` (`"cv"`).
h	Maximum forecast horizon (`"cv"`).

window	"expanding" or "fixed" window ("cv").
cv_score	"rmse" or "mae" ("cv").
seed	Optional seed. The white-noise test of "bj" uses Monte Carlo integration, so its p-values change between runs without one.
verbose	Whether to show progress messages and a progress bar.

Details

- "ic_full" (functional IC) minimises the adaptive functional Hannan-Quinn information criterion over all candidates.
- "ic_auto" (auto functional IC) uses the same adaptive functional Hannan-Quinn information criterion with a stepwise search from a few starting models. Its noise level comes from the largest model.
- "bj" (functional Box-Jenkins) tries the candidates by increasing p + q and stops at the first whose residuals pass the functional portmanteau test.
- "cv" (cross-validation) minimises the RMSPE, or MAPE, of the forecasts from the validation origins, averaged over horizons. A candidate whose fit fails at any origin is dropped.

Value

An object of class farfima_order with the selected p and q and a table of the candidates. "ic_auto" also returns its search_path. plot() shows the criterion or CV score of each candidate with the selected model boxed, or the p-values for "bj".

References

Hannan, E. J. and Quinn, B. G. (1979). The determination of the order of an autoregression. *Journal of the Royal Statistical Society: Series B*.

Kokoszka, P., Rice, G. and Shang, H. L. (2017). Inference for the autocovariance of a functional time series under conditional heteroscedasticity. *Journal of Multivariate Analysis*.

Kuenzer, T. (2024). Estimation of functional ARMA models. *Bernoulli*.

See Also

estimate_operators() for the operators, and farfima() for the whole fit.

Examples

```
x = sim_farfima(n = 300, d = 0, p = 1, q = 0, seed = 1)
o = select_order(x, method = "ic_full", p = 0:2, q = 0:2)
o
plot(o)
```

plot_curves *Rainbow plots of curves*

Description

Draws the curves coloured by time, from red (earliest) to violet (latest). `type = "rainbow"` draws all curves in one panel with a colour bar, and `type = "periods"` draws one panel per period with the other curves in grey.

Usage

```
plot_curves(
  x,
  type = c("rainbow", "periods"),
  periods = 5,
  argvals = NULL,
  xlab = "Grid",
  ylab = "Value",
  main = NULL,
  ...
)
```

Arguments

`x`	A matrix of curves (grid points by time) or an `fts` object.
`type`	`"rainbow"` or `"periods"`.
`periods`	Number of periods of equal length, or the first time point of each period, such as `c(1901, 1921, 1941, 1961, 1981)`.
`argvals`	Grid values. Defaults to the row names of `x` if numeric.
`xlab, ylab, main`	Axis labels and title.
`...`	Other arguments passed to `graphics::matplot()`.

Value

`x`, invisibly.

See Also

`split_curves()`

Examples

```
x = sim_farfima(n = 100, d = 0.3, p = 1, q = 0, no_grid = 21, seed = 1)
plot_curves(x)
plot_curves(x, type = "periods", periods = 4)
```

`predict.farfima` *Forecast from a FARFIMA fit*

Description

Point forecasts and conformal prediction bands up to `h` steps beyond the data. With a test sample, forecasts are also made from every test origin and evaluated.

Usage

```
## S3 method for class 'farfima'
predict(
  object,
  test = NULL,
  h = attr(object, "settings")$h,
  band = attr(object, "settings")$band,
  test_eval = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	A fit from `farfima()`.
`test`	Optional test sample, following the validation sample.
`h`	Maximum forecast horizon, at most the `h` of the fit.
`band`	Conformal prediction bands to return, from those calibrated in `farfima()`.
`test_eval`	Optional curves to check the test forecasts against, in place of `test`. The forecasts are still made from `test`.
`verbose`	Whether to show progress messages and a progress bar.
`...`	Not used.

Details

The conformal prediction bands are those chosen in `farfima()`:

- `"SR"`: scaled residual band, the forecast plus and minus a multiplier of the point-wise scale function;
- `"NS"`: non-conformity score band, with the width set by a quantile of the standardised non-conformity scores;
- `"AC"`: adaptive conformal band, an NS band whose quantile level is updated over the test sample;
- `"PID"`: conformal PID control band, whose multiplier is updated by a proportional term, an integral term and an optional scorecaster;

- `"AcMCP"`: autocorrelated multistep conformal band, an asymmetric PID band shifted by a forecast of the standardised residual curve.

The AC, PID and AcMCP bands are updated from the error rates of earlier bands, once their targets are observed.

In the test evaluation, the memory parameter and the orders stay fixed, and the operators and the mean are re-estimated at each origin. An origin whose fit fails is skipped with a warning. The forecast beyond the data starts after the last test curve, or after the last validation curve when there is no test sample.

Value

An object of class `farfima_forecast`, a list with

- `point_forecast`: the forecast curves beyond the data (grid by `h`);
- `bands`: the conformal prediction bands, with the `lower` and `upper` matrices (grid by `h`) for each band and level (such as `"80%"`);
- `evaluation`, with a test sample: the tables `point` (RMSPE, the root mean squared prediction error, and MAPE, the mean absolute prediction error), `interval` (ECP, CPD and MIS of the bands) and `wis` (weighted interval score).

References

Gibbs, I. and Candes, E. J. (2021). Adaptive conformal inference under distribution shift. *Advances in Neural Information Processing Systems*.

Angelopoulos, A. N., Candes, E. J. and Tibshirani, R. J. (2023). Conformal PID control for time series prediction. *Advances in Neural Information Processing Systems*.

Wang, X. and Hyndman, R. J. (2024). Online conformal inference for multi-step time series forecasting. arXiv:2410.13115.

Wang, X. and Girolimetto, D. (2026). conformalForecast: an R package for sequential conformal prediction in time series. arXiv:2610.21648.

Bracher, J., Ray, E. L., Gneiting, T. and Reich, N. G. (2021). Evaluating epidemic forecasts in an interval format. *PLOS Computational Biology*.

See Also

`plot()` and `as.data.frame()` for the results.

Examples

```
sim = sim_farfima(n = 200, d = 0.3, p = 1, q = 0, no_grid = 51, seed = 1)
blocks = split_curves(sim)
fit = farfima(blocks$train, blocks$valid, d = 0.3, order = c(1, 0), h = 5, trim = 30)
fc = predict(fit, test = blocks$test)
fc
```

`sim_farfima` *Simulate a FARFIMA(p, d, q) process*

Description

Simulates `n` curves from a FARFIMA(p, d, q) model on `no_grid` equally spaced points of [0, 1]. The innovations are Brownian motion curves. They are fractionally integrated with memory parameter `d`, passed through the MA operators and then through the AR recursion. The first 100 curves are dropped as burn-in.

Usage

```
sim_farfima(
  n = 500,
  d = 0.1,
  p = 1,
  q = 0,
  no_grid = 101,
  seed = NULL,
  ar_strength = 0.6,
  ma_strength = 0.6,
  lag_strength = c("equal", "decay")
)
```

Arguments

`n`	Number of curves.
`d`	Memory parameter.
`p`, `q`	AR and MA orders, not both zero.
`no_grid`	Number of grid points.
`seed`	Optional seed.
`ar_strength`, `ma_strength`	Operator strengths ψ and ν.
`lag_strength`	`"equal"` splits the strength equally across lags, and `"decay"` lets it decay geometrically.

Details

The AR kernel is $\exp\{(u^2 + v^2)/2\}$ and the MA kernel is $\min(u, v)$. Each is scaled to operator norm one and multiplied by `ar_strength` or `ma_strength`.

Value

A matrix of curves, `no_grid` by `n`. The innovations are kept in `attr(, "innovations")`.

See Also

split_curves(), farfima()

Examples

```
sim = sim_farfima(n = 200, d = 0.2, p = 1, q = 1, seed = 1)
dim(sim)
```

---

split_curves *Split curves into training, validation and test samples*

Description

Splits the curves in time order.

Usage

```
split_curves(x, train = 0.6, valid = 0.2)
```

Arguments

x	A matrix of curves (grid points by time) or an fts object.
train, valid	Proportions of the curves in the training and validation samples, rounded down. The remaining curves form the test sample.

Value

A list with train, valid and test. test is NULL when train + valid = 1.

See Also

farfima()

Examples

```
x = sim_farfima(n = 100, d = 0.2, p = 1, q = 0, no_grid = 21, seed = 1)
blocks = split_curves(x, train = 0.6, valid = 0.2)
sapply(blocks, ncol)
```

Index

as.data.frame(), *2, 3*, *20*
as.data.frame.farfima_forecast, 3

estimate_memory, 4
estimate_memory(), *2, 3*, *9–12*
estimate_operators, 6
estimate_operators(), *2, 3*, *10, 11*, *17*

far::far(), *9*
farfima, 8
farfima(), *2, 3*, *6, 7*, *11–13*, *15*, *17*, *19*, *22*
farfima-package, 2
fitted(), *2*, *10*, *13*
fitted.farfima, 11
frac_diff, 12
frac_diff(), *3*, *6*, *15*

graphics::barplot(), *16*
graphics::image(), *7*, *16*
graphics::matplot(), *13*, *18*
graphics::plot(), *15*

plot(), *2*, *10*, *12*, *20*
plot.farfima, 13
plot.farfima_forecast, 14
plot.farfima_operators
        *(*estimate_operators*)*, 6
plot.farfima_order, 15
plot_curves, 18
plot_curves(), *2*, *13*
predict(), *2–4*, *9*, *11*, *14, 15*
predict.farfima, 19

residuals(), *2*, *10*, *13*
residuals.farfima *(*fitted.farfima*)*, 11

select_order *(*plot.farfima_order*)*, 15
select_order(), *2, 3*, *7*, *9–11*
sim_farfima, 21
sim_farfima(), *3*
split_curves, 22
split_curves(), *2*, *18*, *22*