Package 'RJDemetra'

August 10, 2020

Type Package

Title Interface to 'JDemetra+' Seasonal Adjustment Software

Version 0.1.6

Description Interface around 'JDemetra+' (<https://github.com/jdemetra/jdemetra-app>), the seasonal adjustment software officially recommended to the members of the European Statistical System (ESS) and the European System of Central Banks.
It offers full access to all options and outputs of 'JDemetra+', including the two leading seasonal adjustment methods TRAMO/SEATS+ and X-12ARIMA/X-13ARIMA-SEATS.

Depends R (>= 3.1.1),

Imports rJava (>= 0.9-8), graphics, grDevices, methods, stats, utils

SystemRequirements Java SE 8 or higher

License EUPL

URL https://github.com/jdemetra/rjdemetra

LazyData TRUE

Suggests knitr, rmarkdown

RoxygenNote 7.1.0

BugReports https://github.com/jdemetra/rjdemetra/issues

Encoding UTF-8

NeedsCompilation no

Author Alain Quartier-la-Tente [aut, cre] (<https://orcid.org/0000-0001-7890-3857>), Anna Michalek [aut], Jean Palate [aut], Raf Baeyens [aut]

Maintainer Alain Quartier-la-Tente <alain.quartier@yahoo.fr>

Repository CRAN

Date/Publication 2020-08-10 17:20:33 UTC

R topics documented:

add_sa_item . . . 2
compute . . . 3
count . . . 4
get_model . . . 5
get_name . . . 6
get_object . . . 7
get_ts . . . 8
ipi_c_eu . . . 9
jSA . . . 11
load_workspace . . . 13
new_workspace . . . 13
plot . . . 14
regarima . . . 16
regarima_spec_tramoseats . . . 20
regarima_spec_x13 . . . 31
save_spec . . . 41
save_workspace . . . 43
specification . . . 44
tramoseats . . . 47
tramoseats_spec . . . 51
user_defined_variables . . . 61
x13 . . . 62
x13_spec . . . 64

Index **76**

`add_sa_item` *Add a seasonally adjust model to a multi-processing*

Description

Function to add a new seasonally adjust object (class `"SA"` or `"jSA"`) in a `workspace` object.

Usage

```
add_sa_item(workspace, multiprocessing, sa_obj, name)
```

Arguments

`workspace`	the workspace to add the seasonally adjust model.
`multiprocessing`	the name or index of the multiprocessing to add the seasonally adjust model.
`sa_obj`	the seasonally adjust object to export.
`name`	the name of the seasonally adjust model in the multiprocessing. By default the name of the `sa_obj` is used.

See Also

load_workspace, save_workspace

Examples

```
dir <- tempdir()
spec_x13 <- x13_spec(spec = "RSA5c", easter.enabled = FALSE)
sa_x13 <- x13(ipi_c_eu[, "FR"], spec = spec_x13)
spec_ts <- tramoseats_spec(spec = "RSA5")
sa_ts <- jtramoseats(ipi_c_eu[, "FR"], spec = spec_ts)

wk <- new_workspace()
new_multiprocessing(wk, "sa1")
add_sa_item(wk, "sa1", sa_x13, "X13")
add_sa_item(wk, "sa1", sa_ts, "TramoSeats")

save_workspace(wk, file.path(dir, "workspace.xml"))
```

---

compute *Compute the multi-processing from a workspace*

Description

Function to compute all the multi-processings or a given one from a workspace. By default the workspace only contains definitions: computation is needed to get the seasonal adjustment model (with get_model).

Usage

```
compute(workspace, i)
```

Arguments

workspace	the workspace to compute.
i	a character or numeric indicating the name or the index of the multiprocessing to compute. By default all the multi-processings are compute.

See Also

get_model

Examples

```
spec_x13 <- x13_spec(spec = "RSA5c", easter.enabled = FALSE)
sa_x13 <- x13(ipi_c_eu[, "FR"], spec = spec_x13)

wk <- new_workspace()
mp <- new_multiprocessing(wk, "sa1")
add_sa_item(wk, "sa1", sa_x13, "X13")
sa_item1 <- get_object(mp, 1)

get_model(sa_item1, wk) # Returns NULL

compute(wk)

get_model(sa_item1, wk) # Returns the SA model sa_x13
```

---

count *Count the number of objects inside a workspace or multiprocessing*

Description

Generics functions to count the number of `multiprocessing` (respectively `sa_item`) inside a `workspace` (respectively `multiprocessing`).

Usage

```
count(x)
```

Arguments

x	the `workspace` or the codemultiprocessing.

See Also

Other functions to get informations from a workspace, multiprocessing or sa_item: `get_model`, `get_name`, `get_ts`.

Examples

```
wk <- new_workspace()
mp <- new_multiprocessing(wk, "sa1")
count(wk) # 1 multiprocessing inside the workspace wk
count(mp) # 0 sa_item inside the multiprocessing mp
```

get_model *Get the seasonally adjusted model from a workspace*

Description

Generics functions to get seasonally adjusted model(s) from workspace, multiprocessing or sa_item object. get_model returns a "SA" objects while get_jmodel returns the Java objects of the models.

Usage

```
get_jmodel(x, workspace, userdefined = NULL, progress_bar = TRUE)

get_model(x, workspace, userdefined = NULL, progress_bar = TRUE)
```

Arguments

x	the object to get the seasonally adjusted model.
workspace	the workspace object where models are stored. If x is a workspace object this parameter is not used.
userdefined	vector with characters for additional output variables. (see x13 or tramoseats).
progress_bar	boolean: if TRUE a progress bar is printed.

Value

get_model() returns a seasonally adjust object (class c("SA","X13") or c("SA","TRAMO_SEATS") or list of seasonally adjust objects:

- if x is a sa_item object, get_model(x) returns a "SA" object (or a jSA object with get_jmodel(x));
- if x is a multiprocessing object, get_ts(x) returns list of length the number of sa_items, each element containing a "SA" object (or a jSA object with get_jmodel(x));
- if x is a workspace object, get_ts(x) returns list of length the number of multiprocessing, each element containing a list of a "SA" object (or a jSA object with get_jmodel(x)).

See Also

Other functions to get informations from a workspace, multiprocessing or sa_item: count, get_name, get_ts.

compute

Examples

```
spec_x13 <- x13_spec(spec = "RSA5c", easter.enabled = FALSE)
sa_x13 <- x13(ipi_c_eu[, "FR"], spec = spec_x13)
spec_ts <- tramoseats_spec(spec = "RSA5")
sa_ts <- tramoseats(ipi_c_eu[, "FR"], spec = spec_ts)

wk <- new_workspace()
mp <- new_multiprocessing(wk, "sa1")
add_sa_item(wk, "sa1", sa_x13, "X13")
add_sa_item(wk, "sa1", sa_ts, "TramoSeats")

compute(wk) # It's important to compute the workspace to get the SA model
sa_item1 <- get_object(mp, 1)

get_model(sa_item1, wk) # Extract the model of the sa_item1: its the object sa_x13

# To get all the models of the multiprocessing mp:
get_model(mp, wk)

# To get all the models of the workspace wk:
get_model(wk)
```

---

get_name *Get the Java name of a multiprocessing or a sa_item*

Description

Generics functions to get the Java name of a `multiprocessing` or a `sa_item`.

Usage

```
get_name(x)
```

Arguments

x	the object to get the name from.

Value

A `character`.

See Also

Other functions to get informations from a workspace, multiprocessing or sa_item: `count`, `get_model`, `get_ts`.

Examples

```
spec_x13 <- x13_spec(spec = "RSA5c", easter.enabled = FALSE)
sa_x13 <- x13(ipi_c_eu[, "FR"], spec = spec_x13)
spec_ts <- tramoseats_spec(spec = "RSA5")
sa_ts <- tramoseats(ipi_c_eu[, "FR"], spec = spec_ts)

wk <- new_workspace()
mp <- new_multiprocessing(wk, "sa1")
add_sa_item(wk, "sa1", sa_x13, "X13")
add_sa_item(wk, "sa1", sa_ts, "TramoSeats")

sa_item1 <- get_object(mp, 1)
sa_item2 <- get_object(mp, 2)

get_name(sa_item1) # returns "X13"
get_name(sa_item2) # returns "TramoSeats"

get_name(mp) # returns "sa1"

# To get all the name of the sa_items inside a multiprocessing:
sapply(get_all_objects(mp), get_name)

# To get all the name of the multiprocessings inside a workspace:
sapply(get_all_objects(wk), get_name)

# To get all the name of the sa_items inside a workspace:
lapply(get_all_objects(wk),function(mp){
  sapply(get_all_objects(mp), get_name)
})
```

---

get_object *Get objects inside a workspace or multiprocessing*

Description

Generics functions to get all (`get_all_objects()`) `multiprocessing` (respectively `sa_item`) from a `workspace` (respectively `multiprocessing`) or to get a given one (`get_object()`) .

Usage

```
get_object(x, pos = 1)

get_all_objects(x)
```

Arguments

`x`	the object where to extract the `multiprocessing` or the `sa_item`.
`pos`	the index of the object to extract.

Value

An object of class `multiprocessing` or `sa_item` (for `get_object()`) or a list of objects of class `multiprocessing` or `sa_item` (for `get_all_objects()`).

See Also

Other functions to get informations from a workspace, multiprocessing or sa_item: `count`, `get_model`, `get_name`, `get_ts`.

Examples

```
sa_x13 <- x13(ipi_c_eu[, "FR"], spec = "RSA5c")

wk <- new_workspace()
mp <- new_multiprocessing(wk, "sa1")
add_sa_item(wk, "sa1", sa_x13, "X13")

# Other way to get the multiprocessing:
mp <- get_object(wk, 1)
# To get the sa_item object:
sa_item <- get_object(mp, 1)
```

---

get_ts *Get the input raw time series*

Description

Generics functions to get the input raw time series of a `workspace`, `multiprocessing`, `sa_item` or `SA` object.

Usage

```
get_ts(x)
```

Arguments

`x`	the object where to get the time series.

Value

`get_ts()` returns a `ts` object or list of `ts` objects:

- if `x` is a `sa_item` or a `SA` object, `get_ts(x)` returns a single `ts` object;
- if `x` is a `multiprocessing` object, `get_ts(x)` returns list of length the number of sa_items, each a `ts` object;
- if `x` is a `workspace` object, `get_ts(x)` returns list of length the number of multiprocessing, each element containing a list of `ts` object.

See Also

Other functions to get informations from a workspace, multiprocessing or sa_item: count, get_model, get_name.

Examples

```
sa_x13 <- x13(ipi_c_eu[, "FR"], spec = "RSA5c")

wk <- new_workspace()
mp <- new_multiprocessing(wk, "sa1")
add_sa_item(wk, "sa1", sa_x13, "X13")
sa_item <- get_object(mp, 1)

  # Extracting from a SA:
get_ts(sa_x13) # Returns the ts object ipi_c_eu[, "FR"]

  # Extracting from a sa_item:
get_ts(sa_item) # Returns the ts object ipi_c_eu[, "FR"]

  # Extracting from a multiprocessing:
# Returns a list of length 1 named "X13" containing the ts object ipi_c_eu[, "FR"]:
get_ts(mp)


  # Extracting from a workspace:
# Returns a list of length 1 named "sa1" containing a list
# of length 1 named "X13" containing the ts object ipi_c_eu[, "FR"]
get_ts(wk)
```

---

ipi_c_eu *Industrial Production Indices in manufacturing in the European Union*

Description

A dataset containing on monthly industrial production indices in manufacturing in the European Union (from sts_inpr_m dataset of Eurostat). Data are based 100 in 2015 and are unadjusted, i.e. neither seasonally adjusted nor calendar adjusted.

Usage

```
ipi_c_eu
```

Format

A monthly ts object from january 1990 to december 2019 with 34 variables.

Details

The dataset contains 34 time series corresponding to the following geographical area

BE	Belgium
BG	Bulgaria
CZ	Czechia
DK	Denmark
DE	Germany (until 1990 former territory of the FRG)
EE	Estonia
IE	Ireland
EL	Greece
ES	Spain
FR	France
HR	Croatia
IT	Italy
CY	Cyprus
LV	Latvia
LT	Lithuania
LU	Luxembourg
HU	Hungary
MT	Malta
NL	Netherlands
AT	Austria
PL	Poland
PT	Portugal
RO	Romania
SI	Slovenia
SK	Slovakia
FI	Finland
SE	Sweden
UK	United Kingdom
NO	Norway
CH	Switzerland
ME	Montenegro
MK	Former Yugoslav Republic of Macedonia, the
RS	Serbia
TR	Turkey
BA	Bosnia and Herzegovina

Source

http://ec.europa.eu/eurostat/wdds/rest/data/v2.1/json/en/sts_inpr_m?nace_r2=C&precision=1&sinceTimePeriod=1980M01&unit=I15&s_adj=NSA

`jSA` *Functions around 'jSA' objects*

Description

`get_dictionary` returns the indicators that can be extracted from `"jSA"` objects, `get_indicators` extract a list of indicators and `jSA2R` returns the corresponding `"SA"`.

Usage

```
get_jspec(x, ...)

get_dictionary(x)

get_indicators(x, ...)

jSA2R(x, userdefined = NULL)
```

Arguments

x	a "jSA" object.
...	characters containing the names of the indicators to extract.
userdefined	userdefined vector with characters for additional output variables (see user_defined_variables). Only used for "SA" objects.

Details

A "jSA" object is a list with three elements:

- "result": the Java object with the results of a seasonal adjustment or a pre-adjustment method.
- "spec": the Java object with the specification of a seasonal adjustment or a pre-adjustment method.
- "dictionary": the Java object with dictionnary of a seasonal adjustment or a pre-adjustment method. In particular, it contains all the user-defined regressors.

get_dictionary returns the list of indicators that can be extracted from a jSA object by the function get_indicators.

jSA2R returns the corresponding formatted seasonal adjustment ("SA" object) or RegARIMA ("regarima" object) model.

get_jspec returns the Java object that contains the specification from an object "jSA", "X13", "TRAMO_SEATS" or "sa_item".

Value

get_dictionary a vector of characters, get_indicators returns a list containing the indicators that are extracted, jSA2R returns a "SA" or a "regarima" object and get_jspec returns a Java object.

Examples

```
myseries <- ipi_c_eu[, "FR"]
mysa <- jx13(myseries, spec = "RSA5c")
get_dictionary(mysa)

get_indicators(mysa, "decomposition.b2", "decomposition.d10")
```

```
# To convert to the R object
jSA2R(mysa)
```

---

load_workspace *Load a 'JDemetra+' workpace*

Description

Function to load a 'JDemetra+' workspace.

Usage

```
load_workspace(file)
```

Arguments

file	the path to the 'JDemetra+' workspace to load. By default a dialog box opens.

Value

An object of class "workspace".

See Also

save_workspace, get_model

---

new_workspace *Create a workspace or a multi-processing*

Description

Functions to create a 'JDemetra+' workspace (new_workspace()) add a multi-processing to it (new_multiprocessing()).

Usage

```
new_workspace()

new_multiprocessing(workspace, name)
```

Arguments

workspace	a workspace object
name	character name of the new multiprocessing

Value

new_workspace() returns an object of class workspace and new_multiprocessing() returns an object of class multiprocessing.

See Also

load_workspace, save_workspace, add_sa_item

Examples

```
# Create and export a empty 'JDemetra+' workspace
wk <- new_workspace()
mp <- new_multiprocessing(wk, "sa1")
```

---

plot *Plotting regarima, decomposition or final results of SA*

Description

Plotting methods for the S3 class objects around the seasonal adjustment: "regarima" for RegARIMA,"decomposition_X11" and "decomposition_SEATS" for the decomposition with X13 and TRAMO-SEATS, "final" for the final SA results and "SA" for the entire seasonal adjustment object. The function plot.SA just calls the function plot.final.

Usage

```
## S3 method for class 'regarima'
plot(
  x,
  which = 1:6,
  caption = list("Residuals", "Histogram of residuals", "Normal Q-Q",
   "ACF of residuals", "PACF of residuals", "Decomposition", list("Y linearised",
    "Calendar effects", "Outliers effects"))[sort(which)],
  ask = prod(par("mfcol")) < length(which) && dev.interactive(),
  ...
)

## S3 method for class 'decomposition_X11'
plot(x, first_date, last_date, caption = "S-I ratio", ylim, ...)

## S3 method for class 'decomposition_SEATS'
plot(x, first_date, last_date, caption = "S-I ratio", ylim, ...)

## S3 method for class 'final'
plot(
```

```
  x,
  first_date,
  last_date,
  forecast = TRUE,
  type_chart = c("sa-trend", "cal-seas-irr"),
  caption = c(`sa-trend` = "Y, Sa, trend", `cal-seas-irr` =
    "Cal., sea., irr.")[type_chart],
  ask = length(type_chart) > 1 && dev.interactive(),
  ylim,
  ...
)

## S3 method for class 'SA'
plot(x, ...)
```

Arguments

x	the object to plot.
which	vector with numerics specifying which graphs should be plotted: (1) "Residuals", (2) "Histogram of residuals", (3) "Normal Q-Q", (4) "ACF of residuals", (5) "PACF of residuals", (6) "Decomposition", (7) "Decomposition - zoom".
caption	list or character with the graphs titles.
ask	logicals. If TRUE, the user will in future be prompted before a new graphical page is started.
...	other parameters.
first_date	the first date to start the plot. If missing the plot starts at the beginning of the time-series.
last_date	the last date to end the plot. If missing the plot ends at the end of the time-series (eventually, including forecast).
ylim	the y limits of the plot.
forecast	logical indicating if forecasts should be included in the plot. If TRUE the forecast is plotted.
type_chart	character vector indicating which type of chart to plot.

Examples

```
myseries <- ipi_c_eu[, "FR"]
mysa <- x13(myseries, spec = c("RSA5c"))
 # RegArima
plot(mysa$regarima) # 6 graphics are plotted by default
# To only plot one graphic (here the residuals) changing the title:
plot(mysa$regarima, which = 1, caption = "Plot of residuals")
plot(mysa$regarima, which = 7)

  # Decomposition
plot(mysa$decomposition) # To plot the S-I ratio
```

```
plot(mysa$decomposition, first_date = c(2010, 1)) # To start the plot in January 2010

  # Final
plot(mysa$final) # 2 graphics are plotted by default
# To only plot one graphic (here raw data, seasonal adjusted data and trend),
# changing the last date and the title
plot(mysa$final, last_date = c(2000, 1),
     caption = "Results", type_chart = "sa-trend")
```

---

regarima *RegARIMA model, pre-adjustment in X13 and TRAMO-SEATS*

Description

regarima/regarima_x13/regarima_tramoseats decomposes the time series in a linear deterministic component and in a stochastic component. The deterministic part of the series can contain outliers, calendar effects and regression effects. The stochastic part is defined by a seasonal multiplicative ARIMA model, as discussed by BOX, G.E.P., and JENKINS, G.M. (1970). jregarima/jregarima_x13/jregarima_tramoseats does the same computation but returns the Java objects without formatting the output

Usage

```
jregarima(series, spec = NA)

jregarima_tramoseats(
  series,
  spec = c("TRfull", "TR0", "TR1", "TR2", "TR3", "TR4", "TR5")
)

jregarima_x13(series, spec = c("RG5c", "RG0", "RG1", "RG2c", "RG3", "RG4c"))

regarima(series, spec = NA)

regarima_tramoseats(
  series,
  spec = c("TRfull", "TR0", "TR1", "TR2", "TR3", "TR4", "TR5")
)

regarima_x13(series, spec = c("RG5c", "RG0", "RG1", "RG2c", "RG3", "RG4c"))
```

Arguments

series	a univariate time series
spec	model specification. For the function: • regarima, object of class c("regarima_spec","X13") or c("regarima_spec","TRAMO_SEATS")) See functions regarima_spec_x13 and regarima_spec_tramoseats.

- `regarima_x13`, predefined X13 'JDemetra+' model specification (see *Details*). The default is "RG5c".
- `regarima_tramoseats`, predefined TRAMO-SEATS 'JDemetra+' model specification (see *Details*). The default is "TRfull".

Details

In the X13 and TRAMO-SEATS seasonal adjustment the first step consists of pre-adjusting the original series with a RegARIMA model, where the original series is corrected for any deterministic effects and missing observations. This step is also referred as linearization of the original series.

The RegARIMA model (model with ARIMA errors) is specified as below.

$$z_t = y_t\beta + x_t$$

where:

- z_t - is the original series;
- $\beta = (\beta_1, ..., \beta_n)$ - a vector of regression coefficients;
- $y_t = (y_{1t}, ..., y_{nt})$ - n regression variables (outliers, calendar effects, user-defined variables);
- x_t - a disturbance that follows the general ARIMA process: $\phi(B)\delta(B)x_t = \theta(B)a_t$; $\phi(B), \delta(B)$ and $\theta(B)$ are the finite polynomials in B; a_t is a white-noise variable with zero mean and a constant variance.

The polynomial $\phi(B)$ is a stationary autoregressive (AR) polynomial in B, which is a product of the stationary regular AR polynomial in B and the stationary seasonal polynomial in B^s:

$$\phi(B) = \phi_p(B)\Phi_{bp}(B^s) = (1 + \phi_1 B + ... + \phi_p B^p)(1 + \Phi_1 B^s + ... + \Phi_{bp} B^{bps})$$

where:

- p - number of regular AR terms (here and in 'JDemetra+' $p \leq 3$);
- bp - number of seasonal AR terms (here and in 'JDemetra+' $bp \leq 1$);
- s - number of observations per year (frequency of the time series).

The polynomial $\theta(B)$ is an invertible moving average (MA) polynomial in B, which is a product of the invertible regular MA polynomial in B and the invertible seasonal MA polynomial in B^s:

$$\theta(B) = \theta_q(B)\Theta_{bq}(B^s) = (1 + \theta_1 B + ... + \theta_q B^q)(1 + \Theta_1 B^s + ... + \Theta_{bq} B^{bqs})$$

where:

- q - number of regular MA terms (here and in 'JDemetra+' $q \leq 3$);
- bq - number of seasonal MA terms (here and in 'JDemetra+' $bq \leq 1$);

The polynomial $\delta(B)$ is the non-stationary AR polynomial in B (unit roots):

$$\delta(B) = (1 - B)^d(1 - B^s)^{d_s}$$

where:

- d - regular differencing order (here and in 'JDemetra+' $d \leq 1$);
- d_s - seasonal differencing order (here and in 'JDemetra+' $d_s \leq 1$);

Notations used for AR and MA processes, model denoted as ARIMA $(P, D, Q)(BP, BD, BQ)$, are consistent with those in 'JDemetra+'.

As regards the available predefined 'JDemetra+' X13 and TRAMO-SEATS model specifications, they are described in the tables below.

X13:

Identifier	Log/level detection	Outliers detection	Calendar effects	ARIMA
RG0	*NA*	*NA*	*NA*	Airline(+mean)
RG1	automatic	AO/LS/TC	*NA*	Airline(+mean)
RG2c	automatic	AO/LS/TC	2 td vars + Easter	Airline(+mean)
RG3	automatic	AO/LS/TC	*NA*	automatic
RG4c	automatic	AO/LS/TC	2 td vars + Easter	automatic
RG5c	automatic	AO/LS/TC	7 td vars + Easter	automatic

TRAMO-SEATS:

Identifier	Log/level detection	Outliers detection	Calendar effects	ARIMA
TR0	*NA*	*NA*	*NA*	Airline(+mean)
TR1	automatic	AO/LS/TC	*NA*	Airline(+mean)
TR2	automatic	AO/LS/TC	2 td vars + Easter	Airline(+mean)
TR3	automatic	AO/LS/TC	*NA*	automatic
TR4	automatic	AO/LS/TC	2 td vars + Easter	automatic
TR5	automatic	AO/LS/TC	7 td vars + Easter	automatic
TRfull	automatic	AO/LS/TC	automatic	automatic

Value

`jregarima/jregarima_x13/jregarima_tramoseats` return a `jSA` object. It contains the Java objects of the result of the preadjustment method without any formatting. Therefore the computation is faster than with `regarima/regarima_x13/regarima_tramoseats`. The results can the seasonal adjustment can be extract by `get_indicators`.

`regarima/regarima_x13/regarima_tramoseats` return an object of class `"regarima"` and sub-class `"X13"` or `"TRAMO_SEATS"`. `regarima_x13` returns an object of class `c("regarima","X13")` and `regarima_tramoseats` an object of class `c("regarima","TRAMO_SEATS")`. For the function `regarima`, the sub-class of the object depends on the used method that is defined by the class of the `spec` object.

An object of class `"regarima"` is a list containing the following components:

`specification` list with the model specification as defined by the `spec` argument. See also Value of the `regarima_spec_x13` and `regarima_spec_tramoseats` functions.

`arma` vector with the orders of the autoregressive (AR), moving average (MA), seasonal AR and seasonal MA processes, as well as with the regular and seasonal differencing orders (P,D,Q) (BP,BD,BQ).

`arima.coefficients`
: matrix with the regular and seasonal AR and MA coefficients. The matrix contains the estimated coefficients, standard errors and t-statistics values. The estimated coefficients can be also extracted with the function `coef` (the output includes also the regression coefficients).

`regression.coefficients`
: matrix with the regression variables (i.e.: mean, calendar effect, outliers and user-defined regressors) coefficients. The matrix contains the estimated coefficients, standard errors and t-statistics values. The estimated coefficients can be also extracted with the function `coef` (output includes also the arima coefficients).

`loglik`
: matrix containing the log-likelihood of the RegARIMA model as well as the associated model selection criteria statistics (AIC, AICC, BIC and BICC) and parameters (`np` = number of parameters in the likelihood, `neffectiveobs` = number of effective observations in the likelihood). These statistics can be also extracted with the function `logLik`.

`model`
: list containing the information on the model specification after its estimation (`spec_rslt`), as well as the decomposed elements of the input series (ts matrix, `effects`). The model specification includes the information on the estimation method (`Model`) and time span (`T.span`), whether the original series was log transformed (`Log transformation`) and details on the regression part of the RegARIMA model; i.e. if it includes a `Mean`, `Trading days` effects (if yes, it provides the number of regressors), `Leap year` effect, `Easter` effect and whether outliers were detected (`Outliers`; if yes, it provides the number of outliers). The decomposed elements of the input series contain the linearised series (`y_lin`) and the deterministic components; i.e.: trading days effect (`tde`), Easter effect (`ee`), other moving holidays effect (`omhe`) and outliers effect (total - `out`, related to irregular - `out_i`, related to trend - `out_t`, related to seasonal - `out_s`).

`residuals`
: the residuals (time series). They can be also extracted with the function `residuals`.

`residuals.stat`
: List containing statistics on the RegARIMA residuals. It provides residuals standard error (`st.error`) and results for the tests on the normality, independence and linearity of the residuals (`tests`) - object of class `c("regarima_rtests","data.frame")`.

`forecast`
: ts matrix containing the forecast of the original series (`fcst`) and it's standard error (`fcsterr`).

References

Info on 'JDemetra+', usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en

BOX G.E.P. and JENKINS G.M. (1970), "Time Series Analysis: Forecasting and Control", Holden-Day, San Francisco.

BOX G.E.P., JENKINS G.M., REINSEL G.C. and LJUNG G.M. (2015), "Time Series Analysis: Forecasting and Control", John Wiley & Sons, Hoboken, N. J., 5th edition.

Examples

```
 # X13 method
myseries <- ipi_c_eu[, "FR"]
myreg <- regarima_x13(myseries, spec ="RG5c")
summary(myreg)
plot(myreg)

myspec1 <- regarima_spec_x13(myreg, tradingdays.option = "WorkingDays")
myreg1 <- regarima(myseries, myspec1)

myspec2 <- regarima_spec_x13(myreg, usrdef.outliersEnabled = TRUE,
             usrdef.outliersType = c("LS", "AO"),
             usrdef.outliersDate = c("2008-10-01", "2002-01-01"),
             usrdef.outliersCoef = c(36, 14),
             transform.function = "None")
myreg2 <- regarima(myseries, myspec2)
myreg2

myspec3 <- regarima_spec_x13(myreg, automdl.enabled = FALSE,
             arima.p = 1, arima.q = 1,
             arima.bp = 0, arima.bq = 1,
             arima.coefEnabled = TRUE,
             arima.coef = c(-0.8, -0.6, 0),
             arima.coefType = c(rep("Fixed", 2), "Undefined"))
s_arimaCoef(myspec3)
myreg3 <- regarima(myseries, myspec3)
summary(myreg3)
plot(myreg3)

 # TRAMO-SEATS method
myspec <- regarima_spec_tramoseats("TRfull")
myreg <- regarima(myseries, myspec)
myreg

myspec2 <- regarima_spec_tramoseats(myspec, tradingdays.mauto = "Unused",
             tradingdays.option = "WorkingDays",
             easter.type = "Standard",
             automdl.enabled = FALSE, arima.mu = TRUE)
myreg2 <- regarima(myseries, myspec2)

var1 <- ts(rnorm(length(myseries))*10, start = start(myseries), frequency = 12)
var2 <- ts(rnorm(length(myseries))*100, start = start(myseries), frequency = 12)
var <- ts.union(var1, var2)
myspec3 <- regarima_spec_tramoseats(myspec,
             usrdef.varEnabled = TRUE, usrdef.var = var)
s_preVar(myspec3)
myreg3 <- regarima(myseries, myspec3)
myreg3
```

`regarima_spec_tramoseats` *RegARIMA model specification, pre-adjustment in TRAMO-SEATS*

Description

Function to create (and/or modify) a `c("regarima_spec","TRAMO_SEATS")` class object with the RegARIMA model specification for the TRAMO-SEATS method. The object can be created from a predefined 'JDemetra+' model specification (a `character`), a previous specification (`c("regarima_spec","TRAMO_SEATS")` object) or a TRAMO-SEATS RegARIMA model (`c("regarima","TRAMO_SEATS")`).

Usage

```
regarima_spec_tramoseats(
  spec = c("TRfull", "TR0", "TR1", "TR2", "TR3", "TR4", "TR5"),
  preliminary.check = NA,
  estimate.from = NA_character_,
  estimate.to = NA_character_,
  estimate.first = NA_integer_,
  estimate.last = NA_integer_,
  estimate.exclFirst = NA_integer_,
  estimate.exclLast = NA_integer_,
  estimate.tol = NA_integer_,
  estimate.eml = NA,
  estimate.urfinal = NA_integer_,
  transform.function = c(NA, "Auto", "None", "Log"),
  transform.fct = NA_integer_,
  usrdef.outliersEnabled = NA,
  usrdef.outliersType = NA,
  usrdef.outliersDate = NA,
  usrdef.outliersCoef = NA,
  usrdef.varEnabled = NA,
  usrdef.var = NA,
  usrdef.varType = NA,
  usrdef.varCoef = NA,
  tradingdays.mauto = c(NA, "Unused", "FTest", "WaldTest"),
  tradingdays.pftd = NA_integer_,
 tradingdays.option = c(NA, "TradingDays", "WorkingDays", "UserDefined", "None"),
  tradingdays.leapyear = NA,
  tradingdays.stocktd = NA_integer_,
  tradingdays.test = c(NA, "Separate_T", "Joint_F", "None"),
 easter.type = c(NA, "Unused", "Standard", "IncludeEaster", "IncludeEasterMonday"),
  easter.julian = NA,
  easter.duration = NA_integer_,
  easter.test = NA,
  outlier.enabled = NA,
  outlier.from = NA_character_,
  outlier.to = NA_character_,
```

```
  outlier.first = NA_integer_,
  outlier.last = NA_integer_,
  outlier.exclFirst = NA_integer_,
  outlier.exclLast = NA_integer_,
  outlier.ao = NA,
  outlier.tc = NA,
  outlier.ls = NA,
  outlier.so = NA,
  outlier.usedefcv = NA,
  outlier.cv = NA_integer_,
  outlier.eml = NA,
  outlier.tcrate = NA_integer_,
  automdl.enabled = NA,
  automdl.acceptdefault = NA,
  automdl.cancel = NA_integer_,
  automdl.ub1 = NA_integer_,
  automdl.ub2 = NA_integer_,
  automdl.armalimit = NA_integer_,
  automdl.reducecv = NA_integer_,
  automdl.ljungboxlimit = NA_integer_,
  automdl.compare = NA,
  arima.mu = NA,
  arima.p = NA_integer_,
  arima.d = NA_integer_,
  arima.q = NA_integer_,
  arima.bp = NA_integer_,
  arima.bd = NA_integer_,
  arima.bq = NA_integer_,
  arima.coefEnabled = NA,
  arima.coef = NA,
  arima.coefType = NA,
  fcst.horizon = NA_integer_
)
```

Arguments

`spec`	model specification. It can be a `character` of predefined 'JDemetra+' model specification (see *Details*), an object of class `c("regarima_spec","TRAMO_SEATS")` or an object of class `c("regarima","TRAMO_SEATS")`. The default is `"TRfull"`.
`preliminary.check`	boolean to check the quality of the input series and exclude highly problematic ones: e.g. these with a number of identical observations and/or missing values above pre-specified threshold values. The time span of the series to be used for the estimation of the RegArima model coefficients (default from 1900-01-01 to 2020-12-31) is controlled by the following six variables: `estimate.from`,`estimate.to`,`estimate.first`,`estimate.last`,`estimate.exc` and `estimate.exclLast`; where `estimate.from` and `estimate.to` have priority over remaining span control variables, `estimate.last` and `estimate.first`

have priority over estimate.exclFirst and estimate.exclLast, and estimate.last has priority over estimate.first.

estimate.from character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). Can be combined with estimate.to.

estimate.to character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). Can be combined with estimate.from.

estimate.first numeric specifying the number of periods considered at the beginning of the series.

estimate.last numeric specifying the number of periods considered at the end of the series.

estimate.exclFirst
numeric specifying the number of periods excluded at the beginning of the series. Can be combined with estimate.exclLast.

estimate.exclLast
numeric specifying the number of periods excluded at the end of the series. Can be combined with estimate.exclFirst.

estimate.tol 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.

estimate.eml logicals, exact maximum likelihood estimation. If TRUE the program performs an exact maximum likelihood estimation. If FASLE the Unconditional Least Squares method is used.

estimate.urfinal
numeric, 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 less than this number, 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.

transform.function
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.

transform.fct numeric controlling the bias in the log/level pre-test: transform.fct > 1 favours levels, transform.fct< 1 favours logs. Considered only when transform.function is set to "Auto".

Control variables for the pre-specified outliers. The pre-specified outliers are used in the model only if they are enabled (usrdef.outliersEnabled=TRUE) and the outliers' type (usrdef.outliersType) and date (usrdef.outliersDate) are provided.

usrdef.outliersEnabled
logicals. If TRUE the program uses the pre-specified outliers.

usrdef.outliersType
vector defining the outliers' type. Possible types are: ("AO") - additive, ("LS") - level shift, ("TC") - transitory change, ("SO") - seasonal outlier. E.g.: usrdef.outliersType =c("AO","AO","LS").

usrdef.outliersDate
: vector defining the outliers' date. The dates should be characters in format "YYYY-MM-DD". E.g.: usrdef.outliersDate=c("2009-10-01","2005-02-01","2003-04-01").

usrdef.outliersCoef
: vector providing fixed coefficients for the outliers. The coefficients can't be fixed if transform.function is set to "Auto" - the series transformation need to be pre-defined. E.g.: usrdef.outliersCoef=c(200,170,20).

Control variables for the user-defined variables:

usrdef.varEnabled
: logicals. If TRUE the program uses the user-defined variables.

usrdef.var
: time series (ts) or matrix of time series (mts) with the user-defined variables.

usrdef.varType
: vector of character(s) defining the user-defined variables component type. Possible types are: "Undefined","Series","Trend","Seasonal","SeasonallyAdjusted","Irregular", The type "Calendar" has to be used with tradingdays.option = "UserDefined" to use user-defined calendar regressors. If not specified, the program will assign the "Undefined" type.

usrdef.varCoef
: vector providing fixed coefficients for the user-defined variables. The coefficients can't be fixed if transform.function is set to "Auto" - the series transformation need to be pre-defined.

tradingdays.mauto
: defines whether the calendar effects should be added to the model manually ("Unused") or automatically. In the automatic selection, the choice of the number of calendar variables can be based on the F-Test ("FTest") or the Wald Test ("WaldTest"); the model with higher F value is chosen, provided that it is higher than tradingdays.pftd).

tradingdays.pftd
: numeric. P-value applied in the test specified by the automatic parameter (tradingdays.mauto) to assess the significance of the pre-tested calendar effects variables and whether they should be included in the RegArima model.

Control variables for the manual selection of calendar effects variables (tradingdays.mauto is set to "Unused"):

tradingdays.option
: defines the type of the trading days regression variables: "TradingDays" - six day-of-the-week regression variables; "WorkingDays" - one working/non-working day contrast variable; "None" - no correction for trading days and working days effects; "UserDefined" - user-defined trading days regressors (regressors have to be defined by the usrdef.var argument with usrdef.varType set to "Calendar" and usrdef.varEnabled = TRUE). "None" has also to be chosen for the "day-of-week effects" correction (tradingdays.stocktd to be modified accordingly).

tradingdays.leapyear
: logicals. Specifies if the leap-year correction should be included. If TRUE the model includes the leap-year effect.

tradingdays.stocktd
: 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). Modifications of this variable are taken into account only when tradingdays.option is set to "None".

`tradingdays.test`
: defines the pre-tests of the trading day effects: `"None"` - calendar variables are used in the model without pre-testing; `"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.

`easter.type`
: specifies the presence and the length of the Easter effect: `"Unused"` - 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.

`easter.julian`
: logicals. If `TRUE` the program uses the Julian Easter (expressed in Gregorian calendar).

`easter.duration`
: numeric indicating the duration of the Easter effect (length in days, between 1 and 15).

`easter.test`
: logicals. If `TRUE` the program performs a t-test for the significance of the Easter effect. The Easter effect is considered as significant if the modulus of t-statistic is greater than 1.96.

`outlier.enabled`
: logicals. If `TRUE` the automatic detection of outliers is enabled in the defined time span.

  The time span of the series to be searched for outliers (default from 1900-01-01 to 2020-12-31) is controlled by the following six variables: `outlier.from`,`outlier.to`,`outlier.first` and `outlier.exclLast`; where `outlier.from` and `outlier.to` have priority over remaining span control variables, `outlier.last` and `outlier.first` have priority over `outlier.exclFirst` and `outlier.exclLast`, and `outlier.last` has priority over `outlier.first`.

`outlier.from`
: character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). Can be combined with `outlier.to`.

`outlier.to`
: character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). Can be combined with `outlier.from`.

`outlier.first`
: numeric specifying the number of periods considered at the beginning of the series.

`outlier.last`
: numeric specifying the number of periods considered at the end of the series.

`outlier.exclFirst`
: numeric specifying the number of periods excluded at the beginning of the series. Can be combined with `outlier.exclLast`.

`outlier.exclLast`
: numeric specifying the number of periods excluded at the end of the series. Can be combined with `outlier.exclFirst`.

`outlier.ao`
: logicals. If `TRUE` the automatic detection of additive outliers is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.tc` logicals. If `TRUE` the automatic detection of transitory changes is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.ls` logicals. If `TRUE` the automatic detection of level shifts is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.so` logicals. If `TRUE` the automatic detection of seasonal outliers is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.usedefcv` logicals. If `TRUE` the critical value for the outliers' detection procedure is automatically determined by the number of observations in the outlier detection time span. If `FALSE` the procedure uses the inputted critical value (`outlier.cv`).

`outlier.cv` numeric. Inputted critical value for the outliers' detection procedure. The modification of this variable is taken in to account only when `outlier.usedefcv` is set to `FALSE`.

`outlier.eml` logicals, exact likelihood estimation method.Controls the method applied for a parameter estimation in the intermediate steps of the automatic detection and correction of outliers. If `TRUE` an exact likelihood estimation method is used, when `FALSE` the fast Hannan-Rissanen method is used.

`outlier.tcrate` numeric. The rate of decay for the transitory change outlier.

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

Control variables for the automatic modelling of the ARIMA model (`automdl.enabled` is set to `TRUE`):

`automdl.acceptdefault` logicals. If `TRUE` the default model (ARIMA(0,1,1)(0,1,1)) may be chosen in the first step of the automatic model identification. If the Ljung-Box Q statistics for the residuals is acceptable, the default model is accepted and no further attempt will be made to identify any other.

`automdl.cancel` numeric, cancelation limit. If the difference in moduli of an AR and an MA roots (when estimating ARIMA(1,0,1)(1,0,1) models in the second step of the automatic identification of the differencing orders) is smaller than cancelation limit, the two roots are assumed equal and cancel out.

`automdl.ub1` numeric, 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.

`automdl.ub2` numeric, 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).

`automdl.armalimit`
: numeric, arma limit. It is the threshold value for t-statistics of ARMA coefficients and constant term used for the final test of model parsimony. If the highest order ARMA coefficient has a t-value less than this value in magnitude, the order of the model is reduced. Also if the constant term has a t-value less than arma limit in magnitude, it is removed from the set of regressors.

`automdl.reducecv`
: numeric, ReduceCV. The percentage by which the outlier's 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)xCV, where CV is the original critical value.

`automdl.ljungboxlimit`
: numeric, Ljung Box limit. 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, the model is rejected, the outlier critical value is reduced, and model and outlier identification (if specified) is redone with a reduced value.

`automdl.compare`
: logicals. 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.

Control variables for the non-automatic modelling of the ARIMA model (`automdl.enabled` is set to `FALSE`):

`arima.mu`
: logicals. If `TRUE`, the mean is considered as part of the ARIMA model.

`arima.p`
: numeric. The order of the non-seasonal autoregressive (AR) polynomial.

`arima.d`
: numeric. Regular differencing order.

`arima.q`
: numeric. The order of the non-seasonal moving average (MA) polynomial.

`arima.bp`
: numeric. The order of the seasonal autoregressive (AR) polynomial.

`arima.bd`
: numeric. Seasonal differencing order.

`arima.bq`
: numeric. The order of the seasonal moving average (MA) polynomial.

Control variables for the user-defined ARMA coefficients. Coefficients can be defined for the regular and seasonal autoregressive (AR) polynomials and moving average (MA) polynomials. The model considers the coefficients only if the procedure for their estimation (`arima.coefType`) is provided, and the number of provided coefficients matches the sum of (regular and seasonal) AR and MA orders (`p,q,bp,bq`).

`arima.coefEnabled`
: logicals. If `TRUE` the program uses the user-defined ARMA coefficients.

`arima.coef`
: vector providing the coefficients for the regular and seasonal AR and MA polynominals. The length of the vector must equal the sum of the regular and seasonal AR and MA orders. The coefficients shall be provided in the 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 `arima.p=1,arima.q=0,arima.bp=1` and `arima.bq=0`.

`arima.coefType` vector defining ARMA coefficients estimation procedure. Possible procedures are: `"Undefined"` - no use of user-defined input (i.e. coefficients are estimated), `"Fixed"` - fixes the coefficients at the value provided by the user, `"Initial"` - the value defined by the user is used as initial condition. For orders for which the coefficients shall not be defined, the `arima.coef` can be set to `NA` or `0` or the `arima.coefType` can be set to `"Undefined"`. E.g.: `arima.coef = c(-0.8,-0.6,NA)`, `arima.coefType = c("Fixed","Fixed","Undefined")`.

`fcst.horizon` numeric, forecasts horizon. Length of the forecasts generated by the RegARIMA model in periods (positive values) or years (negative values). By default the program generates two years forecasts (`fcst.horizon` set to `-2`).

Details

The available predefined 'JDemetra+' model specifications are described in the table below.

Identifier	Log/level detection	Outliers detection	Calendar effects	ARIMA
TR0	*NA*	*NA*	*NA*	Airline(+mean)
TR1	automatic	AO/LS/TC	*NA*	Airline(+mean)
TR2	automatic	AO/LS/TC	2 td vars + Easter	Airline(+mean)
TR3	automatic	AO/LS/TC	*NA*	automatic
TR4	automatic	AO/LS/TC	2 td vars + Easter	automatic
TR5	automatic	AO/LS/TC	7 td vars + Easter	automatic
TRfull	automatic	AO/LS/TC	automatic	automatic

Value

A list of class `c("regarima_spec","TRAMO_SEATS")` with the below components. Each component refers to different part of the RegARIMA model specification, mirroring the arguments of the function (for details see arguments description). Each of the lowest-level component (except span, pre-specified outliers, user-defined variables and pre-specified ARMA coefficients) is structured within a data frame with columns denoting different variables of the model specification and rows referring to: first row - base specification, as provided within the argument `spec`; second row - user modifications as specified by the remaining arguments of the function (e.g.: `arima.d`); and third row - final model specification, values that will be used in the function `regarima`. The final specification (third row) shall include user modifications (row two) unless they were wrongly specified. The pre-specified outliers, user-defined variables and pre-specified ARMA coefficients consist of a list with the `Predefined` (base model specification) and `Final` values.

`estimate` data frame. Variables referring to: `span` - time span for the model estimation, `tolerance` - argument `estimate.tol`, `exact_ml` - argument `estimate.eml`, `urfinal` - argument `esimate.urfinal`. The final values can be also accessed with the function `s_estimate`.

`transform` data frame. Variables referring to: `tfunction` - argument `transform.function`, `fct` - argument `transform.fct`. The final values can be also accessed with the function `s_transform`.

`regression` list including the information on the user-defined variables (`userdef`), `trading.days` effect and `easter` effect. The user-defined part includes: `specification` -

data frame with the information if pre-specified outliers (outlier) and user-defined variables (variables) are included in the model and if fixed coefficients are used (outlier.coef and variables.coef). The final values can be also accessed with the function s_usrdef; outliers - matrixes with the outliers (Predefined and Final). The final outliers can be also accessed with the function s_preOut; and variables - list with the Predefined and Final user-defined variables (series) and its description (description) including the information on the variable type and values of fixed coefficients. The final user-defined variables can be also accessed with the function s_preVar. Within the data frame trading.days variables refer to: automatic - argument tradingdays.mauto, pftd - argument tradingdays.pftd, option - argument tradingdays.option, leapyear - argument tradingdays.leapyear, stocktd - argument tradingdays.stocktd, test - argument tradingdays.test. The final trading.days values can be also accessed with the function s_td. Within the data frame easter variables refer to: type - argument easter.type, julian - argument easter.julian, duration - argument easter.duration, test - argument easter.test. The final easter values can be also accessed with the function s_easter.

outliers — data frame. Variables referring to: ao - argument outlier.ao, tc - argument outlier.tc, ls - argument outlier.ls, so - argument outlier.so, usedefcv - argument outlier.usedefcv, cv - argument outlier.cv, eml - argument outlier.eml, tcrate - argument outlier.tcrate. The final values can be also accessed with the function s_out.

arima — list containing a data frame with the ARIMA settings (specification) and matrixes with the information on the pre-specified ARMA coefficients (coefficients). The matrix Predefined refers to the pre-defined model specification and matrix Final to the final specification. Both matrixes contain the value of the ARMA coefficients and the procedure for its estimation. Within the data frame specification the variable enabled refer to the argument automdl.enabled and all the remaining variables (automdl.acceptdefault,automdl.cancel,automdl.ub1,automdl.ub to the respective function arguments. The final values of the specification can be also accessed with the function s_arima and final pre-specified ARMA coefficients with the function s_arimaCoef.

forecast — data frame with the forecast horizon (argument fcst.horizon). The final value can be also accessed with the function s_fcst.

span — matrix containing the final time span for the model estimation and outliers' detection. Contains the same information as the variable span in the data frames estimate and outliers.The matrix can be also accessed with the function s_span.

References

Info on 'JDemetra+', usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en

Examples

```
myseries <- ipi_c_eu[, "FR"]
```

```
myspec1 <- regarima_spec_tramoseats(spec = "TRfull")
myreg1 <- regarima(myseries, spec = myspec1)

 # Modify a pre-specified model specification
myspec2 <- regarima_spec_tramoseats(spec = "TRfull",
             tradingdays.mauto = "Unused",
             tradingdays.option = "WorkingDays",
             easter.type = "Standard",
             automdl.enabled = FALSE, arima.mu = TRUE)
myreg2 <- regarima(myseries, spec = myspec2)

 # Modify the model specification from a "regarima" object
myspec3 <- regarima_spec_tramoseats(myreg1,
             tradingdays.mauto = "Unused",
             tradingdays.option = "WorkingDays",
             easter.type = "Standard", automdl.enabled = FALSE,
             arima.mu = TRUE)
myreg3 <- regarima(myseries, myspec3)

 # Modify the model specification from a "regarima_spec" object
myspec4 <- regarima_spec_tramoseats(myspec1,
             tradingdays.mauto = "Unused",
             tradingdays.option = "WorkingDays",
             easter.type = "Standard",
             automdl.enabled = FALSE, arima.mu = TRUE)
myreg4 <- regarima(myseries, myspec4)

 # Pre-specified outliers
myspec1 <- regarima_spec_tramoseats(spec = "TRfull",
             usrdef.outliersEnabled = TRUE,
             usrdef.outliersType = c("LS", "LS"),
             usrdef.outliersDate = c("2008-10-01" ,"2003-01-01"),
             usrdef.outliersCoef = c(10, -8), transform.function = "None")
s_preOut(myspec1)
myreg1 <- regarima(myseries, myspec1)
myreg1
s_preOut(myreg1)


 # User-defined variables
var1 <- ts(rnorm(length(myseries))*10, start = start(myseries),
           frequency = 12)
var2 <- ts(rnorm(length(myseries))*100, start = start(myseries),
           frequency = 12)
var <- ts.union(var1, var2)

myspec1 <- regarima_spec_tramoseats(spec = "TRfull",
            usrdef.varEnabled = TRUE, usrdef.var = var)
s_preVar(myspec1)
myreg1 <- regarima(myseries,myspec1)

myspec2 <- regarima_spec_tramoseats(spec = "TRfull",
             usrdef.varEnabled = TRUE,
```

```
             usrdef.var = var, usrdef.varCoef = c(17,-1),
             transform.function = "None")
myreg2 <- regarima(myseries, myspec2)

 # Pre-specified ARMA coefficients
myspec1 <- regarima_spec_tramoseats(spec = "TRfull",
             arima.coefEnabled = TRUE, automdl.enabled = FALSE,
             arima.p = 2, arima.q = 0, arima.bp = 1, arima.bq = 1,
             arima.coef = c(-0.12, -0.12, -0.3, -0.99),
             arima.coefType = rep("Fixed", 4))
myreg1 <- regarima(myseries, myspec1)
myreg1
summary(myreg1)
s_arimaCoef(myspec1)
s_arimaCoef(myreg1)
```

`regarima_spec_x13` *RegARIMA model specification, pre-adjustment in X13*

Description

Function to create (and/or modify) a `c("regarima_spec","X13")` class object with the RegARIMA model specification for the X13 method. The object can be created from a predefined 'JDemetra+' model specification (a `character`), a previous specification (`c("regarima_spec","X13")` object) or a X13 RegARIMA model (`c("regarima","X13")`).

Usage

```
regarima_spec_x13(
  spec = c("RG5c", "RG0", "RG1", "RG2c", "RG3", "RG4c"),
  preliminary.check = NA,
  estimate.from = NA_character_,
  estimate.to = NA_character_,
  estimate.first = NA_integer_,
  estimate.last = NA_integer_,
  estimate.exclFirst = NA_integer_,
  estimate.exclLast = NA_integer_,
  estimate.tol = NA_integer_,
  transform.function = c(NA, "Auto", "None", "Log"),
  transform.adjust = c(NA, "None", "LeapYear", "LengthOfPeriod"),
  transform.aicdiff = NA_integer_,
  usrdef.outliersEnabled = NA,
  usrdef.outliersType = NA,
  usrdef.outliersDate = NA,
  usrdef.outliersCoef = NA,
  usrdef.varEnabled = NA,
  usrdef.var = NA,
```

```
 usrdef.varType = NA,
 usrdef.varCoef = NA,
tradingdays.option = c(NA, "TradingDays", "WorkingDays", "UserDefined", "None"),
 tradingdays.autoadjust = NA,
 tradingdays.leapyear = c(NA, "LeapYear", "LengthOfPeriod", "None"),
 tradingdays.stocktd = NA_integer_,
 tradingdays.test = c(NA, "Remove", "Add", "None"),
 easter.enabled = NA,
 easter.julian = NA,
 easter.duration = NA_integer_,
 easter.test = c(NA, "Add", "Remove", "None"),
 outlier.enabled = NA,
 outlier.from = NA_character_,
 outlier.to = NA_character_,
 outlier.first = NA_integer_,
 outlier.last = NA_integer_,
 outlier.exclFirst = NA_integer_,
 outlier.exclLast = NA_integer_,
 outlier.ao = NA,
 outlier.tc = NA,
 outlier.ls = NA,
 outlier.so = NA,
 outlier.usedefcv = NA,
 outlier.cv = NA_integer_,
 outlier.method = c(NA, "AddOne", "AddAll"),
 outlier.tcrate = NA_integer_,
 automdl.enabled = NA,
 automdl.acceptdefault = NA,
 automdl.cancel = NA_integer_,
 automdl.ub1 = NA_integer_,
 automdl.ub2 = NA_integer_,
 automdl.mixed = NA,
 automdl.balanced = NA,
 automdl.armalimit = NA_integer_,
 automdl.reducecv = NA_integer_,
 automdl.ljungboxlimit = NA_integer_,
 automdl.ubfinal = NA_integer_,
 arima.mu = NA,
 arima.p = NA_integer_,
 arima.d = NA_integer_,
 arima.q = NA_integer_,
 arima.bp = NA_integer_,
 arima.bd = NA_integer_,
 arima.bq = NA_integer_,
 arima.coefEnabled = NA,
 arima.coef = NA,
 arima.coefType = NA,
 fcst.horizon = NA_integer_
```

)

Arguments

spec	model specification. It can be a character of predefined 'JDemetra+' model specification (see *Details*), an object of class c("regarima_spec","X13") or an object of class c("regarima","X13"). The default is "RG5c".
preliminary.check	boolean to check the quality of the input series and exclude highly problematic ones: e.g. these with a number of identical observations and/or missing values above pre-specified threshold values. The time span of the series to be used for the estimation of the RegARIMA model coefficients (default from 1900-01-01 to 2020-12-31) is controlled by the following six variables: estimate.from,estimate.to,estimate.first,estimate.last,estimate.e and estimate.exclLast; where estimate.from and estimate.to have priority over remaining span control variables, estimate.last and estimate.first have priority over estimate.exclFirst and estimate.exclLast, and estimate.last has priority over estimate.first.
estimate.from	character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). Can be combined with estimate.to.
estimate.to	character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). Can be combined with estimate.from.
estimate.first	numeric specifying the number of periods considered at the beginning of the series.
estimate.last	numeric specifying the number of periods considered at the end of the series.
estimate.exclFirst	numeric specifying the number of periods excluded at the beginning of the series. Can be combined with estimate.exclLast.
estimate.exclLast	numeric specifying the number of periods excluded at the end of the series. Can be combined with estimate.exclFirst.
estimate.tol	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.
transform.function	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.
transform.adjust	pre-adjustment of the input series for 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 transform.function is set to "Log".
transform.aicdiff	numeric defining the difference in AICC needed to accept no transformation when the automatic transformation selection is chosen (considered only when transform.function is set to "Auto").

Control variables for the pre-specified outliers. The pre-specified outliers are used in the model only if they are enabled (usrdef.outliersEnabled=TRUE) and the outliers' type (usrdef.outliersType) and date (usrdef.outliersDate) are provided.

usrdef.outliersEnabled
: logicals. If TRUE the program uses the pre-specified outliers.

usrdef.outliersType
: vector defining the outliers' type. Possible types are: ("AO") - additive, ("LS") - level shift, ("TC") - transitory change, ("SO") - seasonal outlier. E.g.: usrdef.outliersType =c("AO","AO","LS").

usrdef.outliersDate
: vector defining the outliers' date. The dates should be characters in format "YYYY-MM-DD". E.g.: usrdef.outliersDate=c("2009-10-01","2005-02-01","2003-04-01").

usrdef.outliersCoef
: vector providing fixed coefficients for the outliers. The coefficients can't be fixed if transform.function is set to "Auto" - the series transformation need to be pre-defined. E.g.: usrdef.outliersCoef=c(200,170,20).

Control variables for the user-defined variables:

usrdef.varEnabled
: logicals. If TRUE the program uses the user-defined variables.

usrdef.var
: time series (ts) or matrix of time series (mts) with the user-defined variables.

usrdef.varType
: vector of character(s) defining the user-defined variables component type. Possible types are: "Undefined","Series","Trend","Seasonal","SeasonallyAdjusted","Irregular", The type "Calendar" has to be used with tradingdays.option = "UserDefined" to use user-defined calendar regressors. If not specified, the program will assign the "Undefined" type.

usrdef.varCoef
: vector providing fixed coefficients for the user-defined variables. The coefficients can't be fixed if transform.function is set to "Auto" - the series transformation need to be pre-defined.

tradingdays.option
: defines the type of the trading days regression variables: "TradingDays" - six day-of-the-week regression variables; "WorkingDays" - one working/non-working day contrast variable; "None" - no correction for trading days and working days effects; "UserDefined" - user-defined trading days regressors (regressors have to be defined by the usrdef.var argument with usrdef.varType set to "Calendar" and usrdef.varEnabled = TRUE). "None" has also to be chosen for the "day-of-week effects" correction (tradingdays.stocktd to be modified accordingly).

tradingdays.autoadjust
: logicals. If TRUE the program corrects automatically for the leap year effect. Modifications of this variable are taken into account only when transform.function is set to "Auto".

tradingdays.leapyear
: option for including the leap-year effect in the model: "LeapYear" - leap year effect; "LengthOfPeriod" - length of period, "None" - no effect included. The leap-year effect can be pre-specified in the model only if the input series was

not pre-adjusted (transform.adjust set to "None") and the automatic correction for the leap-year effect was not selected (tradingdays.autoadjust set to FALSE).

tradingdays.stocktd
: 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). Modifications of this variable are taken into account only when tradingdays.option is set to "None".

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

easter.enabled
: logicals. If TRUE the program considers the Easter effect in the model.

easter.julian
: logicals. If TRUE the program uses the Julian Easter (expressed in Gregorian calendar).

easter.duration
: numeric indicating the duration of the Easter effect (length in days, between 1 and 20).

easter.test
: defines the pre-tests for the significance of the Easter effect based on the t-statistic (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 belong 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.

outlier.enabled
: logicals. If TRUE the automatic detection of outliers is enabled in the defined time span.

  The time span of the series to be searched for outliers (default from 1900-01-01 to 2020-12-31) is controlled by the following six variables: outlier.from,outlier.to,outlier.first and outlier.exclLast; where outlier.from and outlier.to have priority over remaining span control variables, outlier.last and outlier.first have priority over outlier.exclFirst and outlier.exclLast, and outlier.last has priority over outlier.first.

outlier.from
: character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). Can be combined with outlier.to.

outlier.to
: character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). Can be combined with outlier.from.

outlier.first
: numeric specifying the number of periods considered at the beginning of the series.

outlier.last
: numeric specifying the number of periods considered at the end of the series.

outlier.exclFirst
: numeric specifying the number of periods excluded at the beginning of the series. Can be combined with outlier.exclLast.

outlier.exclLast
: numeric specifying the number of periods excluded at the end of the series. Can be combined with outlier.exclFirst.

outlier.ao
: logicals. If TRUE the automatic detection of additive outliers is enabled (outlier.enabled must be also set to TRUE).

outlier.tc
: logicals. If TRUE the automatic detection of transitory changes is enabled (outlier.enabled must be also set to TRUE).

outlier.ls
: logicals. If TRUE the automatic detection of level shifts is enabled (outlier.enabled must be also set to TRUE).

outlier.so
: logicals. If TRUE the automatic detection of seasonal outliers is enabled (outlier.enabled must be also set to TRUE).

outlier.usedefcv
: logicals. If TRUE the critical value for the outliers' detection procedure is automatically determined by the number of observations in the outlier detection time span. If FALSE the procedure uses the inputted critical value (outlier.cv).

outlier.cv
: numeric. Inputted critical value for the outliers' detection procedure. The modification of this variable is taken into account only when outlier.usedefcv is set to FALSE.

outlier.method
: determines how the program successively adds detected outliers to the model. At present only the AddOne method is supported.

outlier.tcrate
: numeric. The rate of decay for the transitory change outlier.

automdl.enabled
: logicals. If TRUE the automatic modelling of the ARIMA model is enabled. If FALSE the parameters of the ARIMA model can be specified.

  Control variables for the automatic modelling of the ARIMA model (automdl.enabled is set to TRUE):

automdl.acceptdefault
: logicals. If TRUE the default model (ARIMA(0,1,1)(0,1,1)) may be chosen in the first step of the automatic model identification. If the Ljung-Box Q statistics for the residuals is acceptable, the default model is accepted and no further attempt will be made to identify any other.

automdl.cancel
: numeric, cancelation limit. If the difference in moduli of an AR and an MA roots (when estimating ARIMA(1,0,1)(1,0,1) models in the second step of the automatic identification of the differencing orders) is smaller than cancelation limit, the two roots are assumed equal and cancel out.

automdl.ub1
: numeric, 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.

automdl.ub2
: numeric, 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).

automdl.mixed logicals. The 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 not AR and MA terms in either the seasonal or non-seasonal.

automdl.balanced
logicals. 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 operator is equal to the order of the combined MA operator).

automdl.armalimit
numeric, arma limit. It is the threshold value for t-statistics of ARMA coefficients and constant term used for the final test of model parsimony. If the highest order ARMA coefficient has a t-value less than this value in magnitude, the order of the model is reduced. Also if the constant term has a t-value less than arma limit in magnitude, it is removed from the set of regressors.

automdl.reducecv
numeric, ReduceCV. The percentage by which the outlier's 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)xCV, where CV is the original critical value.

automdl.ljungboxlimit
numeric, Ljung Box limit. 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, the model is rejected, the outlier critical value is reduced, and model and outlier identification (if specified) is redone with a reduced value.

automdl.ubfinal
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 less than the final unit root limit, 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.

Control variables for the non-automatic modelling of the ARIMA model (automdl.enabled is set to FALSE):

arima.mu logicals. If TRUE, the mean is considered as part of the ARIMA model.

arima.p numeric. The order of the non-seasonal autoregressive (AR) polynomial.

arima.d numeric. Regular differencing order.

arima.q numeric. The order of the non-seasonal moving average (MA) polynomial.

arima.bp numeric. The order of the seasonal autoregressive (AR) polynomial.

arima.bd numeric. Seasonal differencing order.

arima.bq numeric. The order of the seasonal moving average (MA) polynomial.

Control variables for the user-defined ARMA coefficients. Coefficients can be defined for the regular and seasonal autoregressive (AR) polynomials and moving average (MA) polynomials. The model considers the coefficients only if the procedure for their estimation (`arima.coefType`) is provided, and the number of provided coefficients matches the sum of (regular and seasonal) AR and MA orders (`p,q,bp,bq`).

`arima.coefEnabled`

logicals. If `TRUE` the program uses the user-defined ARMA coefficients.

`arima.coef` vector providing the coefficients for the regular and seasonal AR and MA polynominals. The length of the vector must equal the sum of the regular and seasonal AR and MA orders. The coefficients shall be provided in the 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 `arima.p=1,arima.q=0,arima.bp=1` and `arima.bq=0`.

`arima.coefType` vector defining ARMA coefficients estimation procedure. Possible procedures are: `"Undefined"` - no use of user-defined input (i.e. coefficients are estimated), `"Fixed"` - fixes the coefficients at the value provided by the user, `"Initial"` - the value defined by the user is used as initial condition. For orders for which the coefficients shall not be defined, the `arima.coef` can be set to `NA` or `0` or the `arima.coefType` can be set to `"Undefined"`. E.g.: `arima.coef = c(-0.8,-0.6,NA)`, `arima.coefType = c("Fixed","Fixed","Undefined")`.

`fcst.horizon` numeric, forecasts horizon. Length of the forecasts generated by the RegARIMA model in periods (positive values) or years (negative values). By default the program generates two years forecasts (`fcst.horizon` set to `-2`).

Details

The available predefined 'JDemetra+' model specifications are described in the table below.

Identifier	Log/level detection	Outliers detection	Calendar effects	ARIMA
RG0	*NA*	*NA*	*NA*	Airline(+mean)
RG1	automatic	AO/LS/TC	*NA*	Airline(+mean)
RG2c	automatic	AO/LS/TC	2 td vars + Easter	Airline(+mean)
RG3	automatic	AO/LS/TC	*NA*	automatic
RG4c	automatic	AO/LS/TC	2 td vars + Easter	automatic
RG5c	automatic	AO/LS/TC	7 td vars + Easter	automatic

Value

A list of class `c("regarima_spec","X13")` with the below components. Each component refers to different part of the RegARIMA model specification, mirroring the arguments of the function (for details see arguments description). Each of the lowest-level component (except span, pre-specified outliers, user-defined variables and pre-specified ARMA coefficients) is structured within a data frame with columns denoting different variables of the model specification and rows referring to: first row - base specification, as provided within the argument `spec`; second row - user modifications as specified by the remaining arguments of the function (e.g.: `arima.d`); and third row - final model specification, values that will be used in the function `regarima`. The final specification (third row) shall include user modifications (row two) unless they were wrongly specified. The

pre-specified outliers, user-defined variables and pre-specified ARMA coefficients consist of a list with the Predefined (base model specification) and Final values.

estimate — data frame. Variables referring to: span - time span for the model estimation, tolerance - argument estimate.tol. The final values can be also accessed with the function s_estimate.

transform — data frame. Variables referring to: tfunction - argument transform.function, adjust - argument transform.adjust, aicdiff - argument transform.aicdiff. The final values can be also accessed with the function s_transform.

regression — list including the information on the user-defined variables (userdef), trading.days effect and easter effect. The user-defined part includes: specification - data frame with the information if pre-specified outliers (outlier) and user-defined variables (variables) are included in the model and if fixed coefficients are used (outlier.coef and variables.coef). The final values can be also accessed with the function s_usrdef; outliers - matrixes with the outliers (Predefined and Final). The final outliers can be also accessed with the function s_preOut; and variables - list with the Predefined and Final user-defined variables (series) and its description (description) including the information on the variable type and values of fixed coefficients. The final user-defined variables can be also accessed with the function s_preVar. Within the data frame trading.days variables refer to: option - argument tradingdays.option,autoadjust - argument tradingdays.autoadjust,leapyear - argument tradingdays.leapyear,stocktd - argument tradingdays.stocktd,test - argument tradingdays.test. The final trading.days values can be also accessed with the function s_td. Within the data frame easter variables refer to: enabled - argument easter.enabled,julian - argument easter.julian,duration - argument easter.duration,test - argument easter.test. The final easter values can be also accessed with the function s_easter.

outliers — data frame. Variables referring to: enabled - argument outlier.enabled, span - time span for the outliers' detection, ao - argument outlier.ao,tc - argument outlier.tc,ls - argument outlier.ls,so - argument outlier.so,usedefcv - argument outlier.usedefcv,cv - argument outlier.cv,method - argument outlier.method,tcrate - argument outlier.tcrate. The final values can be also accessed with the function s_out.

arima — list containing a data frame with the ARIMA settings (specification) and matrixes with the information on the pre-specified ARMA coefficients (coefficients). The matrix Predefined refers to the pre-defined model specification and matrix Final to the final specification. Both matrixes contain the value of the ARMA coefficients and the procedure for its estimation. Within the data frame specification the variable enabled refer to the argument automdl.enabled and all the remaining variables (automdl.acceptdefault,automdl.cancel,automdl.ub1,automdl.ub
to the respective function arguments. The final values of the specification can be also accessed with the function s_arima and final pre-specified ARMA coefficients with the function s_arimaCoef.

forecast — data frame with the forecast horizon (argument fcst.horizon). The final value can be also accessed with the function s_fcst.

span matrix containing the final time span for the model estimation and outliers' detection. Contains the same information as the variable span in the data frames estimate and outliers. The matrix can be also accessed with the function s_span.

References

Info on 'JDemetra+', usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en

Examples

```
myseries <- ipi_c_eu[, "FR"]
myspec1 <- regarima_spec_x13(spec = "RG5c")
myreg1 <- regarima(myseries, spec = myspec1)

 # Modify a pre-specified model specification
myspec2 <- regarima_spec_x13(spec = "RG5c",
                             tradingdays.option = "WorkingDays")
myreg2 <- regarima(myseries, spec = myspec2)

 # Modify the model specification from a "regarima" object
myspec3 <- regarima_spec_x13(myreg1, tradingdays.option = "WorkingDays")
myreg3 <- regarima(myseries, myspec3)

 # Modify the model specification from a "regarima_spec" object
myspec4 <- regarima_spec_x13(myspec1, tradingdays.option = "WorkingDays")
myreg4 <- regarima(myseries, myspec4)

 # Pre-specified outliers
myspec1 <- regarima_spec_x13(spec = "RG5c", usrdef.outliersEnabled = TRUE,
              usrdef.outliersType = c("LS", "AO"),
              usrdef.outliersDate = c("2008-10-01", "2002-01-01"),
              usrdef.outliersCoef = c(36, 14),
              transform.function = "None")

myreg1 <- regarima(myseries, myspec1)
myreg1
s_preOut(myreg1)


 # User-defined variables
var1 <- ts(rnorm(length(myseries))*10, start = start(myseries),
           frequency = 12)
var2 <- ts(rnorm(length(myseries))*100, start = start(myseries),
           frequency = 12)
var <- ts.union(var1, var2)

myspec1 <- regarima_spec_x13(spec = "RG5c", usrdef.varEnabled = TRUE,
                             usrdef.var = var)
myreg1 <- regarima(myseries, myspec1)
myreg1
```

```
myspec2 <- regarima_spec_x13(spec = "RG5c", usrdef.varEnabled = TRUE,
                             usrdef.var = var1, usrdef.varCoef = 2,
                             transform.function = "None")
myreg2 <- regarima(myseries, myspec2)
s_preVar(myreg2)

 # Pre-specified ARMA coefficients
myspec1 <- regarima_spec_x13(spec = "RG5c", automdl.enabled =FALSE,
             arima.p = 1, arima.q = 1, arima.bp = 0, arima.bq = 1,
             arima.coefEnabled = TRUE, arima.coef = c(-0.8, -0.6, 0),
             arima.coefType = c(rep("Fixed", 2), "Undefined"))

s_arimaCoef(myspec1)
myreg1 <- regarima(myseries, myspec1)
myreg1
```

---

save_spec *Saving and loading a model specification, SA and pre-adjustment in X13 and TRAMO-SEATS*

Description

save_spec saves a SA or RegARIMA model specification. load_spec loads the previously saved model specification.

Usage

```
save_spec(object, file = file.path(tempdir(), "spec.RData"))

load_spec(file = "spec.RData")
```

Arguments

object	object of one of the classes: c("SA_spec","X13"), c("SA_spec","TRAMO_SEATS"), c("SA","X13"), c("SA","TRAMO_SEATS"), c("regarima_spec","X13"), c("regarima_spec","TRAM c("regarima","X13"), c("regarima","TRAMO_SEATS").
file	(path and) name of the file where the model specification will be saved (have been saved).

Details

save_spec saves the final model specification of a "SA_spec", "SA", "regarima_spec" or "regarima" class object. load_spec loads the previously saved model specification. It creates a c("SA_spec","X13"), c("sA_spec","TRAMO_SEATS"), c("regarima_spec","X13") or c("regarima_spec","TRAMO_SEATS") class object, in line with the class of the previously saved model specification.

Value

load_spec returns an object of class "SA_spec" or "regarima_spec".

References

Info on JDemtra+, usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en

Examples

```
myseries <- ipi_c_eu[, "FR"]
myreg1 <- regarima_x13(myseries, spec = "RG5c")
myspec2 <- regarima_spec_x13(myreg1, estimate.from = "2005-10-01", outlier.from = "2010-03-01")
myreg2 <- regarima(myseries, myspec2)

myreg3 <- regarima_tramoseats(myseries, spec = "TRfull")
myspec4 <-regarima_spec_tramoseats(myreg3, tradingdays.mauto = "Unused",
                                   tradingdays.option ="WorkingDays",
                                   easter.type = "Standard",
                                   automdl.enabled = FALSE, arima.mu = TRUE)
myreg4 <-regarima(myseries, myspec4)

myspec6 <- x13_spec("RSA5c")
mysa6 <- x13(myseries, myspec6)

myspec7 <- tramoseats_spec("RSAfull")
mysa7 <- tramoseats(myseries, myspec7)

dir <- tempdir()

 # Save the model specification from a c("regarima_spec","X13") class object
save_spec(myspec2, file.path(dir, "specx13.RData"))
 # Save the model specification from a c("regarima","X13") class object
save_spec(myreg2, file.path(dir,"regx13.RData"))
 # Save the model specification from a c("regarima_spec","TRAMO_SEATS") class object
save_spec(myspec4, file.path(dir,"specTS.RData"))
 # Save the model specification from a c("regarima","TRAMO_SEATS") class object
save_spec(myreg4, file.path(dir,"regTS.RData"))
 # Save model from a c("SA_spec","X13") class object
save_spec(myspec6, file.path(dir,"specFullx13.RData"))
 # Save model from a c("SA","X13") class object
save_spec(mysa6, file.path(dir,"sax13.RData"))
 # Save model from a c("SA_spec","TRAMO_SEATS") class object
save_spec(myspec7, file.path(dir,"specFullTS.RData"))
 # Save model from a c("SA","TRAMO_SEATS") class object
save_spec(mysa7, file.path(dir,"saTS.RData"))

 # Load the model specification
myspec2a <- load_spec(file.path(dir,"specx13.RData"))
myspec2b <- load_spec(file.path(dir,"regx13.RData"))
myspec4a <- load_spec(file.path(dir,"specTS.RData"))
```

```
myspec4b <- load_spec(file.path(dir,"regTS.RData"))
myspec6a <- load_spec(file.path(dir,"specFullx13.RData"))
myspec6b <- load_spec(file.path(dir,"sax13.RData"))
myspec7a <- load_spec(file.path(dir,"specFullTS.RData"))
myspec7b <- load_spec(file.path(dir,"saTS.RData"))


regarima(myseries, myspec2a)
x13(myseries, myspec6a)
tramoseats(myseries, myspec7a)

regarima(myseries, myspec4a)
x13(myseries, myspec6b)
tramoseats(myseries, myspec7b)
```

---

save_workspace *Save a workspace*

Description

Functions save a workspace object into a 'JDemetra+" workspace.

Usage

```
save_workspace(workspace, file)
```

Arguments

workspace	a workspace object to export
file	the path to the export 'JDemetra+' workspace (.xml file). By default a dialog box opens.

Value

A boolean indicating whether the export has suceed.

See Also

load_workspace

Examples

```
dir <- tempdir()
# Create and export a empty 'JDemetra+' workspace
wk <- new_workspace()
new_multiprocessing(wk, "sa1")
save_workspace(wk, file.path(dir, "workspace.xml"))
```

`specification` *Access model specification, SA and pre-adjustment in X13 and TRAMO-SEATS*

Description

Below functions access different parts of the final model specification as included in the `"SA"`,`"regarima"`,`"SA_spec"` and `"regarima_spec"` S3 class objects.

Usage

```
s_estimate(object = NA)

s_transform(object = NA)

s_usrdef(object = NA)

s_preOut(object = NA)

s_preVar(object = NA)

s_td(object = NA)

s_easter(object = NA)

s_out(object = NA)

s_arima(object = NA)

s_arimaCoef(object = NA)

s_fcst(object = NA)

s_span(object = NA)

s_x11(object = NA)

s_seats(object = NA)
```

Arguments

`object`	object of one of the classes: `c("SA","X13")`, `c("SA","TRAMO_SEATS")`, `c("SA_spec","X13")`, `c("SA_spec","TRAMO_SEATS")`, `c("regarima","X13")`, `c("regarima","TRAMO_SEATS")`, `c("regarima_spec","X13")`, `c("regarima_spec","TRAMO_SEATS")`.

Value

- `s_estimate` returns a data.frame with the *estimate* variables

- `s_transform` returns a data.frame with the *transform* varaibles
- `s_usrdef` returns a data.frame with the *user-defined regressors* (outliers and variables) model specification, indicating if those variables are included in the model and if coefficients are pre-specified
- `s_preOut` returns a data.frame with the *pre-specified outliers*
- `s_preVar` returns a list with the information on the user-defined variables, including: `series` - the time series and `description` - data.frame with the variable type and coefficients
- `s_td` returns a data.frame with the *trading.days* variables
- `s_easter` returns a data.frame with the *easter* variables
- `s_out` returns a data.frame with the *outliers* detection variables
- `s_arima` returns a data.frame with the *arima* variables
- `s_arimaCoef` returns a data.frame with the user-specified ARMA coefficients
- `s_fcst` returns a data.frame with the forecast horizon
- `s_span` returns a data.frame with the *span* variables
- `s_x11` returns a data.frame with the *x11* variables
- `s_seats` returns a data.frame with the *seats* variables

References

Info on 'JDemetra+', usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en

Examples

```
myseries <- ipi_c_eu[, "FR"]
myreg1 <- regarima_x13(myseries, spec = "RG5c")
myspec1 <- regarima_spec_x13(myreg1,
             estimate.from = "2005-10-01",
             outlier.from = "2010-03-01")

s_estimate(myreg1)
s_estimate(myspec1)

s_transform(myreg1)
s_transform(myspec1)

s_usrdef(myreg1)
s_usrdef(myspec1)

myspec2 <- regarima_spec_x13(myreg1, usrdef.outliersEnabled = TRUE,
             usrdef.outliersType = c("LS", "AO"),
             usrdef.outliersDate = c("2009-10-01", "2005-02-01"))
myreg2 <- regarima(myseries, myspec2)

s_preOut(myreg2)
s_preOut(myspec2)
```

```
var1 <- ts(rnorm(length(myseries))*10, start = start(myseries), frequency = 12)
var2 <- ts(rnorm(length(myseries))*100, start = start(myseries), frequency = 12)
var3 <- ts.union(var1, var2)
myspec3 <- regarima_spec_x13(spec = "RG5c",
                             usrdef.varEnabled = TRUE,
                             usrdef.var = var3)
myreg3 <- regarima(myseries, myspec3)

s_preVar(myspec3)
s_preVar(myreg3)

s_td(myreg1)
s_td(myspec1)

s_easter(myreg1)
s_easter(myspec1)

s_out(myreg1)
s_out(myspec1)

s_arima(myreg1)
s_arima(myspec1)

myspec4 <- regarima_spec_x13(myreg1, automdl.enabled = FALSE,
             arima.coefEnabled = TRUE,
             arima.p = 1,arima.q = 1, arima.bp = 1, arima.bq = 1,
             arima.coef = rep(0.2, 4),
             arima.coefType = rep("Initial", 4))
myreg4 <- regarima(myseries, myspec4)

s_arimaCoef(myreg4)
s_arimaCoef(myspec4)

s_fcst(myreg1)
s_fcst(myspec1)

s_span(myreg1)
s_span(myspec1)

myspec5 <- x13_spec(spec = "RSA5c", x11.seasonalComp = FALSE)
mysa5 <- x13(myseries, myspec5)

s_x11(mysa5)
s_x11(myspec5)

myspec6 <- tramoseats_spec(spec = "RSAfull", seats.approx = "Noisy")
mysa6 <- tramoseats(myseries, myspec6)

s_seats(mysa6)
s_seats(mysa6)
```

`tramoseats` *Seasonal Adjustment with TRAMO-SEATS*

Description

Function to estimate the seasonally adjusted series (sa) with the TRAMO-SEATS method. This is achieved by decomposing the time series (y) into the: trend-cycle (t), seasonal component (s) and irregular component (i). The final seasonally adjusted series shall be free of seasonal and calendar-related movements. `tramoseats` returns a preformatted result while `jtramoseats` returns the Java objects of the seasonal adjustment.

Usage

```
jtramoseats(
  series,
  spec = c("RSAfull", "RSA0", "RSA1", "RSA2", "RSA3", "RSA4", "RSA5"),
  userdefined = NULL
)

tramoseats(
  series,
  spec = c("RSAfull", "RSA0", "RSA1", "RSA2", "RSA3", "RSA4", "RSA5"),
  userdefined = NULL
)
```

Arguments

`series`	a univariate time series
`spec`	model specification TRAMO-SEATS. It can be a `character` of the predefined TRAMO-SEATS 'JDemetra+' model specification (see *Details*), or an object of class `c("SA_spec","TRAMO_SEATS")`. The default is `"RSAfull"`.
`userdefined`	vector with characters for additional output variables (see `user_defined_variables`).

Details

The first step of the seasonal adjustment consist of pre-adjusting the time series by removing from it the deterministic effects by means of a regression model with ARIMA noise (RegARIMA, see: `regarima`). In the second part, the pre-adjusted series is decomposed into the following components: trend-cycle (t), seasonal component (s) and irregular component (i). The decomposition can be: additive ($y = t + s + i$) or multiplicative ($y = t * s * i$). The final seasonally adjusted series (sa) shall be free of seasonal and calendar-related movements.

In the TRAMO-SEATS method, the second step - SEATS ("Signal Extraction in ARIMA Time Series") - performs an ARIMA-based decomposition of an observed time series into unobserved components. More information on the method can be found on the Bank of Spian website (`https://www.bde.es`).

As regards the available predefined 'JDemetra+' TRAMO-SEATS model specifications, they are described in the table below.

Identifier	Log/level detection	Outliers detection	Calendar effects	ARIMA
RSA0	*NA*	*NA*	*NA*	Airline(+mean)
RSA1	automatic	AO/LS/TC	*NA*	Airline(+mean)
RSA2	automatic	AO/LS/TC	2 td vars + Easter	Airline(+mean)
RSA3	automatic	AO/LS/TC	*NA*	automatic
RSA4	automatic	AO/LS/TC	2 td vars + Easter	automatic
RSA5	automatic	AO/LS/TC	7 td vars + Easter	automatic
RSAfull	automatic	AO/LS/TC	automatic	automatic

Value

`jtramoseats` returns a `jSA` object. It contains the Java objects of the result of the seasonal adjustment without any formatting. Therefore the computation is faster than with `tramoseats`. The results can the seasonal adjustment can be extract by `get_indicators`.

`tramoseats` returns an object of class `c("SA","TRAMO_SEATS")`, a list containing the following components:

`regarima` object of class `c("regarima","TRAMO_SEATS")`. See *Value* of the function `regarima`.

`decomposition` object of class `"decomposition_SEATS"`, five elements list:

- `specification` list with the SEATS algorithm specification. See also function `tramoseats_spec`
- `mode` decomposition mode
- `model` list with the SEATS models: `model,sa,trend,seasonal,transitory,irregular`. Each of them is a matrix with the estimated coefficients.
- `linearized` time series matrix (mts) with the stochastic series decomposition (input series `y_lin`, seasonally adjusted `sa_lin`, trend `t_lin`, seasonal `s_lin`, irregular `i_lin`)
- `components` time series matrix (mts) with the decomposition components (input series `y_cmp`, seasonally adjusted `sa_cmp`, trend `t_cmp`, seasonal `s_cmp`, irregular `i_cmp`)

`final` object of class `c("final","mts","ts","matrix")`. Matrix with the final results of the seasonal adjustment. It includes time series: original time series (`y`), forecast of the original series (`y_f`), trend (`t`), forecast of the trend (`t_f`), seasonally adjusted series (`sa`), forecast of the seasonally adjusted series (`sa_f`), seasonal component (`s`), forecast of the seasonal component (`s_f`), irregular component (`i`) and the forecast of the irregular component (`i_f`).

`diagnostics` object of class `"diagnostics"`, list with three type of diagnostics tests:

- `variance_decomposition` data.frame with the tests on the relative contribution of the components to the stationary portion of the variance in the original series, after the removal of the long term trend.
- `residuals_test` data.frame with the tests on the presence of seasonality in the residuals (includes the statistic, p-value and parameters description)
- `combined_test` combined tests for stable seasonality in the entire series. Two elements list with: `tests_for_stable_seasonality` - data.frame with the tests (includes the statistic, p-value and parameters description) and `combined_seasonality_test` - the summary.

`user_defined` object of class "`user_defined`". List containing the userdefined additional variables defined in the `userdefined` argument.

References

Info on 'JDemetra+', usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en

BOX G.E.P. and JENKINS G.M. (1970), "Time Series Analysis: Forecasting and Control", Holden-Day, San Francisco.

BOX G.E.P., JENKINS G.M., REINSEL G.C. and LJUNG G.M. (2015), "Time Series Analysis: Forecasting and Control", John Wiley & Sons, Hoboken, N. J., 5th edition.

See Also

`tramoseats_spec`, `x13`

Examples

```
myseries <- ipi_c_eu[, "FR"]
myspec <- tramoseats_spec("RSAfull")
mysa <- tramoseats(myseries, myspec)
mysa

# Equivalent to:
mysa1 <- tramoseats(myseries, spec = "RSAfull")
mysa1

var1 <- ts(rnorm(length(myseries))*10, start = start(myseries), frequency = 12)
var2 <- ts(rnorm(length(myseries))*100, start = start(myseries), frequency = 12)
var <- ts.union(var1, var2)
myspec2 <- tramoseats_spec(myspec, tradingdays.mauto = "Unused",
                           tradingdays.option = "WorkingDays",
                           easter.type = "Standard",
                           automdl.enabled = FALSE, arima.mu = TRUE,
                           usrdef.varEnabled = TRUE, usrdef.var = var)
s_preVar(myspec2)
mysa2 <- tramoseats(myseries, myspec2,
                    userdefined = c("decomposition.sa_lin_f",
                                    "decomposition.sa_lin_e"))
mysa2
plot(mysa2)
plot(mysa2$regarima)
plot(mysa2$decomposition)
```

tramoseats_spec *TRAMO-SEATS model specification, SA/TRAMO-SEATS*

Description

Function to create (and/or modify) a c("SA_spec","TRAMO_SEATS") class object with the SA model specification for the TRAMO-SEATS method. It can be done from a pre-defined 'JDemetra+' model specification (a character), a previous specification (c("SA_spec","TRAMO_SEATS") object) or a seasonal adjustment model (c("SA","TRAMO_SEATS") object).

Usage

```
tramoseats_spec(
  spec = c("RSAfull", "RSA0", "RSA1", "RSA2", "RSA3", "RSA4", "RSA5"),
  preliminary.check = NA,
  estimate.from = NA_character_,
  estimate.to = NA_character_,
  estimate.first = NA_integer_,
  estimate.last = NA_integer_,
  estimate.exclFirst = NA_integer_,
  estimate.exclLast = NA_integer_,
  estimate.tol = NA_integer_,
  estimate.eml = NA,
  estimate.urfinal = NA_integer_,
  transform.function = c(NA, "Auto", "None", "Log"),
  transform.fct = NA_integer_,
  usrdef.outliersEnabled = NA,
  usrdef.outliersType = NA,
  usrdef.outliersDate = NA,
  usrdef.outliersCoef = NA,
  usrdef.varEnabled = NA,
  usrdef.var = NA,
  usrdef.varType = NA,
  usrdef.varCoef = NA,
  tradingdays.mauto = c(NA, "Unused", "FTest", "WaldTest"),
  tradingdays.pftd = NA_integer_,
 tradingdays.option = c(NA, "TradingDays", "WorkingDays", "UserDefined", "None"),
  tradingdays.leapyear = NA,
  tradingdays.stocktd = NA_integer_,
  tradingdays.test = c(NA, "Separate_T", "Joint_F", "None"),
 easter.type = c(NA, "Unused", "Standard", "IncludeEaster", "IncludeEasterMonday"),
  easter.julian = NA,
  easter.duration = NA_integer_,
  easter.test = NA,
  outlier.enabled = NA,
  outlier.from = NA_character_,
  outlier.to = NA_character_,
```

```
  outlier.first = NA_integer_,
  outlier.last = NA_integer_,
  outlier.exclFirst = NA_integer_,
  outlier.exclLast = NA_integer_,
  outlier.ao = NA,
  outlier.tc = NA,
  outlier.ls = NA,
  outlier.so = NA,
  outlier.usedefcv = NA,
  outlier.cv = NA_integer_,
  outlier.eml = NA,
  outlier.tcrate = NA_integer_,
  automdl.enabled = NA,
  automdl.acceptdefault = NA,
  automdl.cancel = NA_integer_,
  automdl.ub1 = NA_integer_,
  automdl.ub2 = NA_integer_,
  automdl.armalimit = NA_integer_,
  automdl.reducecv = NA_integer_,
  automdl.ljungboxlimit = NA_integer_,
  automdl.compare = NA,
  arima.mu = NA,
  arima.p = NA_integer_,
  arima.d = NA_integer_,
  arima.q = NA_integer_,
  arima.bp = NA_integer_,
  arima.bd = NA_integer_,
  arima.bq = NA_integer_,
  arima.coefEnabled = NA,
  arima.coef = NA,
  arima.coefType = NA,
  fcst.horizon = NA_integer_,
  seats.predictionLength = NA_integer_,
  seats.approx = c(NA, "None", "Legacy", "Noisy"),
  seats.trendBoundary = NA_integer_,
  seats.seasdBoundary = NA_integer_,
  seats.seasdBoundary1 = NA_integer_,
  seats.seasTol = NA_integer_,
  seats.maBoundary = NA_integer_,
  seats.method = c(NA, "Burman", "KalmanSmoother", "McElroyMatrix")
)
```

Arguments

spec	model specification X13. It can be a `character` of predefined TRAMO-SEATS 'JDemetra+' model specification (see *Details*), an object of class `c("SA_spec","TRAMO_SEATS")` or an object of class `c("SA","TRAMO_SEATS")`. The default is `"RSAfull"`.
preliminary.check	boolean to check the quality of the input series and exclude highly problematic

ones: e.g. these with a number of identical observations and/or missing values above pre-specified threshold values.

The time span of the series to be used for the estimation of the RegArima model coefficients (default from 1900-01-01 to 2020-12-31) is controlled by the following six variables: estimate.from,estimate.to,estimate.first,estimate.last,estimate.exc and estimate.exclLast; where estimate.from and estimate.to have priority over remaining span control variables, estimate.last and estimate.first have priority over estimate.exclFirst and estimate.exclLast, and estimate.last has priority over estimate.first.

estimate.from character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). Can be combined with estimate.to.

estimate.to character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). Can be combined with estimate.from.

estimate.first numeric specifying the number of periods considered at the beginning of the series.

estimate.last numeric specifying the number of periods considered at the end of the series.

estimate.exclFirst
numeric specifying the number of periods excluded at the beginning of the series. Can be combined with estimate.exclLast.

estimate.exclLast
numeric specifying the number of periods excluded at the end of the series. Can be combined with estimate.exclFirst.

estimate.tol 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.

estimate.eml logicals, exact maximum likelihood estimation. If TRUE the program performs an exact maximum likelihood estimation. If FASLE the Unconditional Least Squares method is used.

estimate.urfinal
numeric, 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 less than this number, 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.

transform.function
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.

transform.fct numeric controlling the bias in the log/level pre-test: transform.fct > 1 favours levels, transform.fct< 1 favours logs. Considered only when transform.function is set to "Auto".

Control variables for the pre-specified outliers. The pre-specified outliers are used in the model only if they are enabled (usrdef.outliersEnabled=TRUE) and the outliers' type (usrdef.outliersType) and date (usrdef.outliersDate) are provided.

usrdef.outliersEnabled
: logicals. If TRUE the program uses the pre-specified outliers.

usrdef.outliersType
: vector defining the outliers' type. Possible types are: ("AO") - additive, ("LS") - level shift, ("TC") - transitory change, ("SO") - seasonal outlier. E.g.: usrdef.outliersType =c("AO","AO","LS").

usrdef.outliersDate
: vector defining the outliers' date. The dates should be characters in format "YYYY-MM-DD". E.g.: usrdef.outliersDate=c("2009-10-01","2005-02-01","2003-04-01").

usrdef.outliersCoef
: vector providing fixed coefficients for the outliers. The coefficients can't be fixed if transform.function is set to "Auto" - the series transformation need to be pre-defined. E.g.: usrdef.outliersCoef=c(200,170,20).

Control variables for the user-defined variables:

usrdef.varEnabled
: logicals. If TRUE the program uses the user-defined variables.

usrdef.var
: time series (ts) or matrix of time series (mts) with the user-defined variables.

usrdef.varType
: vector of character(s) defining the user-defined variables component type. Possible types are: "Undefined","Series","Trend","Seasonal","SeasonallyAdjusted","Irregular", The type "Calendar" has to be used with tradingdays.option = "UserDefined" to use user-defined calendar regressors. If not specified, the program will assign the "Undefined" type.

usrdef.varCoef
: vector providing fixed coefficients for the user-defined variables. The coefficients can't be fixed if transform.function is set to "Auto" - the series transformation need to be pre-defined.

tradingdays.mauto
: defines whether the calendar effects should be added to the model manually ("Unused") or automatically. In the automatic selection, the choice of the number of calendar variables can be based on the F-Test ("FTest") or the Wald Test ("WaldTest"); the model with higher F value is chosen, provided that it is higher than tradingdays.pftd).

tradingdays.pftd
: numeric. P-value applied in the test specified by the automatic parameter (tradingdays.mauto) to assess the significance of the pre-tested calendar effects variables and whether they should be included in the RegArima model.

Control variables for the manual selection of calendar effects variables (tradingdays.mauto is set to "Unused"):

tradingdays.option
: defines the type of the trading days regression variables: "TradingDays" - six day-of-the-week regression variables; "WorkingDays" - one working/non-working day contrast variable; "None" - no correction for trading days and working days effects; "UserDefined" - user-defined trading days regressors (regressors have to be defined by the usrdef.var argument with usrdef.varType set to "Calendar" and usrdef.varEnabled = TRUE). "None" has also to be chosen for the "day-of-week effects" correction (tradingdays.stocktd to be modified accordingly).

`tradingdays.leapyear`
: logicals. Specifies if the leap-year correction should be included. If `TRUE` the model includes the leap-year effect.

`tradingdays.stocktd`
: 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). Modifications of this variable are taken into account only when `tradingdays.option` is set to `"None"`.

`tradingdays.test`
: defines the pre-tests of the trading day effects: `"None"` - calendar variables are used in the model without pre-testing; `"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.

`easter.type`
: specifies the presence and the length of the Easter effect: `"Unused"` - 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.

`easter.julian`
: logicals. If `TRUE` the program uses the Julian Easter (expressed in Gregorian calendar).

`easter.duration`
: numeric indicating the duration of the Easter effect (length in days, between 1 and 15).

`easter.test`
: logicals. If `TRUE` the program performs a t-test for the significance of the Easter effect. The Easter effect is considered as significant if the modulus of t-statistic is greater than 1.96.

`outlier.enabled`
: logicals. If `TRUE` the automatic detection of outliers is enabled in the defined time span.

  The time span of the series to be searched for outliers (default from 1900-01-01 to 2020-12-31) is controlled by the following six variables: `outlier.from`,`outlier.to`,`outlier.first` and `outlier.exclLast`; where `outlier.from` and `outlier.to` have priority over remaining span control variables, `outlier.last` and `outlier.first` have priority over `outlier.exclFirst` and `outlier.exclLast`, and `outlier.last` has priority over `outlier.first`.

`outlier.from`
: character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). Can be combined with `outlier.to`.

`outlier.to`
: character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). Can be combined with `outlier.from`.

`outlier.first`
: numeric specifying the number of periods considered at the beginning of the series.

`outlier.last`
: numeric specifying the number of periods considered at the end of the series.

`outlier.exclFirst`
: numeric specifying the number of periods excluded at the beginning of the series. Can be combined with `outlier.exclLast`.

`outlier.exclLast`
: numeric specifying the number of periods excluded at the end of the series. Can be combined with `outlier.exclFirst`.

`outlier.ao`
: logicals. If `TRUE` the automatic detection of additive outliers is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.tc`
: logicals. If `TRUE` the automatic detection of transitory changes is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.ls`
: logicals. If `TRUE` the automatic detection of level shifts is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.so`
: logicals. If `TRUE` the automatic detection of seasonal outliers is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.usedefcv`
: logicals. If `TRUE` the critical value for the outliers' detection procedure is automatically determined by the number of observations in the outlier detection time span. If `FALSE` the procedure uses the inputted critical value (`outlier.cv`).

`outlier.cv`
: numeric. Inputted critical value for the outliers' detection procedure. The modification of this variable is taken in to account only when `outlier.usedefcv` is set to `FALSE`.

`outlier.eml`
: logicals, exact likelihood estimation method.Controls the method applied for a parameter estimation in the intermediate steps of the automatic detection and correction of outliers. If `TRUE` an exact likelihood estimation method is used, when `FALSE` the fast Hannan-Rissanen method is used.

`outlier.tcrate`
: numeric. The rate of decay for the transitory change outlier.

`automdl.enabled`
: logicals. If `TRUE` the automatic modelling of the ARIMA model is enabled. If `FALSE` the parameters of the ARIMA model can be specified.

  Control variables for the automatic modelling of the ARIMA model (`automdl.enabled` is set to `TRUE`):

`automdl.acceptdefault`
: logicals. If `TRUE` the default model (ARIMA(0,1,1)(0,1,1)) may be chosen in the first step of the automatic model identification. If the Ljung-Box Q statistics for the residuals is acceptable, the default model is accepted and no further attempt will be made to identify any other.

`automdl.cancel`
: numeric, cancelation limit. If the difference in moduli of an AR and an MA roots (when estimating ARIMA(1,0,1)(1,0,1) models in the second step of the automatic identification of the differencing orders) is smaller than cancelation limit, the two roots are assumed equal and cancel out.

`automdl.ub1`
: numeric, 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.

`automdl.ub2` numeric, 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).

`automdl.armalimit` numeric, arma limit. It is the threshold value for t-statistics of ARMA coefficients and constant term used for the final test of model parsimony. If the highest order ARMA coefficient has a t-value less than this value in magnitude, the order of the model is reduced. Also if the constant term has a t-value less than arma limit in magnitude, it is removed from the set of regressors.

`automdl.reducecv` numeric, ReduceCV. The percentage by which the outlier's 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)xCV, where CV is the original critical value.

`automdl.ljungboxlimit` numeric, Ljung Box limit. 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, the model is rejected, the outlier critical value is reduced, and model and outlier identification (if specified) is redone with a reduced value.

`automdl.compare` logicals. 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.

Control variables for the non-automatic modelling of the ARIMA model (`automdl.enabled` is set to `FALSE`):

`arima.mu` logicals. If `TRUE`, the mean is considered as part of the ARIMA model.

`arima.p` numeric. The order of the non-seasonal autoregressive (AR) polynomial.

`arima.d` numeric. Regular differencing order.

`arima.q` numeric. The order of the non-seasonal moving average (MA) polynomial.

`arima.bp` numeric. The order of the seasonal autoregressive (AR) polynomial.

`arima.bd` numeric. Seasonal differencing order.

`arima.bq` numeric. The order of the seasonal moving average (MA) polynomial.

Control variables for the user-defined ARMA coefficients. Coefficients can be defined for the regular and seasonal autoregressive (AR) polynomials and moving average (MA) polynomials. The model considers the coefficients only if the procedure for their estimation (`arima.coefType`) is provided, and the number of provided coefficients matches the sum of (regular and seasonal) AR and MA orders (`p,q,bp,bq`).

`arima.coefEnabled`
: logicals. If `TRUE` the program uses the user-defined ARMA coefficients.

`arima.coef`
: vector providing the coefficients for the regular and seasonal AR and MA polynominals. The length of the vector must equal the sum of the regular and seasonal AR and MA orders. The coefficients shall be provided in the 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 `arima.p=1`,`arima.q=0`,`arima.bp=1` and `arima.bq=0`.

`arima.coefType`
: vector defining ARMA coefficients estimation procedure. Possible procedures are: `"Undefined"` - no use of user-defined input (i.e. coefficients are estimated), `"Fixed"` - fixes the coefficients at the value provided by the user, `"Initial"` - the value defined by the user is used as initial condition. For orders for which the coefficients shall not be defined, the `arima.coef` can be set to `NA` or `0` or the `arima.coefType` can be set to `"Undefined"`. E.g.: `arima.coef = c(-0.8,-0.6,NA)`, `arima.coefType = c("Fixed","Fixed","Undefined")`.

`fcst.horizon`
: numeric, forecasts horizon. Length of the forecasts generated by the RegARIMA model in periods (positive values) or years (negative values). By default the program generates two years forecasts (`fcst.horizon` set to `-2`).

`seats.predictionLength`
: integer, number of forecasts used in the decomposition. Negative values correspond to numbers of years.

`seats.approx`
: character, approximation mode. When the ARIMA model estimated by TRAMO does not accept an admissible decomposition, SEATS: `"None"` - performs an approximation; `"Legacy"` - replaces the model with a decomposable one; `"Noisy"` - estimates a new model by adding a white noise to the non-admissible model estimated by TRAMO.

`seats.trendBoundary`
: numeric, trend boundary. The boundary from which an AR root is integrated in the trend component. If the modulus of the inverse real root is greater than Trend boundary, the AR root is integrated in the trend component. Below this value the root is integrated in the transitory component.

`seats.seasdBoundary`
: numeric, seasonal boundary. Boundary from which a negative AR root is integrated in the seasonal component.

`seats.seasdBoundary1`
: numeric, seasonal boundary (unique). Boundary from which a negative AR root is integrated in the seasonal component when the root is the unique seasonal root.

`seats.seasTol`
: numeric, seasonal tolerance. The tolerance (measured in degrees) to allocate the AR non-real roots to the seasonal component (if the modulus of the inverse complex AR root is greater than Trend boundary and the frequency of this root differs from one of the seasonal frequencies by less than Seasonal tolerance) or the transitory component (otherwise).

`seats.maBoundary`
: numeric, MA unit root boundary. When the modulus of an estimated MA root falls in the range (xl, 1), it is set to xl.

`seats.method` character, estimation method of the unobserved components. The choice can be made from: `"Burman"` (default, may result in a significant underestimation of the standard deviations of the components as it may become numerically unstable when some roots of the MA polynomial are near 1); `"KalmanSmoother"` (it is not disturbed by the (quasi-) unit roots in MA); `"McElroyMatrix"` (has the same stability issues as the Burman's algorithm).

Details

The available predefined 'JDemetra+' model specifications are described in the table below.

Identifier	Log/level detection	Outliers detection	Calendar effects	ARIMA
RSA0	*NA*	*NA*	*NA*	Airline(+mean)
RSA1	automatic	AO/LS/TC	*NA*	Airline(+mean)
RSA2	automatic	AO/LS/TC	2 td vars + Easter	Airline(+mean)
RSA3	automatic	AO/LS/TC	*NA*	automatic
RSA4	automatic	AO/LS/TC	2 td vars + Easter	automatic
RSA5	automatic	AO/LS/TC	7 td vars + Easter	automatic
RSAfull	automatic	AO/LS/TC	automatic	automatic

Value

A two-elements list of class `c("SA_spec","TRAMO_SEATS")`: (1) object of class `c("regarima_spec","TRAMO_SEATS")` with the RegARIMA model specification, (2) object of class `c("seats_spec","data.frame")` with the SEATS algorithm specification. Each component refers to different part of the SA model specification, mirroring the arguments of the function (for details see arguments description). Each of the lowest-level component (except span, pre-specified outliers, user-defined variables and pre-specified ARMA coefficients) is structured within a data frame with columns denoting different variables of the model specification and rows referring to: first row - base specification, as provided within the argument `spec`; second row - user modifications as specified by the remaining arguments of the function (e.g.: `arima.d`); and third row - final model specification. The final specification (third row) shall include user modifications (row two) unless they were wrongly specified. The pre-specified outliers, user-defined variables and pre-specified ARMA coefficients consist of a list with the `Predefined` (base model specification) and `Final` values.

`regarima` object of class `c("regarima_spec","TRAMO_SEATS")`. See *Value* of the function `regarima_spec_tramoseats`

`seats` data.frame of class `c("seats_spec","data.frame")`, containing the *seats* variables in line with the names of the arguments variables. The final values can be also accessed with the function `s_seats`.

References

Info on 'JDemetra+', usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en BOX G.E.P. and JENKINS G.M. (1970), "Time Series Analysis: Forecasting and Control", Holden-Day, San Francisco.

BOX G.E.P., JENKINS G.M., REINSEL G.C. and LJUNG G.M. (2015), "Time Series Analysis: Forecasting and Control", John Wiley & Sons, Hoboken, N. J., 5th edition.

See Also

tramoseats

Examples

```
myseries <- ipi_c_eu[, "FR"]
myspec1 <- tramoseats_spec(spec = c("RSAfull"))
mysa1 <- tramoseats(myseries, spec = myspec1)

# Modify a pre-specified model specification
myspec2 <- tramoseats_spec(spec = "RSAfull", tradingdays.mauto = "Unused",
                           tradingdays.option = "WorkingDays",
                           easter.type = "Standard",
                           automdl.enabled = FALSE, arima.mu = TRUE)
mysa2 <- tramoseats(myseries, spec = myspec2)

# Modify the model specification from a "SA" object
myspec3 <- tramoseats_spec(mysa1, tradingdays.mauto = "Unused",
                           tradingdays.option = "WorkingDays",
                       easter.type = "Standard", automdl.enabled = FALSE, arima.mu = TRUE)
mysa3 <- tramoseats(myseries, myspec3)

# Modify the model specification from a "SA_spec" object
myspec4 <- tramoseats_spec(myspec1, tradingdays.mauto = "Unused",
                           tradingdays.option = "WorkingDays",
                       easter.type = "Standard", automdl.enabled = FALSE, arima.mu = TRUE)
mysa4 <- tramoseats(myseries, myspec4)

# Pre-specified outliers
myspec5 <- tramoseats_spec(spec = "RSAfull",
                           usrdef.outliersEnabled = TRUE,
                           usrdef.outliersType = c("LS", "LS"),
                           usrdef.outliersDate = c("2008-10-01", "2003-01-01"),
                           usrdef.outliersCoef = c(10,-8), transform.function = "None")
s_preOut(myspec5)
mysa5 <- tramoseats(myseries, myspec5)
mysa5
s_preOut(mysa5)

# User-defined calendar regressors
var1 <- ts(rnorm(length(myseries))*10, start = start(myseries), frequency = 12)
var2 <- ts(rnorm(length(myseries))*100, start = start(myseries), frequency = 12)
var<- ts.union(var1, var2)

myspec6 <- tramoseats_spec(spec = "RSAfull", tradingdays.option = "UserDefined",
                           usrdef.varEnabled = TRUE, usrdef.var = var,
                           usrdef.varType = c("Calendar", "Calendar"))
s_preVar(myspec6)
mysa6 <- tramoseats(myseries, myspec6)

myspec7 <- tramoseats_spec(spec = "RSAfull", usrdef.varEnabled = TRUE,
```

```
                            usrdef.var = var, usrdef.varCoef = c(17,-1),
                            transform.function = "None")
mysa7 <- tramoseats(myseries, myspec7)

# Pre-specified ARMA coefficients
myspec8 <- tramoseats_spec(spec = "RSAfull",
                            arima.coefEnabled = TRUE, automdl.enabled = FALSE,
                            arima.p = 2, arima.q = 0,
                            arima.bp = 1, arima.bq = 1,
                            arima.coef = c(-0.12, -0.12, -0.3, -0.99),
                            arima.coefType = rep("Fixed", 4))
mysa8 <- tramoseats(myseries, myspec8)
mysa8
s_arimaCoef(myspec8)
s_arimaCoef(mysa8)
```

`user_defined_variables` *Get the names of the user-defined variables*

Description

Function to get the names of the additional output variables that can be defined in `x13` and `tramoseats` with the parameter `userdefined`.

Usage

```
user_defined_variables(sa_object = c("X13-ARIMA", "TRAMO-SEATS"))
```

Arguments

`sa_object`	a character: `"X13-ARIMA"` to get the additional output variables available for the X13-ARIMA method and `"TRAMO-SEATS"` for the TRAMO-SEATS method.

Examples

```
user_defined_variables("X13-ARIMA")
user_defined_variables("TRAMO-SEATS")
```

x13 *Seasonal Adjustment with X-13ARIMA-SEATS*

Description

Functions to estimate the seasonally adjusted series (sa) with the X-13ARIMA-SEATS method. This is achieved by decomposing the time series (y) into the: trend-cycle (t), seasonal component (s) and irregular component (i). The final seasonally adjusted series shall be free of seasonal and calendar-related movements. `x13` returns a preformatted result while `jx13` returns the Java objects of the seasonal adjustment.

Usage

```
jx13(
  series,
  spec = c("RSA5c", "RSA0", "RSA1", "RSA2c", "RSA3", "RSA4c", "X11"),
  userdefined = NULL
)

x13(
  series,
  spec = c("RSA5c", "RSA0", "RSA1", "RSA2c", "RSA3", "RSA4c", "X11"),
  userdefined = NULL
)
```

Arguments

`series`	a univariate time series
`spec`	model specification X13. It can be a `character` of predefined X13 'JDemetra+' model specification (see *Details*), or a specification created by `x13_spec`. The default is `"RSA5c"`.
`userdefined`	vector with characters for additional output variables (see `user_defined_variables`).

Details

The first step of the seasonal adjustment consist of pre-adjusting the time series by removing from it the deterministic effects by means of a regression model with ARIMA noise (RegARIMA, see: `regarima`). In the second part, the pre-adjusted series is decomposed into the following components: trend-cycle (t), seasonal component (s) and irregular component (i). The decomposition can be: additive ($y = t + s + i$), multiplicative ($y = t * s * i$), log-additive ($log(y) = log(t) + log(s) + log(i)$) or pseudo-additive ($y = t * (s + i - 1)$). The final seasonally adjusted series (sa) shall be free of seasonal and calendar-related movements.

In the X13 method, the X11 algorithm (second step) decomposes the time series by means of linear filters. More information on the method can be found on the U.S. Census Bureau website.

As regards the available predefined 'JDemetra+' X13 model specifications, they are described in the table below.

Identifier	Log/level detection	Outliers detection	Calendar effects	ARIMA
RSA0	*NA*	*NA*	*NA*	Airline(+mean)
RSA1	automatic	AO/LS/TC	*NA*	Airline(+mean)
RSA2c	automatic	AO/LS/TC	2 td vars + Easter	Airline(+mean)
RSA3	automatic	AO/LS/TC	*NA*	automatic
RSA4c	automatic	AO/LS/TC	2 td vars + Easter	automatic
RSA5c	automatic	AO/LS/TC	7 td vars + Easter	automatic
X11	*NA*	*NA*	*NA*	NA

Value

`jx13` returns a `jSA` object. It contains the Java objects of the result of the seasonal adjustment without any formatting. Therefore the computation is faster than with `x13`. The results can the seasonal adjustment can be extract by `get_indicators`.

`x13` returns an object of class `c("SA","X13")`, a list containing the following components:

`regarima` object of class `c("regarima","X13")`. See *Value* of the function `regarima`.

`decomposition` object of class `"decomposition_X11"`, six elements list:

- `specification` list with the X11 algorithm specification. See also function `x13_spec`
- `mode` decomposition mode
- `mstats` matrix with the M statistics
- `si_ratio` time series matrix (mts) with the `d8` and `d10` series
- `s_filter` seasonal filters
- `t_filter` trend filter

`final` object of class `c("final","mts","ts","matrix")`. Matrix with the final results of the seasonal adjustment. It includes time series: original time series (`y`), forecast of the original series (`y_f`), trend (`t`), forecast of the trend (`t_f`), seasonally adjusted series (`sa`), forecast of the seasonally adjusted series (`sa_f`), seasonal component (`s`), forecast of the seasonal component (`s_f`), irregular component (`i`) and the forecast of the irregular component (`i_f`).

`diagnostics` object of class `"diagnostics"`, list with three type of diagnostics tests:

- `variance_decomposition` data.frame with the tests on the relative contribution of the components to the stationary portion of the variance in the original series, after the removal of the long term trend.
- `residuals_test` data.frame with the tests on the presence of seasonality in the residuals (includes the statistic, p-value and parameters description)
- `combined_test` combined tests for stable seasonality in the entire series. Two elements list with: `tests_for_stable_seasonality` - data.frame with the tests (includes the statistic, p-value and parameters description) and `combined_seasonality_test` - the summary.

`user_defined` object of class `"user_defined"`. List containing the userdefined additional variables defined in the `userdefined` argument.

References

Info on 'JDemetra+', usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en

BOX G.E.P. and JENKINS G.M. (1970), "Time Series Analysis: Forecasting and Control", Holden-Day, San Francisco.

BOX G.E.P., JENKINS G.M., REINSEL G.C. and LJUNG G.M. (2015), "Time Series Analysis: Forecasting and Control", John Wiley & Sons, Hoboken, N. J., 5th edition.

See Also

x13_spec, tramoseats

Examples

```
myseries <- ipi_c_eu[, "FR"]
mysa <- x13(myseries, spec = "RSA5c")

myspec1 <- x13_spec(mysa, tradingdays.option = "WorkingDays",
            usrdef.outliersEnabled = TRUE,
            usrdef.outliersType = c("LS","AO"),
            usrdef.outliersDate = c("2008-10-01", "2002-01-01"),
            usrdef.outliersCoef = c(36, 14),
            transform.function = "None")
mysa1 <- x13(myseries, myspec1)
mysa1
summary(mysa1$regarima)

myspec2 <- x13_spec(mysa, automdl.enabled =FALSE,
            arima.coefEnabled = TRUE,
            arima.p = 1, arima.q = 1, arima.bp = 0, arima.bq = 1,
            arima.coef = c(-0.8, -0.6, 0),
            arima.coefType = c(rep("Fixed", 2), "Undefined"))
s_arimaCoef(myspec2)
mysa2 <- x13(myseries, myspec2,
             userdefined = c("decomposition.d18", "decomposition.d19"))
mysa2
plot(mysa2)
plot(mysa2$regarima)
plot(mysa2$decomposition)
```

---

x13_spec *X-13ARIMA-SEATS model specification, SA/X13*

Description

Function to create (and/or modify) a `c("SA_spec","X13")` class object with the SA model specification for the X13 method. It can be done from a pre-defined 'JDemetra+' model specification (a `character`), a previous specification (`c("SA_spec","X13")` object) or a seasonal adjustment model (`c("SA","X13")` object).

Usage

```
x13_spec(
  spec = c("RSA5c", "RSA0", "RSA1", "RSA2c", "RSA3", "RSA4c", "X11"),
  preliminary.check = NA,
  estimate.from = NA_character_,
  estimate.to = NA_character_,
  estimate.first = NA_integer_,
  estimate.last = NA_integer_,
  estimate.exclFirst = NA_integer_,
  estimate.exclLast = NA_integer_,
  estimate.tol = NA_integer_,
  transform.function = c(NA, "Auto", "None", "Log"),
  transform.adjust = c(NA, "None", "LeapYear", "LengthOfPeriod"),
  transform.aicdiff = NA_integer_,
  usrdef.outliersEnabled = NA,
  usrdef.outliersType = NA,
  usrdef.outliersDate = NA,
  usrdef.outliersCoef = NA,
  usrdef.varEnabled = NA,
  usrdef.var = NA,
  usrdef.varType = NA,
  usrdef.varCoef = NA,
 tradingdays.option = c(NA, "TradingDays", "WorkingDays", "UserDefined", "None"),
  tradingdays.autoadjust = NA,
  tradingdays.leapyear = c(NA, "LeapYear", "LengthOfPeriod", "None"),
  tradingdays.stocktd = NA_integer_,
  tradingdays.test = c(NA, "Remove", "Add", "None"),
  easter.enabled = NA,
  easter.julian = NA,
  easter.duration = NA_integer_,
  easter.test = c(NA, "Add", "Remove", "None"),
  outlier.enabled = NA,
  outlier.from = NA_character_,
  outlier.to = NA_character_,
  outlier.first = NA_integer_,
  outlier.last = NA_integer_,
  outlier.exclFirst = NA_integer_,
  outlier.exclLast = NA_integer_,
  outlier.ao = NA,
  outlier.tc = NA,
  outlier.ls = NA,
```

```
  outlier.so = NA,
  outlier.usedefcv = NA,
  outlier.cv = NA_integer_,
  outlier.method = c(NA, "AddOne", "AddAll"),
  outlier.tcrate = NA_integer_,
  automdl.enabled = NA,
  automdl.acceptdefault = NA,
  automdl.cancel = NA_integer_,
  automdl.ub1 = NA_integer_,
  automdl.ub2 = NA_integer_,
  automdl.mixed = NA,
  automdl.balanced = NA,
  automdl.armalimit = NA_integer_,
  automdl.reducecv = NA_integer_,
  automdl.ljungboxlimit = NA_integer_,
  automdl.ubfinal = NA_integer_,
  arima.mu = NA,
  arima.p = NA_integer_,
  arima.d = NA_integer_,
  arima.q = NA_integer_,
  arima.bp = NA_integer_,
  arima.bd = NA_integer_,
  arima.bq = NA_integer_,
  arima.coefEnabled = NA,
  arima.coef = NA,
  arima.coefType = NA,
  fcst.horizon = NA_integer_,
  x11.mode = c(NA, "Undefined", "Additive", "Multiplicative", "LogAdditive",
    "PseudoAdditive"),
  x11.seasonalComp = NA,
  x11.lsigma = NA_integer_,
  x11.usigma = NA_integer_,
  x11.trendAuto = NA,
  x11.trendma = NA_integer_,
  x11.seasonalma = NA_character_,
  x11.fcasts = NA_integer_,
  x11.bcasts = NA_integer_,
  x11.calendarSigma = NA,
  x11.sigmaVector = NA,
  x11.excludeFcasts = NA
)
```

Arguments

`spec`	model specification X13. It can be a `character` of predefined X13 'JDemetra+' model specification (see *Details*), an object of class `c("SA_spec","X13")` or an object of class `c("SA","X13")`. The default is `"RSA5c"`.
`preliminary.check`	boolean to check the quality of the input series and exclude highly problematic

ones: e.g. these with a number of identical observations and/or missing values above pre-specified threshold values.

The time span of the series to be used for the estimation of the RegARIMA model coefficients (default from 1900-01-01 to 2020-12-31) is controlled by the following six variables: estimate.from,estimate.to,estimate.first,estimate.last,estimate.e and estimate.exclLast; where estimate.from and estimate.to have priority over remaining span control variables, estimate.last and estimate.first have priority over estimate.exclFirst and estimate.exclLast, and estimate.last has priority over estimate.first.

estimate.from character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). Can be combined with estimate.to.

estimate.to character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). Can be combined with estimate.from.

estimate.first numeric specifying the number of periods considered at the beginning of the series.

estimate.last numeric specifying the number of periods considered at the end of the series.

estimate.exclFirst
numeric specifying the number of periods excluded at the beginning of the series. Can be combined with estimate.exclLast.

estimate.exclLast
numeric specifying the number of periods excluded at the end of the series. Can be combined with estimate.exclFirst.

estimate.tol 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.

transform.function
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.

transform.adjust
pre-adjustment of the input series for 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 transform.function is set to "Log".

transform.aicdiff
numeric defining the difference in AICC needed to accept no transformation when the automatic transformation selection is chosen (considered only when transform.function is set to "Auto").

Control variables for the pre-specified outliers. The pre-specified outliers are used in the model only if they are enabled (usrdef.outliersEnabled=TRUE) and the outliers' type (usrdef.outliersType) and date (usrdef.outliersDate) are provided.

usrdef.outliersEnabled
logicals. If TRUE the program uses the pre-specified outliers.

usrdef.outliersType
: vector defining the outliers' type. Possible types are: ("AO") - additive, ("LS") - level shift, ("TC") - transitory change, ("SO") - seasonal outlier. E.g.: usrdef.outliersType =c("AO","AO","LS").

usrdef.outliersDate
: vector defining the outliers' date. The dates should be characters in format "YYYY-MM-DD". E.g.: usrdef.outliersDate=c("2009-10-01","2005-02-01","2003-04-01").

usrdef.outliersCoef
: vector providing fixed coefficients for the outliers. The coefficients can't be fixed if transform.function is set to "Auto" - the series transformation need to be pre-defined. E.g.: usrdef.outliersCoef=c(200,170,20).

Control variables for the user-defined variables:

usrdef.varEnabled
: logicals. If TRUE the program uses the user-defined variables.

usrdef.var
: time series (ts) or matrix of time series (mts) with the user-defined variables.

usrdef.varType
: vector of character(s) defining the user-defined variables component type. Possible types are: "Undefined","Series","Trend","Seasonal","SeasonallyAdjusted","Irregular", The type "Calendar" has to be used with tradingdays.option = "UserDefined" to use user-defined calendar regressors. If not specified, the program will assign the "Undefined" type.

usrdef.varCoef
: vector providing fixed coefficients for the user-defined variables. The coefficients can't be fixed if transform.function is set to "Auto" - the series transformation need to be pre-defined.

tradingdays.option
: defines the type of the trading days regression variables: "TradingDays" - six day-of-the-week regression variables; "WorkingDays" - one working/non-working day contrast variable; "None" - no correction for trading days and working days effects; "UserDefined" - user-defined trading days regressors (regressors have to be defined by the usrdef.var argument with usrdef.varType set to "Calendar" and usrdef.varEnabled = TRUE). "None" has also to be chosen for the "day-of-week effects" correction (tradingdays.stocktd to be modified accordingly).

tradingdays.autoadjust
: logicals. If TRUE the program corrects automatically for the leap year effect. Modifications of this variable are taken into account only when transform.function is set to "Auto".

tradingdays.leapyear
: option for including the leap-year effect in the model: "LeapYear" - leap year effect; "LengthOfPeriod" - length of period, "None" - no effect included. The leap-year effect can be pre-specified in the model only if the input series was not pre-adjusted (transform.adjust set to "None") and the automatic correction for the leap-year effect was not selected (tradingdays.autoadjust set to FALSE).

tradingdays.stocktd
: 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). Modifications of this variable are taken into account only when tradingdays.option is set to "None".

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

`easter.enabled`
: logicals. If `TRUE` the program considers the Easter effect in the model.

`easter.julian`
: logicals. If `TRUE` the program uses the Julian Easter (expressed in Gregorian calendar).

`easter.duration`
: numeric indicating the duration of the Easter effect (length in days, between 1 and 20).

`easter.test`
: defines the pre-tests for the significance of the Easter effect based on the t-statistic (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 belong 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.

`outlier.enabled`
: logicals. If `TRUE` the automatic detection of outliers is enabled in the defined time span.

  The time span of the series to be searched for outliers (default from 1900-01-01 to 2020-12-31) is controlled by the following six variables: `outlier.from`,`outlier.to`,`outlier.first` and `outlier.exclLast`; where `outlier.from` and `outlier.to` have priority over remaining span control variables, `outlier.last` and `outlier.first` have priority over `outlier.exclFirst` and `outlier.exclLast`, and `outlier.last` has priority over `outlier.first`.

`outlier.from`
: character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). Can be combined with `outlier.to`.

`outlier.to`
: character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). Can be combined with `outlier.from`.

`outlier.first`
: numeric specifying the number of periods considered at the beginning of the series.

`outlier.last`
: numeric specifying the number of periods considered at the end of the series.

`outlier.exclFirst`
: numeric specifying the number of periods excluded at the beginning of the series. Can be combined with `outlier.exclLast`.

`outlier.exclLast`
: numeric specifying the number of periods excluded at the end of the series. Can be combined with `outlier.exclFirst`.

`outlier.ao`
: logicals. If `TRUE` the automatic detection of additive outliers is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.tc`
: logicals. If `TRUE` the automatic detection of transitory changes is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.ls` logicals. If `TRUE` the automatic detection of level shifts is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.so` logicals. If `TRUE` the automatic detection of seasonal outliers is enabled (`outlier.enabled` must be also set to `TRUE`).

`outlier.usedefcv` logicals. If `TRUE` the critical value for the outliers' detection procedure is automatically determined by the number of observations in the outlier detection time span. If `FALSE` the procedure uses the inputted critical value (`outlier.cv`).

`outlier.cv` numeric. Inputted critical value for the outliers' detection procedure. The modification of this variable is taken into account only when `outlier.usedefcv` is set to `FALSE`.

`outlier.method` determines how the program successively adds detected outliers to the model. At present only the `AddOne` method is supported.

`outlier.tcrate` numeric. The rate of decay for the transitory change outlier.

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

Control variables for the automatic modelling of the ARIMA model (`automdl.enabled` is set to `TRUE`):

`automdl.acceptdefault` logicals. If `TRUE` the default model (ARIMA(0,1,1)(0,1,1)) may be chosen in the first step of the automatic model identification. If the Ljung-Box Q statistics for the residuals is acceptable, the default model is accepted and no further attempt will be made to identify any other.

`automdl.cancel` numeric, cancelation limit. If the difference in moduli of an AR and an MA roots (when estimating ARIMA(1,0,1)(1,0,1) models in the second step of the automatic identification of the differencing orders) is smaller than cancelation limit, the two roots are assumed equal and cancel out.

`automdl.ub1` numeric, 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.

`automdl.ub2` numeric, 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).

`automdl.mixed` logicals. The 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 not AR and MA terms in either the seasonal or non-seasonal.

`automdl.balanced`
: logicals. 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 operator is equal to the order of the combined MA operator).

`automdl.armalimit`
: numeric, arma limit. It is the threshold value for t-statistics of ARMA coefficients and constant term used for the final test of model parsimony. If the highest order ARMA coefficient has a t-value less than this value in magnitude, the order of the model is reduced. Also if the constant term has a t-value less than arma limit in magnitude, it is removed from the set of regressors.

`automdl.reducecv`
: numeric, ReduceCV. The percentage by which the outlier's 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)xCV, where CV is the original critical value.

`automdl.ljungboxlimit`
: numeric, Ljung Box limit. 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, the model is rejected, the outlier critical value is reduced, and model and outlier identification (if specified) is redone with a reduced value.

`automdl.ubfinal`
: 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 less than the final unit root limit, 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.

Control variables for the non-automatic modelling of the ARIMA model (`automdl.enabled` is set to `FALSE`):

`arima.mu`
: logicals. If `TRUE`, the mean is considered as part of the ARIMA model.

`arima.p`
: numeric. The order of the non-seasonal autoregressive (AR) polynomial.

`arima.d`
: numeric. Regular differencing order.

`arima.q`
: numeric. The order of the non-seasonal moving average (MA) polynomial.

`arima.bp`
: numeric. The order of the seasonal autoregressive (AR) polynomial.

`arima.bd`
: numeric. Seasonal differencing order.

`arima.bq`
: numeric. The order of the seasonal moving average (MA) polynomial.

Control variables for the user-defined ARMA coefficients. Coefficients can be defined for the regular and seasonal autoregressive (AR) polynomials and moving average (MA) polynomials. The model considers the coefficients only if the procedure for their estimation (`arima.coefType`) is provided, and the number of provided coefficients matches the sum of (regular and seasonal) AR and MA orders (`p,q,bp,bq`).

`arima.coefEnabled`
: logicals. If `TRUE` the program uses the user-defined ARMA coefficients.

arima.coef vector providing the coefficients for the regular and seasonal AR and MA polynominals. The length of the vector must equal the sum of the regular and seasonal AR and MA orders. The coefficients shall be provided in the 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 arima.p=1,arima.q=0,arima.bp=1 and arima.bq=0.

arima.coefType vector defining ARMA coefficients estimation procedure. Possible procedures are: "Undefined" - no use of user-defined input (i.e. coefficients are estimated), "Fixed" - fixes the coefficients at the value provided by the user, "Initial" - the value defined by the user is used as initial condition. For orders for which the coefficients shall not be defined, the arima.coef can be set to NA or 0 or the arima.coefType can be set to "Undefined". E.g.: arima.coef = c(-0.8,-0.6,NA), arima.coefType = c("Fixed","Fixed","Undefined").

fcst.horizon numeric, forecasts horizon. Length of the forecasts generated by the RegARIMA model in periods (positive values) or years (negative values). By default the program generates two years forecasts (fcst.horizon set to -2).

x11.mode character, decomposition mode. Determines the mode of the seasonal adjustment decomposition to be performed: "Undefined" - no assumption concerning the relationship between the time series components is made; "Additive" - assumes an additive relationship; "Multiplicative" - assumes a multiplicative relationship; "LogAdditive" - performs an additive decomposition of the logarithms of the series being adjusted; "PseudoAdditive" - assumes an pseudo-additive relationship. Could be changed by the program, if needed.

x11.seasonalComp logicals. If TRUE the program computes a seasonal component. Otherwise, the seasonal component is not estimated and its values are all set to 0 (additive decomposition) or 1 (multiplicative decomposition).

x11.lsigma numeric, lower sigma boundary for the detection of extreme values.

x11.usigma numeric, upper sigma boundary for the detection of extreme values.

x11.trendAuto logicals, automatic Henderson filter. If TRUE an automatic selection of the Henderson filter's length for the trend estimation is enabled.

x11.trendma numeric, length of the Henderson filter. The user-defined length of the Henderson filter. The option is available when the automatic Henderson filter selection is disabled (x11.trendAuto=FALSE). Should be an odd number in the range (1, 101].

x11.seasonalma vector of character(s) specifying which seasonal moving average (i.e. seasonal filter) will be used to estimate the seasonal factors for the entire series. The vector can be of length: 1 - same seasonal filters for all periods (e.g.: seasonalma=c("Msr")); or period's number - a seasonal filer is defined for each period (e.g. for quarterly series: seasonalma=c("S3X3","Msr","S3X3","Msr")). Possible filters are: "Msr","Stable","X11Default","S3X1","S3X3","S3X5","S3X9","S3X15". "Msr" - the program chooses the final seasonal filter automatically.

x11.fcasts numeric, RegARIMA forecast. Length of the forecasts generated by the RegARIMA model in periods (positive values) or years (negative values).

x11.bcasts numeric, backcast. Length of the backcasts used in X11. Negative figures are translated in years of backcasts.

`x11.calendarSigma`
: character to specify if the standard errors used for extreme values detection and adjustment are computed from 5 year spans of irregulars (`"None"`, the default); separately for each calendar month/quarter (`"All"`); separately for each period only if Cochran's hypothesis test determines that the irregular component is heteroskedastic by calendar month/quarter (`"Signif"`); separately for two complementary sets of calendar months/quarters specified by the x11.sigmaVector parameter (`"Select"`, see parameter `x11.sigmaVector`).

`x11.sigmaVector`
: vector to specifies one of the two groups of periods for whose standard errors used for extreme values detection and adjustment will be computed. Only used if `x11.calendarSigma = "Select"`. Possibe values are: "Group1" and "Group2".

`x11.excludeFcasts`
: logicals, exclude forecats and backcasts. If `TRUE` forecasts and backcasts from the RegARIMA model are not used in the generation of extreme values in the seasonal adjustment routines.

Details

The available predefined 'JDemetra+' model specifications are described in the table below.

Identifier	Log/level detection	Outliers detection	Calendar effects	ARIMA
RSA0	*NA*	*NA*	*NA*	Airline(+mean)
RSA1	automatic	AO/LS/TC	*NA*	Airline(+mean)
RSA2c	automatic	AO/LS/TC	2 td vars + Easter	Airline(+mean)
RSA3	automatic	AO/LS/TC	*NA*	automatic
RSA4c	automatic	AO/LS/TC	2 td vars + Easter	automatic
RSA5c	automatic	AO/LS/TC	7 td vars + Easter	automatic
X11	*NA*	*NA*	*NA*	NA

Value

A two-elements list of class `c("SA_spec","X13")`: (1) object of class `c("regarima_spec","X13")` with the RegARIMA model specification, (2) object of class `c("X11_spec","data.frame")` with the X11 algorithm specification. Each component refers to different part of the SA model specification, mirroring the arguments of the function (for details see arguments description). Each of the lowest-level component (except span, pre-specified outliers, user-defined variables and pre-specified ARMA coefficients) is structured within a data frame with columns denoting different variables of the model specification and rows referring to: first row - base specification, as provided within the argument `spec`; second row - user modifications as specified by the remaining arguments of the function (e.g.: `arima.d`); and third row - final model specification. The final specification (third row) shall include user modifications (row two) unless they were wrongly specified. The pre-specified outliers, user-defined variables and pre-specified ARMA coefficients consist of a list with the `Predefined` (base model specification) and `Final` values.

`regarima`
: object of class `c("regarima_spec","x13")`. See *Value* of the function `regarima_spec_x13`

`x11`
: data.frame of class `c("X11_spec","data.frame")`, containing the *x11* variables in line with the names of the arguments variables. The final values can be also accessed with the function `s_x11`.

References

Info on 'JDemetra+', usage and functions: https://ec.europa.eu/eurostat/cros/content/documentation_en BOX G.E.P. and JENKINS G.M. (1970), "Time Series Analysis: Forecasting and Control", Holden-Day, San Francisco.

BOX G.E.P., JENKINS G.M., REINSEL G.C. and LJUNG G.M. (2015), "Time Series Analysis: Forecasting and Control", John Wiley & Sons, Hoboken, N. J., 5th edition.

See Also

x13

Examples

```
myseries <- ipi_c_eu[, "FR"]
myspec1 <- x13_spec(spec = "RSA5c")
myreg1 <- x13(myseries, spec = myspec1)

# Modify a pre-specified model specification
myspec2 <- x13_spec(spec = "RSA5c", tradingdays.option = "WorkingDays")
myreg2 <- x13(myseries, spec = myspec2)

# Modify the model specification from a "X13" object
 myspec3 <- x13_spec(myreg1, tradingdays.option = "WorkingDays")
 myreg3 <- x13(myseries, myspec3)

# Modify the model specification from a "X13_spec" object
 myspec4 <- x13_spec(myspec1, tradingdays.option = "WorkingDays")
 myreg4 <- x13(myseries, myspec4)

# Pre-specified outliers
 myspec1 <- x13_spec(spec = "RSA5c", usrdef.outliersEnabled = TRUE,
             usrdef.outliersType = c("LS", "AO"),
             usrdef.outliersDate = c("2008-10-01", "2002-01-01"),
             usrdef.outliersCoef = c(36, 14),
             transform.function = "None")

 myreg1 <- x13(myseries, myspec1)
 myreg1
 s_preOut(myreg1)


# User-defined calendar regressors
 var1 <- ts(rnorm(length(myseries))*10, start = start(myseries), frequency = 12)
 var2 <- ts(rnorm(length(myseries))*100, start = start(myseries), frequency = 12)
 var <- ts.union(var1, var2)
 myspec1 <- x13_spec(spec = "RSA5c", tradingdays.option = "UserDefined",
                     usrdef.varEnabled = TRUE,
                     usrdef.var = var,
                     usrdef.varType = c("Calendar", "Calendar"))
 myreg1 <- x13(myseries, myspec1)
```

```
myreg1

myspec2 <- x13_spec(spec = "RSA5c", usrdef.varEnabled = TRUE,
            usrdef.var = var1, usrdef.varCoef = 2,
            transform.function = "None")
myreg2 <- x13(myseries, myspec2)
s_preVar(myreg2)

# Pre-specified ARMA coefficients
myspec1 <- x13_spec(spec = "RSA5c", automdl.enabled = FALSE,
            arima.p = 1, arima.q = 1, arima.bp = 0, arima.bq = 1,
            arima.coefEnabled = TRUE,
            arima.coef = c(-0.8, -0.6, 0),
            arima.coefType = c(rep("Fixed", 2), "Undefined"))

s_arimaCoef(myspec1)
myreg1 <- x13(myseries, myspec1)
myreg1

# Defined seasonal filters
myspec1 <- x13_spec("RSA5c", x11.seasonalma = rep("S3X1", 12))
mysa1 <- x13(myseries, myspec1)
```

Index

∗ **datasets**
    ipi_c_eu, 9

add_sa_item, 2, *14*

coef, *19*
compute, 3, *5*
count, 4, *5, 6*, *8, 9*

get_all_objects *(*get_object*)*, 7
get_dictionary *(*jSA*)*, 11
get_indicators, *18*, *49*, *63*
get_indicators *(*jSA*)*, 11
get_jmodel *(*get_model*)*, 5
get_jspec *(*jSA*)*, 11
get_model, *3, 4*, 5, *6*, *8, 9*, *13*
get_name, *4, 5*, 6, *8, 9*
get_object, 7
get_ts, *4–6*, *8*, 8

ipi_c_eu, 9

jregarima *(*regarima*)*, 16
jregarima_tramoseats *(*regarima*)*, 16
jregarima_x13 *(*regarima*)*, 16
jSA, *5*, 11, *18*, *49*, *63*
jSA2R *(*jSA*)*, 11
jtramoseats *(*tramoseats*)*, 47
jx13 *(*x13*)*, 62

load_spec *(*save_spec*)*, 41
load_workspace, *3*, 13, *14*, *43*
logLik, *19*

new_multiprocessing *(*new_workspace*)*, 13
new_workspace, 13

plot, 14

regarima, 16, *28*, *38*, *47*, *49*, *62, 63*
regarima_spec_tramoseats, *16*, *18*, 20, *59*
regarima_spec_x13, *16*, *18*, 31, *73*
regarima_tramoseats *(*regarima*)*, 16
regarima_x13 *(*regarima*)*, 16
residuals, *19*

s_arima, *29*, *39*
s_arima *(*specification*)*, 44
s_arimaCoef, *29*, *39*
s_arimaCoef *(*specification*)*, 44
s_easter, *29*, *39*
s_easter *(*specification*)*, 44
s_estimate, *28*, *39*
s_estimate *(*specification*)*, 44
s_fcst, *29*, *39*
s_fcst *(*specification*)*, 44
s_out, *29*, *39*
s_out *(*specification*)*, 44
s_preOut, *29*, *39*
s_preOut *(*specification*)*, 44
s_preVar, *29*, *39*
s_preVar *(*specification*)*, 44
s_seats, *59*
s_seats *(*specification*)*, 44
s_span, *29*, *40*
s_span *(*specification*)*, 44
s_td, *29*, *39*
s_td *(*specification*)*, 44
s_transform, *28*, *39*
s_transform *(*specification*)*, 44
s_usrdef, *29*, *39*
s_usrdef *(*specification*)*, 44
s_x11, *73*
s_x11 *(*specification*)*, 44
save_spec, 41
save_workspace, *3*, *13, 14*, 43
specification, 44

tramoseats, *5*, 47, *60, 61*, *64*
tramoseats_spec, *49, 50*, 51
ts, *8*

user_defined_variables, *12*, *47*, 61, *62*

x13, *5*, *50*, *61*, 62, *74*
x13_spec, *62–64*, 64