Package 'RJDemetra'

March 19, 2024

Type Package

Title Interface to 'JDemetra+' Seasonal Adjustment Software

Version 0.2.6

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 (>= 8)

License EUPL

LazyData TRUE

Suggests knitr, rmarkdown

URL <https://jdemetra.github.io/rjdemetra/>,
<https://github.com/jdemetra/rjdemetra>

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

Encoding UTF-8

RoxygenNote 7.3.1

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 2024-03-19 22:10:04 UTC

R **topics documented:**

---

add_sa_item | *Add a seasonally adjusted series to a multi-processing*

Description

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

Usage

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

Arguments

workspace   the workspace to add the seasonally adjusted series to.

multiprocessing

         the name or index of the multiprocessing to add the seasonally adjusted series to.

sa_obj   the seasonally adjusted object to add

name the name of the seasonally adjusted series in the multiprocessing. By default the name of the `sa_obj` is used.

See Also

[load_workspace](), [save_workspace]()

Examples

```
dir <- tempdir()
# Adjustment of a series with the x13 and Tramo-Seats methods
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)

# Creation of a new workspace..
wk <- new_workspace()
# and of the multiprocessing "sa1" that will contain the series
new_multiprocessing(wk, "sa1")
# Addition of the adjusted series to the workspace via the sa1 multiprocessing
add_sa_item(wk, "sa1", sa_x13, "X13")
add_sa_item(wk, "sa1", sa_ts, "TramoSeats")

# Export of the new filled workspace
save_workspace(wk, file.path(dir, "workspace.xml"))
```

---

compute *Compute a workspace multi-processing(s)*

Description

Function to compute all the multiprocessings or only a given one from a workspace. By default, the workspace only contains definitions: computation is needed to recalculate and access the adjusted 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 multiprocessings are computed.

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, "sap1")
add_sa_item(wk, "sap1", 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

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

Usage

```
count(x)
```

Arguments

x   the `workspace` or the `multiprocessing`.

See Also

Other functions to retrieve information from a workspace, multiprocessing or sa_item: [get_model](#), [get_name](#), [get_ts](#).

Examples

```
wk <- new_workspace()
mp <- new_multiprocessing(wk, "sap1")
count(wk) # 1 multiprocessing inside the workspace wk
count(mp) # 0 sa_item inside the multiprocessing mp
```

---

get_all_names   *Get the Java name of all the contained object*

Description

Generic functions to retrieve the Java name of the contained `multiprocessings` or the contained `sa_items`.

Usage

```
get_all_names(x)
```

Arguments

x   An object containing other objects whose names we want to know

Value

A `character` vector containing all the names.

See Also

Other functions to retrieve information from a workspace, multiprocessing or sa_item: get_name, get_position, 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, "sap1")
mp2 <- new_multiprocessing(wk, "sap2")

get_all_names(wk)

add_sa_item(wk, "sap1", sa_x13, "X13")
add_sa_item(wk, "sap1", sa_ts, "TramoSeats")
```

```
get_all_names(mp)
```

get_model   *Get the seasonally adjusted model from a workspace*

Description

Generic functions to retrieve seasonally adjusted model(s) from `workspace`, `multiprocessing` or `sa_item` object. `get_model` returns a `"SA"` object 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 from which to retrieve the seasonally adjusted model.
workspace	the workspace object where models are stored. If x is a workspace object, this parameter is not used.
userdefined	a vector containing the names of additional output variables. (see [x13](#) or [tramoseats](#)).
progress_bar	Boolean: if TRUE, a progress bar is printed.

Value

`get_model()` returns a seasonally adjusted object (class `c("SA", "X13")` or `c("SA", "TRAMO_SEATS")`) or a list of seasonally adjusted 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 a 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 multiprocessings, each element containing a list of `"SA"` object(s) (or [jSA](#) object's) with `get_jmodel(x)`).

See Also

Other functions to retrieve information 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, "sap1")
add_sa_item(wk, "sap1", sa_x13, "X13")
add_sa_item(wk, "sap1", sa_ts, "TramoSeats")

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

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

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

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

---

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

Description

Generic functions to retrieve the Java name of a `multiprocessing` or a `sa_item`.

Usage

```
get_name(x)
```

Arguments

x   the object to retrieve the name from.

Value

A `character`.

See Also

Other functions to retrieve information 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, "sap1")
add_sa_item(wk, "sap1", sa_x13, "X13")
add_sa_item(wk, "sap1", 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 "sap1"

# To retrieve the name of every sa_item in a given multiprocessing:
sapply(get_all_objects(mp), get_name)

# To retrieve the name of every multiprocessing in a given workspace:
sapply(get_all_objects(wk), get_name)

# To retrieve the name of every sa_item in a given 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

Generic functions to retrieve all (get_all_objects()) multiprocessing (respectively sa_item) from a workspace (respectively multiprocessing) or to retrieve a single one (get_object()) .

Usage

```
get_object(x, pos = 1)

get_all_objects(x)
```

Arguments

x	the object in which to store the extracted multiprocessing or 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 retrieve information 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, "sap1")
add_sa_item(wk, "sap1", sa_x13, "X13")

# A way to retrieve the multiprocessing:
mp <- get_object(wk, 1)
# And the sa_item object:
sa_item <- get_object(mp, 1)
```

---

get_position   *Get the position of an object*

Description

Generic functions to retrieve the position of the contained `multiprocessings` or the contained `sa_items`.

Usage

```
get_position(x, name)
```

Arguments

x	An object containing other objects whose names we want to know
name	acharacter specifiing an object

Value

A `integer`

See Also

Other functions to retrieve information from a workspace, multiprocessing or sa_item: get_name, get_all_names, 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, "sap1")
mp2 <- new_multiprocessing(wk, "sap2")

get_position(wk, "sap1")
get_position(wk, "sap2")

add_sa_item(wk, "sap1", sa_x13, "X13")
add_sa_item(wk, "sap1", sa_ts, "TramoSeats")

get_position(mp, "TramoSeats")
get_position(mp, "X13")
```

---

get_ts   *Get the input raw time series*

Description

Generic functions to retrieve the input raw time series of a workspace, multiprocessing, sa_item or SA object.

Usage

```
get_ts(x)
```

Arguments

x   the object from which to retrieve 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 a list of length the number of sa_items, each element being a `ts` object;
- if x is a `workspace` object, `get_ts(x)` returns a list of length the number of multiprocessings, each element being a list of `ts` objects.

See Also

Other functions to retrieve information 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, "sap1")
add_sa_item(wk, "sap1", sa_x13, "X13")
sa_item <- get_object(mp, 1)

  # Extracting the raw time series from an adjusted series:
get_ts(sa_x13) # Returns the ts object ipi_c_eu[, "FR"]

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

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


  # Extracting all raw time series from a workspace:
# Returns a list of length 1 named "sap1" 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 industry 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 2020 with 34 variables.

Details

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

BE	Belgium
BG	Bulgaria
CZ	Czech Republic
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

Eurostat, 'sts_inpr_m' database.

---

jSA   *Functions around 'jSA' objects*

Description

`get_dictionary` returns the indicators that can be extracted from `"jSA"` objects, `get_indicators` extracts a list of indicators `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	a userdefined vector containing the names of additional output variables (see [user_defined_variables](#)). Only used for `"SA"` objects.

Details

A `"jSA"` object is a list of three elements:

- `"result"`: the Java object containing the results of a seasonal adjustment or a pre-adjustment method.
- `"spec"`: the Java object containing the specification of a seasonal adjustment or a pre-adjustment method.
- `"dictionary"`: the Java object containing the dictionary 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 seasonally adjusted (`"SA"` object) or RegARIMA (`"regarima"` object) model.

`get_jspec` returns the Java object that contains the specification of an object. Such object can be of type `"jSA"`, `"X13"`, `"TRAMO_SEATS"` or `"sa_item"`.

Value

`get_dictionary` returns 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 the Java object to an R object
jSA2R(mysa)
```

---

load_workspace　　　　　　　　　*Load a 'JDemetra+' workspace*

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()) and to add a new multi-processing (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

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

---

plot   *Plotting regarima, decomposition or final results of a 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	a numeric vector 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	a string containing the graph title
ask	a Boolean. If TRUE, the user will be prompted before a new graphical page is started.
...	other parameters
first_date	the plot starting date. If missing, the plot starts at the beginning of the time-series.
last_date	the end date of 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	a Boolean indicating if forecasts should be included in the plot. If TRUE, the forecast is plotted.
type_chart	a string 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 plot only one graphic (here, the residuals) and change 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$final) # 2 graphics are plotted by default
# To only plot one graphic (here the raw data, the seasonally adjusted data and the trend),
# To change the last date and the title
plot(mysa$final, last_date = c(2000, 1),
     caption = "Results", type_chart = "sa-trend")
```

---

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

Description

The `regarima/regarima_x13/regarima_tramoseats` functions remove deterministic effects from
the input series (e.g.calendar effects, outliers) using a multivariate regression model with arima er-
rors. The `jregarima/jregarima_x13/jregarima_tramoseats` functions do the same computa-
tion but return the Java objects instead of a formatted 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   an univariate time series

spec   the model specification. For the function:

- regarima: an object of class c("regarima_spec","X13") or c("regarima_spec","TRAMO_SEATS"). See the functions regarima_spec_x13 and regarima_spec_tramoseats.
- regarima_x13: the name of a predefined X13 'JDemetra+' model specification (see *Details*). The default value is "RG5c".
- regarima_tramoseats:the name of a predefined TRAMO-SEATS 'JDemetra+' model specification (see *Details*). The default value is "TRfull".

Details

When seasonally adjusting with X13 and TRAMO-SEATS, the first step consists in 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 to as the linearization of the original series.

The RegARIMA model (model with ARIMA errors) is specified as such:

$$z_t = y_t\beta + x_t$$

where:

- z_t is the original series;
- $\beta = (\beta_1, ..., \beta_n)$ is a vector of regression coefficients;
- $y_t = (y_{1t}, ..., y_{nt})$ are n regression variables (outliers, calendar effects, user-defined variables);
- x_t is a disturbance that follows the general ARIMA process: $\phi(B)\delta(B)x_t = \theta(B)a_t$; where $\phi(B), \delta(B)$ and $\theta(B)$ are finite polynomials in B and 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 is the number of regular AR terms (here and in 'JDemetra+', $p \leq 3$);
- bp is the number of seasonal AR terms (here and in 'JDemetra+', $bp \leq 1$);
- s is the number of observations per year (ie. The time series frequency).

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 is the number of regular MA terms (here and in 'JDemetra+', $q \leq 3$);

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

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

The available predefined 'JDemetra+' X13 and TRAMO-SEATS model specifications 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

The jregarima/jregarima_x13/jregarima_tramoseats functions return a [jSA](#) object that contains the result of the pre-adjustment method without any formatting. Therefore, the computation is faster than with the regarima/regarima_x13/regarima_tramoseats functions. The results of the seasonal adjustment can be extracted with the function [get_indicators](#).

The regarima/regarima_x13/regarima_tramoseats functions 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 spec object class.

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

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

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

arima.coefficients

     a matrix containing the estimated regular and seasonal AR and MA coefficients,
as well as the associated standard errors and t-statistics values. The estimated
coefficients can be also extracted with the function coef (whose output also
includes the regression coefficients).

regression.coefficients

     a matrix containing the estimated regression variables (i.e.: mean, calendar ef-
fect, outliers and user-defined regressors) coefficients, as well as the associated
standard errors and t-statistics values. The estimated coefficients can be also
extracted with the function coef (whose output also includes the arima coeffi-
cients).

loglik   a 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 also be
extracted with the function logLik.

model   a list containing 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 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 so, it provides the number of
regressors), Leap year effect, Easter effect and whether outliers were detected
(Outliers (if so, 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 hol-
idays 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   a list containing statistics on the RegARIMA residuals. It provides the residuals
standard error (st.error) and the results of normality, independence and linear-
ity of the residuals (tests) - object of class c("regarima_rtests","data.frame").

forecast   a ts matrix containing the forecast of the original series (fcst) and its standard
error (fcsterr).

References

More information and examples related to 'JDemetra+' features in the online documentation: https:
//jdemetra-new-documentation.netlify.app/

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 the name (character) of a predefined 'JDemetra+' model specification, 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   the model specification. It can be the name (`character`) of a predefined 'JDeme-
                     tra+' 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

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

                     The time span of the series, which is the (sub)period used to estimate the re-
                     garima model, is controlled by the following six variables: estimate.from,

estimate.to, estimate.first, estimate.last, estimate.exclFirst and estimate.exclLast; where estimate.from and estimate.to have priority over the 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. Default= "All".

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

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

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

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

estimate.exclFirst
                 numeric, the number of periods excluded at the beginning of the series. It can be combined with the parameter estimate.exclLast.

estimate.exclLast
                 numeric, the number of periods excluded at the end of the series. It can be combined with the parameter estimate.exclFirst.

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

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

transform.function
                 the transformation of the input series: "None" = no transformation of the series; "Log" = takes the log of the series; "Auto" = the program tests for the log-level specification.

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. Said pre-specified outliers are used in the model only when enabled (usrdef.outliersEnabled=TRUE) and when the outliers' type (usrdef.outliersType) and date (usrdef.outliersDate) are provided.

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

usrdef.outliersType
                 a 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

    a 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

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

    Control variables for the user-defined variables:

usrdef.varEnabled

    logical If TRUE, the program uses the user-defined variables.

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

usrdef.varType   a vector of character(s) defining the user-defined variables component type. Possible types are: "Undefined", "Series", "Trend", "Seasonal", "SeasonallyAdjusted", "Irregular", "Calendar". To use the user-defined calendar regressors, the type "Calendar" must be defined in conjunction with tradingdays.option = "UserDefined". Otherwise, the program will automatically set usrdef.varType = "Undefined".

usrdef.varCoef   a vector providing fixed coefficients for the user-defined variables. The coefficients can't be fixed if transform.function is set to "Auto" (i.e. if the series transformation needs to be pre-defined).

tradingdays.mauto

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

tradingdays.pftd

    numeric. The p-value used 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

    to choose 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 must be defined by the usrdef.var argument with usrdef.varType set to "Calendar" and usrdef.varEnabled = TRUE). "None" must also be chosen for the "day-of-week effects" correction (and tradingdays.stocktd must be modified accordingly).

tradingdays.leapyear

    logical. 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   acharacter that specifies the presence and the length of the Easter effect: "Unused" = the Easter effect is not considered; "Standard" = influences the period of n days strictly before Easter Sunday; "IncludeEaster" = influences the entire period (n) up to and including Easter Sunday; "IncludeEasterMonday" = influences the entire period (n) up to and including Easter Monday.

easter.julian   logical. 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   logical. 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

> logical. 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 is controlled by the following six variables: `outlier.from`, `outlier.to`, `outlier.first`, `outlier.last`, `outlier.exclFirst` and `outlier.exclLast`; where `outlier.from` and `outlier.to` have priority over the 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   a character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). It can be combined with `outlier.to`.

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

outlier.exclLast

numeric specifying the number of periods excluded at the end of the series. It can be combined with `outlier.exclFirst`.

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

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

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

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

outlier.usedefcv

logical. 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 entered critical value (`outlier.cv`).

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

outlier.eml   logical for the exact likelihood estimation method. It 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

logical. 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

logical. 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 another model.

automdl.cancel   numeric, the cancellation 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 the cancellation limit, the two roots are assumed equal and canceled out.

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

automdl.ub2   numeric, the second unit root limit. When one of the roots in the estimation of the ARIMA(1,0,1)(1,0,1) plus mean model, which is performed in the second step of the automatic model identification procedure, is larger than second unit

root limit in modulus, it is checked if there is a common factor in the corresponding AR and MA polynomials of the ARMA model that can be canceled (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, the arma limit. It is the threshold value for t-statistics of ARMA coefficients and the constant term used for the final test of model parsimony. If the highest order ARMA coefficient has a t-value smaller than this value in magnitude, the order of the model is reduced. If the constant term has a t-value smaller than the ARMA limit in magnitude, it is removed from the set of regressors.

automdl.reducecv

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

automdl.ljungboxlimit

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

automdl.compare

logical. If TRUE, the program compares the model identified by the automatic procedure to the default model (ARIMA(0,1,1)(0,1,1)) and the model with the best fit is selected. Criteria considered are residual diagnostics, the model structure and the number of outliers.

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

arima.mu   logical. 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. The 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. The seasonal differencing order.

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

Control variables for the user-defined ARMA coefficients. Such 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

logical. If TRUE, the program uses the user-defined ARMA coefficients.

arima.coef   a vector providing the coefficients for the regular and seasonal AR and MA
                   polynomials. The length of the vector must be equal to the sum of the regular
                   and seasonal AR and MA orders. The coefficients shall be provided in the fol-
                   lowing 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   avector defining the ARMA coefficients estimation procedure. Possible proce-
                   dures 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, the forecasting horizon. The 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 following components, each refer-
ring to a different part of the RegARIMA model specification, mirroring the arguments of the func-
tion (for details, see the arguments description). Each lowest-level component (except the 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 = the 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 = the 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   a data frame containing Variables referring to: `span` - time span to be used for
                   the 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   a data frame containing variables referring to: tfunction - argument transform.function,
                   fct - argument transform.fct. The final values can be also accessed with the
                   function s_transform.

regression   a list containing 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 coeffi-
                   cients are used (outlier.coef and variables.coef). The final values can be
                   also accessed with the function s_usrdef; outliers - matrixes with the out-
                   liers (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 in-
                   formation on the variable type and values of fixed coefficients. The final user-
                   defined variables can be also accessed with the function s_preVar.

                   The trading.days data frame 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. The easter data frame 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   a 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   a list containing a data frame with the ARIMA settings (specification) and
                   matrices giving information on the pre-specified ARMA coefficients (coefficients).
                   The matrix Predefined refers to the pre-defined model specification and ma-
                   trix Final, to the final specification. Both matrices contain the values of the
                   ARMA coefficients and the procedure for its estimation. In the data frame
                   specification, the variable enabled refers to the argument automdl.enabled
                   and all remaining variables (automdl.acceptdefault, automdl.cancel, automdl.ub1,
                   automdl.ub2, automdl.armalimit, automdl.reducecv, automdl.ljungboxlimit,
                   automdl.compare, arima.mu, arima.p, arima.d, arima.q, arima.bp, arima.bd, arima.bq),
                   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   a data frame with the forecasting horizon (argument fcst.horizon). The final
                   value can be also accessed with the function s_fcst.

span   a matrix containing the final time span for the model estimation and outliers' de-
                   tection. It 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

More information and examples related to 'JDemetra+' features in the online documentation: `https://jdemetra-new-documentation.netlify.app/`

Examples

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

 # To 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)

 # To modify the model specification of 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)

 # To modify the model specification of 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: the 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   the model specification. It can be the name (`character`) of a pre-defined 'JDeme-
                              tra+' model specification (see *Details*), an object of class `c("regarima_spec", "X13")`
                              or an object of class `c("regarima", "X13")`. The default value is `"RG5c"`.

preliminary.check

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

                              The time span of the series, which is the (sub)period used to estimate the re-
                              garima model, is controlled by the following six variables: `estimate.from`,
                              `estimate.to`, `estimate.first`, `estimate.last`, `estimate.exclFirst` and
                              `estimate.exclLast`; where `estimate.from` and `estimate.to` have priority
                              over the 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`. Default= "All".

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

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

estimate.first   a 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

                              a numeric specifying the number of periods excluded at the beginning of the
                              series. It can be combined with the parameter `estimate.exclLast`.

estimate.exclLast

                              a numeric specifying the number of periods excluded at the end of the series. It
                              can be combined with the parameter `estimate.exclFirst`.

estimate.tol   a numeric, convergence tolerance. The absolute changes in the log-likelihood
                              function are compared to this value to check for the convergence of the estima-
                              tion iterations.

transform.function

                              the transformation of the input series: `"None"` = no transformation of the series;
                              `"Log"` = takes the log of the series; `"Auto"` = the program tests for the log-level
                              specification.

transform.adjust

pre-adjustment of the input series for the length of period or leap year effects: "None" = no adjustment; "LeapYear" = leap year effect; "LengthOfPeriod" = length of period. Modifications of this variable are taken into account only when transform.function is set to "Log".

transform.aicdiff

a 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 when enabled (usrdef.outliersEnabled=TRUE) and the outlier type (usrdef.outliersType) and date (usrdef.outliersDate) are provided.

usrdef.outliersEnabled

logical. If TRUE, the program uses the pre-specified outliers.

usrdef.outliersType

a vector defining the outlier 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

a vector defining the outlier dates. 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

a vector providing fixed coefficients for the outliers. The coefficients can't be fixed if transform.function is set to "Auto" i.e. 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

a logical. If TRUE, the program uses the user-defined variables.

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

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

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

tradingdays.option

to specify the set of 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 must be defined by the usrdef.var argument with usrdef.varType set to "Calendar" and usrdef.varEnabled = TRUE). "None" must also be specified for the "day-of-week effects" correction (tradingdays.stocktd to be modified accordingly).

tradingdays.autoadjust

> a logical. 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

> a `character` to specify whether or not to include 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 hasn't been pre-adjusted (`transform.adjust` set to `"None"`) and if the automatic correction for the leap-year effect isn't selected (`tradingdays.autoadjust` set to FALSE).

tradingdays.stocktd

> a numeric indicating the day of the month when inventories and other stock are reported (to denote the last day of the month, set the variable to 31). 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 a logical. If TRUE, the program considers the Easter effect in the model.

easter.julian a logical. If TRUE, the program uses the Julian Easter (expressed in Gregorian calendar).

easter.duration

> a 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 (the Easter effect is considered as significant if the t-statistic is greater than 1.96): `"Add"` = the Easter effect variable is not included in the initial regression model but can be added to the RegARIMA model after the test; `"Remove"` = the Easter effect variable belongs to the initial regression model but can be removed from the RegARIMA model after the test; `"None"` = the Easter effect variable is not pre-tested and is included in the model.

outlier.enabled

> a logical. If TRUE, the automatic detection of outliers is enabled in the defined time span.

> The time span during which outliers will be searched is controlled by the following six variables: `outlier.from`, `outlier.to`, `outlier.first`, `outlier.last`, `outlier.exclFirst` and `outlier.exclLast`; where `outlier.from` and `outlier.to` have priority over the 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 a character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). It can be combined with the parameter `outlier.to`.

outlier.to	a character in format "YYYY-MM-DD" indicating the end of the time span (e.g. "2020-12-31"). it can be combined with the parameter `outlier.from`.
outlier.first	a numeric specifying the number of periods considered at the beginning of the series.
outlier.last	a numeric specifying the number of periods considered at the end of the series.

outlier.exclFirst

a numeric specifying the number of periods excluded at the beginning of the series. It can be combined with the parameter `outlier.exclLast`.

outlier.exclLast

a numeric specifying the number of periods excluded at the end of the series. It can be combined with the parameter `outlier.exclFirst`.

outlier.ao	a logical. If TRUE, the automatic detection of additive outliers is enabled (`outlier.enabled` must be also set to TRUE).
outlier.tc	a logical. If TRUE, the automatic detection of transitory changes is enabled (`outlier.enabled` must be also set to TRUE).
outlier.ls	a logical. If TRUE, the automatic detection of level shifts is enabled (`outlier.enabled` must be also set to TRUE).
outlier.so	a logical. If TRUE, the automatic detection of seasonal outliers is enabled (`outlier.enabled` must be also set to TRUE).

outlier.usedefcv

a logical. If TRUE, the critical value for the outlier detection procedure is automatically determined by the number of observations in the outlier detection time span. If FALSE, the procedure uses the entered critical value (`outlier.cv`).

outlier.cv	a numeric. The entered critical value for the outlier detection procedure. The modification of this variable is only taken into account 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	a numeric. The rate of decay for the transitory change outlier.

automdl.enabled

a logical. 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 (when `automdl.enabled` is set to TRUE):

automdl.acceptdefault

a logical. 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 another model.

automdl.cancel	the cancellation limit (numeric). 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 the cancellation limit, the two roots are assumed equal and cancel out.

automdl.ub1   the first unit root limit (`numeric`). 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 the first unit
                     root limit in modulus, it is set equal to unity.

automdl.ub2   the second unit root limit (`numeric`). 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 corre-
                     sponding AR and MA polynomials of the ARMA model that can be canceled
                     (see `automdl.cancel`). If there is no cancellation, the AR root is set equal to
                     unity (i.e. the differencing order changes).

automdl.mixed   a logical. This variable controls whether ARIMA models with non-seasonal AR
                     and MA terms or seasonal AR and MA terms will be considered in the auto-
                     matic 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, pro-
                     vided there are no AR or MA terms in either the seasonal or non-seasonal terms.

automdl.balanced
                     a logical. If `TRUE`, the automatic model identification procedure will have a pref-
                     erence 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
                     the ARMA limit (`numeric`). 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 smaller than this value in magni-
                     tude, the order of the model is reduced. If the constant term t-value is smaller
                     than the 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)*CV, where CV is the original
                     critical value.

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

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 smaller than the final unit
                     root limit, then a unit root is assumed, the order of the AR polynomial is reduced
                     by one and the appropriate order of the differencing (non-seasonal, seasonal) is
                     increased. The parameter value should be greater than one.

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

arima.mu	logical. 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. The 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. The 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

logical. If TRUE, the program uses the user-defined ARMA coefficients.

arima.coef   a vector providing the coefficients for the regular and seasonal AR and MA polynomials. The vector length must be equal to the sum of the regular and seasonal AR and MA orders. The coefficients shall be provided in the following order: regular AR (*Phi*; p elements), regular MA (*Theta*; q elements), seasonal AR (*BPhi*; bp elements) and seasonal MA (*BTheta*; bq elements). E.g.: arima.coef=c(0.6,0.7) with arima.p=1, arima.q=0,arima.bp=1 and arima.bq=0.

arima.coefType a vector defining the ARMA coefficients estimation procedure. Possible procedures are: "Undefined" = no use of any user-defined input (i.e. coefficients are estimated), "Fixed" = the coefficients are fixed at the value provided by the user, "Initial" = the value defined by the user is used as the initial condition. 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 the forecasting horizon (numeric). The forecast length generated by the RegARIMA model in periods (positive values) or years (negative values). By default, the program generates a two-year forecast (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 following components, each referring to a
different part of the RegARIMA model specification, mirroring the arguments of the function (for
details, see the arguments description). Each 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 modifi-
cations 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
of Predefined (base model specification) and Final values.

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

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

regression   a list containing 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 coeffi-
                cients are used (outlier.coef and variables.coef). The final values can
                also be accessed with the function [s_usrdef](); outliers - matrices with the out-
                liers (Predefined and Final). The final outliers can also be accessed with the
                function [s_preOut](); and variables - a list with the Predefined and Final
                user-defined variables (series) and its description (description) including
                the information on the variable type and the values of fixed coefficients. The
                final user-defined variables can also be accessed with the function [s_preVar]().
                Within the data frame trading.days, the 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 re-
                fer 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   a data frame. Variables referring to: enabled - argument outlier.enabled,
                span - time span for the outlier detection, ao - argument outlier.ao, tc - ar-
                gument 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 also be accessed with the function [s_out]().

arima   a list of a data frame with the ARIMA settings (specification) and matrices
                with the information on the pre-specified ARMA coefficients (coefficients).
                The matrix Predefined refers to the pre-defined model specification, and the

matrix Final to the final specification. Both matrices contain the value of the ARMA coefficients and the procedure for its estimation. In the data frame specification, the variable enabled refers to the argument automdl.enabled and all remaining variables (automdl.acceptdefault, automdl.cancel, automdl.ub1, automdl.ub2, automdl.mixed, automdl.balanced, automdl.armalimit, automdl.reducecv, automdl.ljungboxlimit, automdl.ubfinal, arima.mu, arima.p, arima.d, arima.q, arima.bp, arima.bd, arima.bq), to the respective function arguments. The final values of the specification can be also accessed with the function s_arima and the final pre-specified ARMA coefficients, with the function s_arimaCoef.

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

span   a matrix containing the final time span for the model estimation and outlier detection. It 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

More information and examples related to 'JDemetra+' features in the online documentation: https://jdemetra-new-documentation.netlify.app/

Examples

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

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

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

 # To modify the model specification of 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   an object of one of the following 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   the (path and) name of the file where the model specification will be/has 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

More information and examples related to 'JDemetra+' features in the online documentation: https:
//jdemetra-new-documentation.netlify.app/

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()

 # To save the model specification of a c("regarima_spec","X13") class object
save_spec(myspec2, file.path(dir, "specx13.RData"))
 # To save the model specification of a c("regarima","X13") class object
save_spec(myreg2, file.path(dir,"regx13.RData"))
 # To save the model specification of a c("regarima_spec","TRAMO_SEATS") class object
save_spec(myspec4, file.path(dir,"specTS.RData"))
 # To save the model specification of a c("regarima","TRAMO_SEATS") class object
save_spec(myreg4, file.path(dir,"regTS.RData"))
 # To save the model of a c("SA_spec","X13") class object
save_spec(myspec6, file.path(dir,"specFullx13.RData"))
```

```
 # To save the model of a c("SA","X13") class object
save_spec(mysa6, file.path(dir,"sax13.RData"))
 # To save the model of a c("SA_spec","TRAMO_SEATS") class object
save_spec(myspec7, file.path(dir,"specFullTS.RData"))
 # To save the model of a c("SA","TRAMO_SEATS") class object
save_spec(mysa7, file.path(dir,"saTS.RData"))

 # To load a 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"))

# To use the re-loaded specifications and models:
regarima(myseries, myspec2a)
x13(myseries, myspec6a)
tramoseats(myseries, myspec7a)

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

---

save_workspace   *Save a workspace*

Description

Function to save a `workspace` object into a 'JDemetra+' workspace.

Usage

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

Arguments

workspace	the workspace object to export
file	the path where to export the 'JDemetra+' workspace (.xml file). By default, if not specified, a dialog box opens.

Value

A boolean indicating whether the export is successful.

See Also

[load_workspace](#)

Examples

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

---

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

Description

The following functions enable the access to 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   an object of one of the following 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* variables
- 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 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* variable
- 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

More information and examples related to 'JDemetra+' features in the online documentation: https://jdemetra-new-documentation.netlify.app/

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

Functions 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), the seasonal component (s)
and the irregular component (i). Calendar-related movements can be corrected in the pre-treatment
(TRAMO) step. 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	an univariate time series
spec	a TRAMO-SEATS model specification. It can be the name (character) of a pre-defined TRAMO-SEATS 'JDemetra+' model specification (see *Details*), or an object of class c("SA_spec","TRAMO_SEATS"). The default value is "RSAfull".
userdefined	a character vector containing the additional output variables (see user_defined_variables).

Details

The first step of a seasonal adjustment consists in pre-adjusting the time series with TRAMO. This is done by removing its deterministic effects (calendar and outliers), using a regression model with ARIMA noise (RegARIMA, see: regarima). In the second part, the pre-adjusted series is decomposed by the SEATS algorithm 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 ($y = t * s * i$, in the latter case pre-adjustment and decomposition are performed on $(log(y) = log(t) + log(s) + log(i))$.

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 this method at https://jdemetra-new-documentation.netlify.app/m-seats-decomposition.

The available predefined 'JDemetra+' TRAMO-SEATS 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

jtramoseats returns a jSA object that contains the results of the seasonal adjustment without any formatting. Therefore, the computation is faster than with the function tramoseats. The results of the seasonal adjustment can be extracted with the function get_indicators.

tramoseats returns an object of class c("SA","TRAMO_SEATS"), that is, a list containing :

regarima   an object of class c("regarima","TRAMO_SEATS"). More info in the *Value* section of the function regarima.

decomposition   an object of class "decomposition_SEATS", that is a five-element list:

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

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

diagnostics   an object of class "diagnostics", that is a list containing three types of tests
                         results:

                              • variance_decomposition a data.frame with the tests results on the rela-
                                tive 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 a data.frame with the tests results of the presence of sea-
                                sonality in the residuals (including the statistic test values, the correspond-
                                ing p-values and the parameters description);

                              • combined_test the combined tests for stable seasonality in the entire se-
                                ries. The format is a two-element list with: tests_for_stable_seasonality,
                                a data.frame containing the tests results (including the statistic test value, its
                                p-value and the parameters description), and combined_seasonality_test,
                                the summary.

user_defined   an object of class "user_defined": a list containing the additional userdefined
                         variables.

References

More information and examples related to 'JDemetra+' features in the online documentation: [https:
//jdemetra-new-documentation.netlify.app/](https://jdemetra-new-documentation.netlify.app/)

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

```
#Example 1
myseries <- ipi_c_eu[, "FR"]
myspec <- tramoseats_spec("RSAfull")
mysa <- tramoseats(myseries, myspec)
mysa

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

#Example 2
```

```
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*

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 'JDeme-
tra+' 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
a TRAMO-SEATS model specification. It can be the 'JDemetra+' name (character) of a predefined TRAMO-SEATS 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
a logical to check the quality of the input series and exclude highly problematic series e.g. the series with a number of identical observations and/or missing values above pre-specified threshold values.

The time span of the series, which is the (sub)period used to estimate the regarima model, is controlled by the following six variables: estimate.from, estimate.to, estimate.first, estimate.last, estimate.exclFirst and estimate.exclLast; where estimate.from and estimate.to have priority over the 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. Default= "All".

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

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

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

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

estimate.exclFirst
numeric, the number of periods excluded at the beginning of the series. It can be combined with the parameter estimate.exclLast.

estimate.exclLast
numeric, the number of periods excluded at the end of the series. It can be combined with the parameter estimate.exclFirst.

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

estimate.urfinal

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

transform.function

> the transformation of the input series: "None" = no transformation of the series; "Log" = takes the log of the series; "Auto" = the program tests for the log-level specification.

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. Said pre-specified outliers are used in the model only when enabled (usrdef.outliersEnabled=TRUE) and when the outliers' type (usrdef.outliersType) and date (usrdef.outliersDate) are provided.

usrdef.outliersEnabled

> logical. If TRUE, the program uses the pre-specified outliers.

usrdef.outliersType

> a 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

> a 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

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

> Control variables for the user-defined variables:

usrdef.varEnabled

> logical If TRUE, the program uses the user-defined variables.

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

usrdef.varType a vector of character(s) defining the user-defined variables component type. Possible types are: "Undefined", "Series", "Trend", "Seasonal", "SeasonallyAdjusted", "Irregular", "Calendar". To use the user-defined calendar regressors, the type "Calendar" must be defined in conjunction with tradingdays.option = "UserDefined". Otherwise, the program will automatically set usrdef.varType = "Undefined".

usrdef.varCoef a vector providing fixed coefficients for the user-defined variables. The coefficients can't be fixed if transform.function is set to "Auto" (i.e. if the series transformation needs to be pre-defined).

tradingdays.mauto

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

tradingdays.pftd

> numeric. The p-value used in the test specified by the automatic parameter
> (tradingdays.mauto) to assess the significance of the pre-tested calendar ef-
> fects 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

> to choose 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 (regres-
> sors must be defined by the usrdef.var argument with usrdef.varType set
> to ″Calendar″ and usrdef.varEnabled = TRUE). ″None″ must also be chosen
> for the "day-of-week effects" correction (and tradingdays.stocktd must be
> modified accordingly).

tradingdays.leapyear

> logical. 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). Modifica-
> tions 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   acharacter that specifies the presence and the length of the Easter effect: ″Unused″
> = the Easter effect is not considered; ″Standard″ = influences the period of n
> days strictly before Easter Sunday; ″IncludeEaster″ = influences the entire
> period (n) up to and including Easter Sunday; ″IncludeEasterMonday″ = in-
> fluences the entire period (n) up to and including Easter Monday.

easter.julian   logical. 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   logical. 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

                 logical. 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 is controlled by the follow-
                 ing six variables: outlier.from, outlier.to, outlier.first, outlier.last,
                 outlier.exclFirst and outlier.exclLast; where outlier.from and outlier.to
                 have priority over the 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   a character in format "YYYY-MM-DD" indicating the start of the time span
                 (e.g. "1900-01-01"). It can be combined with outlier.to.

outlier.to   a character in format "YYYY-MM-DD" indicating the end of the time span (e.g.
                 "2020-12-31"). It 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 se-
                 ries. It can be combined with outlier.exclLast.

outlier.exclLast

                 numeric specifying the number of periods excluded at the end of the series. It
                 can be combined with outlier.exclFirst.

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

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

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

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

outlier.usedefcv

                 logical. If TRUE, the critical value for the outliers' detection procedure is auto-
                 matically determined by the number of observations in the outlier detection time
                 span. If FALSE, the procedure uses the entered critical value (outlier.cv).

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

outlier.eml   logical for the exact likelihood estimation method. It controls the method
                 applied for a parameter estimation in the intermediate steps of the automatic de-
                 tection 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

> logical. 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

> logical. 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 another model.

automdl.cancel numeric, the cancellation 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 the cancellation limit, the two roots are assumed equal and canceled out.

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

automdl.ub2 numeric, the second unit root limit. When one of the roots in the estimation of the ARIMA(1,0,1)(1,0,1) plus mean model, which is performed in the second step of the automatic model identification procedure, is larger than second unit root limit in modulus, it is checked if there is a common factor in the corresponding AR and MA polynomials of the ARMA model that can be canceled (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, the arma limit. It is the threshold value for t-statistics of ARMA coefficients and the constant term used for the final test of model parsimony. If the highest order ARMA coefficient has a t-value smaller than this value in magnitude, the order of the model is reduced. If the constant term has a t-value smaller than the ARMA limit in magnitude, it is removed from the set of regressors.

automdl.reducecv

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

automdl.ljungboxlimit

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

automdl.compare

> logical. If TRUE, the program compares the model identified by the automatic procedure to the default model (ARIMA(0,1,1)(0,1,1)) and the model with the

best fit is selected. Criteria considered are residual diagnostics, the model structure and the number of outliers.

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

arima.mu   `logical`. 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`. The 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`. The seasonal differencing order.

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

Control variables for the user-defined ARMA coefficients. Such 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

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

arima.coef   a vector providing the coefficients for the regular and seasonal AR and MA polynomials. The length of the vector must be equal to the sum of the regular and seasonal AR and MA orders. The coefficients shall be provided in the following order: regular AR (*Phi* - p elements), regular MA (*Theta* - q elements), seasonal AR (*BPhi* - bp elements) and seasonal MA (*BTheta* - bq elements). E.g.: `arima.coef=c(0.6,0.7)` with `arima.p=1`, `arima.q=0`,`arima.bp=1` and `arima.bq=0`.

arima.coefType   avector defining the 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, the forecasting horizon. The 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: the number of forecasts used in the decomposition. Negative values correspond to number of years. Default=-1.

seats.approx   character: the 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. Default="Legacy".

seats.trendBoundary

> numeric: the trend boundary. The boundary beyond which an AR root is integrated in the trend component. If the modulus of the inverse real root is greater than the trend boundary, the AR root is integrated in the trend component. Below this value, the root is integrated in the transitory component. Possible values [0,1]. Default=0.5.

seats.seasdBoundary

> numeric: the seasonal boundary. The boundary beyond which a negative AR root is integrated in the seasonal component. If the modulus of the inverse negative real root is greater (or equal) than Seasonal boundary, the AR root is integrated into the seasonal component. Otherwise the root is integrated into the trend or transitory component. Possible values [0,1]. Default=0.8.

seats.seasdBoundary1

> numeric: the seasonal boundary (unique). The boundary beyond which a negative AR root is integrated in the seasonal component, when the root is the unique seasonal root. If the modulus of the inverse negative real root is greater (or equal) than Seasonal boundary, the AR root is integrated into the seasonal component. Otherwise the root is integrated into the trend or transitory component. Possible values [0,1]. Default=0.8.

seats.seasTol numeric: the 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 the 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). Possible values in [0,10]. Default value 2.

seats.maBoundary

> numeric: the MA unit root boundary. When the modulus of an estimated MA root falls in the range (xl, 1), it is set to xl. Possible values [0.9,1]. Default=0.95.

seats.method character: the estimation method for the unobserved components. The choice can be made from:

> - "Burman": the default value. May result in a significant underestimation of the components' standard deviation, 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": it 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-element list of class c("SA_spec", "TRAMO_SEATS"), containing: (1) an object of class c("regarima_spec", "TRAMO_SEATS") with the RegARIMA model specification, (2) an object of class c("seats_spec", "data.frame") with the SEATS algorithm specification. Each component refers to a different part of the SA model specification, mirroring the arguments of the function (for details see the function arguments in the description). Each lowest-level component (except span, pre-specified outliers, user-defined variables and pre-specified ARMA coefficients) is structured as a data frame with columns denoting different variables of the model specification and rows referring to:

- first row: the 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: the 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 of Predefined (base model specification) and Final values.

- regarima: an object of class c("regarima_spec", "TRAMO_SEATS"). See *Value* of the function regarima_spec_tramoseats.
- seats: a 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 also be accessed with the function s_seats.

References

More information and examples related to 'JDemetra+' features in the online documentation: https://jdemetra-new-documentation.netlify.app/

See Also

tramoseats

Examples

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

# To 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)

# To modify the model specification of 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)

# To modify the model specification of 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

> *Display a list of all the available output objects (series, parameters,*
> *diagnostics)*

Description

Function generating a comprehensive list of available output variables (series, parameters, diagnostics) from the estimation process with `x13` and `tramoseats`. Some items are available in the default estimation output but the remainder can be added using the `userdefined` parameter.

Usage

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

Arguments

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

Value

a vector containing the names of all the available output objects (series, diagnostics, parameters)

References

More information and examples related to 'JDemetra+' features in the online documentation: https://jdemetra-new-documentation.netlify.app/

Examples

```
y<- ipi_c_eu[, "FR"]
user_defined_variables("X13-ARIMA")
m <- x13(y,"RSA5c", userdefined=c("b20","ycal","residuals.kurtosis" ))
m$user_defined$b20
m$user_defined$ycal
m$user_defined$residuals.kurtosis
user_defined_variables("TRAMO-SEATS")
m <- tramoseats(y,"RSAfull", userdefined=c("ycal","variancedecomposition.seasonality"))
m$user_defined$ycal
m$user_defined$variancedecomposition.seasonality
```

---

x13 *Seasonal Adjustment with X13-ARIMA*

Description

Functions to estimate the seasonally adjusted series (sa) with the X13-ARIMA method. This is achieved by decomposing the time series (y) into the trend-cycle (t), the seasonal component (s) and the irregular component (i). Calendar-related movements can be corrected in the pre-treatment (regarima) step. x13 returns a preformatted result while jx13 returns the Java objects resulting from 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   an univariate time series

spec   the x13 model specification. It can be the name (character) of a pre-defined X13 'JDemetra+' model specification (see *Details*) or of a specification created with the x13_spec function. The default value is "RSA5c".

userdefined   a character vector containing the additional output variables (see user_defined_variables).

Details

The first step of a seasonal adjustment consists in pre-adjusting the time series. This is done by removing its deterministic effects (calendar and outliers), using a regression model with ARIMA noise (RegARIMA, see: regarima). In the second part, the pre-adjusted series is decomposed by the X11 algorithm 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 ($y = t * s * i$). More information on the X11 algorithm at https://jdemetra-new-documentation.netlify.app/m-x11-decomposition.

The available pre-defined 'JDemetra+' X13 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

jx13 returns the result of the seasonal adjustment in a Java ([jSA](#)) object, without any formatting. Therefore, the computation is faster than with the x13 function. The results of the seasonal adjustment can be extracted with the function [get_indicators](#).

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

regarima   an object of class c("regarima","X13"). More info in the *Value* section of the function [regarima](#).

decomposition   an object of class "decomposition_X11", that is a six-element list:

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

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

diagnostics   an object of class "diagnostics", that is a list containing three types of tests results:

- variance_decomposition a data.frame with the tests results 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 a data.frame with the tests results of the presence of seasonality in the residuals (including the statistic test values, the corresponding p-values and the parameters description);
- combined_test the combined tests for stable seasonality in the entire series. The format is a two elements list with: tests_for_stable_seasonality, a data.frame containing the tests results (including the statistic test value, its p-value and the parameters description), and combined_seasonality_test, the summary.

user_defined   an object of class "user_defined": a list containing the additional userdefined variables.

References

More information and examples related to 'JDemetra+' features in the online documentation: https: //jdemetra-new-documentation.netlify.app/

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 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   an x13 model specification. It can be the 'JDemetra+' name (character) of
                   a 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

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

                   The time span of the series, which is the (sub)period used to estimate the re-
                   garima model, is controlled by the following six variables: estimate.from,
                   estimate.to, estimate.first, estimate.last, estimate.exclFirst and
                   estimate.exclLast; where estimate.from and estimate.to have priority

over the 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. Default= "All".

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

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

estimate.first   a 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

                   a numeric specifying the number of periods excluded at the beginning of the
                   series. It can be combined with the parameter estimate.exclLast.

estimate.exclLast

                   a numeric specifying the number of periods excluded at the end of the series. It
                   can be combined with the parameter estimate.exclFirst.

estimate.tol   a numeric, convergence tolerance. The absolute changes in the log-likelihood
                   function are compared to this value to check for the convergence of the estima-
                   tion iterations.

transform.function

                   the transformation of the input series: "None" = no transformation of the series;
                   "Log" = takes the log of the series; "Auto" = the program tests for the log-level
                   specification.

transform.adjust

                   pre-adjustment of the input series for the length of period or leap year effects:
                   "None" = no adjustment; "LeapYear" = leap year effect; "LengthOfPeriod" =
                   length of period. Modifications of this variable are taken into account only when
                   transform.function is set to "Log".

transform.aicdiff

                   a 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 when enabled (usrdef.outliersEnabled=TRUE) and
                   the outlier type (usrdef.outliersType) and date (usrdef.outliersDate) are
                   provided.

usrdef.outliersEnabled

                   logical. If TRUE, the program uses the pre-specified outliers.

usrdef.outliersType

                   a vector defining the outlier 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

                   a vector defining the outlier dates. 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

a vector providing fixed coefficients for the outliers. The coefficients can't be fixed if `transform.function` is set to "Auto" i.e. 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

a logical. If `TRUE`, the program uses the user-defined variables.

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

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

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

tradingdays.option

to specify the set of 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 must be defined by the `usrdef.var` argument with `usrdef.varType` set to "Calendar" and `usrdef.varEnabled` = TRUE). "None" must also be specified for the "day-of-week effects" correction (`tradingdays.stocktd` to be modified accordingly).

tradingdays.autoadjust

a logical. 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

a `character` to specify whether or not to include 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 hasn't been pre-adjusted (`transform.adjust` set to "None") and if the automatic correction for the leap-year effect isn't selected (`tradingdays.autoadjust` set to `FALSE`).

tradingdays.stocktd

a numeric indicating the day of the month when inventories and other stock are reported (to denote the last day of the month, set the variable to 31). 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   a logical. If TRUE, the program considers the Easter effect in the model.

easter.julian   a logical. If TRUE, the program uses the Julian Easter (expressed in Gregorian calendar).

easter.duration

a 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 (the Easter effect is considered as significant if the t-statistic is greater than 1.96): ″Add″ = the Easter effect variable is not included in the initial regression model but can be added to the RegARIMA model after the test; ″Remove″ = the Easter effect variable belongs to the initial regression model but can be removed from the RegARIMA model after the test; ″None″ = the Easter effect variable is not pre-tested and is included in the model.

outlier.enabled

a logical. If TRUE, the automatic detection of outliers is enabled in the defined time span.

The time span during which outliers will be searched is controlled by the following six variables: outlier.from, outlier.to, outlier.first, outlier.last, outlier.exclFirst and outlier.exclLast; where outlier.from and outlier.to have priority over the 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   a character in format "YYYY-MM-DD" indicating the start of the time span (e.g. "1900-01-01"). It can be combined with the parameter outlier.to.

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

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

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

outlier.exclFirst

a numeric specifying the number of periods excluded at the beginning of the series. It can be combined with the parameter outlier.exclLast.

outlier.exclLast

a numeric specifying the number of periods excluded at the end of the series. It can be combined with the parameter outlier.exclFirst.

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

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

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

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

outlier.usedefcv

a logical. If TRUE, the critical value for the outlier detection procedure is automatically determined by the number of observations in the outlier detection time span. If FALSE, the procedure uses the entered critical value (outlier.cv).

outlier.cv   a numeric. The entered critical value for the outlier detection procedure. The modification of this variable is only taken into account 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   a numeric. The rate of decay for the transitory change outlier.

automdl.enabled

a logical. 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 (when automdl.enabled is set to TRUE):

automdl.acceptdefault

a logical. 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 another model.

automdl.cancel   the cancellation limit (numeric). 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 the cancellation limit, the two roots are assumed equal and cancel out.

automdl.ub1   the first unit root limit (numeric). 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 the first unit root limit in modulus, it is set equal to unity.

automdl.ub2   the second unit root limit (numeric). 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 canceled (see automdl.cancel). If there is no cancellation, the AR root is set equal to unity (i.e. the differencing order changes).

automdl.mixed   a logical. This variable controls whether ARIMA models with non-seasonal AR and MA terms or seasonal AR and MA terms will be considered in the automatic model identification procedure. If FALSE, a model with AR and MA terms in both the seasonal and non-seasonal parts of the model can be acceptable, provided there are no AR or MA terms in either the seasonal or non-seasonal terms.

automdl.balanced

a logical. If TRUE, the automatic model identification procedure will have a preference for balanced models (i.e. models for which the order of the combined AR and differencing operator is equal to the order of the combined MA operator).

automdl.armalimit

the ARMA limit (`numeric`). 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 smaller than this value in magnitude, the order of the model is reduced. If the constant term t-value is smaller than the 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)*CV, where CV is the original critical value.

automdl.ljungboxlimit

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

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 smaller than the final unit root limit, then a unit root is assumed, the order of the AR polynomial is reduced by one and the appropriate order of the differencing (non-seasonal, seasonal) is increased. The parameter value should be greater than one.

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

arima.mu   logical. 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. The 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. The 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

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

arima.coef   a vector providing the coefficients for the regular and seasonal AR and MA polynomials. The vector length must be equal to the sum of the regular and seasonal AR and MA orders. The coefficients shall be provided in the following order: regular AR (*Phi*; p elements), regular MA (*Theta*; q elements),

	seasonal AR (*BPhi*; bp elements) and seasonal MA (*BTheta*; bq elements). E.g.: `arima.coef=c(0.6,0.7)` with `arima.p=1`, `arima.q=0,arima.bp=1` and `arima.bq=0`.
arima.coefType	a vector defining the ARMA coefficients estimation procedure. Possible procedures are: "Undefined" = no use of any user-defined input (i.e. coefficients are estimated), "Fixed" = the coefficients are fixed at the value provided by the user, "Initial" = the value defined by the user is used as the initial condition. 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	the forecasting horizon (numeric). The forecast length generated by the RegARIMA model in periods (positive values) or years (negative values). By default, the program generates a two-year forecast (`fcst.horizon` set to `-2`).
x11.mode	character: the 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	
	logical: 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: the lower sigma boundary for the detection of extreme values, > 0.5, default=1.5.
x11.usigma	numeric: the upper sigma boundary for the detection of extreme values, > lsigma, default=2.5.
x11.trendAuto	logical: automatic Henderson filter. If TRUE, an automatic selection of the Henderson filter's length for the trend estimation is enabled.
x11.trendma	numeric: the 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	a 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 - the same seasonal filter is used for all periods (e.g.: 'seasonal.filter = "Msr"' or 'seasonal.filter = "S3X3"'); or have a different value for each quarter (length 4) or each month (length 12) - (e.g. for quarterly series: 'seasonal.filter = 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: the number of forecasts generated by the RegARIMA model in periods (positive values) or years (negative values).Default value: fcasts=-1.

x11.bcasts   numeric: the number of backcasts used in X11. Negative figures are translated
                 in years of backcasts. Default value: bcasts=0.

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 het-
                 eroskedastic by calendar month/quarter ("Signif"); separately for two com-
                 plementary sets of calendar months/quarters specified by the x11.sigmaVector
                 parameter ("Select", see parameter x11.sigmaVector).

x11.sigmaVector

                 a vector to specify one of the two groups of periods for whose standard er-
                 rors used for extreme values detection and adjustment will be computed. Only
                 used if x11.calendarSigma = "Select". Possible values are: "Group1" and
                 "Group2".

x11.excludeFcasts

                 logical: to exclude forecasts and backcasts. If TRUE, the RegARIMA model
                 forecasts and backcasts are not used during the detection 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-element list of class c("SA_spec", "X13"), containing: (1) an object of class c("regarima_spec",
"X13") with the RegARIMA model specification; (2) an object of class c("X11_spec", "data.frame")
with the X11 algorithm specification. Each component refers to different parts of the SA model
specification, mirroring the arguments of the function (for details, see the function arguments in the
description). Each lowest-level component (except span, pre-specified outliers, user-defined vari-
ables and pre-specified ARMA coefficients) is structured as a data frame with columns denoting
different variables of the model specification and rows referring to:

  • first row: the 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: the 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 of `Predefined` (base model specification) and `Final` values.

- regarima: an object of class c("regarima_spec", "x13"). See *Value* of the function `regarima_spec_x13`.
- x11: a 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

More information and examples related to 'JDemetra+' features in the online documentation: `https://jdemetra-new-documentation.netlify.app/` 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)

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

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

# To modify the model specification of 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

# To define a seasonal filter
 myspec1 <- x13_spec("RSA5c", x11.seasonalma = rep("S3X1", 12))
 mysa1 <- x13(myseries, myspec1)
```

Index