Package 'rjd3toolkit'

January 13, 2026

Type Package

Title Utility Functions Around 'JDemetra+ 3.0'

Version 3.6.0

Description R Interface to 'JDemetra+ 3.x' (<https://github.com/jdemetra>) time series analysis software. It provides functions allowing to model time series (create outlier regressors, user-defined calendar regressors, Unobserved Components AutoRegressive Integrated Moving Average (UCARIMA) models...), to test the presence of trading days or seasonal effects and also to set specifications in pre-adjustment and benchmarking when using 'rjd3x13' or 'rjd3tramoseats'.

License EUPL

URL https://github.com/rjdverse/rjd3toolkit,
https://rjdverse.github.io/rjd3toolkit/

BugReports https://github.com/rjdverse/rjd3toolkit/issues

Depends R (>= 4.1)

Imports checkmate, graphics, methods, rJava (>= 1.0-6), rjd3jars, RProtoBuf (>= 0.4.20), stats, utils

Encoding UTF-8

Language en-GB

LazyData TRUE

RoxygenNote 7.3.3

SystemRequirements Java (>= 17)

Collate 'utils.R' 'jd2r.R' 'protobuf.R' 'arima.R' 'calendars.R' 'calendarts.R' 'decomposition.R' 'deprecated.R' 'differencing.R' 'display.R' 'distributions.R' 'generics.R' 'jd3rslts.R' 'modellingcontext.R' 'procresults.R' 'regarima_generic.R' 'regarima_rslts.R' 'spec_benchmarking.R' 'spec_regarima.R' 'splines.R' 'tests_regular.R' 'tests_seasonality.R' 'tests_td.R' 'timeseries.R' 'variables.R' 'zzz.R'

NeedsCompilation no

Author Jean Palate [aut],
Alain Quartier-la-Tente [aut] (ORCID:
<https://orcid.org/0000-0001-7890-3857>),
Tanguy Barthelemy [aut, cre, art],
Anna Smyk [aut]

Maintainer Tanguy Barthelemy <tanguy.barthelemy@insee.fr>

Repository CRAN

Date/Publication 2026-01-13 18:00:02 UTC

Contents

`.add_ud_var` *Add user-defined variable to a SA model*

Description

Add user-defined variable to a SA model

Usage

```
.add_ud_var(x, jx, userdefined = NULL, out_class = NULL, result = FALSE)
```

Arguments

`x`	The model of SA
`jx`	Reference to a Java object
`userdefined`	vector containing the names of the object to extract.
`out_class`	Java class of the result object
`result`	Boolean. Does `jx` contains the results? Default to FALSE.

Value

A new model with same class as `x`

`.likelihood` *Information on the (log-)likelihood*

Description

Function allowing to gather information on likelihood estimation

Usage

```
.likelihood(
  nobs,
  neffectiveobs = NA,
  nparams = 0,
  ll,
  adjustedll = NA,
  aic,
  aicc,
  bic,
  bicc,
  ssq
)
```

Arguments

`nobs`	Number of observations
`neffectiveobs`	Number of effective observations. NA if the same as nobs.
`nparams`	Number of hyper-parameters
`ll`	Log-likelihood
`adjustedll`	Adjusted log-likelihood when the series has been transformed
`aic`	AIC
`aicc`	AICC
`bic`	BIC
`bicc`	BIC corrected for the length
`ssq`	Sum of the squared residuals

Value

Returns a java object of class JD3_LIKELIHOOD.

Examples

```
# Values used below are taken from the following estimation
# m <- rjd3x13::x13(rjd3toolkit::ABS$X0.2.09.10.M, "rsa3")
# m$result$preprocessing$estimation$likelihood
ll_estimation <- .likelihood(425, 4, 7, 720.2, -2147.407, 4308.14, 4309.09,
                              4333.96, 433.962, 0.0418)
```

.r2jd_tsdata *Java Utility Functions*

Description

These functions are used in all JDemetra+ 3.0 packages to easily interact between R and Java objects.

Usage

```
.r2jd_tsdata(s)

.r2jd_tsdomain(period, startYear, startPeriod, length)

.jd2r_tsdata(s)

.jd2r_mts(s)

.jd2r_lts(s)

.jd2r_matrix(s)

.r2jd_matrix(s)

.jdomain(period, start, end)

.enum_sextract(type, p)

.enum_sof(type, code)

.enum_extract(type, p)

.enum_of(type, code, prefix)
```

```
.r2p_parameter(r)

.p2r_parameter(p)

.r2p_parameters(r)

.r2p_lparameters(r)

.p2r_parameters(p)

.p2r_parameters_rslt(p)

.p2r_parameters_rsltx(p)

.p2r_test(p)

.p2r_matrix(p)

.p2r_tsdata(p)

.r2p_tsdata(r)

.p2r_parameters_estimation(p)

.p2r_likelihood(p)

.p2r_date(p)

.r2p_date(s)

.p2r_span(span)

.r2p_span(rspan)

.p2r_arima(p)

.p2r_ucarima(p)

.p2r_spec_sarima(spec)

.r2p_spec_sarima(r)

.p2r_outliers(p)

.r2p_outliers(r)

.p2r_sequences(p)
```

```
.r2p_sequences(r)

.p2r_iv(p)

.r2p_iv(r)

.p2r_ivs(p)

.r2p_ivs(r)

.p2r_ramps(p)

.r2p_ramps(r)

.p2r_uservars(p)

.r2p_uservars(r)

.p2r_variables(p)

.p2r_sa_decomposition(p, full = FALSE)

.p2r_sa_diagnostics(p)

.p2r_spec_benchmarking(p)

.r2p_spec_benchmarking(r)

.r2jd_sarima(model)

.jd2r_ucarima(jucm)

.p2jd_calendar(pcalendar)

.r2p_calendar(r)

.proc_numeric(rslt, name)

.proc_vector(rslt, name)

.proc_int(rslt, name)

.proc_bool(rslt, name)

.proc_ts(rslt, name)

.proc_str(rslt, name)
```

```
.proc_desc(rslt, name)

.proc_test(rslt, name)

.proc_parameter(rslt, name)

.proc_parameters(rslt, name)

.proc_matrix(rslt, name)

.proc_data(rslt, name)

.proc_dictionary(name)

.proc_dictionary2(jobj)

.proc_likelihood(jrslt, prefix)

.r2p_moniker(r)

.p2r_moniker(p)

.r2p_datasupplier(name, r)

.p2r_metadata(p)

.r2p_metadata(r, type)

.p2r_ts(p)

.r2p_ts(r)

.p2r_tscollection(p)

.r2p_tscollection(r)

.r2jd_ts(s)

.jd2r_ts(js)

.r2jd_tscollection(s)

.jd2r_tscollection(js)

.p2r_datasupplier(p)

.r2p_datasuppliers(r)
```

```
.p2r_datasuppliers(p)

.p2jd_variables(p)

.jd2p_variables(jd)

.jd2r_variables(jcals)

.r2jd_variables(r)

.p2r_context(p)

.r2p_context(r)

.p2jd_context(p)

.jd2p_context(jd)

.jd2r_modellingcontext(jcontext)

.r2jd_modellingcontext(r)

.p2r_calendars(p)

.r2p_calendars(r)

.p2jd_calendars(p)

.jd2p_calendars(jd)

.jd2r_calendars(jcals)

.r2jd_calendars(r)

.jd3_object(jobjRef, subclasses = NULL, result = FALSE)

.p2r_regarima_rslts(p)

.r2jd_tmp_ts(s, name)

.r2jd_make_ts(source, id, type = "All")

.r2jd_make_tscollection(source, id, type = "All")

current_java_version

minimal_java_version
```

```
get_date_min()

get_date_max()
```

Arguments

`s`	Time series
`startYear`	Initial year in the time domain
`startPeriod`	Initial period in the time domain(1 for the first period)
`length`	Length
`p`, `r`, `spec`, `jucm`, `start`, `end`, `name`, `period`, `type`, `code`, `prefix`, `span`, `rspan`, `full`, `rslt`, `jd`, `jcontext`, `jobjRef`, `jcals`, `subclasses`, `result`, `pcalendar`	parameters.
`model`	Model
`jobj`	Java object
`jrslt`	Reference to a Java object
`js`	Java time series
`source`	Source of the time series information
`id`	Identifier of the time series information (source-dependent)

Format

An object of class `integer` of length 1.

An object of class `numeric` of length 1.

`.tsmoniker` *Create a Moniker*

Description

Create a Moniker

Usage

```
.tsmoniker(source, id)
```

Arguments

`source`	Source of the time series.
`id`	Id of the time series.

Value

Returns a java object of class JD3_TSMONIKER.

Examples

```
source <- "Txt"
# id is split due to length restrictions
id1 <- "demetra://tsprovider/Txt/20111201/SERIES?datePattern=dd%2FMM%2Fyyyy&delimiter=SEMICOLON&"
id2 <- "file=C%3A%5CDocuments%5CIPI%5CData%5CIPI_nace4.csv#seriesIndex=0"
id <- paste0(id1, id2)
moniker <- .tsmoniker(source, id)
```

ABS *Retail trade statistics in Australia*

Description

Retail trade statistics in Australia

Usage

```
ABS
```

Format

An object of class `data.frame` with 425 rows and 22 columns.

Source

ABS

Examples

```
data(ABS)
```

---

add_outlier *Manage Outliers/Ramps in Specification*

Description

Generic function to add outliers or Ramp regressors (`add_outlier()` and `add_ramp()`) to a specification or to remove them (`remove_outlier()` and `remove_ramp()`).

Usage

```
add_outlier(x, type, date, name = sprintf("%s (%s)", type, date), coef = 0)

remove_outlier(x, type = NULL, date = NULL, name = NULL)

add_ramp(x, start, end, name = sprintf("rp.%s - %s", start, end), coef = 0)

remove_ramp(x, start = NULL, end = NULL, name = NULL)
```

Arguments

`x`	the specification to customize, must be a "SPEC" class object (see details).
`type, date`	type and date of the outliers. Possible `type` are: `"AO"` = additive, `"LS"` = level shift, `"TC"` = transitory change and `"SO"` = seasonal outlier.
`name`	the name of the variable (to format print).
`coef`	the coefficient if needs to be fixed. If equal to 0 the outliers/ramps coefficients are estimated.
`start, end`	dates of the ramp regressor.

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with `rjd3x13::x13_spec()` (or "JD3_REGARIMA_SPEC" generated with `rjd3x13::spec_regarima()` or "JD3_TRAMOSEATS_SPEC" generated with `rjd3tramoseats::spec_tramoseats()` or "JD3_TRAMO_SPEC" generated with `rjd3tramoseats::spec_tramo()`). If a Seasonal adjustment process is performed, each type of Outlier will be allocated to a pre-defined component after the decomposition: "AO" and "TC" to the irregular, "LS" and Ramps to the trend.

Value

The modified specification (with/without outliers or ramp)

References

More information on outliers and other auxiliary variables in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

`add_usrdefvar`, `intervention_variable`

Examples

```
init_spec <- x13_spec_default

# Adding outlier on year 2012
new_spec <- add_outlier(init_spec, type = "AO", date = "2012-01-01")
# Removing outlier on year 2012
new_spec <- remove_outlier(new_spec, type = "AO", date = "2012-01-01")

# Adding ramp on year 2012
new_spec2 <- add_ramp(init_spec, start = "2012-01-01", end = "2012-12-01")
# Removing ramp on year 2012
new_spec2 <- remove_ramp(new_spec2, start = "2012-01-01", end = "2012-12-01")
```

add_usrdefvar *Add a User-Defined Variable to Pre-Processing Specification.*

Description

Function allowing to add any user-defined regressor to a specification and allocate its effect to a selected component, excepted to the calendar component. To add user-defined calendar regressors, set_tradingdays. Once added to a specification, the external regressor(s) will also have to be added to a modelling context before being used in an estimation process. see modelling_context and example.

Usage

```
add_usrdefvar(
  x,
  group = "r",
  name,
  label = paste0(group, ".", name),
  lag = 0,
  coef = NULL,
  regeffect = c("Undefined", "Trend", "Seasonal", "Irregular", "Series",
    "SeasonallyAdjusted")
)
```

Arguments

x	the specification to customize, must be a "SPEC" class object (see details).
group, name	the name of the regressor in the format "group.name", by default "r.name" by default if group NULL "group.name" has to be the same as in modelling_context (see examples)
label	the label of the variable to be displayed when printing specification or results. By default equals to group.name.
lag	integer defining if the user-defined variable should be lagged. By default (lag = 0), the regressor x_t is not lagged. If lag = 1, then x_{t-1} is used.
coef	the coefficient, if needs to be fixed.
regeffect	component to which the effect of the user-defined variable will be assigned. By default ("Undefined"), see details.

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with rjd3x13::x13_spec() (or "JD3_REGARIMA_SPEC" generated with rjd3x13::spec_regarima() or "JD3_TRAMOSEATS_SPEC" generated with rjd3tramoseats::spec_tramoseats() or "JD3_TRAMO_SPEC" generated with rjd3tramoseats::spec_tramo()). Components to which the effect of the regressor can be allocated:

- "Undefined" : the effect of the regressor is assigned to an additional component, the variable is used to improve the pre-processing step, but is not removed from the series for the decomposition.
  - "Trend": after the decomposition the effect is allocated to the trend component, like a Level-Shift
  - "Irregular": after the decomposition the effect is allocated to the irregular component, like an Additive-outlier
  - "Seasonal": after the decomposition the effect is allocated to the seasonal component, like a Seasonal-outlier
  - "Series": after the decomposition the effect is allocated to the raw series: $yc_t = y_t + effect$
  - "SeasonallyAdjusted": after the decomposition the effect is allocated to the seasonally adjusted series: $sa_t = T + I + effect$

Value

The modified specification (with new user-defined variables)

References

More information on outliers and other auxiliary variables in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

set_tradingdays, intervention_variable

Examples

```
# Creating one or several external regressors (TS objects),
# which will be gathered in one or several groups
iv1 <- intervention_variable(
    frequency = 12,
    start = c(2000, 1),
    length = 60,
    starts = "2001-01-01",
    ends = "2001-12-01"
)
iv2 <- intervention_variable(
    frequency = 12,
    start = c(2000, 1),
    length = 60,
    starts = "2001-01-01",
    ends = "2001-12-01",
    delta = 1
)

# Using one variable in a a seasonal adjustment process
# Regressors as a list of two groups reg1 and reg2
```

```
vars <- list(
    reg1 = list(x = iv1),
    reg2 = list(x = iv2)
)

# Creating the modelling context
my_context <- modelling_context(variables = vars)

# Customize a default specification
init_spec <- x13_spec_default

# Regressors have to be added one by one
new_spec <- add_usrdefvar(init_spec, name = "reg1.iv1", regeffect = "Trend")
new_spec <- add_usrdefvar(new_spec, name = "reg2.iv2", regeffect = "Trend", coef = 0.7)
```

---

aggregate *Aggregation of time series*

Description

Makes a frequency change of this series.

Usage

```
aggregate(
  s,
  nfreq = 1,
  conversion = c("Sum", "Average", "First", "Last", "Min", "Max"),
  complete = TRUE
)
```

Arguments

`s`	the input time series.
`nfreq`	the new frequency. Must be la divisor of the frequency of `s`.
`conversion`	Aggregation mode: sum (`"Sum"`), average (`"Average"`), first observation (`"First"`), last observation (`"Last"`), minimum (`"Min"`), maximum (`"Max"`).
`complete`	Boolean indicating if the observation for a given period in the new series is set missing if some data in the original series are missing.

Value

A new time series of frequency `nfreq`.

Examples

```
s <- ABS$X0.2.09.10.M
# Annual sum
aggregate(s, nfreq = 1, conversion = "Sum") # first and last years removed
aggregate(s, nfreq = 1, conversion = "Sum", complete = FALSE)
# Quarterly mean
aggregate(s, nfreq = 4, conversion = "Average")
```

---

arima_difference *Remove an arima model from an existing one.*

Description

More exactly, m_diff = m_left - m_right iff m_left = m_right + m_diff.

Usage

```
arima_difference(left, right, simplify = TRUE)
```

Arguments

`left`	Left operand (JD3_ARIMA object)
`right`	Right operand (JD3_ARIMA object)
`simplify`	Simplify the results if possible (common roots in the auto-regressive and in the moving average polynomials, including unit roots)

Value

a `"JD3_ARIMA"` model.

Examples

```
mod1 <- arima_model(delta = c(1, -2, 1))
mod2 <- arima_model(variance = .01)
diff <- arima_difference(mod1, mod2)
sum <- arima_sum(diff, mod2)
# sum should be equal to mod1
```

arima_model *ARIMA Model*

Description

ARIMA Model

Usage

```
arima_model(name = "arima", ar = 1, delta = 1, ma = 1, variance = 1)
```

Arguments

name	Name of the model.
ar	coefficients of the regular auto-regressive polynomial (1 + ar(1)B + ar(2)B + ...). True signs.
delta	non stationary auto-regressive polynomial.
ma	coefficients of the regular moving average polynomial (1 + ma(1)B + ma(2)B + ...). True signs.
variance	variance.

Value

a "JD3_ARIMA" model.

Examples

```
model <- arima_model("trend", ar = c(1, -.8), delta = c(1, -1), ma = c(1, -.5), var = 100)
```

arima_properties *Properties of an ARIMA model*

Description

The (pseudo-)spectrum and the auto-covariances of the model are returned

Usage

```
arima_properties(model, nspectrum = 601, nac = 36)
```

Arguments

`model`	a "JD3_ARIMA" model (created with `arima_model()`).
`nspectrum`	number of points to calculate the spectrum; th points are uniformly distributed in [0, pi]
`nac`	maximum lag at which to calculate the auto-covariances; if the model is non-stationary, the auto-covariances are computed on its stationary transformation.

Value

A list with the auto-covariances and with the (pseudo-)spectrum

Examples

```
mod1 <- arima_model(ar = c(0.1, 0.2), delta = c(1, -1), ma = 0)
arima_properties(mod1)
```

---

arima_sum *Sum ARIMA Models*

Description

Sum ARIMA Models

Usage

```
arima_sum(...)
```

Arguments

`...`	list of ARIMA models (created with `arima_model()`).

Details

Adds several Arima models, considering that their innovations are independent. The sum of two Arima models is computed as follows: the auto-regressive parts (stationary and non stationary of the aggregated model are the smaller common multiple of the corresponding polynomials of the components. The sum of the acf of the modified moving average polynomials is then computed and factorized, to get the moving average polynomial and innovation variance of the sum.

Value

a "JD3_ARIMA" model.

Examples

```
mod1 <- arima_model(ar = c(0.1, 0.2), delta = 0, ma = 0)
mod2 <- arima_model(ar = 0, delta = 0, ma = c(0.4))
arima_sum(mod1, mod2)
```

---

autocorrelations *Autocorrelation Functions*

Description

Autocorrelation Functions

Usage

```
autocorrelations(data, mean = TRUE, n = 15)

autocorrelations_partial(data, mean = TRUE, n = 15)

autocorrelations_inverse(data, nar = 30, n = 15)
```

Arguments

`data`	data being tested.
`mean`	Mean correction. If `TRUE`, the auto-correlations are computed as usual. If `FALSE`, we consider that the (known) mean is 0 and that the series has been corrected for it.
`n`	maximum lag at which to calculate the stats.
`nar`	number of AR lags used to compute inverse autocorrelations.

Value

`autocorrelations()` returns a vector of length `n` with the autocorrelations. `autocorrelations_partial()` returns a vector of length `n` with the partial autocorrelations. `autocorrelations_inverse()` returns a vector of length `n` with the inverse autocorrelations.

Examples

```
x <- ABS$X0.2.09.10.M
autocorrelations(x)
autocorrelations_partial(x)
autocorrelations_inverse(x)
```

`Births` *Number of births registered in France from 1968 to 2024*

Description

Daily number of births recorded in France (metropolitan + DOM), covering the period from January 1, 1968 to December 31, 2024.

Usage

```
Births
```

Format

A data frame with 20,820 rows and 2 variables:

- date: Date of the value (from 1968-01-01 to 2024-12-31)
- births: Number of daily births (1254–2830)

Details

The dataset corresponds to the INSEE series **T79jnais**. The raw data can be downloaded as a CSV file here: https://www.insee.fr/fr/statistiques/fichier/8582123/T79jnais.csv

Source

INSEE, Statistiques de l'état civil – https://www.insee.fr/fr/statistiques/8582123?sommaire=8582147

Examples

```
data(Births)
plot(Births$date, Births$births,
     type = "l",
     main = "Daily births in France",
     ylab = "Number of births",
     xlab = "date")
```

bsplines *B-Splines*

Description

B-Splines

Usage

```
bsplines(order = 4, knots, pos)
```

Arguments

`order`	Order of the splines (4 for cubic)
`knots`	Knots of the splines (in [0, period[)
`pos`	Requested positions (in [0, period[). The rows of the returned matrix will correspond to those positions

Value

A matrix (len(pos) x len(knots))

Examples

```
s<-bsplines(knots = c(0,.2,.3, .9,.95, 1), pos=seq(0,1,0.01))
matplot(s, type='l')
```

calendar_td *Trading day regressors with pre-defined holidays*

Description

Allows to generate trading day regressors (as many as defined groups), taking into account 7 or less different types of days, from Monday to Sunday, and specific holidays,which are to defined beforehand in a calendar using the functions `national_calendar`,`weighted_calendar` or `Chained_calendar`.

Usage

```
calendar_td(
  calendar = national_calendar(),
  frequency,
  start,
  length,
  s,
  groups = c(1, 2, 3, 4, 5, 6, 0),
  holiday = 7,
  contrasts = TRUE
)
```

Arguments

`calendar`	The calendar containing the required holidays
`frequency`	Frequency of the series, number of periods per year (12, 4, 3, 2...)
`start, length`	First date (array with the first year and the first period, for instance `c(1980, 1)`) and number of periods of the output variables. Can also be provided with the `s` argument
`s`	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
`groups`	Groups of days. The length of the array must be 7. It indicates to what group each week day belongs. The first item corresponds to Mondays and the last one to Sundays. The group used for contrasts (usually Sundays) is identified by 0. The other groups are identified by 1, 2,... n (<= 6). For instance, usual trading days are defined by c(1, 2, 3, 4, 5, 6, 0), week days by c(1, 1, 1, 1, 1, 0, 0), week days, Saturdays, Sundays by c(1, 1, 1, 1, 1, 2, 0) etc.
`holiday`	Day to aggregate holidays with. (holidays are considered as that day). 1 for Monday... 7 for Sunday. Doesn't necessary belong to the 0-group.
`contrasts`	If true, the variables are defined by contrasts with the 0-group. Otherwise, raw number of days is provided.

Details

Aggregated values for monthly or quarterly are the numbers of days belonging to a given group, holidays are all summed together in of those groups. Contrasts are the differences between the number of days in a given group (1 to 6) and the number of days in the reference group (0). Regressors are corrected for long-term mean if `contrasts = TRUE`.

Value

Time series (object of class `c("ts","mts","matrix")`) corresponding to each group, starting with the 0-group (`contrasts = FALSE`) or the 1-group (`contrasts = TRUE`).

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

national_calendar, td

Examples

```
BE <- national_calendar(list(
    fixed_day(7, 21),
    special_day("NEWYEAR"),
    special_day("CHRISTMAS"),
    special_day("MAYDAY"),
    special_day("EASTERMONDAY"),
    special_day("ASCENSION"),
    special_day("WHITMONDAY"),
    special_day("ASSUMPTION"),
    special_day("ALLSAINTSDAY"),
    special_day("ARMISTICE")
))
calendar_td(BE, 12, c(1980, 1), 240,
    holiday = 7, groups = c(1, 1, 1, 2, 2, 3, 0),
    contrasts = FALSE
)
```

---

chained_calendar *Create a Chained Calendar*

Description

Allows to combine two calendars, one before and one after a given date.

Usage

```
chained_calendar(calendar1, calendar2, break_date)
```

Arguments

calendar1, calendar2	calendars to chain.
break_date	the break date in the format "YYYY-MM-DD".

Details

A chained calendar is an useful option when major changes in the composition of the holidays take place. In such a case two calendars describing the situation before and after the change of regime can be defined and bound together, one before the break and one after the break.

Value

returns an object of class `c("JD3_CHAINEDCALENDAR","JD3_CALENDARDEFINITION")`

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

`national_calendar`, `weighted_calendar`

Examples

```
Belgium <- national_calendar(list(special_day("NEWYEAR"), fixed_day(7, 21)))
France <- national_calendar(list(special_day("NEWYEAR"), fixed_day(7, 14)))
chained_cal <- chained_calendar(France, Belgium, "2000-01-01")
```

---

clean_extremities — *Removal of missing values at the beginning/end*

Description

Removal of missing values at the beginning/end

Usage

```
clean_extremities(s)
```

Arguments

s	Original series

Value

Cleaned series

Examples

```
y <- window(ABS$X0.2.09.10.M, start = 1982, end = 2018, extend = TRUE)
y
clean_extremities(y)
```

compare_annual_totals *Compare the annual totals of two series*

Description

Usually a raw series and the corresponding seasonally adjusted series

Usage

```
compare_annual_totals(raw, sa)
```

Arguments

raw	Raw series
sa	Seasonally adjusted series

Value

The largest annual difference (in percentage of the average level of the seasonally adjusted series)

Examples

```
s1<- rjd3toolkit::ABS$X0.2.09.10.M
# two raw series for example's sake
s2 <- rjd3toolkit::ABS$X0.2.08.10.M
compare_annual_totals(s1,s2)
```

data_to_ts *Promote a R time series to a "full JDemetra+ time series"*

Description

Promote a R time series to a "full JDemetra+ time series"

Usage

```
data_to_ts(s, name)
```

Arguments

s	R time series (class TS)
name	name of the series

Value

Returns a java object of class JD3_TS

Examples

```
s <- ABS$X0.2.09.10.M
t <- data_to_ts(s, "test")
```

`daysOf` *Provides a list of dates corresponding to each period of the given time series*

Description

Provides a list of dates corresponding to each period of the given time series

Usage

```
daysOf(ts, pos = 1)
```

Arguments

ts	A time series
pos	The position of the first considered period.

Value

A list of the starting dates of each period

Examples

```
daysOf(Retail$BookStores)
```

`density_chi2` *The Chi-Squared Distribution*

Description

Density, (cumulative) distribution function and random generation for chi-squared distribution.

Usage

```
density_chi2(df, x)

cdf_chi2(df, x)

random_chi2(df, n)
```

Arguments

df	degrees of freedom.
x	vector of quantiles.
n	number of observations.

Value

The functions density_XXX and cdf_t return numeric vectors of same length as x. The functions random_XXX return random number (numeric vectors) of length n.

Examples

```
density_chi2(df = 3, 1:10)
cdf_chi2(df = 3, 1:10)
random_chi2(df = 3, n = 10)
```

---

density_gamma *The Gamma Distribution*

Description

Density, (cumulative) distribution function and random generation for Gamma distribution.

Usage

```
density_gamma(shape, scale, x)

cdf_gamma(shape, scale, x)

random_gamma(shape, scale, n)
```

Arguments

shape, scale	shape and scale parameters.
x	vector of quantiles.
n	number of observations.

Value

The functions density_XXX and cdf_t return numeric vectors of same length as x. The functions random_XXX return random number (numeric vectors) of length n.

Examples

```
density_gamma(shape = 1, scale = 2, x = 1:10)
cdf_gamma(shape = 1, scale = 2, x = 1:10)
random_gamma(shape = 1, scale = 2, n = 10)
```

density_inverse_gamma *The Inverse-Gamma Distribution*

Description

Density, (cumulative) distribution function and random generation for inverse-gamma distribution.

Usage

```
density_inverse_gamma(shape, scale, x)

cdf_inverse_gamma(shape, scale, x)

random_inverse_gamma(shape, scale, n)
```

Arguments

shape, scale	shape and scale parameters.
x	vector of quantiles.
n	number of observations.

Value

The functions density_XXX and cdf_t return numeric vectors of same length as x. The functions random_XXX return random number (numeric vectors) of length n.

Examples

```
density_inverse_gamma(shape = 1, scale = 2, x = 1:10)
cdf_inverse_gamma(shape = 1, scale = 2, x = 1:10)
random_inverse_gamma(shape = 1, scale = 2, n = 10)
```

density_inverse_gaussian *The Inverse-Gaussian Distribution*

Description

Density, (cumulative) distribution function and random generation for inverse-gaussian distribution.

Usage

```
density_inverse_gaussian(shape, scale, x)

cdf_inverse_gaussian(shape, scale, x)

random_inverse_gaussian(shape, scale, n)
```

Arguments

`shape, scale`	shape and scale parameters.
`x`	vector of quantiles.
`n`	number of observations.

Value

The functions density_XXX and cdf_t return numeric vectors of same length as `x`. The functions random_XXX return random number (numeric vectors) of length `n`.

Examples

```
density_inverse_gaussian(shape = 1, scale = 2, x = 1:10)
random_inverse_gaussian(shape = 1, scale = 2, n = 10)
```

`density_t` *The Student Distribution*

Description

Probability Density Function (PDF), Cumulative Density Function (CDF) and generation of random variables following a Student distribution.

Usage

```
density_t(df, x)

cdf_t(df, x)

random_t(df, n)
```

Arguments

`df`	degrees of freedom.
`x`	vector of quantiles.
`n`	number of observations.

Value

The functions density_XXX and cdf_t return numeric vectors of same length as `x`. The functions random_XXX return random number (numeric vectors) of length `n`.

Examples

```
# Probability density function of T with 2 degrees of freedom.
z <- density_t(df = 2, .01 * seq(-100, 100, 1))
# Generating a random vector with each component drawn from a T(2) distribution
z <- random_t(2, 100)
# Computing the probabilty that the random variable X following a T distribution
# with df degrees of freedom is lower than x
z <- cdf_t(df = 12, x = 1.2)
z
z <- cdf_t(df = 12, x = c(0:10)) # array of values
z
```

---

deprecated-rjd3toolkit *Deprecated functions*

Description

Use `sa_decomposition()` instead of `sa.decomposition()`.

Usage

```
sa.decomposition(x, ...)
```

Arguments

`x`	the object to print.
`...`	further arguments.

Value

`"JD3_SADECOMPOSITION"` object.

---

diagnostics *Generic Diagnostics Function*

Description

Generic Diagnostics Function

Usage

```
diagnostics(x, ...)

## S3 method for class 'JD3'
diagnostics(x, ...)
```

Arguments

x	the object to extract diagnostics.
...	further arguments.

Value

"No diagnostic" or a list with the diagnostic part of the model

Examples

```
decompo <- sadecomposition(
    y = ts(c(112, 118, 132, 129, 121, 135), start = 2000, frequency = 12L),
   sa = ts(c(121.72, 124.52, 125.4, 128.91, 128.84, 126.73), start = 2000, frequency = 12L),
   t = ts(c(122.24, 124.33, 126.21, 127.61, 127.8, 126.94), start = 2000, frequency = 12L),
    s = ts(c(0.92, 0.95, 1.05, 1, 0.94, 1.07), start = 2000, frequency = 12L),
    i = ts(c(1, 1, 0.99, 1.01, 1.01, 1), start = 2000, frequency = 12L),
    mul = TRUE
)
diagnostics(decompo)
```

---

dictionary *Get Dictionary and Result*

Description

Extract dictionary of a "JD3_ProcResults" object (dictionary()) and extract a specific value (result()) or a list of values (user_defined()).

Usage

```
dictionary(object)

result(object, id)

user_defined(object, userdefined = NULL)
```

Arguments

object	the java object.
id	the name of the object to extract.
userdefined	vector containing the names of the object to extract.

Value

the function dictionary() returns a character vector with the items that can be extracted from object. The result() function extract an item from the object. The user_defined() function do the same thing as result() but can also extract several element at once and encapsulate the items in a user_defined class object.

`differences` *Differencing of a series*

Description

Differencing of a series

Usage

```
differences(data, lags = 1, mean = TRUE)
```

Arguments

`data`	The series to be differenced.
`lags`	Lags of the differencing.
`mean`	Apply a mean correction at the end of the differencing process.

Value

The differenced series.

Examples

```
differences(Retail$BookStores, c(1, 1, 12), FALSE)
```

`differencing_fast` *The series is differenced till its variance is decreasing.*

Description

Automatic differencing

Usage

```
differencing_fast(data, period, mad = TRUE, centile = 90, k = 1.2)
```

Arguments

`data`	Series being differenced.
`period`	Period considered in the automatic differencing.
`mad`	Use of MAD in the computation of the variance (true by default).
`centile`	Percentage of the data used for computing the variance (90 by default).
`k`	tolerance in the decrease of the variance. The algorithm stops if the new variance is higher than k*the old variance. k should be equal or slightly higher than 1 (1.2 by default)

Value

Stationary transformation

- ddata: data after differencing
- mean: mean correction
- differences:
  - lag: $ddata(t) = data(t) - data(t - lag)$
  - order: order of the differencing

Examples

```
differencing_fast(log(ABS$X0.2.09.10.M), 12)
```

---

do_stationary *Automatic stationary transformation*

Description

Automatic processing (identification of the order of the differencing) based on auto-correlations and on mean correction. The series should not be seasonal. Source: Tramo

Usage

```
do_stationary(data, period)
```

Arguments

data	Series being differenced.
period	Period of the series.

Value

Stationary transformation

- ddata: data after differencing
- mean: mean correction
- differences:
  - lag: $ddata(t) = data(t) - data(t - lag)$
  - order: order of the differencing

Examples

```
do_stationary(log(ABS$X0.2.09.10.M), 12)
```

`easter_dates` *Display Easter Sunday dates in given period*

Description

Allows to display the date of Easter Sunday for each year, in the defined period. Dates are displayed in "YYYY-MM-DD" format and as a number of days since January 1st 1970.

Usage

```
easter_dates(year0, year1, julian = FALSE)
```

Arguments

`year0`, `year1`	starting year and ending year
`julian`	Boolean indicating if Julian calendar must be used.

Value

a named numeric vector. Names are the dates in format "YYYY-MM-DD", values are number of days since January 1st 1970.

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

`national_calendar`, `easter_day`

Examples

```
# Dates from 2018(included) to 2023 (included)
easter_dates(2018, 2023)
```

easter_day *Set a Holiday on an Easter related day*

Description

Allows to define a holiday which date is related to Easter Sunday.

Usage

```
easter_day(offset, julian = FALSE, weight = 1, validity = NULL)
```

Arguments

offset	The position of the holiday in relation to Easter Sunday, measured in days (can be positive or negative).
julian	Boolean indicating if Julian calendar must be used.
weight	weight associated to the holiday.
validity	validity period: either `NULL` (full sample) or a named list with `"start"` and/or "end" dates in the format `"YYYY-MM-DD"`.

Value

returns an object of class `c("JD3_EASTERDAY","JD3_HOLIDAY")`

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

national_calendar, fixed_day,special_day,fixed_week_day

Examples

```
easter_day(1) # Easter Monday
easter_day(-2) # Easter Good Friday
# Corpus Christi 60 days after Easter
# Sunday in Julian calendar with weight 0.5, from January 2000 to December 2020
easter_day(
    offset = 60, julian = TRUE, weight = 0.5,
    validity = list(start = "2000-01-01", end = "2020-12-01")
)
```

`easter_variable` *Easter regressor*

Description

Allows to generate a regressor taking into account the (Julian) Easter effect in monthly or quarterly time series.

Usage

```
easter_variable(
  frequency,
  start,
  length,
  s,
  duration = 6,
  endpos = -1,
  correction = c("Simple", "PreComputed", "Theoretical", "None")
)

julianeaster_variable(frequency, start, length, s, duration = 6)
```

Arguments

`frequency`	Frequency of the series, number of periods per year (12, 4, 3, 2...)
`start, length`	First date (array with the first year and the first period, for instance `c(1980, 1)`) and number of periods of the output variables. Can also be provided with the `s` argument
`s`	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
`duration`	Duration (length in days) of the Easter effect. (value between 1 and 20, default =6)
`endpos`	Position of the end of the Easter effect, relatively to Easter: -1(default): before Easter Sunday, 0: on Easter Sunday, 1: on Easter Monday)
`correction`	mean correction option. Simple"(default), "PreComputed", "Theoretical" or "None".

Value

A time series (object of class `"ts"`)

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

calendar_td

Examples

```
# Monthly regressor, five-year long, duration 8 days, effect finishing on Easter Monday
ee <- easter_variable(12, c(2020, 1), length = 5 * 12, duration = 8, endpos = 1)
je <- julianeaster_variable(12, c(2020, 1), length = 5 * 12, duration = 8)
```

Electricity — *French national electricity consumtion*

Description

French national electricity consumtion

Usage

```
Electricity
```

Format

A data frame with 210384 rows and 3 variables:

- Date: Date of the event (from January 1, 2012 to December 31, 2023)
- Hours: Timestamp of the event (from 00:00 AM to 11:30 PM)
- Consumtion: number of daily birth (29124–102098)

Source

https://www.rte-france.com/en/data-publications/eco2mix/download-indicators

Examples

```
data(Electricity)
```

`Exports` *Belgian exports to European countries*

Description

Belgian exports to European countries

Usage

```
Exports
```

Format

An object of class `list` of length 34.

Source

NBB

Examples

```
data(Exports)
```

`fixed_day` *Set a holiday on a Fixed Day*

Description

It creates a holiday falling on a fixed day each year, with an optional weight and period of validity, like Christmas which is always celebrated on December 25th.

Usage

```
fixed_day(month, day, weight = 1, validity = NULL)
```

Arguments

`month, day`	the month and the day of the fixed day to add.
`weight`	weight associated to the holiday.
`validity`	validity period: either `NULL` (full sample) or a named list with `"start"` and/or "end" dates in the format `"YYYY-MM-DD"`.

Value

returns an object of class `c("JD3_FIXEDDAY","JD3_HOLIDAY")`

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

national_calendar, special_day,easter_day

Examples

```
day <- fixed_day(7, 21, .9)
day # July 21st, with weight=0.9, on the whole sample
day <- fixed_day(12, 25, .5, validity = list(start = "2010-01-01"))
day # December 25th, with weight=0.5, from January 2010
day <- fixed_day(12, 25, .5, validity = list(start = "1968-02-01", end = "2010-01-01"))
day # December 25th, with weight=0.9, from February 1968 until January 2010
```

---

fixed_week_day *Set a Holiday on a Fixed Week Day*

Description

Allows to define a holiday falling on a fixed week day each year, like Labour Day in the USA which is always celebrated on the first Monday of September.

Usage

```
fixed_week_day(month, week, dayofweek, weight = 1, validity = NULL)
```

Arguments

month	month of the holiday: from 1 (January) to 12 (December).
week	position of the specified week day in the month: from 1 (first week of the month) to 5. Should be always lower than 5. -1 for the last dayofweek of the month.
dayofweek	day of the week: from 1 (Monday) to 7 (Sunday).
weight	weight associated to the holiday.
validity	validity period: either NULL (full sample) or a named list with "start" and/or "end" dates in the format "YYYY-MM-DD".

Value

returns an object of class c("JD3_FIXEDWEEKDAY","JD3_HOLIDAY")

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

national_calendar, fixed_day,special_day,easter_day

Examples

```
day <- fixed_week_day(9, 1, 1) # first Monday(1) of September.
day
```

---

holidays *Daily calendar regressors corresponding to holidays*

Description

Allows to generate daily regressors (dummy variables) corresponding to each holiday of a pre-defined calendar.

Usage

```
holidays(
  calendar,
  start,
  length,
  nonworking = c(6, 7),
  type = c("Skip", "All", "NextWorkingDay", "PreviousWorkingDay"),
  single = FALSE
)
```

Arguments

calendar The calendar in which the holidays are defined.

start Starting date for the regressors, format "YYYY-MM-DD".

length Length of the regressors in days.

nonworking Indexes of non working days (Monday=1, Sunday=7).

type Adjustment type when a holiday falls on a week-end:

- "NextWorkingDay": the holiday is set to the next day,
- "PreviousWorkingDay": the holiday is set to the previous day,
- "Skip": holidays corresponding to non working days are simply skipped in the matrix,
- "All": (holidays are always put in the matrix, even if they correspond to a non working day.

single Boolean indication if a single variable (TRUE) should be returned or a matrix (FALSE, the default) containing the different holidays in separate columns.

Details

The pre-defined in a calendar has to be created with the functions national_calendar or weighted_calendar or weighted_calendar. A many regressors as defined holidays are generated, when the holiday occurs the value is 1 and 0 otherwise. This kind of non-aggregated regressors are used for calendar correction in daily data.

Value

A matrix (class "matrix") where each column is associated to a holiday (in the order of creation of the holiday) and each row to a date.

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

calendar_td

Examples

```
BE <- national_calendar(list(
    fixed_day(7, 21),
    special_day("NEWYEAR"),
    special_day("CHRISTMAS"),
    special_day("MAYDAY"),
    special_day("EASTERMONDAY"),
    special_day("ASCENSION"),
    special_day("WHITMONDAY"),
    special_day("ASSUMPTION"),
    special_day("ALLSAINTSDAY"),
    special_day("ARMISTICE")
))
q <- holidays(BE, "2021-01-01", 366 * 10, type = "All")
plot(apply(q, 1, max))
```

Imports *Belgian imports from European countries*

Description

Belgian imports from European countries

Usage

```
Imports
```

Format

An object of class `list` of length 34.

Source

NBB

Examples

```
data(Imports)
```

`intervention_variable` *Intervention variable*

Description

Function allowing to create external regressors as sequences of zeros and ones. The generated variables will have to be added with `add_usrdefvar` function will require a modelling context definition with `modelling_context` to be used in an estimation process.

Usage

```
intervention_variable(
  frequency,
  start,
  length,
  s,
  starts,
  ends,
  delta = 0,
  seasonaldelta = 0
)
```

Arguments

`frequency`	Frequency of the series, number of periods per year (12, 4, 3, 2...)
`start, length`	First date (array with the first year and the first period, for instance `c(1980, 1)`) and number of periods of the output variables. Can also be provided with the `s` argument
`s`	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
`starts, ends`	characters specifying sequences of starts/ends dates for the intervention variable. Can be characters or integers.
`delta`	regular differencing order.
`seasonaldelta`	seasonal differencing order.

Details

Intervention variables are combinations of any possible sequence of ones and zeros (the sequence of ones being defined by the parameters `starts` and `ends`) and of $\frac{1}{(1-B)^d}$ and $\frac{1}{(1-B^s)^D}$ where B is the backwards operator, s is the frequency of the time series, d and D are the parameters `delta` and `seasonaldelta`.

For example, with `delta = 0` and `seasonaldelta = 0` we get temporary level shifts defined by the parameters `starts` and `ends`. With `delta = 1` and `seasonaldelta = 0` we get the cumulative sum of temporary level shifts, once differenced the regressor will become a classical level shift.

Value

a `ts` object

References

More information on auxiliary variables in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

`modelling_context`, `add_usrdefvar`

Examples

```
iv1 <- intervention_variable(
    frequency = 12,
    start = c(2000, 1),
    length = 60,
    starts = "2001-01-01",
    ends = "2001-12-01"
)
plot(iv1)
iv2 <- intervention_variable(
    frequency = 12,
    start = c(2000, 1),
    length = 60,
    starts = "2001-01-01",
    ends = "2001-12-01",
    delta = 1
)
plot(iv2)

# Using one variable in a a seasonal adjustment process
# Regressors as a list of two groups reg1 and reg2
vars <- list(
    reg1 = list(x = iv1),
    reg2 = list(x = iv2)
)

# creating the modelling context
my_context <- modelling_context(variables = vars)
```

```
# customize a default specification
init_spec <- x13_spec_default
new_spec <- add_usrdefvar(init_spec, name = "reg1.iv1", regeffect = "Trend")
```

---

jd3_print *JD3 print functions*

Description

JD3 print functions

Usage

```
## S3 method for class 'JD3_ARIMA'
print(x, ...)

## S3 method for class 'JD3_UCARIMA'
print(x, ...)

## S3 method for class 'JD3_SARIMA'
print(x, ...)

## S3 method for class 'JD3_SARIMA_ESTIMATION'
print(x, digits = max(3L, getOption("digits") - 3L), ...)

## S3 method for class 'JD3_SPAN'
print(x, ...)

## S3 method for class 'JD3_LIKELIHOOD'
print(x, summary_info = getOption("summary_info"), ...)

## S3 method for class 'JD3_REGARIMA_RSLTS'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  summary_info = getOption("summary_info"),
  ...
)
```

Arguments

`x`	the object to print.
`...`	further unused parameters.
`digits`	minimum number of significant digits to be used for most numbers.

`summary_info`	boolean indicating if a message suggesting the use of the summary function for more details should be printed. By default used the option `"summary_info"` it used, which initialized to `TRUE`.

Value

The object is returned invisibly.

`ljungbox` *Ljung-Box Test*

Description

Compute Ljung-Box test to check the independence of a data.

Usage

```
ljungbox(data, k = 1, lag = 1, nhp = 0, sign = 0, mean = TRUE)
```

Arguments

`data`	data being tested.
`k`	number of auto-correlations used in the test
`lag`	number of lags used between two auto-correlations.
`nhp`	number of hyper parameters (to correct the degree of freedom)
`sign`	if `sign = 1`, only positive auto-correlations are considered in the test. If `sign = -1`, only negative auto-correlations are considered. If `sign = 0`, all auto-correlations are integrated in the test.
`mean`	Mean correction. If `TRUE`, the auto-correlations are computed as usual. If `FALSE`, we consider that the (known) mean is 0 and that the series has been corrected for it.

Value

A `c("JD3_TEST", "JD3")` object (see `statisticaltest()` for details).

Examples

```
ljungbox(random_t(2, 100), lag = 24, k = 1)
ljungbox(ABS$X0.2.09.10.M, lag = 24, k = 1)
```

`long_term_mean` *Display Long-term means for a set of calendar regressors*

Description

Given a pre-defined calendar and set of groups, the function displays the long-term means which would be used to seasonally adjust the corresponding regressors, as the final value using contrasts is "number of days in the group - long term mean".

Usage

```
long_term_mean(
  calendar,
  frequency,
  groups = c(1, 2, 3, 4, 5, 6, 0),
  holiday = 7
)
```

Arguments

`calendar`	The calendar containing the required holidays
`frequency`	Frequency of the series, number of periods per year (12, 4, 3, 2...)
`groups`	Groups of days. The length of the array must be 7. It indicates to what group each week day belongs. The first item corresponds to Mondays and the last one to Sundays. The group used for contrasts (usually Sundays) is identified by 0. The other groups are identified by 1, 2,... n (<= 6). For instance, usual trading days are defined by c(1, 2, 3, 4, 5, 6, 0), week days by c(1, 1, 1, 1, 1, 0, 0), week days, Saturdays, Sundays by c(1, 1, 1, 1, 1, 2, 0) etc.
`holiday`	Day to aggregate holidays with. (holidays are considered as that day). 1 for Monday... 7 for Sunday. Doesn't necessary belong to the 0-group.

Details

A long-term mean is a probability based computation of the average value for every period in every group. (see references). For monthly regressors there are 12 types of periods (January to December).

Value

returns an object of class `c("matrix","array")` with the long term means corresponding to each group/period, starting with the 0-group.

Examples

```
BE <- national_calendar(list(
    fixed_day(7, 21),
    special_day("NEWYEAR"),
    special_day("CHRISTMAS"),
```

```
    special_day("MAYDAY"),
    special_day("EASTERMONDAY"),
    special_day("ASCENSION"),
    special_day("WHITMONDAY"),
    special_day("ASSUMPTION"),
    special_day("ALLSAINTSDAY"),
    special_day("ARMISTICE")
))
lt <- long_term_mean(BE, 12,
    groups = c(1, 1, 1, 1, 1, 0, 0),
    holiday = 7
)
```

lp_variable *Leap Year regressor*

Description

Allows to generate a regressor correcting for the leap year or length-of-period effect.

Usage

```
lp_variable(
  frequency,
  start,
  length,
  s,
  type = c("LeapYear", "LengthOfPeriod")
)
```

Arguments

frequency	Frequency of the series, number of periods per year (12, 4, 3, 2...)
start, length	First date (array with the first year and the first period, for instance `c(1980, 1)`) and number of periods of the output variables. Can also be provided with the `s` argument
s	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
type	the modelling of the leap year effect: as a contrast variable (`type = "LeapYear"`, default) or by a length-of-month (or length-of-quarter; `type = "LengthOfPeriod"`).

Value

Time series (object of class `"ts"`)

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

calendar_td

Examples

```
# Leap years occur in year 2000, 2004, 2008 and 2012
lp_variable(4, start = c(2000, 1), length = 4 * 13)
lper <- lp_variable(12, c(2000, 1), length = 10 * 12, type = "LengthOfPeriod")
```

---

mad *Compute a robust median absolute deviation (MAD)*

Description

Compute a robust median absolute deviation (MAD)

Usage

```
mad(data, centile = 50, medianCorrected = TRUE)
```

Arguments

data	The data for which we compute the robust deviation
centile	The centile used to exclude extreme values (only the "centile" part of the data are is to compute the mad)
medianCorrected	TRUE if the series is corrected for its median, FALSE if the median is supposed to be 0

Value

The median absolute deviation

Examples

```
y <- rnorm(1000)
m <- rjd3toolkit::mad(y, centile = 70)
```

`modelling_context` *Create modelling context*

Description

Function allowing to include calendars and external regressors in a format that makes them usable in an estimation process (reg-arima or tramo modelling, stand alone or as pre-processing in seasonal adjustment). The regressors can be created with functions available in the package or come from any other source, provided they are `ts` class objects.

Usage

```
modelling_context(calendars = NULL, variables = NULL)
```

Arguments

`calendars`	list of calendars.
`variables`	list of variables.

Value

list of calendars and variables

References

More information on auxiliary variables in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

`add_usrdefvar`, `intervention_variable`

Examples

```
# Creating one or several external regressors (TS objects), which will
# be gathered in one or several groups
iv1 <- intervention_variable(12, c(2000, 1), 60,
    starts = "2001-01-01", ends = "2001-12-01"
)
iv2 <- intervention_variable(12, c(2000, 1), 60,
    starts = "2001-01-01", ends = "2001-12-01", delta = 1
)

# Regressors as a list of two groups reg1 and reg2
vars <- list(reg1 = list(x = iv1), reg2 = list(x = iv2))

# Creating the modelling context
my_context <- modelling_context(variables = vars)
```

monotonic_cspline *Monotonic cubic spline*

Description

Monotonic cubic spline

Usage

```
monotonic_cspline(x, y, pos)
```

Arguments

x	Abscissas of the knots
y	Ordinates of the knots
pos	Requested positions

Value

An array corresponding to the values of the spline at the requested positions

Examples

```
s<-monotonic_cspline(x = c(0,.2,.3, .9,.95), y= c(1,3,5,8,12), pos=seq(0,1,0.01))
plot(s, type='l')
```

national_calendar *Create a National Calendar*

Description

Will create a calendar as a list of days corresponding to the required holidays. The holidays have to be generated by one of these functions: `fixed_day()`, `fixed_week_day()`, `easter_day()`, `special_day()` or `single_day()`.

Usage

```
national_calendar(days = list(), mean_correction = TRUE)
```

Arguments

days	list of holidays to be taken into account in the calendar
mean_correction	TRUE if the variables generated by this calendar will contain long term mean corrections (default). FALSE otherwise.

Value

returns an object of class c("JD3_CALENDAR","JD3_CALENDARDEFINITION")

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

chained_calendar, weighted_calendar

Examples

```
# Fictional calendar using all possibilities to set the required holidays
MyCalendar <- national_calendar(list(
    fixed_day(7, 21),
    special_day("NEWYEAR"),
    special_day("CHRISTMAS"),
    fixed_week_day(7, 2, 3), # second Wednesday of July
    special_day("MAYDAY"),
    easter_day(1), # Easter Monday
    easter_day(-2), # Good Friday
    single_day("2001-09-11"), # appearing once
    special_day("ASCENSION"),
    easter_day(
        offset = 60, julian = FALSE, weight = 0.5,
        validity = list(start = "2000-01-01", end = "2020-12-01")
    ), # Corpus Christi
    special_day("WHITMONDAY"),
    special_day("ASSUMPTION"),
    special_day("ALLSAINTSDAY"),
    special_day("ARMISTICE")
))
```

natural_cspline *Natural cubic spline*

Description

Natural cubic spline

Usage

```
natural_cspline(x, y, pos)
```

Arguments

x	Abscissas of the knots
y	Ordinates of the knots
pos	Requested positions

Value

An array corresponding to the values of the spline at the requested positions

Examples

```
s<-natural_cspline(x = c(0,.2,.3, .9,.95), y= c(1,3,5,8,12), pos=seq(0,1,0.01))
plot(s, type='l')
```

---

normality_tests *Normality Tests*

Description

Set of functions to test the normality of a time series.

Usage

```
bowmanshenton(data)

doornikhansen(data)

jarquebera(data, k = 0, sample = TRUE)

skewness(data)

kurtosis(data)
```

Arguments

data	data being tested.
k	number of degrees of freedom to be subtracted if the input time series is a series of residuals.
sample	boolean indicating if unbiased empirical moments should be computed.

Value

A `c("JD3_TEST", "JD3")` object (see `statisticaltest` for details).

Functions

- bowmanshenton(): Bowman-Shenton test
- doornikhansen(): Doornik-Hansen test
- jarquebera(): Jarque-Bera test
- skewness(): Skewness test
- kurtosis(): Kurtosis test

Examples

```
x <- rnorm(100) # null
bowmanshenton(x)
doornikhansen(x)
jarquebera(x)
skewness(x)
kurtosis(x)

x <- random_t(2, 100) # alternative
bowmanshenton(x)
doornikhansen(x)
jarquebera(x)
skewness(x)
kurtosis(x)
```

---

outliers_variables *Generating Outlier regressors*

Description

Generating Outlier regressors

Usage

```
ao_variable(frequency, start, length, s, pos, date = NULL)

tc_variable(frequency, start, length, s, pos, date = NULL, rate = 0.7)

ls_variable(frequency, start, length, s, pos, date = NULL, zeroended = TRUE)

so_variable(frequency, start, length, s, pos, date = NULL, zeroended = TRUE)
```

Arguments

frequency	Frequency of the series, number of periods per year (12, 4, 3, 2...)
start, length	First date (array with the first year and the first period, for instance c(1980, 1)) and number of periods of the output variables. Can also be provided with the s argument

`s`	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
`pos, date`	the date of the outlier, defined by the position in period compared to the first date (`pos` parameter) or by a specific `date` defined in the format `"YYYY-MM-DD"`.
`rate`	the decay rate of the transitory change regressor (see details).
`zeroended`	Boolean indicating if the regressor should end by 0 (`zeroended = TRUE`, default) or 1 (`zeroended = FALSE`), argument valid only for LS and SO.

Details

An additive outlier (AO, `ao_variable`) is defined as:

$$AO_t = \begin{cases} 1 & \text{if } t = t_0 \\ 0 & \text{if } t \neq t_0 \end{cases}$$

A level shift (LS, `ls_variable`) is defined as (if `zeroended = TRUE`):

$$LS_t = \begin{cases} -1 & \text{if } t < t_0 \\ 0 & \text{if } t \geq t_0 \end{cases}$$

A transitory change (TC, `tc_variable`) is defined as:

$$TC_t = \begin{cases} 0 & \text{if } t < t_0 \\ \alpha^{t-t_0} & t \geq t_0 \end{cases}$$

A seasonal outlier (SO, `so_variable`) is defined as (if `zeroended = TRUE`):

$$SO_t = \begin{cases} 0 & \text{if } t \geq t_0 \\ -1 & \text{if } t < t_0 \text{ and } t \text{ same periode as } t_0 \\ -\frac{1}{s-1} & \text{otherwise} \end{cases}$$

Value

a `ts` object

Examples

```
# Outliers in February 2002
ao <- ao_variable(12, c(2000, 1), length = 12 * 4, date = "2002-02-01")
ls <- ls_variable(12, c(2000, 1), length = 12 * 4, date = "2002-02-01")
tc <- tc_variable(12, c(2000, 1), length = 12 * 4, date = "2002-02-01")
so <- so_variable(12, c(2000, 1), length = 12 * 4, date = "2002-02-01")
plot.ts(ts.union(ao, ls, tc, so),
    plot.type = "single",
    col = c("black", "orange", "green", "gray")
)
```

periodic_bsplines   *Periodic B-Splines*

Description

Periodic B-Splines

Usage

```
periodic_bsplines(order = 4, period = 1, knots, pos)
```

Arguments

order	Order of the splines (4 for cubic)
period	Period of the splines (1 by default)
knots	Knots of the splines (in [0, period[)
pos	Requested positions (in [0, period[). The rows of the returned matrix will correspond to those positions

Value

A matrix (len(pos) x len(knots))

Examples

```
s<-periodic_bsplines(knots = c(0,.2,.3, .9,.95), pos=seq(0,1,0.01))
matplot(s, type='l')
```

periodic_cspline   *Periodic cubic spline*

Description

Periodic cubic spline

Usage

```
periodic_cspline(x, y, pos)
```

Arguments

x	Abscissas of the knots
y	Ordinates of the knots
pos	Requested positions

Value

An array corresponding to the values of the spline at the requested positions

Examples

```
s<-periodic_cspline(x = c(0,.2,.3, .9,.95, 1), y= c(1,3,8,5,12, 1), pos=seq(0,1,0.01))
plot(s, type='l')
```

---

periodic_csplines *Periodic cardinal cubic splines*

Description

Periodic cardinal cubic splines

Usage

```
periodic_csplines(x, pos)
```

Arguments

x	Abscissas of the knots
pos	Requested positions

Value

A matrix (len(pos) x len(knots))

Examples

```
s<-periodic_csplines(x = c(0,.2,.3, .9,.95, 1), pos=seq(0,1,0.01))
matplot(s, type='l')
```

periodic_dummies *Periodic dummies and contrasts*

Description

Periodic dummies and contrasts

Usage

```
periodic_dummies(frequency, start, length, s)

periodic_contrasts(frequency, start, length, s)
```

Arguments

frequency	Frequency of the series, number of periods per year (12, 4, 3, 2...)
start, length	First date (array with the first year and the first period, for instance c(1980, 1)) and number of periods of the output variables. Can also be provided with the s argument
s	time series used to get the dates for the trading days variables. If supplied the parameters frequency, start and length are ignored.

Details

The function periodic_dummies() creates as many time series as types of periods in a year (4 or 12) with the value one only for one given type of period (ex Q1) The periodic_contrasts() function is based on periodic_dummies but adds -1 to the period preceding a 1.

Value

a mts object with frequency column

Examples

```
# periodic dummies for a quarterly series
p <- periodic_dummies(4, c(2000, 1), 60)
# periodic contrasts for a quarterly series
q <- periodic_contrasts(4, c(2000, 1), 60)
q[1:9, ]
```

print_calendars — *Calendars Print Methods*

Description

Print functions for calendars

Usage

```
## S3 method for class 'JD3_FIXEDDAY'
print(x, ...)

## S3 method for class 'JD3_FIXEDWEEKDAY'
print(x, ...)

## S3 method for class 'JD3_EASTERDAY'
print(x, ...)

## S3 method for class 'JD3_SPECIALDAY'
print(x, ...)

## S3 method for class 'JD3_SINGLEDAY'
print(x, ...)

## S3 method for class 'JD3_CALENDAR'
print(x, ...)

## S3 method for class 'JD3_CHAINEDCALENDAR'
print(x, ...)

## S3 method for class 'JD3_WEIGHTEDCALENDAR'
print(x, ...)
```

Arguments

x	The object.
...	other unused parameters.

Value

The object is returned invisibly.

r2jd_calendarts *Create Java CalendarTimeSeries*

Description

Create Java CalendarTimeSeries

Usage

```
r2jd_calendarts(calendarobs)
```

Arguments

calendarobs	list.

Value

a Java object

Examples

```
# example code
obs <- list(
    list(start = as.Date("1980-01-01"), end = as.Date("1999-12-31"), value = 2000),
    list(start = as.Date("2000-01-01"), end = as.Date("2010-01-01"), value = 1000)
)
jobj <- r2jd_calendarts(obs)
```

ramp_variable *Ramp regressor*

Description

Ramp regressor

Usage

```
ramp_variable(frequency, start, length, s, range)
```

Arguments

`frequency`	Frequency of the series, number of periods per year (12, 4, 3, 2...)
`start, length`	First date (array with the first year and the first period, for instance `c(1980, 1)`) and number of periods of the output variables. Can also be provided with the `s` argument
`s`	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
`range`	the range of the regressor. A vector of length 2 containing the datesin the format `"YYYY-MM-DD"` or the position in the series, in number of periods from counting from the series start.

Details

A ramp between two dates t_0 and t_1 is defined as:

$$RP_t = \begin{cases} -1 & \text{if } t \geq t_0 \\ \frac{t-t_0}{t_1-t_0} - 1 & t_0 < t < t_1 \\ 0 & t \leq t_1 \end{cases}$$

Value

a `ts` object

Examples

```
# Ramp variable from January 2001 to September 2001
rp <- ramp_variable(12, c(2000, 1), length = 12 * 4, range = c(13, 21))
# Or equivalently
rp <- ramp_variable(12, c(2000, 1), length = 12 * 4, range = c("2001-01-01", "2001-09-02"))
plot.ts(rp)
```

`rangemean_tstat` *Range-Mean Regression*

Description

Function to perform a range-mean regression, trimmed to avoid outlier distortion. The can be used to select whether the original series will be transformed into log or maintain in level.

Usage

```
rangemean_tstat(data, period = 0, groupsize = 0, trim = 0)
```

Arguments

- `data` — data to test.
- `period` — periodicity of the data.
- `groupsize` — number of observations per group (before being trimmed). The default group size (`groupsize = 0`) is computed as followed:
  - if `period = 12` or `period = 6`, it is equal to `12`;
  - if `period = 4` it is equal to `12` if the data has at least 166 observations, `8` otherwise;
  - if `period = 3` or `period = 2` it is equal to `12` if the data has at least 166 observations, `6` otherwise;
  - if `period = 1` it is equal to `9` if the data has at least 166 observations, `5` otherwise;
  - it is equal to `period` otherwise.
- `trim` — number of trimmed observations.

Details

First, the data is divided into n groups of successive observations of length l (`groupsize`). That is, the first group is formed with the first l observations, the second group is formed with observations $1+l$ to $2l$, etc. Then, for each group i, the observations are sorted and the `trim` smallest and largest observations are rejected (to avoid outlier distortion). With the other observations, the range (noted y_i) and mean (noted m_i) are computed.

Finally, the following regression is performed :

$$y_t = \alpha + \beta m_t + u_t.$$

The function `rangemean_tstat` returns the T-statistic associated to β. If it is significantly higher than 0, log transformation is recommended.

Value

T-Stat of the slope of the range-mean regression.

Examples

```
y <- ABS$X0.2.09.10.M
# Multiplicative pattern
plot(y)
period <- 12
rm_t <- rangemean_tstat(y, period = period, groupsize = period)
rm_t # higher than 0
# Can be tested:
pt(rm_t, period - 2, lower.tail = FALSE)
# Or :
1 - cdf_t(period - 2, rm_t)

# Close to 0
rm_t_log <- rangemean_tstat(log(y), period = period, groupsize = period)
```

```
rm_t_log
pt(rm_t_log, period - 2, lower.tail = FALSE)
```

reload_dictionaries *Reload dictionaries*

Description

Reload dictionaries

Usage

```
reload_dictionaries()
```

Value

invisibly NULL

Examples

```
reload_dictionaries()
```

Retail *US Retail trade statistics*

Description

US Retail trade statistics

Usage

```
Retail
```

Format

An object of class list of length 62.

Source

US-Census Bureau

Examples

```
data(Retail)
```

`runstests` *Runs Tests around the mean or the median*

Description

Functions to compute runs test around the mean or the median (`testofruns`) or up and down runs test (`testofupdownruns`) to check randomness of a data.

Usage

```
testofruns(data, mean = TRUE, number = TRUE)

testofupdownruns(data, number = TRUE)
```

Arguments

`data`	data being tested.
`mean`	If `TRUE`, runs around the mean. Otherwise, runs around the median.
`number`	If `TRUE`, test the number of runs. Otherwise, test the lengths of the runs.

Value

A `c("JD3_TEST", "JD3")` object (see `statisticaltest()` for details).

Functions

- `testofruns()`: Runs test around mean or median
- `testofupdownruns()`: up and down runs test

Examples

```
x <- random_t(5, 1000)
# random values
testofruns(x)
testofupdownruns(x)
# non-random values
testofruns(ABS$X0.2.09.10.M)
testofupdownruns(ABS$X0.2.09.10.M)
```

sadecomposition *Generic Function for Seasonal Adjustment Decomposition*

Description

Generic function to format the seasonal adjustment decomposition components. `sa_decomposition()` is a generic function defined in other packages.

Usage

```
sadecomposition(y, sa, t, s, i, mul)

## S3 method for class 'JD3_SADECOMPOSITION'
print(x, n_last_obs = frequency(x$series), ...)

## S3 method for class 'JD3_SADECOMPOSITION'
plot(
  x,
  first_date = NULL,
  last_date = NULL,
  type_chart = c("sa-trend", "seas-irr"),
  caption = c(`sa-trend` = "Y, Sa, trend", `seas-irr` = "Sea., irr.")[type_chart],
  colors = c(y = "#F0B400", t = "#1E6C0B", sa = "#155692", s = "#1E6C0B", i = "#155692"),
  ...
)

sa_decomposition(x, ...)
```

Arguments

`y, sa, t, s, i, mul`	seasonal adjustment decomposition parameters.
`x`	the object to print.
`n_last_obs`	number of observations to print (by default equal to the frequency of the series).
`...`	further arguments.
`first_date, last_date`	first and last date to plot (by default all the data is used).
`type_chart`	the chart to plot: `"sa-trend"` (by default) plots the input time series, the seasonally adjusted and the trend; `"seas-irr"` plots the seasonal and the irregular components.
`caption`	the caption of the plot.
`colors`	the colours used in the plot.

Value

`"JD3_SADECOMPOSITION"` object.

Examples

```
decompo <- sadecomposition(
    y = ts(c(112, 118, 132, 129, 121, 135), start = 2000, frequency = 12L),
   sa = ts(c(121.72, 124.52, 125.4, 128.91, 128.84, 126.73), start = 2000, frequency = 12L),
   t = ts(c(122.24, 124.33, 126.21, 127.61, 127.8, 126.94), start = 2000, frequency = 12L),
    s = ts(c(0.92, 0.95, 1.05, 1, 0.94, 1.07), start = 2000, frequency = 12L),
    i = ts(c(1, 1, 0.99, 1.01, 1.01, 1), start = 2000, frequency = 12L),
    mul = TRUE
)
print(decompo)
plot(decompo)
```

---

sarima_decompose *Decompose SARIMA Model into three components trend, seasonal, irregular*

Description

Decompose SARIMA Model into three components trend, seasonal, irregular

Usage

```
sarima_decompose(model, rmod = 0, epsphi = 0)
```

Arguments

model	SARIMA model to decompose.
rmod	trend threshold.
epsphi	seasonal tolerance (in degrees).

Value

An UCARIMA model

Examples

```
model <- sarima_model(period = 12, d = 1, bd = 1, theta = -0.6, btheta = -0.5)
ucm <- sarima_decompose(model)
```

`sarima_estimate` *Estimate SARIMA Model*

Description

Estimate SARIMA Model

Usage

```
sarima_estimate(
  x,
  order = c(0, 0, 0),
  seasonal = list(order = c(0, 0, 0), period = NA),
  mean = FALSE,
  xreg = NULL,
  eps = 1e-09
)
```

Arguments

`x`	an univariate time series (class Ts object).
`order`	vector specifying of the non-seasonal part of the ARIMA model: the AR order, the degree of differencing, and the MA order.
`seasonal`	specification of the seasonal part of the ARIMA model and the seasonal frequency (by default equals to `frequency(x)`). Either a list with components `order` and `period` or a numeric vector specifying the seasonal order (the default period is then used).
`mean`	should the SARIMA model include an intercept term.
`xreg`	vector or matrix of external regressors.
`eps`	precision.

Value

An object of class `JD3_SARIMA_ESTIMATE` containing:

- the estimated parameters,
- the raw data,
- the regressors,
- the standard errors,
- the log-likelihood (with the number of observations, the number of effective observations, the number of parameters, the log-likelihood, the adjusted log-likelihood, the AIC, the AICC, the BIC, the BICC, and the sum of squares),
- the residuals,
- the orders of the model.

Examples

```
y <- ABS$X0.2.09.10.M
sarima_estimate(y, order = c(0, 1, 1), seasonal = c(0, 1, 1))
```

---

sarima_hannan_rissanen
Estimate ARIMA Model with Hannan-Rissanen method

Description

Estimate ARIMA Model with Hannan-Rissanen method

Usage

```
sarima_hannan_rissanen(
  x,
  order = c(0, 0, 0),
  seasonal = list(order = c(0, 0, 0), period = NA),
  initialization = c("Ols", "Levinson", "Burg"),
  biasCorrection = TRUE,
  finalCorrection = TRUE
)
```

Arguments

`x`	an univariate time series (TS object).
`order`	vector specifying of the non-seasonal part of the ARIMA model: the AR order, the degree of differencing, and the MA order.
`seasonal`	specification of the seasonal part of the ARIMA model and the seasonal frequency (by default equals to `frequency(x)`). Either a list with components `order` and `period` or a numeric vector specifying the seasonal order (the default period is then used).
`initialization`	Algorithm used in the computation of the long order auto-regressive model (used to estimate the innovations)
`biasCorrection`	Bias correction
`finalCorrection`	Final correction as implemented in Tramo

Value

An object of class `JD3_SARIMA` with the estimated coefficient.

Examples

```
y <- ABS$X0.2.09.10.M
model<- sarima_hannan_rissanen(y, order = c(0, 1, 1), seasonal = c(0, 1, 1))
```

`sarima_model` *Seasonal ARIMA model (Box-Jenkins)*

Description

Seasonal ARIMA model (Box-Jenkins)

Usage

```
sarima_model(
  name = "sarima",
  period,
  phi = NULL,
  d = 0,
  theta = NULL,
  bphi = NULL,
  bd = 0,
  btheta = NULL
)
```

Arguments

name	name of the model.
period	period of the model.
phi	coefficients of the regular auto-regressive polynomial $(1 + \phi_1 B + \phi_2 B + ...)$. True signs.
d	regular differencing order.
theta	coefficients of the regular moving average polynomial $(1 + \theta_1 B + \theta_2 B + ...)$. True signs.
bphi	coefficients of the seasonal auto-regressive polynomial. True signs.
bd	seasonal differencing order.
btheta	coefficients of the seasonal moving average polynomial. True signs.

Value

A "JD3_SARIMA" model.

Examples

```
mod1 <- sarima_model(period = 12, d = 1, bd = 1, theta = 0.2, btheta = 0.2)
```

sarima_properties *SARIMA Properties*

Description

SARIMA Properties

Usage

```
sarima_properties(model, nspectrum = 601, nacf = 36)
```

Arguments

model	a "JD3_SARIMA" model (created with sarima_model()).
nspectrum	number of points in [0, pi] to calculate the spectrum.
nacf	maximum lag at which to calculate the acf.

Value

List with the acf and the spectrum of the model.

Examples

```
mod1 <- sarima_model(period = 12, d = 1, bd = 1, theta = 0.2, btheta = 0.2)
sarima_properties(mod1)
```

sarima_random *Simulate Seasonal ARIMA*

Description

Simulate Seasonal ARIMA

Usage

```
sarima_random(model, length, stde = 1, tdegree = 0, seed = -1)
```

Arguments

model	a "JD3_SARIMA" model (see sarima_model() function).
length	length of the output series.
stde	deviation of the normal distribution of the innovations of the simulated series. Unused if tdegree is larger than 0.
tdegree	degrees of freedom of the T distribution of the innovations. tdegree = 0 if normal distribution is used.
seed	seed of the random numbers generator. Negative values mean random seeds.

Value

a numeric vector with the simulated series.

Examples

```
# Airline model
s_model <- sarima_model(period = 12, d = 1, bd = 1, theta = 0.2, btheta = 0.2)
x <- sarima_random(s_model, length = 64, seed = 0)
plot(x, type = "l")
```

---

sa_preprocessing *Generic Preprocessing Function*

Description

Generic function for preprocessing defined in other packages.

Usage

```
sa_preprocessing(x, ...)
```

Arguments

x, ...	parameters.

Value

a list, the preprocessing part of a model.

---

seasonality_canovahansen *Canova-Hansen seasonality test*

Description

Canova-Hansen seasonality test

Usage

```
seasonality_canovahansen(
  data,
  period,
  type = c("Contrast", "Dummy", "Trigonometric"),
  lag1 = TRUE,
  kernel = c("Bartlett", "Square", "Welch", "Tukey", "Hamming", "Parzen"),
  order = NA,
  start = 1
)
```

Arguments

`data`	the input data.
`period`	Tested periodicity. Can be missing if the input is a time series
`type`	Trigonometric variables, seasonal dummies or seasonal contrasts.
`lag1`	Lagged variable in the regression model.
`kernel`	Kernel used to compute the robust Newey-West covariance matrix.
`order`	The truncation parameter used to compute the robust Newey-West covariance matrix.
`start`	Position of the first observation of the series

Value

list with the FTest on seasonal variables, the joint test and the details for the stability of the different seasonal variables

Examples

```
s <- log(ABS$X0.2.20.10.M)
seasonality_canovahansen(s, 12, type = "Contrast")
seasonality_canovahansen(s, 12, type = "Trigonometric")
```

---

seasonality_canovahansen_trigs

Canova-Hansen test using trigonometric variables

Description

Canova-Hansen test using trigonometric variables

Usage

```
seasonality_canovahansen_trigs(
  data,
  periods,
  lag1 = TRUE,
  kernel = c("Bartlett", "Square", "Welch", "Tukey", "Hamming", "Parzen"),
  order = NA,
  original = FALSE
)
```

Arguments

`data`	the input data.
`periods`	Periodicities.
`lag1`	Lagged variable in the regression model.
`kernel`	Kernel used to compute the robust Newey-West covariance matrix.
`order`	The truncation parameter used to compute the robust Newey-West covariance matrix.
`original`	`TRUE` for original algorithm, `FALSE` for solution proposed by T. Proietti (based on Ox code).

Value

a numeric vector

Examples

```
s <- log(ABS$X0.2.20.10.M)
freqs <- seq(0.01, 0.5, 0.001)
sct <- seasonality_canovahansen_trigs(s, 1 / freqs, original = FALSE)
plot(sct, type = "l")
```

---

seasonality_combined *"X12" Test On Seasonality*

Description

"X12" Test On Seasonality

Usage

```
seasonality_combined(
  data,
  period = NA,
  firstperiod = cycle(data)[1],
  mul = TRUE
)
```

Arguments

`data`	the input data.
`period`	Tested periodicity. Can be missing if the input is a time series
`firstperiod`	Position in a cycle of the first obs. For example, for a monthly, `firstperiod` = 1 means January. If `data` is not a `"ts"` object, `firstperiod` = 1 by default.
`mul`	boolean indicating if the seasonal decomposition is multiplicative (`mul` = `TRUE`) or additive (`mul` = `FALSE`).

Details

Combined test on the presence of identifiable seasonality (see Ladiray and Quenneville, 1999).

Value

a `list` with several seasonnality tests (kruskalwallis, stable and evolutive)

Examples

```
s <- do_stationary(log(ABS$X0.2.09.10.M))$ddata
seasonality_combined(s)
seasonality_combined(random_t(2, 1000), 7)
```

`seasonality_f` *F-test on seasonal dummies*

Description

F-test on seasonal dummies

Usage

```
seasonality_f(data, period = NA, model = c("AR", "D1", "WN"), nyears = 0)
```

Arguments

`data`	the input data.
`period`	Tested periodicity. Can be missing if the input is a time series
`model`	the model to use for the residuals.
`nyears`	Number of periods or number of cycles considered in the test, at the end of the series: in periods (positive value) or years (negative values). By default (`nyears = 0`), the entire sample is used.

Details

Estimation of a model with seasonal dummies. Joint F-test on the coefficients of the dummies.

Value

A `c("JD3_TEST", "JD3")` object (see `statisticaltest()` for details).

Examples

```
seasonality_f(ABS$X0.2.09.10.M, model = "D1")
seasonality_f(random_t(2, 1000), 7)
```

`seasonality_friedman` *Friedman Seasonality Test*

Description

Friedman Seasonality Test

Usage

```
seasonality_friedman(data, period = NA, nyears = 0)
```

Arguments

`data`	the input data.
`period`	Tested periodicity. Can be missing if the input is a time series
`nyears`	Number of periods or number of cycles considered in the test, at the end of the series: in periods (positive value) or years (negative values). By default (`nyears = 0`), the entire sample is used.

Details

Non parametric test ("ANOVA"-type).

Value

A `c("JD3_TEST", "JD3")` object (see `statisticaltest()` for details).

Examples

```
s <- do_stationary(log(ABS$X0.2.09.10.M))$ddata
seasonality_friedman(s)
seasonality_friedman(random_t(2, 1000), 12)
```

`seasonality_kruskalwallis` *Kruskall-Wallis Seasonality Test*

Description

Kruskall-Wallis Seasonality Test

Usage

```
seasonality_kruskalwallis(data, period, nyears = 0)
```

Arguments

`data`	the input data.
`period`	Tested periodicity. Can be missing if the input is a time series
`nyears`	Number of periods or number of cycles considered in the test, at the end of the series: in periods (positive value) or years (negative values). By default (`nyears = 0`), the entire sample is used.

Details

Non parametric test on the ranks.

Value

A `c("JD3_TEST", "JD3")` object (see `statisticaltest()` for details).

Examples

```
s <- do_stationary(log(ABS$X0.2.09.10.M))$ddata
seasonality_kruskalwallis(s)
seasonality_kruskalwallis(random_t(2, 1000), 7)
```

---

seasonality_modified_qs *Modified QS Seasonality Test (Maravall)*

Description

Modified QS Seasonality Test (Maravall)

Usage

```
seasonality_modified_qs(data, period = NA, nyears = 0)
```

Arguments

`data`	the input data.
`period`	Tested periodicity. Can be missing if the input is a time series
`nyears`	Number of periods or number of cycles considered in the test, at the end of the series: in periods (positive value) or years (negative values). By default (`nyears = 0`), the entire sample is used.

Details

Thresholds for p-values: p.9=2.49, p.95=3.83, p.99=7.06, p.999=11.88. Computed on 100.000.000 random series (different lengths). Remark: the length of the series has some impact on the p-values, mainly on short series. Not critical.

Value

The value of the test

Examples

```
s <- do_stationary(log(ABS$X0.2.09.10.M))$ddata
seasonality_modified_qs(s)
```

---

seasonality_periodogram

Periodogram Seasonality Test

Description

Periodogram Seasonality Test

Usage

```
seasonality_periodogram(data, period = NA, nyears = 0)
```

Arguments

`data`	the input data.
`period`	Tested periodicity. Can be missing if the input is a time series
`nyears`	Number of periods or number of cycles considered in the test, at the end of the series: in periods (positive value) or years (negative values). By default (`nyears = 0`), the entire sample is used.

Details

Tests on the sum of a periodogram at seasonal frequencies.

Value

A `c("JD3_TEST", "JD3")` object (see `statisticaltest()` for details).

Examples

```
s <- do_stationary(log(ABS$X0.2.09.10.M))$ddata
seasonality_periodogram(s)
seasonality_periodogram(random_t(2, 1000), 7)
```

seasonality_qs *QS (seasonal Ljung-Box) test.*

Description

QS (seasonal Ljung-Box) test.

Usage

```
seasonality_qs(data, period = NA, nyears = 0, type = 1)
```

Arguments

data	the input data.
period	Tested periodicity. Can be missing if the input is a time series
nyears	Number of periods or number of cycles considered in the test, at the end of the series: in periods (positive value) or years (negative values). By default (nyears = 0), the entire sample is used.
type	1 for positive autocorrelations, -1 for negative autocorrelations, 0 for all autocorrelations. By default (type = 1)

Value

A c("JD3_TEST", "JD3") object (see statisticaltest() for details).

Examples

```
s <- do_stationary(log(ABS$X0.2.09.10.M))$ddata
seasonality_qs(s)
seasonality_qs(random_t(2, 1000), 7)
```

set_arima *Set ARIMA Model Structure in Pre-Processing Specification*

Description

Function allowing to customize the ARIMA model structure when the automatic modelling is disabled.(see example)

Usage

```
set_arima(
  x,
  mean = NA,
  mean.type = c(NA, "Undefined", "Fixed", "Initial"),
  p = NA,
  d = NA,
  q = NA,
  bp = NA,
  bd = NA,
  bq = NA,
  coef = NA,
  coef.type = c(NA, "Undefined", "Fixed", "Initial")
)
```

Arguments

x	the specification to customize, must be a "SPEC" class object (see details).
mean	to fix the coefficient of the mean. If `mean = 0`, the mean is disabled.
mean.type	a character defining the mean coefficient estimation procedure. Possible procedures are: `"Undefined"` = no use of any user-defined input (i.e. coefficient is estimated), `"Fixed"` = the coefficients are fixed at the value provided by the user, `"Initial"` = the value defined by the user is used as the initial condition.
p, d, q, bp, bd, bq	to specify the order of the SARIMA model in the form ARIMA(p,d,q)(bp,bd,bd).
coef	a vector providing the coefficients for the regular and seasonal AR and MA polynomials. The vector length must be equal to the sum of the regular and seasonal AR and MA orders. The coefficients shall be provided in the following order: regular AR (*Phi*; `p` elements), regular MA (*Theta*; `q` elements), seasonal AR (*BPhi*; `bp` elements) and seasonal MA (*BTheta*; `bq` elements). E.g.: `arima.coef=c(0.6,0.7)` with `p=1, q=0,bp=1` and `bq=0`.
coef.type	a vector defining the ARMA coefficients estimation procedure. Possible procedures are: `"Undefined"` = no use of any user-defined input (i.e. coefficients are estimated), `"Fixed"` = the coefficients are fixed at the value provided by the user, `"Initial"` = the value defined by the user is used as the initial condition.

Details

`x` specification parameter must be a JD3_X13_SPEC" class object generated with `rjd3x13::x13_spec()` (or "JD3_REGARIMA_SPEC" generated with `rjd3x13::spec_regarima()` or "JD3_TRAMOSEATS_SPEC" generated with `rjd3tramoseats::spec_tramoseats()` or "JD3_TRAMO_SPEC" generated with `rjd3tramoseats::spec_tramo()`).

Value

The modified specification (with new ARIMA model)

References

More information on reg-arima modelling in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

set_automodel, set_transform

Examples

```
# Customize a default specification
init_spec <- x13_spec_default

# Disable automatic arima modelling
new_spec <- set_automodel(init_spec, enabled = FALSE)

# Customize arima model
new_spec <- set_arima(
    x = new_spec,
    mean = 0.2,
    mean.type = "Fixed",
    p = 1,
    d = 2,
    q = 0,
    bp = 1,
    bd = 1,
    bq = 0,
    coef = c(0.6, 0.7),
    coef.type = c("Initial", "Fixed")
)
```

---

set_automodel *Set Arima Model Identification in Pre-Processing Specification*

Description

Function allowing to customize Arima model identification procedure.

Usage

```
set_automodel(
  x,
  enabled = NA,
  acceptdefault = NA,
  cancel = NA,
  ub1 = NA,
  ub2 = NA,
  reducecv = NA,
```

```
  ljungboxlimit = NA,
  tsig = NA,
  ubfinal = NA,
  checkmu = NA,
  mixed = NA,
  balanced = NA,
  amicompare = NA
)
```

Arguments

`x` the specification to customize, must be a "SPEC" class object (see details).

`enabled` `logical`. If `TRUE`, the automatic modelling of the ARIMA model is enabled. If `FALSE`, the parameters of the ARIMA model can be specified.

`acceptdefault` `logical`. If `TRUE`, the default model (ARIMA(0,1,1)(0,1,1)) will be chosen in the first step of the automatic model identification, if the Ljung-Box Q statistics for the residuals are acceptable. No further attempt will be made to identify a better model. Default = `FALSE`

`cancel` `numeric` cancellation limit. A limit for the AR and the MA roots to be assumed equal. This option is used in the automatic identification of the differencing order. If the difference in moduli of an AR and an MA root (when estimating ARIMA(1,0,1)(1,0,1) models in the second step of the automatic identification of the differencing polynomial) is smaller than cancellation limit, the two roots cancel out. Default = 0.1.

`ub1` `numeric`, the first unit root limit. It is the threshold value for the initial unit root test in the automatic differencing procedure. When one of the roots in the estimation of the ARIMA(2,0,0)(1,0,0) plus mean model, performed in the first step of the automatic model identification procedure, is larger than first unit root limit in modulus, it is set equal to unity. Default = 1.030928.

`ub2` `numeric`, the second unit root limit. When one of the roots in the estimation of the ARIMA(1,0,1)(1,0,1) plus mean model, which is performed in the second step of the automatic model identification procedure, is larger than second unit root limit in modulus, it is checked if there is a common factor in the corresponding AR and MA polynomials of the ARMA model that can be cancelled (see `automdl.cancel`). If there is no cancellation, the AR root is set equal to unity (i.e. the differencing order changes). Default = 1.136364.

`reducecv` `numeric`, ReduceCV. The percentage by which the outlier critical value will be reduced when an identified model is found to have a Ljung-Box statistic with an unacceptable confidence coefficient. The parameter should be between 0 and 1, and will only be active when automatic outlier identification is enabled. The reduced critical value will be set to (1 - ReduceCV) x CV, where CV is the original critical value. Default = 0.14268.

`ljungboxlimit` `numeric`, the Ljung Box limit, setting the acceptance criterion for the confidence intervals of the Ljung-Box Q-statistic. If the LjungBox Q statistics for the residuals of a final model is greater than Ljung Box limit, then the model is rejected, the outlier critical value is reduced, and model and outlier identification (if specified) is redone with a reduced value. Default = 0.95.

tsig	numeric, the arma limit. It is the threshold value for t-statistics of ARMA coefficients and the constant term used for the final test of model parsimony. If the highest order ARMA coefficient has a t-value smaller than this value in magnitude, the order of the model is reduced. If the constant term has a t-value smaller than the ARMA limit in magnitude, it is removed from the set of regressors. Default=1.
ubfinal	(REGARIMA/X13 Specific) numeric, final unit root limit. The threshold value for the final unit root test. If the magnitude of an AR root for the final model is smaller than the final unit root limit, then a unit root is assumed, the order of the AR polynomial is reduced by one and the appropriate order of the differencing (non-seasonal, seasonal) is increased. The parameter value should be greater than one. Default = 1.05.
checkmu	(REGARIMA/X13 Specific) logical indicating if the automatic model selection checks the significance of the constant term.
mixed	(REGARIMA/X13 Specific) logical. This variable controls whether ARIMA models with non-seasonal AR and MA terms or seasonal AR and MA terms will be considered in the automatic model identification procedure. If FALSE, a model with AR and MA terms in both the seasonal and non-seasonal parts of the model can be acceptable, provided there are no AR or MA terms in either the seasonal or non-seasonal terms.
balanced	(REGARIMA/X13 Specific) logical If TRUE, the automatic model identification procedure will have a preference for balanced models (i.e. models for which the order of the combined AR and differencing operators is equal to the order of the combined MA operators). Default = FALSE
amicompare	(TRAMO Specific) logical. If TRUE, the program compares the model identified by the automatic procedure to the default model ($ARIMA(0,1,1)(0,1,1)$) and the model with the best fit is selected. Criteria considered are residual diagnostics, the model structure and the number of outliers.

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with rjd3x13::x13_spec() (or "JD3_REGARIMA_SPEC" generated with rjd3x13::spec_regarima() or "JD3_TRAMOSEATS_SPEC" generated with rjd3tramoseats::spec_tramoseats() or "JD3_TRAMO_SPEC" generated with rjd3tramoseats::spec_tramo()).

Value

The modified specification (with new ARIMA parameters)

References

More information on reg-arima modelling in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

set_arima, set_transform

Examples

```
# Customize a default specification
init_spec <- x13_spec_default
new_spec <- set_automodel(x = init_spec, enabled = FALSE, acceptdefault = TRUE)
```

`set_basic` *Set estimation sub-span and quality check specification*

Description

Function allowing to check if the series can be processed and to define a sub-span on which estimation will be performed

Usage

```
set_basic(
  x,
  type = c(NA, "All", "From", "To", "Between", "Last", "First", "Excluding"),
  d0 = NULL,
  d1 = NULL,
  n0 = 0,
  n1 = 0,
  preliminary.check = NA,
  preprocessing = NA
)
```

Arguments

`x` the specification to customize, must be a "SPEC" class object (see details).

`type`, `d0`, `d1`, `n0`, `n1`
: parameters to specify the sub-span .

  `d0` and `d1` characters in the format "YYYY-MM-DD" to specify first/last date of the span when `type` equals to `"From"`, `"To"` or `"Between"`. Date corresponding to `d0` will be included in the sub-span Date corresponding to `d1` will be excluded from the sub span

  `n0` and `n1` numeric to specify the number of periods at the beginning/end of the series to be used for defining the sub-span (`type` equals to `"First"`, `"Last"`) or to exclude (`type` equals to `"Excluding"`).

`preliminary.check`
: a Boolean to check the quality of the input series and exclude highly problematic ones (e.g. the series with a number of identical observations and/or missing values above pre-specified threshold values).

`preprocessing` (REGARIMA/X13 Specific) a Boolean to enable/disable the pre-processing. Option disabled for the moment.

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with `rjd3x13::x13_spec()` (or "JD3_REGARIMA_SPEC" generated with `rjd3x13::spec_regarima()` or "JD3_TRAMOSEATS_SPEC" generated with `rjd3tramoseats::spec_tramoseats()` or "JD3_TRAMO_SPEC" generated with `rjd3tramoseats::spec_tramo()`).

Value

The modified specification with new estimation span

References

More information in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

set_estimate, set_arima

Examples

```
# Customize a default specification
init_spec <- x13_spec_default

# Estimation on sub-span between two dates (date d1 is excluded)
new_spec <- set_basic(
    init_spec,
    type = "Between",
    d0 = "2014-01-01",
    d1 = "2019-01-01",
    preliminary.check = TRUE,
    preprocessing = TRUE
)

# Estimation on the first 60 observations
new_spec <- set_basic(
    init_spec,
    type = "First",
    n0 = 60,
    preliminary.check = TRUE,
    preprocessing = TRUE
)

# Estimation on the last 60 observations
new_spec <- set_basic(
    init_spec,
    type = "Last",
    n1 = 60,
    preliminary.check = TRUE,
    preprocessing = TRUE
```

```
)

# Estimation excluding 60 observations at the beginning and 36 at the end of the series
new_spec <-set_basic(
    init_spec,
    type = "Excluding",
    n0 = 60,
    n1 = 36,
    preliminary.check = TRUE,
    preprocessing = TRUE
)
```

---

set_benchmarking *Set Benchmarking Specification*

Description

Function allowing to perform a benchmarking procedure after the decomposition step in a seasonal adjustment (disabled by default). Here benchmarking refers to a procedure ensuring consistency over the year between seasonally adjusted and raw (or calendar adjusted) data, as seasonal adjustment can cause discrepancies between the annual totals of seasonally adjusted series and the corresponding annual totals of raw (or calendar adjusted) series.

Usage

```
set_benchmarking(
  x,
  enabled = NA,
  target = c(NA, "CalendarAdjusted", "Original"),
  rho = NA,
  lambda = NA,
  forecast = NA,
  bias = c("None", "Additive", "Multiplicative")
)
```

Arguments

`x`	the specification to customize, must be a "SPEC" class object (see details).
`enabled`	Boolean to enable the user to perform benchmarking.
`target`	specifies the target series for the benchmarking procedure, which can be the raw series (`"Normal"`); or the series adjusted for calendar effects (`"CalendarAdjusted"`).
`rho`	the value of the AR(1) parameter (set between 0 and 1) in the function used for benchmarking. Default =1.
`lambda`	a parameter in the function used for benchmarking that relates to the weights in the regression equation; it is typically equal to 0, 1/2 or 1.

forecast
: Boolean indicating if the forecasts of the seasonally adjusted series and of the target variable (`target`) are used in the benchmarking computation so that the benchmarking constrain is also applied to the forecasting period.

bias
: Character. Bias correction factor. No systematic bias is considered by default. See `vignette(topic = "rjd3bench", package = "rjd3bench")` for more details.

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with `rjd3x13::x13_spec()` (or "JD3_REGARIMA_SPEC" generated with `rjd3x13::spec_regarima()` or "JD3_TRAMOSEATS_SPEC" generated with `rjd3tramoseats::spec_tramoseats()` or "JD3_TRAMO_SPEC" generated with `rjd3tramoseats::spec_tramo()`).

Value

The modified specification with new estimation span

References

More information on benchmarking in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

Examples

```
init_spec <- x13_spec_default
new_spec <- set_benchmarking(
    x = init_spec,
    enabled = TRUE,
    target = "Original",
    rho = 0.8,
    lambda = 0.5,
    forecast = FALSE,
    bias = "None"
)
```

---

set_easter — *Set Easter effect correction in Pre-Processing Specification*

Description

Set Easter effect correction in Pre-Processing Specification

Usage

```
set_easter(
  x,
  enabled = NA,
  julian = NA,
  duration = NA,
  test = c(NA, "Add", "Remove", "None"),
  coef = NA,
  coef.type = c(NA, "Estimated", "Fixed"),
  type = c(NA, "Unused", "Standard", "IncludeEaster", "IncludeEasterMonday")
)
```

Arguments

`x`	the specification to customize, must be a "SPEC" class object (see details).
`enabled`	a logical indicating if the program considers the Easter effect in the pre-processing model. Default = TRUE.
`julian`	a logical indicating if the program uses the Julian Easter (expressed in Gregorian calendar).
`duration`	a numeric indicating the duration of the Easter effect (length in days, between 1 and 20). Default value = 8 in REGARIMA/X-13 and 6 in TRAMO.
`test`	defines the pre-tests for the significance of the Easter effect based on the t-statistic (the Easter effect is considered as significant if the t-statistic is greater than 1.96): `"Add"` = the Easter effect variable is not included in the initial regression model but can be added to the RegARIMA model after the test; `"Remove"` = the Easter effect variable belongs to the initial regression model but can be removed from the RegARIMA model after the test; `"None"` = the Easter effect variable is not pre-tested and is included in the model.
`coef`	to set the coefficient of the easter regressor.(Test parameter has to be set to `"None"`)
`coef.type`	a character defining the easter regressor coefficient estimation procedure. Possible procedures are: `"Estimated"` = coefficient is estimated, `"Fixed"` = the coefficients is fixed. By default the coefficient is estimated.
`type`	(TRAMO specific) a `character` that specifies the presence and the length of the Easter effect: `"Unused"` = the Easter effect is not considered; `"Standard"` = influences the period of `n` days strictly before Easter Sunday; `"IncludeEaster"` = influences the entire period (`n`) up to and including Easter Sunday; `"IncludeEasterMonday"` = influences the entire period (`n`) up to and including Easter Monday.

Details

`x` specification parameter must be a JD3_X13_SPEC" class object generated with `rjd3x13::x13_spec()` (or "JD3_REGARIMA_SPEC" generated with `rjd3x13::spec_regarima()` or "JD3_TRAMOSEATS_SPEC" generated with `rjd3tramoseats::spec_tramoseats()` or "JD3_TRAMO_SPEC" generated with `rjd3tramoseats::spec_tramo()`).

Value

The modified specification (with new easter parameters)

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

easter_variable, easter_day

Examples

```
# Customize a default specification
init_spec <- x13_spec_default
new_spec <- set_easter(
    x = init_spec,
    enabled = TRUE,
    duration = 12,
    test = "None",
    type = "IncludeEasterMonday"
)
```

---

set_estimate *Set Numeric Estimation Parameters and Modelling Span*

Description

Function allowing to define numeric boundaries for estimation and to define a sub-span on which reg-arima (tramo) modelling will be performed (pre-processing step)

Usage

```
set_estimate(
  x,
  type = c(NA, "All", "From", "To", "Between", "Last", "First", "Excluding"),
  d0 = NULL,
  d1 = NULL,
  n0 = 0,
  n1 = 0,
  tol = NA,
  exact.ml = NA,
  unit.root.limit = NA
)
```

Arguments

x — the specification to customize, must be a "SPEC" class object (see details).

type, d0, d1, n0, n1 — parameters to specify the sub-span .

d0 and d1 characters in the format "YYYY-MM-DD" to specify first/last date of the span when type equals to "From", "To" or "Between". Date corresponding to d0 will be included in the sub-span Date corresponding to d1 will be excluded from the sub span

n0 and n1 numeric to specify the number of periods at the beginning/end of the series to be used for defining the sub-span (type equals to "First", "Last") or to exclude (type equals to "Excluding").

tol — a numeric, convergence tolerance. The absolute changes in the log-likelihood function are compared to this value to check for the convergence of the estimation iterations. (The default setting is 0.0000001)

exact.ml — (TRAMO specific) logical, the exact maximum likelihood estimation. If TRUE, the program performs an exact maximum likelihood estimation. If FASLE, the Unconditional Least Squares method is used. (Default=TRUE)

unit.root.limit — (TRAMO specific) numeric, the final unit root limit. The threshold value for the final unit root test for identification of differencing orders. If the magnitude of an AR root for the final model is smaller than this number, then a unit root is assumed, the order of the AR polynomial is reduced by one and the appropriate order of the differencing (non-seasonal, seasonal) is increased.(Default value: 0.96)

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with rjd3x13::x13_spec() (or "JD3_REGARIMA_SPEC" generated with rjd3x13::spec_regarima() or "JD3_TRAMOSEATS_SPEC" generated with rjd3tramoseats::spec_tramoseats() or "JD3_TRAMO_SPEC" generated with rjd3tramoseats::spec_tramo()).

Value

The modified specification (with new estimation parameters)

References

More in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

set_basic, set_arima

Examples

```
# Customize a default specification
init_spec <- tramoseats_spec_default
new_spec <- set_estimate(
    x = init_spec,
    type = "From",
    d0 = "2012-01-01",
    tol = 0.0000002,
    exact.ml = FALSE,
    unit.root.limit = 0.98
)
```

---

set_outlier *Set Outlier Detection Parameters*

Description

Function allowing to customize the automatic outlier detection process built in in the pre-processing step (regarima or tramo).

Usage

```
set_outlier(
  x,
 span.type = c(NA, "All", "From", "To", "Between", "Last", "First", "Excluding"),
  d0 = NULL,
  d1 = NULL,
  n0 = 0,
  n1 = 0,
  outliers.type = NA,
  critical.value = NA,
  tc.rate = NA,
  method = c(NA, "AddOne", "AddAll"),
  maxiter = NA,
  lsrun = NA,
  eml.est = NA
)
```

Arguments

`x` the specification to customize, must be a "SPEC" class object (see details).

`span.type`, `d0`, `d1`, `n0`, `n1`
parameters to specify the sub-span on which outliers will be detected.

- `d0` and `d1` characters in the format "YYYY-MM-DD" to specify first/last date of the span when `type` equals to `"From"`, `"To"` or `"Between"`.

- n0 and n1 numerics to specify the number of periods at the beginning/end of the series to be used for the span (type equals to "From", "To") or to exclude (type equals to "Excluding").

outliers.type — vector of characters of the outliers to be automatically detected.

- "AO" for additive outliers,
- "TC" for transitory changes,
- "LS" for level shifts,
- "SO" for seasonal outliers. For example outliers.type = c("AO", "LS") to enable the detection of additive outliers and level shifts. If outliers.type = NULL or outliers.type = character(), automatic detection of outliers is disabled. Default value = outliers.type = c("AO", "LS", "TC")

critical.value — numeric. Critical value for the outlier detection procedure. If equal to 0 the critical value is automatically determined by the number of observations in the outlier detection time span. (Default value = 4 REGARIMA/X13 and 3.5 in TRAMO)

tc.rate — the rate of decay for the transitory change outlier. (Default = 0.7).

method — (REGARIMA/X13 Specific) determines how the program successively adds detected outliers to the model. Currently, only the "AddOne" method is supported.

maxiter — (REGARIMA/X13 Specific) maximum number of iterations (Default = 30).

lsrun — (REGARIMA/X13 Specific) number of successive level shifts to test for cancellation (Default = 0).

eml.est — (TRAMO Specific) logical for the exact likelihood estimation method. It controls the method applied for parameter estimation in the intermediate steps. If TRUE, an exact likelihood estimation method is used. When FALSE, the fast Hannan-Rissanen method is used.

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with rjd3x13::x13_spec() (or "JD3_REGARIMA_SPEC" generated with rjd3x13::spec_regarima() or "JD3_TRAMOSEATS_SPEC" generated with rjd3tramoseats::spec_tramoseats() or "JD3_TRAMO_SPEC" generated with rjd3tramoseats::spec_tramo()).

If a Seasonal adjustment process is performed, each type of Outlier will be allocated to a pre-defined component after the decomposition: "AO" and "TC" to the irregular, "LS" to the trend and "SO" to seasonal component.

Value

The modified specification (with new outlier parameters)

References

More information on outliers and other auxiliary variables in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

add_outlier, add_usrdefvar

Examples

```
# Customize a default specification
init_spec <- tramoseats_spec_default
new_spec <- set_outlier(
    x = init_spec,
    span.type = "From",
    d0 = "2012-01-01",
    outliers.type = c("LS", "AO"),
    critical.value = 5,
    tc.rate = 0.85
)
```

---

set_tradingdays — *Set Calendar effects correction in Pre-Processing Specification*

Description

Function allowing to select the trading-days regressors to be used for calendar correction in the pre-processing step of a seasonal adjustment procedure. The default is "TradingDays", with easter specific effect enabled. (see set_easter)

All the built-in regressors are meant to correct for type of day effect but don't take into account any holiday. To do so user-defined regressors have to be built.

Usage

```
set_tradingdays(
  x,
  option = c(NA, "TradingDays", "WorkingDays", "TD2c", "TD3", "TD3c", "TD4", "None",
    "UserDefined"),
  calendar.name = NA,
  uservariable = NA,
  stocktd = NA,
  test = c(NA, "None", "Remove", "Add", "Separate_T", "Joint_F"),
  coef = NA,
  coef.type = c(NA, "Fixed", "Estimated"),
  automatic = c(NA, "Unused", "FTest", "WaldTest", "Aic", "Bic"),
  pftd = NA,
  autoadjust = NA,
  leapyear = c(NA, "LeapYear", "LengthOfPeriod", "None"),
  leapyear.coef = NA,
  leapyear.coef.type = c(NA, "Fixed", "Estimated")
)
```

Arguments

`x` the specification to customize, must be a "SPEC" class object (see details).

`option` to specify the set of trading days regression variables:

- `"TradingDays"` = six contrast variables, each type of day (from Monday to Saturday) vs Sundays;
- `"WorkingDays"` = one working (week days) vs non-working (week-ends) day contrast variable;
- `"TD2c"` = one working (Mondays to Saturdays) vs non-working (Sundays) day contrast variable;
- `"TD3"` = two contrast variables: week-days vs Sundays and Saturdays vs Sundays;
- `"TD3c"` = two contrast variables: week-days (Mondays to Thursdays) vs Sundays and Fridays+Saturdays vs Sundays;
- `"TD4"` = three contrast variables: week-days (Mondays to Thursdays) vs Sundays, Fridays vs Sundays, Saturdays vs Sundays;
- `"None"` = no correction for trading days;
- `"UserDefined"` = userdefined trading days regressors.

`calendar.name` name (string) of the user-defined calendar to be taken into account when generating built-in regressors set in `option` (if not `"UserDefined"`).(see examples)

`uservariable` a vector of characters to specify the name of user-defined calendar regressors. When specified, automatically set `option = "UserDefined"`. Names have to be the same as in `modelling_context`, see example.

`stocktd` a numeric indicating the day of the month when inventories and other stock are reported (to denote the last day of the month, set the variable to 31). When specified, automatically set `option = "None"`. See `stock_td` function for details.

`test` defines the pre-tests for the significance of the trading day regression variables based on the AICC statistics: `"None"` = the trading day variables are not pretested and are included in the model;

(REGARIMA/X-13 specific)

- `"Add"` = the trading day variables are not included in the initial regression model but can be added to the RegARIMA model after the test;
- `"Remove"` = the trading day variables belong to the initial regression model but can be removed from the RegARIMA model after the test;

(TRAMO specific)

- `"Separate_T"` = a t-test is applied to each trading day variable separately and the trading day variables are included in the RegArima model if at least one t-statistic is greater than 2.6 or if two t-statistics are greater than 2.0 (in absolute terms);
- `"Joint_F"` = a joint F-test of significance of all the trading day variables. The trading day effect is significant if the F statistic is greater than 0.95.

`coef` vector of coefficients for the trading-days regressors.

`coef.type`, `leapyear.coef.type`
vector defining if the coefficients are fixed or estimated.

automatic	defines whether the calendar effects should be added to the model manually ("Unused") or automatically. During the automatic selection, the choice of the number of calendar variables can be based on the F-Test ("FTest", TRAMO specific), the Wald Test ("WaldTest"), or by minimizing AIC or BIC; the model with higher F-value is chosen, provided that it is higher than pftd).
pftd	(TRAMO SPECIFIC) numeric. The p-value used to assess the significance of the pre-tested calendar effects.
autoadjust	a logical indicating if the program corrects automatically the raw series for the leap year effect if the leap year regressor is significant. Only used when the data is log transformed.
leapyear	a character to specify whether or not to include the leap-year effect in the model: • "LeapYear" = leap year effect; • "LengthOfPeriod" = length of period (REGARIMA/X-13 specific), • "None" = no effect included. Default: a leap year effect regressor is included with any built-in set of trading day regressors.
leapyear.coef	coefficient of the leap year regressor.

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with rjd3x13::x13_spec() (or "JD3_REGARIMA_SPEC" generated with rjd3x13::spec_regarima() or "JD3_TRAMOSEATS_SPEC" generated with rjd3tramoseats::spec_tramoseats() or "JD3_TRAMO_SPEC" generated with rjd3tramoseats::spec_tramo()).

Value

The modified specification (with new trading days variables)

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

modelling_context, calendar_td

Examples

```
# Pre-defined regressors
y_raw <- ABS$X0.2.09.10.M

# Customize a default specification
init_spec <- x13_spec_default

# Estimation on sub-span between two dates (date d1 is excluded)
new_spec <- set_tradingdays(
```

```
    init_spec,
    option = "TD4",
    test = "None",
    coef = c(0.7, NA, 0.5),
    coef.type = c("Fixed", "Estimated", "Fixed"),
    leapyear = "LengthOfPeriod",
    leapyear.coef = 0.6
)

# Pre-defined regressors based on user-defined calendar
### create a calendar
BE <- national_calendar(list(
    fixed_day(7, 21),
    special_day("NEWYEAR"),
    special_day("CHRISTMAS"),
    special_day("MAYDAY"),
    special_day("EASTERMONDAY"),
    special_day("ASCENSION"),
    special_day("WHITMONDAY"),
    special_day("ASSUMPTION"),
    special_day("ALLSAINTSDAY"),
    special_day("ARMISTICE")
))
## Put into a context
my_context <- modelling_context(calendars = list(cal = BE))

## Modify the specification
new_spec <- set_tradingdays(
    init_spec,
    option = "TradingDays",
    calendar.name = "cal"
)

## Estimate with context
# sa <- rjd3x13::x13(y_raw, new_spec, context = my_context)

regs_td <- rjd3toolkit::td(
    s = y_raw,
    groups = c(1, 2, 0, 4, 5, 6, 3),
    contrasts = TRUE
)

variables <- list(
    Monday = regs_td[, 1],
    Tuesday = regs_td[, 2],
    Wednesday = regs_td[, 3],
    Thursday = regs_td[, 4],
    Friday = regs_td[, 5],
    Saturday = regs_td[, 6]
)
# Add regressors to context
my_context <- modelling_context(variables = variables)
```

```
# Create a new spec (here default group name: r)
new_spec <- set_tradingdays(
    init_spec,
    option = "UserDefined",
    uservariable = c("r.Monday", "r.Tuesday", "r.Wednesday",
                     "r.Thursday", "r.Friday", "r.Saturday"),
    test = "None"
)

# Estimate with context
# sa <- rjd3x13::x13(y_raw, new_spec, context = my_context)
```

`set_transform` *Set Log-level Transformation and Decomposition scheme in Pre-Processing Specification*

Description

Set Log-level Transformation and Decomposition scheme in Pre-Processing Specification

Usage

```
set_transform(
  x,
  fun = c(NA, "Auto", "Log", "None"),
  adjust = c(NA, "None", "LeapYear", "LengthOfPeriod"),
  outliers = NA,
  aicdiff = NA,
  fct = NA
)
```

Arguments

`x`	the specification to customize, must be a "SPEC" class object (see details).
`fun`	the transformation of the input series: `"None"` = no transformation of the series; `"Log"` = takes the log of the series; `"Auto"` = the program tests for the log-level specification.
`adjust`	pre-adjustment of the input series for the length of period or leap year effects: `"None"` = no adjustment; `"LeapYear"` = leap year effect; `"LengthOfPeriod"` = length of period. Modifications of this variable are taken into account only when `function = "Log"`.
`outliers`	Boolean indicating if a pre-correction for large outliers (AO and LS only) should be done in the test for the log-level specification (`fun = "Auto"`). By default to `FALSE`.
`aicdiff`	(REGARIMA/X-13 specific) a numeric defining the difference in AICC needed to accept no transformation when the automatic transformation selection is chosen (considered only when `fun = "Auto"`). Default= -2.

fct	(TRAMO specific) `numeric` controlling the bias in the log/level pre-test: `transform.fct>` 1 favours levels, `transform.fct<` 1 favours logs. Considered only when `fun = "Auto"`.

Details

x specification parameter must be a JD3_X13_SPEC" class object generated with `rjd3x13::x13_spec()` (or "JD3_REGARIMA_SPEC" generated with `rjd3x13::spec_regarima()` or "JD3_TRAMOSEATS_SPEC" generated with `rjd3tramoseats::spec_tramoseats()` or "JD3_TRAMO_SPEC" generated with `rjd3tramoseats::spec_tramo()`).

Value

The modified specification (with log/level transformation scheme)

References

More information in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

set_outlier, set_tradingdays

Examples

```
# Customize a default specification
init_spec <- x13_spec_default
new_spec <- set_transform(x = init_spec, fun = "Log", outliers = TRUE)
```

---

single_day — *Set a holiday on a Single Day*

Description

Allows to set a holiday as a once-occurring event.

Usage

```
single_day(date, weight = 1)
```

Arguments

date	the date of the holiday in the format `"YYYY-MM-DD"`.
weight	weight associated to the holiday.

Value

returns an object of class c("JD3_SINGLEDAY","JD3_HOLIDAY") (with name of the event, date, offset...)

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

national_calendar, fixed_day, special_day,easter_day

Examples

```
single_day("1999-03-19")
```

---

special_day *List of Pre-Defined Holidays to choose from*

Description

Allows to define a holiday choosing from a list of pre-specified events, equivalent to use fixed_day or easter_day functions.

Usage

```
special_day(event, offset = 0, weight = 1, validity = NULL)
```

Arguments

event	the event to add (see details).
offset	The position of the holiday in relation to the selected pre-specified holiday measured in days (can be positive or negative). By default offset = 0.
weight	weight associated to the holiday.
validity	validity period: either NULL (full sample) or a named list with "start" and/or "end" dates in the format "YYYY-MM-DD".

Details

Possible values :

NEWYEAR	Fixed holiday, falls on January, 1st.
SHROVEMONDAY	Moving holiday, falls on the Monday before Ash Wednesday (48 days before Easter Sunday).
SHROVETUESDAY	Moving holiday, falls on the Tuesday before Ash Wednesday (47 days before Easter Sunday).
ASHWEDNESDAY	Moving holiday, occurring 46 days before Easter Sunday.
MAUNDYTHURSDAY	Moving holiday, falls on the Thursday before Easter.
GOODFRIDAY	Moving holiday, falls on the Friday before Easter.
EASTER	Moving holiday, falls between March 22nd and April 25th.
EASTERMONDAY	Moving holiday, falls on the day after Easter.
ASCENSION	Moving holiday, celebrated on a Thursday, 39 days after Easter.
PENTECOST	Moving holiday, celebrated 49 days after Easter Sunday.
WHITMONDAY	Moving holiday, falling on the day after Pentecost.
CORPUSCHRISTI	Moving holiday, celebrated 60 days after Easter Sunday.
JULIANEASTER	
MAYDAY	Fixed holiday, falls on May, 1st.
ASSUMPTION	Fixed holiday, falls on August, 15th.
HALLOWEEN	Fixed holiday, falls on October, 31st.
ALLSAINTSDAY	Fixed holiday, falls on November, 1st.
ARMISTICE	Fixed holiday, falls on November, 11th.
CHRISTMAS	Fixed holiday, falls on December, 25th.

Value

returns an object of class `c("JD3_SPECIALDAY","JD3_HOLIDAY")` (with name of the event, date, offset...)

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

national_calendar, fixed_day, easter_day

Examples

```
# To add Easter Monday
special_day("EASTERMONDAY")
# To define a holiday for the day after Christmas, with validity and weight
special_day("CHRISTMAS",
    offset = 1, weight = 0.8,
    validity = list(start = "2000-01-01", end = "2020-12-01")
)
```

statisticaltest *Generic Function For 'JDemetra+' Tests*

Description

Generic function to format the results of 'JDemetra+' tests.

Usage

```
statisticaltest(val, pval, dist = NULL)

## S3 method for class 'JD3_TEST'
print(x, details = FALSE, ...)
```

Arguments

`val, pval, dist`	statistical parameters.
`x`	the object to print.
`details`	boolean indicating if the statistical distribution should be printed.
`...`	further arguments (ignored).

Value

`c("JD3_TEST", "JD3")` object that is a list of three parameters:

- `value` the statistical value of the test.
- `pvalue` the p-value of the test.
- `distribution` the statistical distribution used.

Examples

```
udr_test <- testofupdownruns(random_t(5, 1000))
udr_test # default print
print(udr_test, details = TRUE) # with the distribution

test <- statisticaltest(val = 45, pval = 0.1)
print(test)
```

stock_td *Trading day Regressor for Stock series*

Description

Allows to generate a specific regressor for correcting trading days effects in Stock series.

Usage

```
stock_td(frequency, start, length, s, w = 31)
```

Arguments

`frequency`	Frequency of the series, number of periods per year (12, 4, 3, 2...)
`start, length`	First date (array with the first year and the first period, for instance `c(1980, 1)`) and number of periods of the output variables. Can also be provided with the `s` argument
`s`	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
`w`	indicates day of the month when inventories and other stocks are reported. (to denote the last day of the month enter 31).

Details

The regressor will have the value -1 if the w-th day is a Sunday, 1 if it is a Monday as 0 otherwise.

Value

Time series (object of class `c("ts","mts","matrix")`).

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

`calendar_td`

Examples

```
stock_td(frequency = 12L, start = c(1990L, 1L), length = 480L, w = 1L)
```

td *Trading day regressors without holidays*

Description

Allows to generate trading day regressors (as many as defined groups), taking into account 7 or less different types of days, from Monday to Sunday, but no specific holidays. Regressors are not corrected for long term mean.

Usage

```
td(
  frequency,
  start,
  length,
  s,
  groups = c(1, 2, 3, 4, 5, 6, 0),
  contrasts = TRUE
)
```

Arguments

`frequency`	Frequency of the series, number of periods per year (12, 4, 3, 2...)
`start, length`	First date (array with the first year and the first period, for instance `c(1980, 1)`) and number of periods of the output variables. Can also be provided with the `s` argument
`s`	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
`groups`	Groups of days. The length of the array must be 7. It indicates to what group each week day belongs. The first item corresponds to Mondays and the last one to Sundays. The group used for contrasts (usually Sundays) is identified by 0. The other groups are identified by 1, 2,... n (<= 6). For instance, usual trading days are defined by c(1, 2, 3, 4, 5, 6, 0), week days by c(1, 1, 1, 1, 1, 0, 0), week days, Saturdays, Sundays by c(1, 1, 1, 1, 1, 2, 0) etc.
`contrasts`	If true, the variables are defined by contrasts with the 0-group. Otherwise, raw number of days is provided.

Details

Aggregated values for monthly or quarterly are the numbers of days belonging to a given group. Contrasts are the differences between the number of days in a given group (1 to 6) and the number of days in the reference group (0).

Value

Time series (object of class `c("ts","mts","matrix")`) corresponding to each group, starting with the 0-group (`contrasts = FALSE`) or the 1-group (`contrasts = TRUE`).

References

More information on calendar correction in JDemetra+ online documentation: https://jdemetra-new-documentation.netlify.app/a-calendar-correction

See Also

calendar_td

Examples

```
# Monthly regressors for Trading Days: each type of day is different
# contrasts to Sundays (6 series)
regs_td <- td(12, c(2020, 1), 60, groups = c(1, 2, 3, 4, 5, 6, 0), contrasts = TRUE)
# Quarterly regressors for Working Days: week days are similar
# contrasts to week-end days (1 series)
regs_wd <- td(4, c(2020, 1), 60, groups = c(1, 1, 1, 1, 1, 0, 0), contrasts = TRUE)
```

---

td_canovahansen *Canova-Hansen test for stable trading days*

Description

Canova-Hansen test for stable trading days

Usage

```
td_canovahansen(
  s,
  differencing,
  kernel = c("Bartlett", "Square", "Welch", "Tukey", "Hamming", "Parzen"),
  order = NA
)
```

Arguments

s	a ts object that corresponds to the input time series to test.
differencing	Differencing lags.
kernel	Kernel used to compute the robust covariance matrix.
order	The truncation parameter used to compute the robust covariance matrix.

Value

list with the ftest on td, the joint test and the details for the stability of the different days (starting with Mondays).

Examples

```
s <- log(ABS$X0.2.20.10.M)
td_canovahansen(s, c(1, 12))
```

---

td_f *Residual Trading Days Test*

Description

Residual Trading Days Test

Usage

```
td_f(
  s,
  model = c("D1", "DY", "DYD1", "WN", "AIRLINE", "R011", "R100"),
  nyears = 0
)
```

Arguments

`s`	a `ts` object that corresponds to the input time series to test.
`model`	the model to use for the residuals. See details.
`nyears`	`integer` that corresponds to the length of the sub series, starting from the end of the series, to be used for the test: in number of periods (positive value) or years (negative values). By default (`nyears = 0`), the entire sample is used.

Details

The function performs a residual seasonality test that is a joint F-Test on the coefficients of trading days regressors. Several specifications can be used on the model:

- `model = "WN"` the following model is used:

$$y_t - \bar{y} = \beta TD_t + \varepsilon_t$$

- `model = "D1"` (the default) the following model is used:

$$\Delta y_t - \overline{\Delta y} = \beta \Delta TD_t + \varepsilon_t$$

- `model = "DY"` the following model is used:

$$\Delta_s y_t - \overline{\Delta_s y} = \beta \Delta_s TD_t + \varepsilon_t$$

- `model = "DYD1"` the following model is used:

$$\Delta_s \Delta y_t - \overline{\Delta_s \Delta y} = \beta \Delta_s \Delta TD_t + \varepsilon_t$$

- model = "AIRLINE" the following model is used:

$$y_t = \beta TD_t + \varepsilon_t \text{ with } \varepsilon_t \sim ARIMA(0,1,1)(0,1,1)$$

- model = "R011" the following model is used:

$$y_t = \beta TD_t + \varepsilon_t \text{ with } \varepsilon_t \sim ARIMA(0,1,1)$$

- model = "R100" the following model is used:

$$y_t = \alpha_0 + \alpha_1 y_{t-1} + \beta TD_t + \varepsilon_t$$

Value

a JD3_TEST object with value, p-value and information about the distribution

Examples

```
td_f(ABS$X0.2.09.10.M)
```

`td_timevarying` *Likelihood ratio test on time varying trading days*

Description

Likelihood ratio test on time varying trading days

Usage

```
td_timevarying(s, groups = c(1, 2, 3, 4, 5, 6, 0), contrasts = FALSE)
```

Arguments

s	The tested time series
groups	The groups of days used to generate the regression variables.
contrasts	The covariance matrix of the multivariate random walk model used for the time-varying coefficients are related to the contrasts if TRUE, on the actual number of days (all the days are driven by the same variance) if FALSE.

Value

A Chi2 test

Examples

```
s <- log(ABS$X0.2.20.10.M)
td_timevarying(s)
```

to_ts *Creates a time series object*

Description

Creates a time series object

Usage

```
to_ts(source, id, type = "All")
```

Arguments

source	Source of the time series
id	Identifier of the time series (source-dependent)
type	Type of the requested information (Data, Metadata...). All by default.

Value

An object of type "JD3_TS". List containing the identifiers, the data and the metadata

Examples

```
source <- "Txt"
# id is split due to length restrictions
id1 <- "demetra://tsprovider/Txt/20111201/SERIES?datePattern=dd%2FMM%2Fyyyy&delimiter=SEMICOLON&"
id2 <- "file=C%3A%5CDocuments%5CIPI%5CData%5CIPI_nace4.csv#seriesIndex=0"
id <- paste0(id1, id2)

to_ts(source, id)
```

to_tscollection *Creates a collection of time series*

Description

Creates a collection of time series

Usage

```
to_tscollection(source, id, type = "All")
```

Arguments

source	Source of the collection of time series
id	Identifier of the collection of time series (source-dependent)
type	Type of the requested information (Data, Metadata...). All by default.

Value

An object of type "JD3_TSCOLLECTION". List containing the identifiers, the metadata and all the series (data).

Examples

```
# id is split due to length restrictions
id1 <- "demetra://tsprovider/Txt/20111201/SERIES?datePattern=dd%2FMM%2Fyyyy&delimiter=SEMICOLON&"
id2 <- "file=C%3A%5CDocuments%5CIPI%5CData%5CIPI_nace4.csv#seriesIndex=0"
id <- paste0(id1, id2)
source <- "Txt"
my_collection <- to_tscollection(source, id)
```

---

tramoseats_spec_default

Default Tramo-Seats specification

Description

Default Tramo-Seats specification

Usage

```
tramoseats_spec_default
```

Format

An object of class `JD3_TRAMOSEATS_SPEC` of length 3.

Examples

```
data(tramoseats_spec_default)
```

`trigonometric_variables` *Trigonometric variables*

Description

Computes trigonometric variables at different frequencies.

Usage

```
trigonometric_variables(frequency, start, length, s, seasonal_frequency = NULL)
```

Arguments

`frequency`	Frequency of the series, number of periods per year (12, 4, 3, 2...)
`start, length`	First date (array with the first year and the first period, for instance `c(1980, 1)`) and number of periods of the output variables. Can also be provided with the `s` argument
`s`	time series used to get the dates for the trading days variables. If supplied the parameters `frequency`, `start` and `length` are ignored.
`seasonal_frequency`	the seasonal frequencies. By default the fundamental seasonal frequency and all the harmonics are used.

Details

Denote by P the value of `frequency` (= the period) and f_1, ..., f_n the frequencies provides by `seasonal_frequency` (if `seasonal_frequency` = `NULL` then $n = \lfloor P/2 \rfloor$ and f_i=i).

`trigonometric_variables` returns a matrix of size $length \times (2n)$.

For each date t associated to the period m ($m \in [1, P]$), the columns $2i$ and $2i - 1$ are equal to:

$$\cos\left(\frac{2\pi}{P} \times m \times f_i\right) \text{ and } \sin\left(\frac{2\pi}{P} \times m \times f_i\right)$$

Take for example the case when the first date (`date`) is a January, `frequency` = `12` (monthly time series), `length` = `12` and `seasonal_frequency` = `NULL`. The first frequency, $\lambda_1 = 2\pi/12$ represents the fundamental seasonal frequency and the other frequencies ($\lambda_2 = 2\pi/12 \times 2$, ..., $\lambda_6 = 2\pi/12 \times 6$) are the five harmonics. The output matrix will be equal to:

$$\begin{pmatrix} \cos(\lambda_1) & \sin(\lambda_1) & \cdots & \cos(\lambda_6) & \sin(\lambda_6) \\ \cos(\lambda_1 \times 2) & \sin(\lambda_1 \times 2) & \cdots & \cos(\lambda_6 \times 2) & \sin(\lambda_6 \times 2) \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \cos(\lambda_1 \times 12) & \sin(\lambda_1 \times 12) & \cdots & \cos(\lambda_6 \times 12) & \sin(\lambda_6 \times 12) \end{pmatrix}$$

Value

a mts object with 2 columns

Examples

```
trigonometric_variables(
    frequency = 12,
    length = 480,
    start = c(1990, 1),
    seasonal_frequency = 3
)
```

---

tsdata_of — *Create ts object with values and dates*

Description

Create ts object with values and dates

Usage

```
tsdata_of(values, dates)
```

Arguments

`values`	Values of the time series
`dates`	Dates of the values (could be any date inside the considered period)

Value

A `ts` object. The frequency will be identified automatically and missing values will be added in need be. The identified frequency will be the lowest frequency that match the figures. The provided data can contain missing values (NA)

Examples

```
# Annual series
s <- tsdata_of(c(1, 2, 3, 4), c("1990-01-01", "1995-01-01", "1996-01-01",
        "2000-11-01"))
# Quarterly series
t <- tsdata_of(c(1, 2, 3, NA, 4), c("1990-01-01", "1995-01-01", "1996-01-01",
        "2000-08-01", "2000-11-01"))
```

ts_adjust — *Multiplicative adjustment of a time series for leap year / length of periods*

Description

Multiplicative adjustment of a time series for leap year / length of periods

Usage

```
ts_adjust(s, method = c("LeapYear", "LengthOfPeriod"), reverse = FALSE)
```

Arguments

`s`	The original time series
`method`	`"LeapYear"`: correction for leap year `"LengthOfPeriod"`: correction for the length of periods
`reverse`	Adjustment or reverse operation

Value

The interpolated series

Examples

```
y <- ABS$X0.2.09.10.M
ts_adjust(y)
# with reverse we can find the
all.equal(ts_adjust(ts_adjust(y), reverse = TRUE), y)
```

ts_interpolate — *Interpolation of a time series with missing values*

Description

Interpolation of a time series with missing values

Usage

```
ts_interpolate(s, method = c("airline", "average"))
```

Arguments

`s`	The original time series
`method`	airline: interpolation through an estimated airline model average: interpolation using the average of the previous and next non missing values

Value

The interpolated series

Examples

```
ts_interpolate(AirPassengers)

x <- AirPassengers
x[50:60] <- NA
ts_interpolate(x)
```

---

ucarima_canonical   *Makes a UCARIMA model canonical*

Description

More specifically, put all the noise of the components in one dedicated component

Usage

```
ucarima_canonical(ucm, cmp = 0, adjust = TRUE)
```

Arguments

`ucm`	An UCARIMA model returned by `ucarima_model()`.
`cmp`	Index of the component that will contain the noises; 0 if a new component with all the noises will be added to the model
`adjust`	If TRUE, some noise could be added to the model to ensure that all the components has positive (pseudo-)spectrum

Value

A new UCARIMA model

Examples

```
mod1 <- arima_model("trend", delta = c(1, -2, 1))
mod2 <- arima_model("noise", var = 1600)
hp <- ucarima_model(components = list(mod1, mod2))
hpc <- ucarima_canonical(hp, cmp = 2)
```

ucarima_estimate *Estimate UCARIMA Model*

Description

Estimate UCARIMA Model

Usage

```
ucarima_estimate(x, ucm, stdev = TRUE)
```

Arguments

x	Univariate time series
ucm	An UCARIMA model returned by `ucarima_model()`.
stdev	TRUE if standard deviation of the components are computed

Value

A matrix containing the different components and their standard deviations if stdev is TRUE.

Examples

```
mod1 <- arima_model("trend", delta = c(1, -2, 1))
mod2 <- arima_model("noise", var = 16)
hp <- ucarima_model(components = list(mod1, mod2))
s <- log(aggregate(Retail$AutomobileDealers))
all <- ucarima_estimate(s, hp, stdev = TRUE)
plot(s, type = "l")
t <- ts(all[, 1], frequency = frequency(s), start = start(s))
lines(t, col = "blue")
```

ucarima_model *Creates an UCARIMA model, which is composed of ARIMA models with independent innovations.*

Description

Creates an UCARIMA model, which is composed of ARIMA models with independent innovations.

Usage

```
ucarima_model(model = NULL, components, complements = NULL, checkmodel = FALSE)
```

Arguments

`model`	The reduced model. Usually not provided.
`components`	The ARIMA models representing the components
`complements`	Complements of (some) components. Usually not provided
`checkmodel`	When the model is provided and *checkmodel* is TRUE, we check that it indeed corresponds to the reduced form of the components; similar controls are applied on complements. Currently not implemented

Value

A list with the reduced model, the components and their complements

Examples

```
mod1 <- arima_model("trend", delta = c(1, -2, 1))
mod2 <- arima_model("noise", var = 1600)
hp <- ucarima_model(components = list(mod1, mod2))
print(hp$model)
```

---

ucarima_wk — *Wiener Kolmogorov Estimators*

Description

Wiener Kolmogorov Estimators

Usage

```
ucarima_wk(ucm, cmp, signal = TRUE, nspectrum = 601, nwk = 300)
```

Arguments

`ucm`	An UCARIMA model returned by `ucarima_model()`.
`cmp`	Index of the component for which we want to compute the filter
`signal`	TRUE for the signal (component), FALSE for the noise (complement)
`nspectrum`	Number of points used to compute the (pseudo-) spectrum of the estimator
`nwk`	Number of weights of the Wiener-Kolmogorov filter returned in the result

Value

A list with the (pseudo-)spectrum, the weights of the filter and the squared-gain function (with the same number of points as the spectrum)

Examples

```
mod1 <- arima_model("trend", delta = c(1, -2, 1))
mod2 <- arima_model("noise", var = 1600)
hp <- ucarima_model(components = list(mod1, mod2))
wk1 <- ucarima_wk(hp, 1, nwk = 50)
wk2 <- ucarima_wk(hp, 2)
plot(wk1$filter, type = "h")
```

---

weighted_calendar *Create a Composite Calendar*

Description

Allows to combine two or more calendars into one calendar, weighting all the holidays of each of them.

Usage

```
weighted_calendar(calendars, weights)
```

Arguments

`calendars`	list of calendars.
`weights`	vector of weights associated to each calendar.

Details

Composite calendars are useful for a series that including data from more than one country/region. They can be used, for example, to create the calendar for the European Union or to create the national calendar for a country, in which regional holidays are celebrated. For example, in Germany public holidays are determined by the federal states. Therefore, Epiphany is celebrated only in Baden-Wurttemberg, Bavaria and in Saxony-Anhalt, while from 1994 Day of Repentance and Prayer is celebrated only in Saxony.

Value

returns an object of class `c("JD3_WEIGHTEDCALENDAR", "JD3_CALENDARDEFINITION")`

References

More information on calendar correction in JDemetra+ online documentation: `https://jdemetra-new-documentation.netlify.app/a-calendar-correction`

See Also

`national_calendar`, `chained_calendar`

Examples

```
Belgium <- national_calendar(list(special_day("NEWYEAR"), fixed_day(7, 21)))
France <- national_calendar(list(special_day("NEWYEAR"), fixed_day(7, 14)))
composite_calendar <- weighted_calendar(list(France, Belgium), weights = c(1, 2))
```

---

x13_spec_default *Default X13 specification*

Description

Default X13 specification

Usage

```
x13_spec_default
```

Format

An object of class `JD3_X13_SPEC` of length 3.

Examples

```
data(x13_spec_default)
```

Index