Package 'tsgc'

September 1, 2026

Title Time Series Methods Based on Growth Curves

Version 2.0.0

Description Provides tools for modelling and forecasting epidemic trajectories using a dynamic Gompertz model within a state space framework, with the Kalman filter for robust estimation of non-linear growth. Includes a reinitialization feature to adapt to new waves, and a leading-indicator extension that uses a related series moving ahead of the variable of interest (e.g. cases ahead of hospitalisations) to improve short-horizon forecasts, with model and lag selection via rolling-origin cross-validation. Applicable to data at daily, monthly, quarterly, or annual frequency, and to non-epidemic trajectories with similar dynamics, such as innovation diffusion and product adoption. Includes functions for data preprocessing, model fitting, forecast visualization, and accuracy evaluation using standard error measures. Methods are described in Harvey and Kattuman (2020) <doi:10.1162/99608f92.828f40de>, Harvey and Kattuman (2021) <doi:10.1098/rsif.2021.0179>, and Ashby, Harvey, Kattuman, Tang, and Thamotheram (2024) <https://www.jbs.cam.ac.uk/wp-content/uploads/2024/03/cchle-tsgc-paper-2024.pdf>.

URL https://github.com/edwintang903/tsgc

License GPL (>= 3)

Encoding UTF-8

VignetteBuilder knitr

Suggests ggfortify, knitr, RColorBrewer, rmarkdown, ggforce, gridExtra, latex2exp, here, testthat, dplyr, ggthemes

Config/testthat/edition 3

Imports KFAS, xts, ggplot2 (>= 3.5.0), zoo, magrittr, tidyr, methods, abind, purrr, scales, kableExtra

BugReports https://github.com/edwintang903/tsgc/issues

Depends R (>= 3.5.0)

LazyData true

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

NeedsCompilation no

Author Michael Ashby [aut, cre],
Paul Kattuman [aut],
Andrew Harvey [aut],
Edwin Tang [aut],
Craig Thamotheram [aut],
Guglielmo Secchi [aut],
Cambridge Centre for Health Leadership & Enterprise, Cambridge Judge Business School, University of Cambridge [fnd],
UK Health Security Agency [fnd],
Keynes Fund, Faculty of Economics, University of Cambridge [fnd] (Supported A. Harvey),
University of Cambridge Social Science Impact Fund [fnd] (Supported P. Kattuman),
Cambridge Mathematics Placements Programme, University of Cambridge [fnd] (Supported E. Tang),
Caldecott Bursary Fund, Magdalene College, University of Cambridge [fnd] (Supported E. Tang),
Chancellor's Scholarship Scheme, University of Warwick [fnd] (Supported E. Tang),
Statistics Centre for Doctoral Training, University of Warwick [fnd] (Supported E. Tang)

Maintainer Michael Ashby <mwa22@cam.ac.uk>

Repository CRAN

Date/Publication 2026-09-01 12:50:02 UTC

Contents

alphahat . . . 4
argmax . . . 5
as.matrix.idx_series . . . 6
as.numeric.idx_series . . . 6
as_idx_series . . . 7
att . . . 7
cross_val . . . 8
df2ldl . . . 9
df2ldl_lead . . . 10
england . . . 10
england_weather_2021 . . . 11
estimate . . . 12
estimate_r0 . . . 12
etrading_apps . . . 14
FilterResults-class . . . 14
FilterResultsLI-class . . . 17
gauteng . . . 20

gauteng_weather_2021 . . . 21
gety . . . 22
gety.hat . . . 22
get_timeframe . . . 23
get_V . . . 24
head.idx_series . . . 24
idx_axis_opts . . . 25
idx_calendar . . . 26
idx_calendar_multi_step . . . 28
idx_calendar_offset . . . 29
idx_calendar_step . . . 30
idx_cbind . . . 31
idx_diff . . . 31
idx_lag . . . 32
idx_ncol . . . 32
idx_offset_to_pos . . . 33
idx_positions . . . 33
idx_range . . . 34
idx_rbind . . . 34
idx_series . . . 35
idx_step . . . 36
idx_step_add . . . 37
idx_to_date . . . 38
idx_to_pos . . . 39
idx_values . . . 40
is_idx_calendar . . . 41
is_idx_series . . . 41
is_idx_step . . . 42
is_multi_step_pattern . . . 42
length.idx_series . . . 43
mapes . . . 43
matrixKFS . . . 44
modelKFS . . . 44
multi_step_pattern . . . 45
NCOL.idx_series . . . 46
nintendo_sales . . . 46
output . . . 47
plot.SSModelDynamicGompertz . . . 48
plot.SSModelLeadingIndicator . . . 49
plot_compare_forecast . . . 50
plot_forecast . . . 51
plot_gy_ci . . . 52
plot_gy_components . . . 53
plot_holdout . . . 53
plot_log_forecast . . . 54
plot_r0 . . . 55
print.FilterResults . . . 56
print.FilterResultsLI . . . 57

print.idx_calendar . . . 58
print.idx_series . . . 58
print.idx_step . . . 59
print.multi_step_pattern . . . 59
print.SSModelDynamicGompertz . . . 60
print.SSModelLeadingIndicator . . . 60
print_model_diagnostics . . . 61
Ptt . . . 62
reinitialise_dataframe . . . 62
seasonalComp . . . 63
SSModelDynamicGompertz-class . . . 64
SSModelLeadingIndicator-class . . . 66
summary.FilterResults . . . 68
summary.FilterResultsLI . . . 68
summary.SSModelDynamicGompertz . . . 69
summary.SSModelLeadingIndicator . . . 70
tail.idx_series . . . 71
ukitaly . . . 71
write_results . . . 72
xts_to_idx . . . 73
[.idx_series . . . 74

Index **75**

`alphahat` *Extract alphahat from a KFS object*

Description

Accessor method to access `alphahat`, the smoothed state estimates, from a fitted `KFS` object.

Usage

```
alphahat(object)
```

Arguments

`object`	A `KFS` object.

Value

The smoothed state estimates `alphahat`.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
tsgc:::alphahat(output(res))
```

`argmax` *Return index and value of maximum*

Description

Similar to Python's argmax function.

Usage

```
argmax(x, decreasing = TRUE)
```

Arguments

`x`	Object to have its maximum found; either an `idx_series` or a plain numeric vector.
`decreasing`	Logical value indicating whether `x` should be ordered in decreasing order. Default is `TRUE`. Setting this to `FALSE` would find the minimum.

Value

If `x` is an `idx_series`, a length-1 `idx_series` at the position of the maximum. Otherwise the maximum value.

Examples

```
x <- idx_series(cumsum(rpois(30, 5)) + 1, start = 1)
argmax(x)
```

as.matrix.idx_series *Coerce an* idx_series *to a matrix*

Description

Coerce an idx_series to a matrix

Usage

```
## S3 method for class 'idx_series'
as.matrix(x, ...)
```

Arguments

x	An idx_series object.
...	Unused.

as.numeric.idx_series *Coerce an* idx_series *to a plain numeric vector or matrix*

Description

as.numeric() is a primitive generic in base R and does not always reliably dispatch S3 methods registered only under the as.numeric.* name. as.double.idx_series is also provided for the same reason.

Usage

```
## S3 method for class 'idx_series'
as.numeric(x, ...)

## S3 method for class 'idx_series'
as.double(x, ...)
```

Arguments

x	An idx_series object.
...	Unused.

as_idx_series *Coerce to* idx_series

Description

Coerces plain vectors/matrices to idx_series, or passes through an idx_series unchanged. Convenience helper so that internal functions can accept either raw data or an already constructed idx_series.

Usage

```
as_idx_series(x, start = 1L)
```

Arguments

x	A numeric vector, matrix or idx_series object.
start	Integer start position, only used if x is not already an idx_series. Defaults to 1.

Value

An idx_series object.

att *Extract filtered state estimates from a KFS object*

Description

Accessor method to access the filtered state estimates in a KFS object.

Usage

```
att(object)
```

Arguments

object	A KFS object.

Value

The filtered state estimates.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
tsgc:::att(output(res))
```

`cross_val` *Walk-Forward Validation for Model Comparison Using Mean Absolute Percentage Error (MAPE)*

Description

This function performs a walk-forward validation to compare forecasting performance across different models specified by the user. It returns a data frame of a user-specified error metric (e.g. MAPE, MAE) for forecasts `n.ahead` positions ahead, using the given models with varying end positions.

Usage

```
cross_val(
  Y,
  model_list,
  est.end,
  n.ahead = 7,
  n.estimate = 1,
  gap = 1,
  xpred_targ.full = NULL,
  xpred_lead.full = NULL,
  LeadIndCol = 1,
  criterion = "mape"
)
```

Arguments

`Y`	An `idx_series` representing the cumulative data series. If a Leading Indicator model is compared, Y should include columns for both the leading indicator and the target variable. The specific column for the leading indicator can be designated using the `LeadIndCol` parameter.
`model_list`	A list containing `SSModelDynamicGompertz` or `SSModelLeadingIndicator` objects, to be compared in a cross validation procedure.
`est.end`	The initial estimation end position for model fitting. Starting from this position, the function re-estimates the model and evaluates the performance for each lag in `all_lags` every `gap` positions, over `n.estimate` steps.

n.ahead	Integer specifying the number of positions to forecast ahead for MAPE evaluation.
n.estimate	Integer indicating the total number of walk-forward validation steps to report.
gap	Integer specifying the position gap between two successive validations, where the model is re-estimated and evaluated during the walk-forward validation.
xpred_targ.full	An idx_series containing the values of exogenous variables for the target variable over the estimation and prediction time frame.
xpred_lead.full	(Only required for leading indicator models) An idx_series containing the values of exogenous variables for the leading indicator over the estimation and prediction time frame.
LeadIndCol	(Only required for leading indicator models) Integer representing the column number in Y that contains the leading indicator.
criterion	A string object indicating how to compare between different models. Available choices are "mape" (by default), "smape", "mae" and "rmse".

Value

A table summarizing the chosen error metric for each model in model_list across the specified positions.

---

df2ldl *Compute log growth rate of cumulated dataset*

Description

Helper method to compute the log growth rates of cumulated variables.

Usage

```
df2ldl(dt)
```

Arguments

dt	Cumulated data series, as an idx_series with exactly one column.

Value

An idx_series of log growth rates of the cumulated variable inputted via the parameter dt.

Examples

```
x <- idx_series(cumsum(rpois(30, 5)) + 1)
df2ldl(x)
```

df2ldl_lead — *Compute successive increments and log growth rate of 2-variable cumulated dataset*

Description

Helper method to compute the successive increments and log growth rates of cumulated variables. It will compute the successive increments and log cumulative growth rate for each column in the 2-column series, which will then be used to predict or estimate with the leading indicator model.

Usage

```
df2ldl_lead(data, LeadIndCol = 1)
```

Arguments

data	Cumulated data series as an idx_series with 2 columns: leading indicator and target variable. Can specify which column is leading indicator by LeadIndCol parameter.
LeadIndCol	Column number of data that contains the leading indicator. An integer that can only take values 1 (by default) or 2.

Value

A list of idx_series with the original cumulative variables, successive increments and log growth rates: cLead, cTarg, newLead, newTarg, LDLlead, LDLtarg.

england — *Daily Covid-19 metrics for England.*

Description

Daily Covid-19 metrics for England.

Usage

```
data(england)
```

Format

An object of class "xts" with columns:

cum_cases Cumulative cases of Covid-19

cum_admissions Cumulative hospital admissions with Covid-19 since the start of the pandemic.

cum_deaths Cumulative deaths within 28 days of a positive test for Covid-19 by death date since the start of the pandemic.

hospital_cases Number of patients in hospital with confirmed Covid-19 each day. Note this is NOT cumulative.

References

Downloaded from https://ukhsa-dashboard.data.gov.uk/topics/covid-19. Now at https://ukhsa-dashboard.data.gov.uk/covid-19-archive-data-download. Full documentation from UKHSA included in downloads from the website.

Examples

```
data(england)
# plot daily cases
plot(diff(england))
```

---

england_weather_2021 *Daily Weather data of England in 2021.*

Description

Daily Weather data of England in 2021.

Usage

```
data(england_weather_2021)
```

Format

An object of class `"xts"` with columns:

temperature_C Daily mean air temperature in degree Celsius (C) at 2 metres

max_temp_C Daily maximum temperature in degree Celsius (C) at 2 metres

relhum_percnt Relative humidity at 2 metres (%)

windspd_mtrs_p_sec Wind speed in metres per second (m/s) at 10 metres

precip_mtrs Precipitation (millimetres per day; mm/day)

References

Downloaded from https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels?tab=overview and aggregated across the latitude and longitude corners for England by the authors.

Examples

```
data(england_weather_2021)
```

estimate *Estimate an SSModelDynamicGompertz or SSModelLeadingIndicator model*

Description

Accessor method that calls the `estimate` method of a `SSModelDynamicGompertz` or `SSModelLeadingIndicator` object.

Usage

```
estimate(model)
```

Arguments

`model`	A `SSModelDynamicGompertz` or `SSModelLeadingIndicator` object.

Value

A `FilterResults` or `FilterResultsLI` object containing the estimation results.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
estimate(model)
```

estimate_r0 *Estimate the reproduction number from a fitted model*

Description

Computes the (instantaneous) reproduction number $R_t = \exp\{g_t \times gen_int\}$, where g_t is the filtered or smoothed growth rate of the incidence variable y returned by `res$get_gy_ci()`, and `gen_int` is the mean generation interval of the disease/process being modelled (in the same integer-position units as `res`). This estimate, and its confidence interval, is a deterministic transformation of `get_gy_ci()`'s output; `res` must already have been produced by `estimate()`.

Usage

```
estimate_r0(
  res,
  gen_int,
  n.ahead = 7,
  smoothed = FALSE,
  confidence.level = 0.68
)
```

Arguments

res	A FilterResults or FilterResultsLI object, obtained from the estimate() method.
gen_int	The mean generation interval, in the same integer-position units as res (e.g. if positions are days, this is the mean generation interval in days).
n.ahead	Number of most recent integer positions to return, taken from the end of res's filtered/smoothed range. Default is 7.
smoothed	Logical value indicating whether to use the smoothed (TRUE) or filtered (FALSE, default) growth rate estimates underlying R_t. Passed through to res$get_gy_ci().
confidence.level	Confidence level for the confidence interval around R_t. Default is 0.68, one standard deviation for a normally distributed random variable. Passed through to res$get_gy_ci().

Value

A data frame containing Position, fit, lower and upper. If res$calendar is available, a Date column is included before Position. Each row gives the estimated reproduction number and its confidence interval over one of the last n.ahead integer positions.

Examples

```
library(tsgc)
set.seed(1)
Y <- idx_series(cumsum(rpois(120, 8)) + 1, start = 1)
cal <- idx_calendar(anchor = as.Date("2021-01-01"), anchor_pos = 1L,
                    amount = 1, unit = "days")
model <- SSModelDynamicGompertz$new(Y = Y, q = NULL, end = 100,
                                    calendar = cal)
res <- estimate(model)
estimate_r0(res, gen_int = 5, n.ahead = 7)
```

etrading_apps *Monthly downloads of French leading apps from January 2017 to July 2021*

Description

Monthly downloads of selected leading apps that allow for online share trading in France from January 2017 to July 2021, by app.

Usage

```
data(etrading_apps)
```

Format

An object of class "xts" with cumulative downloads of each trading app

References

Taken from https://www.statista.com/statistics/1260008/etrading-app-downloads-france/

Examples

```
data(etrading_apps)
```

FilterResults-class *Class for estimated Dynamic Gompertz Curve model*

Description

Class for estimated Dynamic Gompertz Curve model and contains methods to extract smoothed/filtered estimates of the states, the level of the incidence variable y, and forecasts of y. The output from the estimate method of the SSModelDynamicGompertz class is of the class FilterResults.

Fields

`data` An `idx_series` object containing the non-reinitialized cumulated variable.

`xpred_logical` Logical value indicating whether exogenous predictors were used to estimate the FilterResults object.

`index` The integer positions of the observations used in estimation (of `data`).

`reinit.idx` The reinitialisation position (a single integer) of the estimated `SSModelDynamicGompertz` model (if applicable).

`ar1` Logical value indicating whether an ar1 component should be included in the model.

`output` A `KFS` results object obtained after fitting a `SSModelDynamicGompertz` model.

`xpred.new` An `idx_series` object containing exogenous predictors to be used in prediction. Defaults to `NULL`, and should be provided if xpred is used for model estimation.

`sea.period` The period of seasonality, inherited from the estimated `SSModelDynamicGompertz` model. For a day-of-the-week effect with daily data, this would be 7.

`calendar` An optional `idx_calendar` object, inherited unchanged from the `SSModelDynamicGompertz` model this was estimated from. Purely cosmetic: describes how the integer positions in `data` map back to calendar time, for use by plotting only. Defaults to `NULL`.

Methods

`get_growth_y(smoothed = FALSE, return.components = FALSE)` Returns the growth rate of the incidence (y) of the cumulated variable (Y). Computed as

$$g_t = \exp\{\delta_t\} + \gamma_t.$$

Parameters:

- `smoothed` Logical value indicating whether to use the smoothed estimates of δ and γ to compute the growth rate (`TRUE`), or the contemporaneous filtered estimates (`FALSE`). Default is `FALSE`.
- `return.components` Logical value indicating whether to return the estimates of δ and γ as well as the estimates of the growth rate, or just the growth rate. Default is `FALSE`.

Return Value: `idx_series` objects containing smoothed/filtered growth rates and components (δ and γ), where applicable.

`get_gy_ci(smoothed = FALSE, confidence.level = 0.68)` Returns the growth rate of the incidence (y) of the cumulated variable (Y). Computed as

$$g_t = \exp\{\delta_t\} + \gamma_t.$$

Parameters:

- `smoothed` Logical value indicating whether to use the smoothed estimates of δ and γ to compute the growth rate (`TRUE`), or the contemporaneous filtered estimates (`FALSE`). Default is `FALSE`.
- `confidence.level` Confidence level for the confidence interval. Default is 0.68, which is one standard deviation for a normally distributed random variable.

Return Value: `idx_series` object containing smoothed/filtered growth rates and upper and lower bounds for the confidence intervals.

`initialize( data, xpred_logical, index, reinit.idx, ar1, output, sea.period, xpred.new = NULL, calendar =` Create an instance of the `FilterResults` class with fields defined earlier in the fields section.

`mapes(n.ahead, Y)` Computes five metrics, including Mean Absolute Percentage Error (MAPE), for forecasts against a holdout sample. For more details, please refer to mapes.

sMAPE is computed as mean(100 * abs(Actual - Forecast) / (Actual + Forecast)) i.e. this uses (Actual + Forecast), not the more common (|Actual| + |Forecast|) / 2, in the denominator. As a result this statistic ranges over [0, 100], NOT the [0, 200] range associated with the textbook sMAPE definition that divides by the mean of the absolute values. Do not compare this sMAPE value directly against sMAPE figures computed with the conventional 0-200 formula.

`predict_all( n.ahead, sea.on = TRUE, return.all = FALSE, confidence.level = 0.68 )` Returns forecasts of the incidence variable y, the state variables and the conditional covariance matrix for the states.

Parameters:

- `n.ahead` The number of forecasts you wish to create from the end of your sample period.
- `sea.on` Logical value indicating whether seasonal components should be included in the state-space model or not. Default is `TRUE`.
- `return.all` Logical value indicating whether to return all filtered estimates and forecasts. (`TRUE`) or only the forecasts (`FALSE`). Default is `FALSE`.
- `confidence.level` The confidence level for the log growth rate that should be used to compute. Confidence intervals only reported for the incidence variable y.

Return Value: `idx_series` objects containing the forecast (and filtered, where applicable) level of y (`y.hat`), δ (`level.t.t`), γ (`slope.t.t`), vector of states including the seasonals where applicable (`a.t.t`) and covariance matrix of all states including seasonals where applicable (`P.t.t`).

`predict_level( n.ahead, confidence.level = 0.68, sea.on = TRUE, return.diff = TRUE )` Forecast the cumulated variable or the incidence of it. This function returns the forecast of the cumulated variable Y, or the forecast of the incidence of the cumulated variable, y. For example, in the case of an epidemic, y might be daily new cases of the disease and Y the cumulative number of recorded infections.

Parameters:

- `n.ahead` The number of periods ahead you wish to forecast from the end of the estimation window.
- `confidence.level` The confidence level for the log growth rate that should be used to compute the forecast intervals of y.
- `sea.on` Logical value indicating whether to return the prediction of just the trend or prediction incorporating seasonality. Deafults to `TRUE`.
- `return.diff` Logical value indicating whether to return the cumulated variable, Y, or the incidence of it, y (i.e., the first difference of the cumulated variable). Default is `TRUE`.

Return Value: `idx_series` object containing the point forecasts and upper and lower bounds of the forecast interval.

`print()` Provides a quick glimpse of model states and standard errors.

`print_estimation_results()` Prints a table of estimated parameters in a format ready to paste into LaTeX.

`summary()` Supplies details of the FilterResults object, such as estimated parameter values, start and end positions of estimation.

References

Harvey, A. C. and Kattuman, P. (2021). A Farewell to R: Time Series Models for Tracking and Forecasting Epidemics, Journal of the Royal Society Interface, vol 18(182): 20210179

Examples

```
library(tsgc)
set.seed(1)
Y <- idx_series(cumsum(rpois(120, 8)) + 1, start = 1)

# Specify a model
model <- SSModelDynamicGompertz$new(Y = Y, q = 0.005, end = 100)

# Estimate a specified model
res <- estimate(model)

# Show summary of object
summary(res)

# Print a short description of the object
print(res)

# Print estimation results
res$print_estimation_results()

# Forecast 7 periods ahead from the end of the estimation window
res$predict_level(n.ahead = 7,
  confidence.level = 0.68, sea.on=TRUE)

# Forecast 7 periods ahead from the model and return filtered states
res$predict_all(n.ahead = 7, return.all = TRUE)

# Return the filtered growth rate and its components
res$get_growth_y(return.components = TRUE)

# Return smoothed growth rate of incidence variable and its confidence
# interval
res$get_gy_ci(smoothed = TRUE, confidence.level = 0.68)

# Return MAPE of forecast
res$mapes(n.ahead=7,Y)
```

FilterResultsLI-class *Class for the estimated Leading Indicator Model*

Description

A class that holds the information of an estimated Leading Indicator model. Contains methods to extract smoothed/filtered estimates of the states, the level of the incidence variable y, and forecasts of y. The output from the estimate method of the SSModelLeadingIndicator class is of the class FilterResultsLI.

Fields

`data` An `idx_series` object with cumulated variables: lagged leading indicator and target variable (plus their increments and log-growth rates - see `df2ldl_lead`).

`output` A `KFS` results object obtained after fitting a `SSModelLeadingIndicator` model.

`n.lag` Number of integer positions the leading indicator is lagged by, inherited from the estimated `SSModelLeadingIndicator` model.

`sea.period` The period of seasonality, inherited from the estimated `SSModelLeadingIndicator` model. For a day-of-the-week effect with daily data, this would be 7.

`LeadIndCol` The column in `data` that contains the leading indicator, inherited from the estimated `SSModelLeadingIndicator` model.

`xpred_logical` Vector of length 2 with logical values, indicating whether there are exogenous predictors for leading series and target series respectively.

`xpred_lead.new` An `idx_series` object containing the values of exogenous variables for the leading indicator over the prediction time frame.

`xpred_targ.new` An `idx_series` object containing the values of exogenous variables for the target variable over the prediction time frame.

`start` Integer position marking the start of the estimation period.

`end` Integer position marking the end of the estimation period.

`calendar` An optional `idx_calendar` object, inherited unchanged from the `SSModelLeadingIndicator` model this was estimated from. Purely cosmetic: describes how the integer positions in `data` map back to calendar time, for use by plotting only. Defaults to `NULL`.

Methods

`get_growth_y(smoothed = FALSE, return.components = FALSE)` Returns the growth rate of the incidence (y) of the cumulated variable (Y). Computed as

$$g_t = \exp\{\delta_t\} + \gamma_t.$$

Parameters:

- `smoothed` Logical value indicating whether to use the smoothed estimates of δ and γ to compute the growth rate (`TRUE`), or the contemporaneous filtered estimates (`FALSE`). Default is `FALSE`.
- `return.components` Logical value indicating whether to return the estimates of δ and γ as well as the estimates of the growth rate, or just the growth rate. Default is `FALSE`.

Return Value: `idx_series` objects containing smoothed/filtered growth rates and components (δ and γ), where applicable.

`get_gy_ci(smoothed = FALSE, confidence.level = 0.68)` Returns the growth rate of the incidence (y) of the cumulated variable (Y). Computed as

$$g_t = \exp\{\delta_t\} + \gamma_t.$$

Parameters:

- `smoothed` Logical value indicating whether to use the smoothed estimates of δ and γ to compute the growth rate (`TRUE`), or the contemporaneous filtered estimates (`FALSE`). Default is `FALSE`.

- `confidence.level` Confidence level for the confidence interval. Default is 0.68, which is one standard deviation for a normally distributed random variable.

Return Value: `idx_series` object containing smoothed/filtered growth rates and upper and lower bounds for the confidence intervals.

`initialize( data, output, n.lag, sea.period, LeadIndCol, xpred_logical, start, end, xpred_lead.new = NULL` Create an instance of the `FilterResultsLI` class with fields defined earlier in the fields section.

`mapes(n.ahead, Y)` Computes five metrics, including Mean Absolute Percentage Error (MAPE), for forecasts against a holdout sample. For more details, please refer to mapes.

sMAPE is computed as mean(100 * abs(Actual - Forecast) / (Actual + Forecast)) i.e. this uses (Actual + Forecast), not the more common (|Actual| + |Forecast|) / 2, in the denominator. As a result this statistic ranges over [0, 100], NOT the [0, 200] range associated with the textbook sMAPE definition that divides by the mean of the absolute values. Do not compare this sMAPE value directly against sMAPE figures computed with the conventional 0-200 formula.

`predict_all( n.ahead, sea.on = TRUE, return.all = FALSE, confidence.level = 0.68 )` Returns forecasts of the incidence variable y, the state variables and the conditional covariance matrix for the states.

Parameters:

- `n.ahead` The number of forecasts you wish to create from the end of your sample period.
- `sea.on` Logical value indicating whether to return the prediction of just the trend or prediction incorporating seasonality. Default is `TRUE`.
- `return.all` Logical value indicating whether to return all filtered estimates and forecasts (`TRUE`) or only the forecasts (`FALSE`). Default is `FALSE`.

Return Value: `idx_series` objects containing the forecast (and filtered, where applicable) level of y (`y.hat`), δ (`level.t.t`), γ (`slope.t.t`), vector of states including the seasonals where applicable (`a.t.t`) and covariance matrix of all states including seasonals where applicable (`P.t.t`).

`predict_level(n.ahead = n.lag, confidence.level = 0.68, sea.on = TRUE)` Forecast the cumulated variable or the incidence of it. This function returns the forecast of the cumulated variable Y, or the forecast of the incidence of the cumulated variable, y. For example, in the case of an epidemic, y might be daily new cases of the disease and

Parameters:

- `n.ahead` The number of periods ahead you wish to forecast from the end of the estimation window. Default is `n.lag`.
- `sea.on` Logical value indicating whether to return the prediction of just the trend or prediction incorporating seasonality. Deafults to `TRUE`.
- `confidence.level` The confidence level for the log growth rate that should be used to compute the forecast intervals of y.

Return Value: An `idx_series` object containing the point forecasts and upper and lower bounds of the forecast interval.

`print()` Provides a quick glimpse of model states and standard errors.

`print_estimation_results()` Prints a table of estimated parameters in a format ready to paste into LaTeX.

`summary()` Supplies details of the FilterResultsLI object, such as estimated parameter values, start and end positions of estimation.

References

Harvey, A. (2021). TIME SERIES MODELLING OF EPIDEMICS: LEADING INDICATORS, CONTROL GROUPS AND POLICY ASSESSMENT. National Institute Economic Review, 257, 83-100. doi:10.1017/nie.2021.21

Examples

```
library(tsgc)
set.seed(1)
lead <- cumsum(rpois(150, 6)) + 1
targ <- cumsum(rpois(150, 8)) + 1
Y <- idx_series(cbind(lead, targ), start = 1)

# Define and estimate the model
model <- SSModelLeadingIndicator(Y = Y, n.lag = 5, q = NULL, LeadIndCol = 1,
  sea.period = 0, start = 1, end = 100)
res <- estimate(model)

# Print estimation results
res$print_estimation_results()

# Forecast 7 positions ahead from the end of the estimation window
res$predict_level(n.ahead = 7, confidence.level = 0.68)

# Forecast 7 positions ahead from the model and return filtered states
res$predict_all(n.ahead = 7, return.all = TRUE)

# Return the filtered growth rate and its components
res$get_growth_y(return.components = TRUE)

# Return smoothed growth rate of incidence variable and its confidence
# interval
res$get_gy_ci(smoothed = TRUE, confidence.level = 0.68)

# Return MAPE of forecast
res$mapes(n.ahead = 7, Y = Y)
```

---

gauteng *Cumulative daily cases of Covid-19 in the South African province of Gauteng.*

Description

Cumulative daily cases of Covid-19 in the South African province of Gauteng.

Usage

```
data(gauteng)
```

Format

An object of class "xts" with columns:

cum_cases Cumulative cases of Covid-19 from 10th March 2020

References

Downloaded from https://sacoronavirus.co.za/

Examples

```
data(gauteng)
# plot daily cases
plot(diff(gauteng))
```

---

gauteng_weather_2021 *Daily weather data of the South African province of Gauteng in 2021*

Description

Daily weather data of the South African province of Gauteng in 2021

Usage

```
data(gauteng_weather_2021)
```

Format

An object of class "xts" with columns:

windspd_mtrs_p_sec Wind speed in metres per second (m/s) at 10 metres

relhum_percnt Relative humidity at 2 metres (%)

temperature_C Daily mean air temperature in degree Celsius (C) at 2 metres

max_temp_C Daily maximum temperature in degree Celsius (C) at 2 metres

precip_mtrs Precipitation (millimetres per day; mm/day)

References

Downloaded from https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels?tab=overview and aggregated across the latitude and longitude corners for Gauteng province, South Africa by the authors.

Examples

```
data(gauteng_weather_2021)
```

gety *Extract time series y from an SSModel object*

Description

Accessor method to access the time series y in an SSModel object.

Usage

```
gety(object)
```

Arguments

object	An SSModel object.

Value

The time series y.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
tsgc:::gety(modelKFS(output(res)))
```

gety.hat *Extract prediction y.hat from predict.all output*

Description

Accessor method to access the prediction y.hat in the output of predict_all.

Usage

```
gety.hat(object)
```

Arguments

object	An object returned by predict_all.

Value

The predicted values `y.hat`.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
all_predictions <- res$predict_all(n.ahead = 7)
tsgc:::gety.hat(all_predictions)
```

`get_timeframe` *Subsetting* `idx_series` *objects given start and end positions*

Description

Helper method to subset an `idx_series` for a specified range of integer positions.

Usage

```
get_timeframe(df, start, end = NULL)
```

Arguments

`df`	An `idx_series` object, or `NULL` if no data are supplied.
`start`	Start position (integer) of the range.
`end`	End position (integer) of the range. Defaults to the last position in `df`.

Value

An `idx_series` containing the selected observations, or `NULL` if `df` is `NULL`.

Examples

```
x <- idx_series(cumsum(rpois(30, 5)) + 1, start = 1)
get_timeframe(x, 5, 10)
get_timeframe(x, 5)
```

get_V *Extract error covariance matrix of smoothed states from a KFS object*

Description

Accessor method to access the non-diffuse part of the error covariance matrix of the smoothed states in a KFS object.

Usage

```
get_V(object)
```

Arguments

object	A KFS object.

Value

The error covariance matrix of the smoothed states.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
get_V(output(res))
```

head.idx_series *First observations of an* idx_series

Description

Returns the first n observations of x, in position order. Unlike indexing x directly with [, n is a count of observations (as for utils::head), not a set of absolute positions, so head(x, 6) always returns the first six observations regardless of x$start.

Usage

```
## S3 method for class 'idx_series'
head(x, n = 6L, ...)
```

Arguments

x	An idx_series object.
n	Number of observations to return from the start of x. Default 6L. If n exceeds length(x), the full series is returned.
...	Unused.

Value

An idx_series of length min(n, length(x)).

---

idx_axis_opts *Options controlling how the x-axis of* idx_series *plots is displayed*

Description

Bundles the small set of cosmetic choices that decide how plotting functions in this file turn integer idx_series positions (and an optional idx_calendar) into an x-axis, and whether to attach a small info box summarising the calendar. Pass an idx_axis_opts object via the axis argument of the various plot.*/plot_* functions; NULL (the default everywhere) reproduces the previous behaviour exactly.

Usage

```
idx_axis_opts(
  mode = c("auto", "position", "steps", "time_since", "date"),
  info_box = FALSE,
  pattern_n = NULL
)
```

Arguments

mode	One of: • "auto" (default): "date" if a calendar with posixct = TRUE is supplied, else "position". • "position": plain integer idx_series positions. • "steps": steps from the calendar's anchor position, i.e. position - anchor_pos. Labelled "Steps from *anchor_name*" if the calendar has an anchor_name, else "Steps from anchor". • "time_since": the pattern-weighted calendar offset from the anchor (via idx_calendar_offset()), expressed in calendar$units. Labelled "Time since *anchor_name* (unit)" or "Time since anchor (unit)". Also valid when calendar$posixct is TRUE: this simply plots the numeric offset instead of a real date, which can be more readable than dates for e.g. hourly/irregular series. • "date": real calendar dates via idx_to_date() (the original behaviour). Requires a calendar.

	"steps", "time_since" and "date" require a calendar; they raise an error otherwise. "position" and "auto" work with or without one.
info_box	A single logical. If TRUE, adds a caption to the plot summarising the calendar: anchor (and its name, if set), step size/ unit, and pattern. Defaults to FALSE. Ignored if no calendar is supplied.
pattern_n	NULL (default) to show the entire pattern in the info box, or a single positive integer to truncate the displayed pattern to its first pattern_n values (with a trailing "...").

Value

An object of class idx_axis_opts.

Examples

```
idx_axis_opts(mode = "steps", info_box = TRUE)
idx_axis_opts(mode = "time_since", info_box = TRUE, pattern_n = 5)
```

---

idx_calendar *Construct an* idx_calendar

Description

idx_calendar describes how to translate the integer positions used by idx_series back into calendar time. It holds no data of its own and is not used by the statistical engine of this package (estimation, filtering, forecasting); it exists for use at the edges of the package - plotting, printing, reporting - where a human needs to see a date rather than an integer position.

An idx_calendar is defined by:

- anchor: a reference point in calendar time (e.g. a Date, or a plain number if the series has no true calendar meaning - e.g. picoseconds since some experiment start).
- anchor_pos: the integer position (in the same units as idx_series$start) that lines up with anchor.
- amount/unit: the size of a single step between consecutive integer positions, e.g. amount = 2.5, unit = "hours" or amount = 1, unit = "days". amount may be non-integer.
- pattern: a repeating vector of step multipliers, used when consecutive integer positions do not correspond to evenly spaced calendar steps. For example, business days (weekends skipped) would be pattern = c(1, 1, 1, 1, 3): four single-day steps, Mon-Thu, then a 3-day step from Fri to the following Mon. Defaults to 1, i.e. every step is a single, uniform amount.
- pattern_start: the index into pattern that anchor_pos sits at, i.e. which position in the repeating cycle the anchor falls on. Defaults to 1.

This object deliberately does no calendar arithmetic itself beyond plain numeric offsets (number of steps x amount, in units). Any interpretation of that numeric offset as a true calendar date/time (including POSIXct-specific concerns such as DST or variable month lengths) is left to the plotting/translation layer, which is expected to use idx_calendar_offset() together with the posixct flag on this object to decide how to turn the offset into an actual date.

Usage

```
idx_calendar(
  anchor,
  anchor_pos = 1L,
  amount = 1,
  unit = "days",
  pattern = 1,
  pattern_start = 1L,
  posixct = FALSE,
  anchor_name = NULL
)
```

Arguments

anchor	A single reference point in calendar time. Typically a Date or POSIXct, but may be a plain number for series with no true calendar meaning.
anchor_pos	A single integer: the idx_series position that anchor corresponds to.
amount	A single positive number: the size of one step between consecutive integer positions, in units. May be non-integer (e.g. 2.5 for "two and a half hours").
unit	A single string naming the unit that amount is measured in (e.g. "seconds", "hours", "days", "weeks", "months", "quarters", "years", or any other unit meaningful to the caller/plotting layer). This package places no constraints on the set of valid units; interpretation is left entirely to the consumer (e.g. the plotting layer's POSIXct conversion).
pattern	A numeric vector of step multipliers, recycled cyclically across consecutive integer positions. Defaults to 1 (every step is a uniform amount). For example, business days (weekends skipped) would use c(1, 1, 1, 1, 3).
pattern_start	A single positive integer: the index into pattern at which anchor_pos falls. Defaults to 1.
posixct	A single logical. If TRUE, hints to the plotting/ translation layer that anchor should be treated as (or coerced to) a POSIXct/Date, and that calendar-aware arithmetic (e.g. seq.POSIXt, month/quarter-aware stepping) should be used to interpret offsets rather than plain numeric addition. Defaults to FALSE. This object does not act on the flag itself; it only stores it for the translation layer to consult.
anchor_name	An optional single string giving a human-readable name for anchor (e.g. "outbreak start", "launch"). When supplied, the plotting layer can use it to label axes/info boxes as e.g. "steps from launch" or "time since launch" instead of the generic "steps from anchor". Defaults to NULL (no name; generic wording is used).

Value

An object of class idx_calendar.

Examples

```
# Daily series, anchored so that position 1 is 2024-01-01.
cal <- idx_calendar(anchor = as.Date("2024-01-01"), anchor_pos = 1L,
                    amount = 1, unit = "days", posixct = TRUE)
idx_to_date(cal, 1:5)

# Business-day series (weekends skipped), anchor at a Monday.
cal_bd <- idx_calendar(anchor = as.Date("2024-01-01"), anchor_pos = 1L,
                       amount = 1, unit = "days",
                       pattern = c(1, 1, 1, 1, 3), pattern_start = 1L,
                       posixct = TRUE)
idx_to_date(cal_bd, 1:10)

# Non-calendar series: picosecond timesteps, arbitrary numeric anchor.
cal_ps <- idx_calendar(anchor = 0, anchor_pos = 1L,
                       amount = 2.5, unit = "picoseconds")
idx_to_date(cal_ps, 1:5)
```

---

idx_calendar_multi_step

Construct an idx_calendar *from a* multi_step_pattern

Description

Third, most general constructor for idx_calendar. Use this when consecutive positions cycle through genuinely different kinds of steps - not just different multiples of one step (that case is covered by the numeric pattern argument of idx_calendar/ idx_calendar_step). For example, "3 days, 3 days, then 1 month, repeating" needs a multi_step_pattern since two slots are fixed-duration and one is calendar-relative.

Unlike the numeric pattern mechanism (which pre-computes a closed-form pattern-weighted step count, see idx_calendar_offset), a multi_step_pattern is applied by walking one slot at a time from anchor_pos to the target position, via idx_step_add. This is required because heterogeneous steps (e.g. days mixed with months) cannot be collapsed into a single scalar multiplier the way a uniform-unit pattern can - so this constructor's calendars are O(distance from anchor) to convert per position, rather than O(1).

Usage

```
idx_calendar_multi_step(
  anchor,
  multi_step,
  anchor_pos = 1L,
  pattern_start = 1L,
  posixct = FALSE,
  anchor_name = NULL
)
```

Arguments

anchor	A single reference point in calendar time (see idx_calendar).
multi_step	A multi_step_pattern object: the repeating cycle of idx_steps.
anchor_pos, posixct, anchor_name	As in idx_calendar.
pattern_start	A single positive integer: the index into multi_step at which anchor_pos falls (the slot for the step *out of* anchor_pos). Defaults to 1.

Value

An object of class idx_calendar.

Examples

```
# 3 days, 3 days, then 1 month, repeating.
cal <- idx_calendar_multi_step(
  anchor = as.Date("2024-01-01"),
  multi_step = multi_step_pattern(idx_step(days = 3), idx_step(days = 3),
                                  idx_step(months = 1)),
  posixct = TRUE
)
idx_to_date(cal, 1:5)
```

---

idx_calendar_offset *Cumulative pattern-weighted step offset between two positions*

Description

Computes, for one or more integer idx_series positions, the number of base amount-sized steps that separate each position from cal$anchor_pos, accounting for the repeating pattern of step multipliers. This is a pure integer/numeric computation - no calendar arithmetic and no units are applied here; the result is in multiples of cal$amount, positive for positions after the anchor and negative for positions before it. It is intended to be consumed by idx_to_date (or directly by the plotting layer's own POSIXct-aware conversion).

Usage

```
idx_calendar_offset(cal, pos)
```

Arguments

cal	An idx_calendar object.
pos	An integer vector of idx_series positions.

Value

A numeric vector, the same length as `pos`, giving the pattern-weighted step offset (in units of `cal$amount`) of each position relative to `cal$anchor_pos`.

`idx_calendar_step` *Construct an* `idx_calendar` *from a compound* `idx_step`

Description

Secondary constructor for `idx_calendar`, taking a compound `idx_step` (a combination of years, quarters, months, weeks, days, hours, minutes, seconds) as the step size instead of a single `amount`/`unit` pair. Use this when a single unit cannot express the step - e.g. "3 seconds after each quarter" or "a month and 13 seconds". For a plain single-unit step, the primary `idx_calendar` constructor is simpler.

Usage

```
idx_calendar_step(
  anchor,
  step,
  anchor_pos = 1L,
  pattern = 1,
  pattern_start = 1L,
  posixct = FALSE,
  anchor_name = NULL
)
```

Arguments

`anchor`	A single reference point in calendar time (see `idx_calendar`).
`step`	An `idx_step` object giving the size of one step between consecutive integer positions.
`anchor_pos`, `pattern`, `pattern_start`, `posixct`, `anchor_name`	As in `idx_calendar`.

Value

An object of class `idx_calendar`.

Examples

```
# Data recorded 3 seconds after each quarter boundary.
cal <- idx_calendar_step(
  anchor = as.POSIXct("2024-01-01 00:00:03", tz = "UTC"),
  step = idx_step(quarters = 1, seconds = 3),
  posixct = TRUE
)
idx_to_date(cal, 1:4)
```

idx_cbind *Combine (column-bind)* idx_series *objects*

Description

Column-binds one or more idx_series objects that share the same start and length, analogous to cbind for matrices/xts objects.

Usage

```
idx_cbind(...)
```

Arguments

...	idx_series objects to combine.

Value

A single idx_series with combined columns.

idx_diff *Take a lagged difference of an* idx_series

Description

Index-based analogue of diff() for calendar series: returns $x_t - x_{t-lag}$ for each position t, dropping the first lag positions (which have no valid lag within the series).

Usage

```
idx_diff(x, lag = 1L)
```

Arguments

x	An idx_series object.
lag	Integer number of positions to lag by. Default 1.

Value

An idx_series, shorter than x by lag.

idx_lag *Lag an* idx_series

Description

Index-based analogue of stats::lag(): shifts the series forward by k positions. The returned series still covers length(x) observations, but its start is shifted by k, i.e. position t of the result holds the value that was at position $t - k$ in x.

Usage

```
idx_lag(x, k = 1L)
```

Arguments

x	An idx_series object.
k	Integer number of positions to lag by. Default 1.

Value

An idx_series with the same length as x.

idx_ncol *Number of columns in an* idx_series

Description

base::NCOL() is not an S3 generic, so it will not dispatch on idx_series objects. Use this function instead when you need the number of columns/variables in an idx_series.

Usage

```
idx_ncol(x)
```

Arguments

x	An idx_series object.

idx_offset_to_pos *Translate a plain numeric offset to an* idx_series *integer position*

Description

Inverse of idx_to_date's plain-arithmetic branch (posixct = FALSE, or a non-Date/POSIXct cal$anchor): converts a value already expressed in cal$anchor's own units/scale (e.g. a picosecond count) into the integer idx_series position that corresponds to it.

Only relevant when you actually have an idx_calendar with a non-Date/POSIXct anchor (e.g. a numeric anchor for sub-second or otherwise non-calendar timesteps) and want to look up a position from a value in that anchor's scale. If your series has no calendar meaning at all - no dates, no anchor, nothing to translate - you don't need an idx_calendar or this function: idx_series positions (start/end arguments throughout this package) are already plain integers, so just use them directly (see idx_range).

Usage

```
idx_offset_to_pos(cal, value)
```

Arguments

cal	An idx_calendar object.
value	A single number, in the same units as cal$anchor (i.e. cal$anchor + n * cal$amount for some integer n), or a vector of such.

Value

An integer (vector, matching value): the idx_series position(s) corresponding to value.

Examples

```
cal <- idx_calendar(anchor = 0, anchor_pos = 1L, amount = 2.5,
                    unit = "picoseconds")
idx_offset_to_pos(cal, 12.5)
```

idx_positions *Integer positions of an* idx_series

Description

Returns the full vector of integer positions associated with each observation in x. This is the index-based analogue of zoo::index().

Usage

```
idx_positions(x)
```

Arguments

x	An idx_series object.

Value

An integer vector.

---

idx_range *Integer position range spanned by an* idx_series

Description

Returns the first and last integer position covered by x, i.e. c(x$start, x$start + length(x) - 1).

Usage

```
idx_range(x)
```

Arguments

x	An idx_series object.

Value

An integer vector of length 2: c(first, last).

---

idx_rbind *Bind two* idx_series *sequentially*

Description

Concatenates x then y in position order. y must start exactly one position after x ends.

Usage

```
idx_rbind(x, y)
```

Arguments

x, y	idx_series objects to concatenate.

Value

A single, combined idx_series.

idx_series *Index-anchored series*

Description

idx_series is the core data structure used throughout the analysis parts of this package in place of calendar-indexed objects (such as xts). A series is stored as a plain numeric vector or matrix of observations together with a single integer start, which is the integer position (relative to some external anchor point, e.g. the first observation ever collected) of the first row of data. All internal computation - estimation, filtering, forecasting, reinitialisation - is done purely in terms of these integer positions. This keeps the statistical engine of the package completely free of calendar/time-of-day concerns.

Calendar time is reintroduced only at the very edges of the package (e.g. plotting, printing to the user) by a separate, cosmetic translation layer that maps integer positions back to dates given an anchor date. That translation layer is out of scope here.

Usage

```
idx_series(data, start = 1L)
```

Arguments

data	A numeric vector or matrix of observations. For a matrix, rows are observations (in position order) and columns are variables.
start	A single positive integer giving the position of the first observation in data. Defaults to 1.

Value

An object of class idx_series.

Examples

```
x <- idx_series(cumsum(rpois(50, 5)) + 1, start = 1)
length(x)
idx_range(x)
x[10:20]
```

idx_step *A compound calendar step*

Description

idx_step describes the size of a single step between consecutive idx_series integer positions as a combination of calendar-relative and fixed-duration components: years, quarters, months, weeks, days, hours, minutes, and seconds. Each component is independent - they are not normalized or collapsed into one another (e.g. idx_step(months = 1, days = 31) is not simplified to two months; it is applied as "one month, then 31 days"). This lets a step express things a single amount/unit pair cannot, such as "3 seconds after each quarter" (idx_step(quarters = 1, seconds = 3)) or "a month and 13 seconds" (idx_step(months = 1, seconds = 13)).

When applied (see idx_step_add), components are added to an anchor date in a fixed order, from largest calendar-relative unit to smallest fixed-duration unit: years, then quarters, then months (each calendar-aware, i.e. respecting variable month/quarter length), then weeks, days, hours, minutes, seconds (each fixed-duration). This order is applied once per full step multiple - i.e. for n steps from the anchor, every component is scaled by n before being added, not added once and then repeated.

A step with any of hours, minutes, or seconds non-zero requires a POSIXct anchor (not a plain Date or yearqtr/yearmon), since those units need a time-of-day component to be meaningful.

Usage

```
idx_step(
  years = 0,
  quarters = 0,
  months = 0,
  weeks = 0,
  days = 0,
  hours = 0,
  minutes = 0,
  seconds = 0
)
```

Arguments

years, quarters, months, weeks, days, hours, minutes, seconds
: Single non-negative, finite numbers giving the count of each component in one step. All default to 0. At least one must be non-zero.

Value

An object of class idx_step.

Examples

```
# A single quarter, nudged 3 seconds later - e.g. data recorded 3
# seconds after each quarter boundary.
```

```
idx_step(quarters = 1, seconds = 3)

# A month and 13 seconds - the extra seconds accumulate with each step,
# same as the calendar-relative part.
idx_step(months = 1, seconds = 13)

# Equivalent to amount = 1, unit = "days".
idx_step(days = 1)
```

`idx_step_add` *Apply an* `idx_step` *to an anchor date, scaled by a step count*

Description

Adds `n` whole steps of `step` to `anchor`, one component at a time, from largest calendar-relative unit to smallest fixed-duration unit (years, quarters, months, weeks, days, hours, minutes, seconds), each scaled by `n` before being added. Calendar-relative components (years/quarters/months) use calendar-aware stepping (`seq.Date`/`seq.POSIXt`, or the equivalent fractional-year arithmetic for `yearqtr`/`yearmon` anchors); fixed-duration components (weeks/days/hours/minutes/seconds) use plain duration addition.

Usage

```
idx_step_add(anchor, step, n)
```

Arguments

`anchor`	A `Date`, `POSIXct`, `yearqtr`, or `yearmon` to step from.
`step`	An `idx_step` object.
`n`	An integer (or vector of integers): the number of whole steps to apply. May be negative.

Value

A vector the same length as `n`, the same class as `anchor` (or `POSIXct`, if a sub-day component forces promotion from `Date`).

idx_to_date *Translate* idx_series *positions to calendar time*

Description

Converts one or more integer idx_series positions into calendar time using an idx_calendar.

When cal$posixct is TRUE and cal$anchor is a Date/POSIXct, and cal$unit is one of the standard calendar units - "seconds", "minutes", "hours", "days", "weeks", "months", "quarters", "years" (singular or plural) - calendar-aware stepping is used via seq.Date/seq.POSIXt's by argument. This is exact for every one of these units: seconds through weeks are fixed-duration and so plain offset arithmetic would already agree with it, but months/quarters/years are calendar-relative (variable length - e.g. not every month has the same number of days) and genuinely require this calendar-aware stepping to land on the correct date (respecting month-end/DST rules) rather than a fixed number of days.

cal$anchor may also be a zoo yearqtr or yearmon, for series that are naturally indexed at quarterly or monthly resolution rather than daily.

Internally, every idx_calendar (whether built via the primary amount/unit constructor or via idx_calendar_step) carries a compound cal$step (see idx_step), and calendar-aware stepping is delegated uniformly to idx_step_add. This is what allows steps that mix calendar-relative and fixed-duration components, e.g. "3 seconds after each quarter" or "a month and 13 seconds" - a single amount/unit pair cannot express these, but a compound idx_step can.

Otherwise (posixct = FALSE, or a non-Date/POSIXct/yearqtr/yearmon anchor) the conversion falls back to plain numeric offset arithmetic: it multiplies the pattern-weighted step offset (see idx_calendar_offset) by cal$amount and adds the result to cal$anchor. This supports arbitrary non-calendar units (e.g. picoseconds) uniformly, but is not available for calendars built via idx_calendar_step with posixct = FALSE, since those have no single amount to multiply by (see idx_calendar_step); such calendars require posixct = TRUE.

Usage

```
idx_to_date(cal, pos)
```

Arguments

cal	An idx_calendar object.
pos	An integer vector of idx_series positions.

Value

A vector, the same length as pos, giving the calendar time (or plain number, for non-calendar cal$anchor) for each position.

idx_to_pos *Translate a calendar date to an* idx_series *integer position*

Description

Inverse of idx_to_date: converts a calendar date into the integer position (relative to cal) that corresponds to it. Intended for locating estimation start/end dates, or for aligning a second series to an already-converted idx_series (see xts_to_idx). Requires a calendar-anchored cal (cal$posixct = TRUE and a Date/POSIXct/ yearqtr/yearmon cal$anchor); use idx_offset_to_pos instead for non-calendar (e.g. sub-second/arbitrary numeric) calendars.

cal may be built via any of the three idx_calendar constructors:

- The primary idx_calendar constructor (a single amount/unit pair). For the calendar-relative units "months", "quarters", "years", the position is found by counting whole calendar steps between cal$anchor and date (the inverse of the calendar-aware stepping used by idx_to_date), rather than via a fixed-length difference - a quarter is not a fixed number of days. For the fixed-duration units "seconds", "minutes", "hours", "days", "weeks", positions are computed as a plain time difference scaled by cal$amount; this is exact for these units. A repeating numeric pattern is respected. Because patterned offsets are piecewise rather than a single amount that can be divided in closed form, the inverse is found by an exact binary search over integer positions. Dates that do not land on a whole step boundary are rejected rather than rounded.
- idx_calendar_step (a compound idx_step) and idx_calendar_multi_step (a multi_step_pattern) calendars have no single amount/unit pair to invert in closed form in general, so date is instead located by search: idx_calendar_step calendars repeat one compound idx_step every position, and idx_step_add is strictly monotonic in the step count, so this uses an O(log distance) binary search; idx_calendar_multi_step calendars cycle through heterogeneous steps that cannot be searched this way, so this walks one step at a time from cal$anchor_pos instead, an O(distance) linear walk. Both are exact, and both ignore any numeric pattern (a multi_step_pattern calendar has no numeric pattern to ignore in the first place - it uses pattern_start only as the multi_step slot index).

A yearqtr/yearmon cal$anchor is only valid together with cal$unit of "quarters"/"months" respectively (matching idx_step_add's restriction that these anchors have no sub-quarter/sub-month resolution); date is coerced to the same class as cal$anchor and whole-step boundaries are found via fractional-year arithmetic, consistent with idx_to_date.

Usage

```
idx_to_pos(cal, date)
```

Arguments

cal	An idx_calendar object with posixct = TRUE and a Date/POSIXct/yearqtr/yearmon anchor.

date — A single date/time. For cal$unit of "months", "quarters", or "years", coerced via as.Date (or to yearqtr/yearmon, matching cal$anchor). For all other units, coerced via as.POSIXct if cal$anchor is POSIXct, or as.Date otherwise - so a character string with a time-of-day component (e.g. "2024-01-01 13:30:00") is preserved where relevant (sub-day units).

Value

A single integer: the idx_series position corresponding to date.

Examples

```
cal <- idx_calendar(anchor = as.Date("2024-01-01"), anchor_pos = 1L,
                    amount = 1, unit = "days", posixct = TRUE)
idx_to_pos(cal, "2024-01-10")

# A multi_step_pattern calendar (walked, not closed-form).
cal_ms <- idx_calendar_multi_step(
  anchor = as.Date("2024-01-01"),
  multi_step = multi_step_pattern(idx_step(days = 3), idx_step(days = 3),
                                  idx_step(months = 1)),
  posixct = TRUE
)
idx_to_pos(cal_ms, "2024-02-07")
```

---

idx_values *Explicit numeric extraction for* idx_series

Description

Portable, always-dispatches accessor for the raw numeric data underlying an idx_series. Prefer this over as.numeric() in package-internal code, since it does not depend on S3 dispatch of a primitive generic.

Usage

```
idx_values(x)
```

Arguments

x — An idx_series object.

Value

A plain numeric vector (or matrix, if x wraps a matrix).

is_idx_calendar *Test whether an object is an* idx_calendar

Description

Test whether an object is an idx_calendar

Usage

```
is_idx_calendar(x)
```

Arguments

x	Object to test.

Value

Logical.

is_idx_series *Test whether an object is an* idx_series

Description

Test whether an object is an idx_series

Usage

```
is_idx_series(x)
```

Arguments

x	Object to test.

Value

Logical.

is_idx_step *Test whether an object is an* idx_step

Description

Test whether an object is an idx_step

Usage

```
is_idx_step(x)
```

Arguments

x	Object to test.

Value

Logical.

is_multi_step_pattern *Test whether an object is a* multi_step_pattern

Description

Test whether an object is a multi_step_pattern

Usage

```
is_multi_step_pattern(x)
```

Arguments

x	Object to test.

Value

Logical.

length.idx_series *Number of observations in an* idx_series

Description

Number of observations in an idx_series

Usage

```
## S3 method for class 'idx_series'
length(x, ...)
```

Arguments

x	An idx_series object.
...	Unused.

mapes *Compute Mean Absolute Percentage Error (MAPE) for Forecasts Against a Holdout Sample*

Description

This is a helper function that calculates five error metrics of a forecast generated by time series growth curve (tsgc) models. It compares the forecasted values to a holdout sample, providing a measure of forecast accuracy.

Usage

```
mapes(res, n.ahead, Y)
```

Arguments

res	A FilterResults or FilterResultsLI object, obtained from estimate() method.
n.ahead	Integer specifying the number of periods to forecast ahead.
Y	An idx_series object containing the original cumulative dataset.

Value

A list containing five error metrics for the forecast, with element names

- mape: mean absolute percentage error
- smape: symmetric mean absolute percentage error (between 0 to 100)
- mae: mean absolute error
- rmse: root mean squared error
- coverage: Percentage of holdout sample data points that lie inside the confidence interval for predictions

matrixKFS *Extract matrices of the observation, state, and disturbance equations from a KFS object*

Description

Accessor method to access the matrices used in the observation, state, and disturbance equations of a KFS object.

Usage

```
matrixKFS(object, matrix)
```

Arguments

object	A KFS object.
matrix	Character string naming a matrix component of SSModel, e.g. "H", "T", "R", "Q".

Value

The requested matrix.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
tsgc:::matrixKFS(output(res), "Z")
```

modelKFS *Extract SSModel object within a KFS object*

Description

Accessor method to access the fitted SSModel.

Usage

```
modelKFS(object)
```

Arguments

object	A KFS object.

Value

The SSModel object underlying the KFS fit.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
modelKFS(output(res))
```

---

multi_step_pattern *A repeating cycle of distinct* idx_steps

Description

multi_step_pattern describes a repeating cycle where each slot can be a completely different idx_step - not just a different multiple of one fixed step. This is for cases the numeric pattern argument of idx_calendar/ idx_calendar_step cannot express, because that mechanism only ever scales a single amount/unit (or a single compound idx_step) by different multipliers around the cycle - it cannot mix step *kinds*. For example, "3 days, 3 days, then 1 month, repeating" needs two of the three slots to be fixed-duration and one to be calendar-relative; multi_step_pattern lets each slot carry its own idx_step: multi_step_pattern(idx_step(days = 3), idx_step(days = 3), idx_step(months = 1)).

Use this via idx_calendar_multi_step (a third, most-general constructor for idx_calendar, alongside the primary amount/unit constructor and the idx_calendar_step compound-step constructor). The existing numeric pattern mechanism is unchanged and remains the simpler choice whenever every slot in the cycle is just a different multiple of the same single step.

Usage

```
multi_step_pattern(...)
```

Arguments

...	One or more idx_step objects, given in the order they repeat.

Value

An object of class multi_step_pattern: a list of idx_step objects.

Examples

```
# 3 days, 3 days, then 1 month - repeating.
multi_step_pattern(idx_step(days = 3), idx_step(days = 3), idx_step(months = 1))
```

`NCOL.idx_series` *Number of columns in an* `idx_series` *(*`NCOL` *method)*

Description

Provided for convenience when calling `NCOL()` directly on an `idx_series`; note that since `base::NCOL` is not an S3 generic, this method is only reached if `NCOL` is called as `NCOL.idx_series()` directly, or after `NCOL` has been made generic (e.g. via `Matrix` or similar). Prefer `idx_ncol` for portable code within this package.

Usage

```
NCOL.idx_series(x)
```

Arguments

`x`	An `idx_series` object.

`nintendo_sales` *Quarterly Sales of Nintendo consoles from 2004 Q4 to 2024 Q1*

Description

Quarterly Sales of Nintendo consoles from 2004 Q4 to 2024 Q1

Usage

```
data(nintendo_sales)
```

Format

An object of class `"xts"` with columns:

wii Cumulative Wii sales

wiiu Cumulative Wii U sales

switch_all Cumulative Switch sales (original, lite and OLED combined)

switch_base Cumulative original Switch sales

switch_Lite Cumulative Switch Lite sales

switch_OLED Cumulative switch OLED sales

3ds Cumulative 3DS sales

nds Cumulative NDS sales

References

Taken from https://vgsales.fandom.com/wiki/Wii

Examples

```
data(nintendo_sales)
```

`output` *Extract output of FilterResults or FilterResultsLI*

Description

Accessor method to access the fitted KFS model from `FilterResults` or `FilterResultsLI`.

Usage

```
output(object)
```

Arguments

`object`	A `FilterResults` or `FilterResultsLI` object.

Value

The fitted KFS model.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
output(res)
```

`plot.SSModelDynamicGompertz` *Plot the cumulated dataset underlying a fitted Gompertz model*

Description

Plots the lagged differences of the cumulated dataset `Y` stored in a `SSModelDynamicGompertz` object, optionally overlaid with a centred moving average.

Usage

```
## S3 method for class 'SSModelDynamicGompertz'
plot(
  x,
  title = NULL,
  series.name = "target variable",
  MA_period = 7,
  axis = NULL,
  ...
)
```

Arguments

`x`	A `SSModelDynamicGompertz` object.
`title`	Title for the plot. `NULL` (default) for no title.
`series.name`	Name of the series being plotted, used in the y-axis label. Default is `"target variable"`.
`MA_period`	Number of positions in the centred moving average to overlay. `0` or `1` disables the moving average. Default `7`.
`axis`	An `idx_axis_opts` object controlling x-axis mode and info box, or `NULL` (default) for the previous default behaviour (dates if a POSIXct calendar is present, else positions).
`...`	Unused.

Value

A `ggplot2` plot.

plot.SSModelLeadingIndicator *Plot the leading indicator and target variable underlying a fitted Leading Indicator model*

Description

Plots the daily incidence (new cases) of the leading indicator and target variable stored in a SSModelLeadingIndicator object, optionally on a log scale.

Usage

```
## S3 method for class 'SSModelLeadingIndicator'
plot(
  x,
  title = NULL,
  series.name.lead = "Leading Indicator",
  series.name.target = "Target Variable",
  take.log = TRUE,
  axis = NULL,
  ...
)
```

Arguments

x	A SSModelLeadingIndicator object.
title	Title for the plot. NULL (default) for no title.
series.name.lead	Name of the leading indicator series, used in the legend. Default "Leading Indicator".
series.name.target	Name of the target variable series, used in the legend. Default "Target Variable".
take.log	Logical value indicating whether to plot the log of the daily incidence. Default TRUE.
axis	An idx_axis_opts object controlling x-axis mode and info box, or NULL (default) for the previous default behaviour.
...	Unused.

Value

A ggplot2 plot.

`plot_compare_forecast` *Forecast comparison plot*

Description

Plots forecasts from a list of fitted models on the same axes for visual comparison.

Usage

```
plot_compare_forecast(
  results,
  n.ahead = 14,
  sea.on = TRUE,
  actual = NULL,
  title = "Comparison of forecasts",
  labels = NULL,
  axis = NULL
)
```

Arguments

`results`	A list of `FilterResults` or `FilterResultsLI` objects, obtained from the `estimate()` method.
`n.ahead`	Duration of the holdout sample. Default is 14.
`sea.on`	Logical value indicating whether to plot the seasonality-adjusted forecasts. Defaults to `TRUE`.
`actual`	An `idx_series` of actual cumulative values. The function automatically extracts the observations corresponding to the prediction period, so the complete series may be provided.
`title`	Title for the plot. Defaults to `"Comparison of forecasts"`.
`labels`	Character vector of legend labels, one per element of `results`, or `NULL` (default). If `NULL`, labels are taken from `names(results)` if set (e.g. `list(a = res1, b = res2)`), and otherwise default to the deparsed expression used for each element in the `results` list call (e.g. `plot_compare_forecast(list(res, res.reinit))` labels the two lines `"res"` and `"res.reinit"`).
`axis`	An `idx_axis_opts` object controlling x-axis mode and info box, or `NULL` (default) for the previous default behaviour. Uses the calendar of `results[[1]]`.

Value

A `ggplot2` plot.

`plot_forecast` *Plots the forecast of new cases (the difference of the cumulated variable)*

Description

Plots actual values of the difference in the cumulated variable, the forecast of the cumulated variable (including seasonal components, where specified) and forecast intervals around the forecast. The forecast intervals are based on the prediction intervals for $\ln(g_t)$.

Usage

```
plot_forecast(
  res,
  n.ahead = 14,
  confidence.level = 0.68,
  title = NULL,
  plt.start = NULL,
  series.name = "target variable",
  axis = NULL
)
```

Arguments

`res`	A `FilterResults` or `FilterResultsLI` object, obtained from the `estimate()` method.
`n.ahead`	Number of forecasts (i.e. number of positions ahead to forecast from the end of the estimation window). Default is 14.
`confidence.level`	Width of the prediction interval for $\ln g_t$, used in forecasts of $y_t = \Delta Y_t$. Default is 0.68, approximately one standard deviation for a Normal distribution.
`title`	Title for the plot. `NULL` (default) for no title.
`plt.start`	First integer position of actual data (from the estimation sample) to plot. `NULL` (default) plots all data in the estimation window.
`series.name`	Name of the series the growth rate is being computed for, e.g. `'cases'`. Default `"target variable"`.
`axis`	An `idx_axis_opts` object controlling x-axis mode and info box, or `NULL` (default) for the previous default behaviour.

Value

A `ggplot2` plot.

Examples

```
library(tsgc)
set.seed(1)
Y <- idx_series(cumsum(rpois(120, 8)) + 1, start = 1)
model <- SSModelDynamicGompertz$new(Y = Y, q = 0.005, end = 100)
res <- estimate(model)
plot_forecast(res, n.ahead = 7, series.name = "cases")
```

`plot_gy_ci` *Plots the growth rate of the cumulative variable with confidence intervals*

Description

Plots the smoothed/filtered growth rate of the difference in the cumulated variable (g_y) and the associated confidence intervals.

Usage

```
plot_gy_ci(
  res,
  plt.start = NULL,
  smoothed = FALSE,
  title = NULL,
  series.name = NULL,
  pad.right = NULL,
  axis = NULL
)
```

Arguments

`res`	A `FilterResults` or `FilterResultsLI` object, obtained from the `estimate()` method.
`plt.start`	Plot start position. `NULL` (default) is the start of the estimation sample.
`smoothed`	Logical value indicating whether to use the smoothed estimates. Default `FALSE` (filtered estimates).
`title`	Title for the plot. `NULL` (default) for no title.
`series.name`	Name of the series the growth rate is being computed for, e.g. `'New cases'`.
`pad.right`	Numerical value for the number of positions of blank space to leave on the right of the graph.
`axis`	An `idx_axis_opts` object controlling x-axis mode and info box, or `NULL` (default) for the previous default behaviour.

Value

A `ggplot2` plot.

plot_gy_components *Plots the growth rate, level and slope of the log cumulative growth rate*

Description

Plots the smoothed/filtered growth rate of the difference in the cumulated variable (g_y), the smoothed/filtered growth rate of the cumulated variable (g), and the smoothed/filtered slope of $\ln(g)$, γ. Following Harvey and Kattuman (2021), we compute $g_{y,t}$ as

$$g_{y,t} = \exp(\delta_t) + \gamma_t.$$

Usage

```
plot_gy_components(
  res,
  plt.start = NULL,
  smoothed = FALSE,
  title = NULL,
  axis = NULL
)
```

Arguments

res	A FilterResults or FilterResultsLI object, obtained from the estimate() method.
plt.start	Plot start position. NULL (default) is the start of the estimation sample.
smoothed	Logical value indicating whether to use the smoothed estimates of δ and γ. Default FALSE (filtered estimates).
title	Title for the plot. NULL (default) for no title.
axis	An idx_axis_opts object controlling x-axis mode and info box, or NULL (default) for the previous default behaviour.

Value

A ggplot2 plot.

plot_holdout *Plots the forecast of new cases over a holdout sample*

Description

Plots actual values of the difference in the cumulated variable, the forecast of the cumulated variable (including seasonal components, where specified) and forecast intervals around the forecast, plus the actual outcomes from the holdout sample. Also reports the mean absolute percentage error over the holdout sample.

Usage

```
plot_holdout(
  res,
  Y,
  n.ahead = 14,
  confidence.level = 0.68,
  series.name = "target variable",
  title = NULL,
  caption = NULL,
  axis = NULL
)
```

Arguments

res	A FilterResults or FilterResultsLI object, obtained from the estimate() method.
Y	Values of the cumulated variable, including the holdout sample.
n.ahead	Duration of the holdout sample. Default is 14.
confidence.level	Width of the prediction interval for $\ln(g_t)$. Default 0.68.
series.name	Name of the variable being forecast, for the y-axis label.
title	Title for the plot. NULL (default) for no title.
caption	Caption for the plot. NULL (default) for no caption.
axis	An idx_axis_opts object controlling x-axis mode and info box, or NULL (default) for the previous default behaviour.

Value

A ggplot2 plot.

---

plot_log_forecast *Plots forecast and realised values of the log cumulative growth rate*

Description

Plots actual and filtered values of the log cumulative growth rate ($\ln(g_t)$) in the estimation sample and the forecast and realised log cumulative growth rate out of the estimation sample.

Usage

```
plot_log_forecast(
  res,
  Y,
  n.ahead = 14,
  plt.start = NULL,
```

```
  title = "",
  caption = "",
  axis = NULL
)
```

Arguments

res	A FilterResults or FilterResultsLI object, obtained from the estimate() method.
Y	Cumulated dataset (idx_series) containing future values.
n.ahead	Number of positions ahead from the end of the sample to forecast. Default is 14.
plt.start	Plot start position. NULL (default) is the start of the estimation sample.
title	Plot title.
caption	Plot caption.
axis	An idx_axis_opts object controlling x-axis mode and info box, or NULL (default) for the previous default behaviour. If info_box = TRUE, the info box is appended below any caption already supplied.

Value

A ggplot2 plot.

---

plot_r0 *Plot the estimated reproduction number R_t*

Description

Plots R_t, as computed by estimate_r0, together with its confidence interval, over the most recent n.ahead integer positions of res. A horizontal reference line is drawn at $R_t = 1$, the threshold separating a growing from a shrinking epidemic/process.

Usage

```
plot_r0(
  res,
  gen_int,
  n.ahead = 7,
  smoothed = FALSE,
  confidence.level = 0.68,
  title = NULL,
  axis = NULL
)
```

Arguments

res	A FilterResults or FilterResultsLI object, obtained from the estimate() method.
gen_int	The mean generation interval, in the same integer-position units as res (e.g. if positions are days, this is the mean generation interval in days). Passed through to estimate_r0().
n.ahead	Number of most recent integer positions to plot, taken from the end of res's filtered/smoothed range. Default 7. Passed through to estimate_r0().
smoothed	Logical value indicating whether to use the smoothed (TRUE) or filtered (FALSE, default) growth rate estimates underlying R_t. Passed through to estimate_r0().
confidence.level	Confidence level for the confidence interval around R_t. Default 0.68. Passed through to estimate_r0().
title	Title for the plot. NULL (default) for no title.
axis	An idx_axis_opts object controlling x-axis mode and info box, or NULL (default) for the previous default behaviour (dates if a POSIXct calendar is present, else positions).

Value

A ggplot2 plot.

Examples

```
library(tsgc)
set.seed(1)
Y <- idx_series(cumsum(rpois(120, 8)) + 1, start = 1)
model <- SSModelDynamicGompertz$new(Y = Y, q = NULL, end = 100)
res <- estimate(model)
plot_r0(res, gen_int = 5, n.ahead = 7)
```

---

print.FilterResults *Calling print method for FilterResults class*

Description

Accessor method to print a short description for the objects of FilterResults class

Usage

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

Arguments

x	A FilterResults object
...	Additional arguments.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
# Specify a model
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
# Estimate a specified model
res <- estimate(model)

# Return short description of fitted model
print(res)
```

---

print.FilterResultsLI *Calling print method for FilterResultsLI class*

Description

Accessor method to print a short description for the objects of FilterResultsLI class

Usage

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

Arguments

x	A FilterResultsLI object
...	Additional arguments.

Examples

```
library(tsgc)
data(england, package = "tsgc")
conv <- xts_to_idx(england[, 1:2])

out_eng <- SSModelLeadingIndicator(
  Y = conv$series, n.lag = 4, sea.period = 7, LeadIndCol = 1,
  calendar = conv$calendar,
  start = idx_to_pos(conv$calendar, "2021-04-30"),
  end = idx_to_pos(conv$calendar, "2021-07-24"))
```

```
res_eng<-estimate(out_eng)
print(res_eng)
```

`print.idx_calendar` *Print an* `idx_calendar`

Description

Print an `idx_calendar`

Usage

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

Arguments

`x`	An `idx_calendar` object.
`...`	Unused.

`print.idx_series` *Print an* `idx_series`

Description

Print an `idx_series`

Usage

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

Arguments

`x`	An `idx_series` object.
`...`	Unused.

print.idx_step *Print an* idx_step

Description

Print an idx_step

Usage

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

Arguments

x	An idx_step object.
...	Unused.

print.multi_step_pattern *Print a* multi_step_pattern

Description

Print a multi_step_pattern

Usage

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

Arguments

x	A multi_step_pattern object.
...	Unused.

print.SSModelDynamicGompertz

Calling print method for SSModelDynamicGompertz class

Description

Accessor method to print a short description for the objects of `SSModelDynamicGompertz` class

Usage

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

Arguments

`x`	A `SSModelDynamicGompertz` object
`...`	Additional arguments.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)

# Specify a model
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-06"))

# Print a short description of the model object
print(model)
```

print.SSModelLeadingIndicator

Calling print method for classes in tsgc

Description

Accessor method to print a short description for the objects of `SSModelLeadingIndicator` class

Usage

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

Arguments

x	A SSModelLeadingIndicator object
...	Additional arguments.

Examples

```
library(tsgc)
data(england, package = "tsgc")
conv <- xts_to_idx(england[, 1:2])

# Specify a model
out_eng <- SSModelLeadingIndicator(
  Y = conv$series, n.lag = 4, sea.period = 7, LeadIndCol = 1,
  calendar = conv$calendar,
  start = idx_to_pos(conv$calendar, "2021-04-30"),
  end = idx_to_pos(conv$calendar, "2021-07-24"))

# Print a short description of the model object
print(out_eng)
```

---

print_model_diagnostics

Report fitted diagnostics for a FilterResults/FilterResultsLI object

---

Description

summary() on FilterResults/ FilterResultsLI reports estimated variance parameters and model states, but does not report the model's log-likelihood or residual diagnostics. print_model_diagnostics() reports these two additional, complementary diagnostics; it does not replace or duplicate what summary() already prints.

Usage

```
print_model_diagnostics(res)
```

Arguments

res	A FilterResults or FilterResultsLI object.

Value

NULL, invisibly. Called for its printed output.

Ptt *Extract error covariance matrix of filtered states from a KFS object*

Description

Accessor method to access the non-diffuse part of the error covariance matrix of the filtered states in a KFS object.

Usage

```
Ptt(object)
```

Arguments

object	A KFS object.

Value

The error covariance matrix of the filtered states.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
tsgc:::Ptt(output(res))
```

reinitialise_dataframe *Reinitialise a series at a given position*

Description

Takes a cumulated series and re-bases it so that it starts from zero at `reinit.idx - 1`.

Usage

```
reinitialise_dataframe(dt, reinit.idx)
```

Arguments

dt	Cumulated data series, as an idx_series with exactly one column.
reinit.idx	Integer position at which reinitialisation should occur (i.e. $t = r$, using the notation in the vignette).

Value

The reinitialised series, as an `idx_series` starting at `reinit.idx`.

Examples

```
x <- idx_series(cumsum(rpois(30, 5)) + 1, start = 1)
reinitialise_dataframe(x, 10)
```

`seasonalComp` *Extract number of seasonal components used in a KFS object*

Description

Accessor method to access the number of seasonal components used in a `KFS` object.

Usage

```
seasonalComp(object)
```

Arguments

`object`	A `KFS` object.

Value

The seasonal component specification stored on the model terms.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
res <- estimate(model)
seasonalComp(output(res))
```

`SSModelDynamicGompertz-class`
Class for designing a Dynamic Gompertz Curve State-Space Model

Description

Class for Dynamic Gompertz Curve State-Space Model Object, which encapsulates model settings and provides methods to obtain a FilterResults object.

The dynamic Gompertz model with an integrated random walk (IRW) trend is defined as:

$$\ln g_t = \delta_t + \varepsilon_t, \quad \varepsilon_t \sim NID(0, \sigma_\varepsilon^2), \quad t = 2, ..., T,$$

where Y_t is the cumulative variable, $y_t = \Delta Y_t$, and

$$\ln g_t = \ln y_t - \ln Y_{t-1}.$$

The trend component follows:

$$\delta_t = \delta_{t-1} + \gamma_{t-1},$$

$$\gamma_t = \gamma_{t-1} + \zeta_t, \quad \zeta_t \sim NID(0, \sigma_\zeta^2).$$

Here, the observation disturbances ε_t and slope disturbances ζ_t are independent and normally distributed. The signal-to-noise ratio, $q_\zeta = \sigma_\zeta^2/\sigma_\varepsilon^2$, determines how rapidly the slope adjusts to new observations—higher values lead to faster changes, while lower values induce smoothness. For models without seasonal terms (`sea.period = 0`), the priors are given by:

$$\begin{pmatrix}\delta_1 & \gamma_1\end{pmatrix} \sim N(a_1, P_1).$$

The diffuse prior is defined as $P_1 = \kappa I_{2\times 2}$ with $\kappa \to \infty$, implemented via the `KFAS` package (Helske, 2017). For models with a seasonal component (`sea.period>1`), the prior mean vector a_1 and prior covariance matrix P_1 are extended accordingly.

See the vignette for details on the state disturbance variance matrix Q and the observation noise variance $H = \sigma_\varepsilon^2$.

This class also supports the implementation of the reinitialisation procedure, described in the vignette and also summarised below. Let $t = r$ denote the re-initialization position and r_0 denote the position at which the cumulative series is set to 0. As the growth rate of cumulative cases is defined as $g_t \equiv \frac{y_t}{Y_{t-1}}$, we have:

$$\ln g_t = \ln y_t - \ln Y_{t-1} \quad t = 1, \ldots, r$$

$$\ln g_t^r = \ln y_t - \ln Y_{t-1}^r \quad t = r+1, \ldots, T$$

$$Y_t^r = Y_{t-1}^r + y_t \quad t = r, \ldots, T$$

where Y_t^r is the cumulative cases after re-initialization. We choose to set the cumulative cases to zero at $r_0 = r - 1, Y_{r-1}^r = 0$, such that the growth rate of cumulative cases is available from $t = r + 1$ onwards. We reinitialise the model by specifying the prior distribution for the initial states appropriately. See the vignette for details.

Fields

`Y` The cumulated variable, as an `idx_series`. Must be strictly increasing.

`q` The signal-to-noise ratio (ratio of slope to irregular variance). Defaults to `'NULL'`, in which case no signal-to-noise ratio will be imposed. Instead, it will be estimated.

`sea.period` A positive integer specifying the period of seasonality used in the trigonometric seasonal component of the model. For example, use `7` for daily data to model day-of-the-week effects. A value of `0` disables the seasonal component entirely. The default is `7`, which is suitable for capturing weekly seasonality in daily time series.

`reinit.idx` (Only needed for reinitialization.) The reinitialisation position r, as a single integer. Defaults to `NULL`, which represents the non-reinitialized version.

`original.results` (Only needed for reinitialization.) Rather than re-estimating the model up to the `reinit.idx`, a `FilterResults` class object can be specified here and the parameters for the reinitialisation will be taken from this object. Default is `NULL`. This parameter is optional.

`use.presample.info` (Only needed for reinitialization.) Logical value denoting whether or not to use information from before the reinitialisation position in the reinitialisation procedure. Default is `TRUE`. If `FALSE`, the model is estimated from scratch from the reinitialisation position and no attempt to use information from before the reinitialisation position is made.

`xpred` An `idx_series` object containing the dataset of exogenous variables to include in the model. Defaults to `NULL`.

`ar1` Logical value indicating whether an ar1 component should be included in the model. Default is `FALSE`.

`start` Integer position marking the start of the estimation period.

`end` Integer position marking the end of the estimation period.

`calendar` An optional `idx_calendar` object describing how the integer positions used by `Y` (and the rest of this model's `idx_series` fields) map back to calendar time. Purely cosmetic: never consulted by estimation/filtering, only carried along so that plotting can later translate positions to dates. Defaults to `NULL`, in which case plots will be labelled with raw integer positions instead of dates.

Methods

`estimate()` Estimates the dynamic Gompertz curve model when applied to an object of class `SSModelDynamicGompertz`.

Return Value: An object of class `FilterResults` containing the result output for the estimated dynamic Gompertz curve model.

`initialize( Y, q = NULL, sea.period = 7, reinit.idx = NULL, original.results = NULL, use.presample.info = T`
Create an instance of the `SSModelDynamicGompertz` class. Parameters are defined in 'fields' section.

Usage: `SSModelDynamicGompertz$new(Y = y, q = 0.005, reinit.idx = 45)`

`print()` Provides a quick description of the SSModelDynamicGompertz object, providing model states and standard errors.

`summary()` Supplies details of the SSModelDynamicGompertz object, such as estimated parameter values, start and end positions of the estimation window.

Examples

```
library(tsgc)
set.seed(1)
Y <- idx_series(cumsum(rpois(120, 8)) + 1, start = 1)

# Specify a model
model <- SSModelDynamicGompertz$new(Y = Y, q = 0.005, end = 100)

# Show summary of the model object
summary(model)

# Print a short description of the model object
print(model)

# Estimate a specified model
res <- estimate(model)
res
```

SSModelLeadingIndicator-class

Class for designing a Leading Indicator Model

Description

A class for specifying the parameters of a leading indicator model. The model settings are stored in the fields of this object, and the class contains methods to obtain a FilterResultsLI object for further analysis.

Fields

`Y` A cumulated `idx_series` with 2 columns: a leading indicator and a target variable. Both the target variable and the lagged leading indicator must be strictly increasing within the estimation window.

`q` The signal-to-noise ratio (ratio of slope error variance to target variable observation error variance). Defaults to `'NULL'`, in which case no signal-to-noise ratio will be imposed. Instead, it will be estimated.

`sea.period` A positive integer specifying the period of seasonality used in the trigonometric seasonal component of the model. For example, use `7` for daily data to model day-of-the-week effects. A value of `0` disables the seasonal component entirely. The default is `7`, which is suitable for capturing weekly seasonality in daily time series.

`n.lag` Number of integer positions to lag the leading indicator by.

`xpred_lead` An `idx_series` object containing the values of exogenous variables for the leading indicator. Dataset must contain values for all positions in the estimation time frame. Defaults to NULL, indicating no exogenous variables are needed for the leading indicator.

xpred_targ An idx_series object containing the values of exogenous variables for the target variable. Dataset must contain values for all positions in the estimation time frame. Defaults to NULL, indicating no exogenous variables are needed for the target variable.

LeadIndCol The column in Y that contains the leading indicator. Defaults to 1.

start Integer position marking the start of the estimation period.

end Integer position marking the end of the estimation period.

calendar An optional idx_calendar object describing how the integer positions used by Y map back to calendar time. Since the leading indicator and target series in Y are already combined onto one shared position axis (see Y), a single calendar correctly applies to both. Purely cosmetic: never consulted by estimation/filtering, only carried along so that plotting can later translate positions to dates. Defaults to NULL, in which case plots will be labelled with raw integer positions instead of dates. It is the user's responsibility to ensure the leading indicator and target series were aligned onto a shared, consistent calendar before being combined into Y; this class does not attempt to reconcile two different calendars.

Methods

estimate() Estimates the Leading Indicator model when applied to an object of class SSModelLeadingIndicator.

Return Value: An object of class FilterResultsLI containing the result output for the estimated Leading Indicator model.

initialize(Y, n.lag, sea.period = 7, q = NULL, LeadIndCol = 1, xpred_lead = NULL, xpred_targ = NULL, start = Create an instance of the SSModelLeadingIndicator class with the fields laid out at the beginning of the documentation.

print() Provides a quick description of the SSModelLeadingIndicator object, providing model states and standard errors.

summary() Supplies details of the SSModelLeadingIndicator object, such as estimated parameter values, start and end positions of estimation.

Examples

```
library(tsgc)
set.seed(1)
lead <- cumsum(rpois(120, 6)) + 1
targ <- cumsum(rpois(120, 8)) + 1
Y <- idx_series(cbind(lead, targ), start = 1)

# Specify a model with the estimation timeframe
model <- SSModelLeadingIndicator(Y = Y, n.lag = 5, sea.period = 0,
  LeadIndCol = 1, start = 1, end = 100)

# Show summary of the model object
summary(model)

# Print a short description of the model object
print(model)

# Estimate a specified model
res <- estimate(model)
```

```
res
```

`summary.FilterResults` *Calling summary method for FilterResults*

Description

Accessor method to show a summary for the objects of `FilterResults` class

Usage

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

Arguments

`object`	A `FilterResults` object
`...`	Additional arguments.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)
# Specify a model
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-20"))
# Estimate a specified model
res <- estimate(model)

# Return KFS object in output of res
summary(res)
```

`summary.FilterResultsLI`
Calling summary method for FilterResultsLI

Description

Accessor method to show a summary for the objects of `FilterResultsLI` class

Usage

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

Arguments

object	A FilterResultsLI object
...	Additional arguments.

Examples

```
library(tsgc)
data(england, package = "tsgc")
conv <- xts_to_idx(england[, 1:2])

out_eng <- SSModelLeadingIndicator(
  Y = conv$series, n.lag = 4, sea.period = 7, LeadIndCol = 1,
  calendar = conv$calendar,
  start = idx_to_pos(conv$calendar, "2021-04-30"),
  end = idx_to_pos(conv$calendar, "2021-07-24"))

res_eng<-estimate(out_eng)
summary(res_eng)
```

---

summary.SSModelDynamicGompertz

Calling summary method for SSModelDynamicGompertz class

Description

Accessor method to show a summary for the objects of SSModelDynamicGompertz class

Usage

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

Arguments

object	A SSModelDynamicGompertz object
...	Additional arguments.

Examples

```
library(tsgc)
data(gauteng, package = "tsgc")
conv <- xts_to_idx(gauteng)

# Specify a model
model <- SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
                                    calendar = conv$calendar,
                                    end = idx_to_pos(conv$calendar, "2020-07-06"))
```

```
# Show summary of the model object
summary(model)
```

---

summary.SSModelLeadingIndicator *Calling summary method for classes in tsgc*

Description

Accessor method to show a summary for the objects of SSModelLeadingIndicator class

Usage

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

Arguments

object	A SSModelLeadingIndicator object
...	Additional arguments.

Examples

```
library(tsgc)
data(england, package = "tsgc")
conv <- xts_to_idx(england[, 1:2])

# Specify a model
out_eng <- SSModelLeadingIndicator(
  Y = conv$series, n.lag = 4, sea.period = 7, LeadIndCol = 1,
  calendar = conv$calendar,
  start = idx_to_pos(conv$calendar, "2021-04-30"),
  end = idx_to_pos(conv$calendar, "2021-07-24"))

summary(out_eng)
```

tail.idx_series *Last observations of an* idx_series

Description

Returns the last n observations of x, in position order. As with head.idx_series, n is a count of observations, not a set of absolute positions.

Usage

```
## S3 method for class 'idx_series'
tail(x, n = 6L, ...)
```

Arguments

x	An idx_series object.
n	Number of observations to return from the end of x. Default 6L. If n exceeds length(x), the full series is returned.
...	Unused.

Value

An idx_series of length min(n, length(x)).

ukitaly *Cumulative daily cases of Covid-19 in Italy and UK, before 14 Dec 2020.*

Description

Cumulative daily cases of Covid-19 in Italy and UK, before 14 Dec 2020.

Usage

```
data(ukitaly)
```

Format

An object of class "xts" with columns:

Italy Cumulative cases of Covid-19 in Italy

UK Cumulative cases of Covid-19 in the UK

References

Downloaded from https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide

Examples

```
data(ukitaly)
# plot daily cases
plot(diff(ukitaly))
```

`write_results` *Write a selection of relevant results to disk*

Description

Function writes the following results to csv files which get saved in the location specified in `res.dir`: forecast new cases or incidence variable, y; the filtered level and slope of $\ln g$, δ and γ; filtered estimates of g_y and the confidence intervals for these estimates.

Usage

```
write_results(res, res.dir, n.ahead, prefix = "", confidence.level = 0.68)
```

Arguments

`res`	Results object of class `FilterResults` or `FilterResultsLI`, obtained from '`estimate()`' method.
`res.dir`	File path to save the results to. A character string.
`n.ahead`	Number of periods ahead to forecast. A positive integer.
`prefix`	The prefix to be added to the file names generated. A character string.
`confidence.level`	Confidence level to use for the confidence interval on the forecasts $\ln(g_t)$.

Value

A number of csv files saved in the directory specified in `res.dir`.

Examples

```
# Not run as do not wish to save to local disk when compiling documentation.
# Below will run if copied and pasted into console.
library(tsgc)
library(here)

res.dir <- tempdir()
data(gauteng,package="tsgc")
conv <- xts_to_idx(gauteng)
res <- estimate(SSModelDynamicGompertz$new(Y = conv$series, q = 0.005,
calendar = conv$calendar,
end = idx_to_pos(conv$calendar, "2020-07-06")))

tsgc::write_results(
```

```
res=res, res.dir = res.dir, prefix="dyn_gompertz",n.ahead = 14,
confidence.level = 0.68)
```

`xts_to_idx` *Convert an* `xts`/`zoo` *object to an* `idx_series`

Description

Converts a calendar-indexed `xts` or `zoo` object into an `idx_series`, together with an `idx_calendar` describing how to translate back to calendar time. This is the standard entry point for bringing calendar-indexed data (the user's own `xts`/`zoo`/data frame data, or one of the package's bundled datasets) into the `idx_series`-based analysis functions.

By default (`detect = TRUE`), the step size and any repeating gap pattern are auto-detected from `x`'s index via `idx_detect_calendar_pattern` (searching the full unit ladder years/quarters/months/weeks/days/hours/minutes/s so callers do not need to work out `amount`/`unit`/`pattern` themselves for common cases: a plain daily/weekly/etc. index, a business-day index (weekends skipped), a monthly/quarterly/yearly `Date` index, or a `yearqtr`/`yearmon` index. This only recognises a single `amount`/`unit` pair (optionally repeated via an integer vector `pattern`); it does not attempt to detect compound (`idx_step`) or heterogeneous (`multi_step_pattern`) calendars - build those manually via `idx_calendar_step`/ `idx_calendar_multi_step` if needed. If detection fails (the index has no such regular pattern, e.g. genuinely irregular gaps), `xts_to_idx` raises an error explaining why, rather than silently falling back to an incorrect calendar; pass `detect = FALSE` (and set `amount`/`unit` yourself) for indices detection can't handle.

Usage

```
xts_to_idx(x, start.pos = 1L, detect = TRUE, amount = 1, unit = "days")
```

Arguments

`x`	An `xts` or `zoo` object with a `Date`, `POSIXct`, `yearqtr`, or `yearmon` index of at least 2 rows (needed to detect a step size), and no duplicate index values.
`start.pos`	Integer position that the first row of `x` should be assigned. Use this to align `x` with another, already-converted `idx_series` that starts at a different calendar date - e.g. `start.pos = idx_to_pos(other_cal, zoo::index(x)[1])`. Defaults to `1L`.
`detect`	A single logical. If `TRUE` (default), auto-detect `amount`/`unit`/`pattern` from `x`'s index (see Description). If `FALSE`, use `amount`/`unit` as supplied (previous behaviour; defaults to `amount = 1, unit = "days"` to match this function's original hardcoded assumption).
`amount, unit`	Used only when `detect = FALSE`; passed straight through to `idx_calendar`.

Value

A list with two elements: `series`, an `idx_series` holding `x`'s values, and `calendar`, an `idx_calendar` anchoring `series`'s positions to `x`'s original dates.

Examples

```
# Daily data - detected automatically.
x <- xts::xts(cumsum(rpois(30, 5)) + 1, order.by = Sys.Date() - 29:0)
conv <- xts_to_idx(x)
conv$series
conv$calendar

# Business-day data (weekends skipped) - pattern detected automatically.
bdays <- seq(as.Date("2024-01-01"), by = "day", length.out = 40)
bdays <- bdays[!weekdays(bdays) %in% c("Saturday", "Sunday")]
xb <- xts::xts(cumsum(rpois(length(bdays), 5)) + 1, order.by = bdays)
conv_b <- xts_to_idx(xb)
conv_b$calendar

# Monthly Date index - detected as amount = 1, unit = "months" (not days).
months <- seq(as.Date("2024-01-01"), by = "month", length.out = 24)
xm <- xts::xts(cumsum(rpois(24, 5)) + 1, order.by = months)
conv_m <- xts_to_idx(xm)
conv_m$calendar
```

`[.idx_series` *Subset an* `idx_series` *by position*

Description

Subsets an `idx_series` using integer positions (in the same units as `x$start`), not row numbers within `data`. For example, if `x$start == 10`, then `x[10:12]` returns the first three observations of `x`.

Usage

```
## S3 method for class 'idx_series'
x[i, j, ...]
```

Arguments

`x`	An `idx_series` object.
`i`	Integer vector of positions to select.
`j`	Optional column selector (for matrix-valued series).
`...`	Unused.

Index

∗ **datasets**
    england, 10
    england_weather_2021, 11
    etrading_apps, 14
    gauteng, 20
    gauteng_weather_2021, 21
    nintendo_sales, 46
    ukitaly, 71
[.idx_series, 74

alphahat, 4
argmax, 5
as.double.idx_series
        *(*as.numeric.idx_series*)*, 6
as.matrix.idx_series, 6
as.numeric.idx_series, 6
as_idx_series, 7
att, 7

cross_val, 8

df2ldl, 9
df2ldl_lead, 10, *18*

england, 10
england_weather_2021, 11
estimate, 12
estimate_r0, 12, *55*
etrading_apps, 14

FilterResults, *61*
FilterResults *(*FilterResults-class*)*, 14
FilterResults-class, 14
FilterResultsLI, *61*
FilterResultsLI
        *(*FilterResultsLI-class*)*, 17
FilterResultsLI-class, 17

gauteng, 20
gauteng_weather_2021, 21
get_timeframe, 23
get_V, 24
gety, 22
gety.hat, 22

head.idx_series, 24, *71*

idx_axis_opts, 25, *48–56*
idx_calendar, 26, *28–30*, *39*, *45*, *73*
idx_calendar_multi_step, 28, *39*, *45*, *73*
idx_calendar_offset, *28*, 29, *38*
idx_calendar_step, *28*, 30, *38, 39*, *45*, *73*
idx_cbind, 31
idx_diff, 31
idx_lag, 32
idx_ncol, 32, *46*
idx_offset_to_pos, 33, *39*
idx_positions, 33
idx_range, *33*, 34
idx_rbind, 34
idx_series, *26*, 35, *73*
idx_step, *30*, 36, *38*, *45*
idx_step_add, *28*, *36*, 37, *38, 39*
idx_to_date, *29*, *33*, 38, *39*
idx_to_pos, 39
idx_values, 40
is_idx_calendar, 41
is_idx_series, 41
is_idx_step, 42
is_multi_step_pattern, 42

length.idx_series, 43

mapes, *15*, *19*, 43
matrixKFS, 44
modelKFS, 44
multi_step_pattern, *28, 29*, 45

NCOL.idx_series, 46
nintendo_sales, 46

output, 47

plot.SSModelDynamicGompertz, 48
plot.SSModelLeadingIndicator, 49
plot_compare_forecast, 50
plot_forecast, 51
plot_gy_ci, 52
plot_gy_components, 53
plot_holdout, 53
plot_log_forecast, 54
plot_r0, 55
print.FilterResults, 56
print.FilterResultsLI, 57
print.idx_calendar, 58
print.idx_series, 58
print.idx_step, 59
print.multi_step_pattern, 59
print.SSModelDynamicGompertz, 60
print.SSModelLeadingIndicator, 60
print_model_diagnostics, 61
Ptt, 62

reinitialise_dataframe, 62

seasonalComp, 63
seq.Date, *37, 38*
seq.POSIXt, *37, 38*
SSModelDynamicGompertz
        *(*SSModelDynamicGompertz-class*)*, 64
SSModelDynamicGompertz-class, 64
SSModelLeadingIndicator
        *(*SSModelLeadingIndicator-class*)*, 66
SSModelLeadingIndicator-class, 66
summary.FilterResults, 68
summary.FilterResultsLI, 68
summary.SSModelDynamicGompertz, 69
summary.SSModelLeadingIndicator, 70

tail.idx_series, 71

ukitaly, 71

write_results, 72

xts_to_idx, *39*, 73