Package: rjd3revisions (via r-universe)

October 23, 2024

Type Package

Title Revision analysis with 'JDemetra+ 3.x'

Version 1.4.0.9000

Description Revision analysis tool part of 'JDemetra+ 3.x' (<https://github.com/jdemetra>) time series analysis software. It performs a battery of tests on revisions and submit a report with the results. The various tests enable the users to detect potential bias and sources of inefficiency in preliminary estimates.

License EUPL | file LICENSE

URL https://github.com/rjdverse/rjd3revisions, https://rjdverse.github.io/rjd3revisions/

BugReports https://github.com/rjdverse/rjd3revisions/issues

SystemRequirements Java (>= 17)

Depends R (>= 4.1.0)

Imports rJava (>= 1.0-6), rjd3toolkit (>= 3.0.1), checkmate, tools, utils, stats, grDevices, graphics, methods

Suggests kableExtra, testthat (>= 3.0.0), knitr, rmarkdown, readxl, flextable

Remotes github::rjdverse/rjd3toolkit

Collate 'rjd3revisions-package.R' 'check.R' 'conversion.R' 'format_table.R' 'report.R' 'revisions.R' 'revision_analysis.R' 'simulate.R' 'tests.R' 'thresholds.R' 'vintages.R' 'zzz.R'

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

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

RemoteUrl https://github.com/rjdverse/rjd3revisions

RemoteRef HEAD

RemoteSha 38b9dcb0b5577e060170af52efcd7471117b3e34

Contents

bias . . . 3
check_date_year . . . 3
check_horizontal . . . 6
check_long . . . 7
check_vertical . . . 8
cointegration . . . 9
create_vintages . . . 9
create_vintages_from_csv . . . 13
create_vintages_from_xlsx . . . 14
descriptive_statistics . . . 15
efficiencyModel1 . . . 16
efficiencyModel2 . . . 17
get_revisions . . . 17
orthogonallyModel1 . . . 18
orthogonallyModel2 . . . 19
plot.rjd3rev_revisions . . . 20
plot.rjd3rev_vintages . . . 20
print.rjd3rev_revisions . . . 21
print.rjd3rev_rslts . . . 21
print.rjd3rev_vintages . . . 22
render_report . . . 22
revision_analysis . . . 24
set_all_thresholds_to_default . . . 26
set_thresholds_to_default . . . 27
signalnoise . . . 27
simulate_long . . . 28
slope_and_drift . . . 29
summary.rjd3rev_revisions . . . 30
summary.rjd3rev_rslts . . . 30
summary.rjd3rev_vintages . . . 31
theil . . . 31
theil2 . . . 32
unitroot . . . 33
vecm . . . 34
View . . . 35
View.rjd3rev_rslts . . . 35

Index **36**

bias *Estimate bias using t-test and augmented t-test*

Description

Estimate bias using t-test and augmented t-test

Usage

```
bias(revisions.view, na.zero = FALSE)
```

Arguments

revisions.view	mts object. Vertical or diagonal view of the get_revisions() output
na.zero	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

revision_analysis(), render_report()

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4)
revisions <- get_revisions(vintages, gap = 1)
bias(revisions$diagonal_view)
```

check_date_year *Check vector with date*

Description

Useful functions to check if a vector represent dates object

Usage

```
check_date_year(x)

check_date_quarter(x)

check_date_month(x)

check_format_date(x, date_format = "%Y-%m-%d")
```

Arguments

x	a vector of Date, character, integer or POSIXt object representing date object
date_format	character string (or vector of string) corresponding to the format(s) used in x

Details

The function check_date_year checks if the pattern AAAA is recognised. If so, the date will be assimilated with the first January of each year AAAA. The function check_date_quarter checks if the quarterly formats. The accepted formats (for example for the third quarter of 2000) are:

- 2000 T3
- 2000 Q3
- 2000t3
- 2000q3
- 2000T3
- 2000Q3
- 2000 t3
- 2000 q3 If one of the previous formats is recognised, the date will be assimilated with the first day of the quarter of the year (For example 2000 Q3 is assimilated to 2000-07-01). The function check_date_month checks if the monthly formats. The accepted formats (for example for march of 2000) are:
- 2000 M3
- 2000 M03
- 2000 m3
- 2000 m03
- 2000M3
- 2000M03
- 2000m3
- 2000m03 If one of the previous formats is recognised, the date will be assimilated with the first day of the month of the year (For example 2000 M3 is assimilated to 2000-03-01). The function check_format_date checks if the object x match the pattern (or one of the patterns) date_format.

Value

a boolean

Examples

```
# check_date_year ------------------------------------------------

# Good date (representing years)
check_date_year(x = c("2000", "2001", "2002", "2003"))
check_date_year(x = 2020:2024)

# Bad date
check_date_year(x = "2000 ")
check_date_year(x = 1:4)

# check_date_quarter ---------------------------------------------

# Good date
check_date_quarter(x = c("2000 q2", "2000 q3", "2000 q4", "2001 q1"))
check_date_quarter(x = c("2010T1", "2010T2", "2010T3", "2010T4"))
check_date_quarter(x = c("2020Q1", "2020Q2", "2020Q3", "2020Q4"))
check_date_quarter(x = c("2020Q01", "2020Q02", "2020Q03", "2020Q04"))

# Bad date
check_date_quarter(x = "2000 ")
check_date_quarter(x = 1:4)
check_date_quarter(x = "2000 q 2")
check_date_quarter(x = "2000 q12")

# check_date_month -----------------------------------------------

# Good date (representing years)
check_date_month(x = c("2000 m2", "2000 m3", "2000 m4", "2000 m5"))
check_date_month(x = c("2010M9", "2010M10", "2010M11", "2010M12"))
check_date_month(x = c("2020M111", "2020M12", "2021M01", "2021M02"))
check_date_month(x = c("2020M01", "2020M02", "2020M03", "2020M04"))

# Bad date
check_date_month(x = "2000 ")
check_date_month(x = 1:4)
check_date_month(x = "2000 m 2")
check_date_month(x = "2000 m13")

# check_format_date ----------------------------------------------

# Good date (representing years)
check_format_date(x = c("2000-01-01", "2000-02-01", "2000-03-01", "2000-04-01",
                        "2000-05-01", "2000-06-01", "2000-07-01", "2000-08-01",
                        "2000-09-01", "2000-10-01"),
                  date_format = "%Y-%m-%d")
check_format_date(x = c("01/08/2010", "01/09/2010", "01/10/2010", "01/11/2010",
                        "01/12/2010", "01/01/2011", "01/02/2011", "01/03/2011",
```

```
                        "01/04/2011", "01/05/2011"),
                  date_format = "%d/%m/%Y")
check_format_date(x = c("2000-01-01", "2000-02-01", "2000-03-01", "2000-04-01",
                        "2000-05-01", "2000-06-01", "2000-07-01", "2000-08-01",
                        "2000-09-01", "2000-10-01"),
                  date_format = c("%Y-%m-%d", "%d/%m/%Y"))

# Bad date
check_format_date(x = c("2000-01-01", "2000-02-01", "2000-03-01", "2000-04-01",
                        "2000-05-01", "2000-06-01", "2000-07-01", "2000-08-01",
                        "2000-09-01", "2000-10-01"),
                  date_format = "%d/%m/%Y")
check_format_date(x = c("01/08/2010", "01/09/2010", "01/10/2010", "01/11/2010",
                        "01/12/2010", "01/01/2011", "01/02/2011", "01/03/2011",
                        "01/04/2011", "01/05/2011"),
                  date_format = "%Y-%m-%d")
```

`check_horizontal` *Check horizontal format*

Description

Check horizontal format

Usage

```
check_horizontal(x, ...)

## S3 method for class 'data.frame'
check_horizontal(x, ...)

## S3 method for class 'matrix'
check_horizontal(x, date_format = "%Y-%m-%d")

## Default S3 method:
check_horizontal(x, ...)
```

Arguments

`x`	a formatted `data.frame` containing the input in the horizontal format
`...`	Arguments to be passed to `check_horizontal` according to the class of the object `x`
`date_format`	`character` string corresponding to the format used in the input data.frame for the revision dates.

Value

the same input but with date formatted

Examples

```
long_format <- rjd3revisions:::simulate_long(
    start_period = as.Date("2020-01-01"),
    n_period = 24,
    n_revision = 6,
    periodicity = 12L
)
horizontal_format <- rjd3revisions:::from_long_to_horizontal(long_format)
check_horizontal(horizontal_format)
```

---

check_long *Check long format*

Description

Check long format

Usage

```
check_long(x, date_format = "%Y-%m-%d")
```

Arguments

x	a formatted data.frame containing the input in the long format
date_format	character string corresponding to the format used in the input data.frame for the revision dates.

Value

the same input but with column and date formatted

Examples

```
long_format <- rjd3revisions:::simulate_long(
    start_period = as.Date("2020-01-01"),
    n_period = 24,
    n_revision = 6,
    periodicity = 12L
)
check_long(long_format)
```

check_vertical *Check vertical format*

Description

Check vertical format

Usage

```
check_vertical(x, ...)

## S3 method for class 'mts'
check_vertical(x, periodicity, date_format = "%Y-%m-%d", ...)

## S3 method for class 'data.frame'
check_vertical(x, ...)

## S3 method for class 'matrix'
check_vertical(x, periodicity, date_format = "%Y-%m-%d", ...)

## Default S3 method:
check_vertical(x, ...)
```

Arguments

x	a formatted data.frame containing the input in the vertical format
...	Arguments to be passed to check_vertical according to the class of the object x
periodicity	Integer. Periodicity of the time period (12, 4 or 1 for resp. monthly, quarterly or annual data)
date_format	character string corresponding to the format used in the input data.frame for the revision dates.

Value

the same input but in a ts object and with revision date formatted

Examples

```
long_format <- rjd3revisions:::simulate_long(
    start_period = as.Date("2020-01-01"),
    n_period = 24,
    n_revision = 6,
    periodicity = 12L
)
vertical_format <- rjd3revisions:::from_long_to_vertical(long_format, periodicity = 12L)
check_vertical(vertical_format)
```

`cointegration` *Cointegration tests (Engle-Granger)*

Description

Cointegration tests (Engle-Granger)

Usage

```
cointegration(vintages.view, adfk = 1, na.zero = FALSE)
```

Arguments

`vintages.view`	mts object. Vertical or diagonal view of the `create_vintages()` output
`adfk`	Number of lags to consider for ADF
`na.zero`	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

`revision_analysis()`, `render_report()`

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
cointegration(vintages$diagonal_view)
```

`create_vintages` *Create vintage tables*

Description

Create vintage tables from data.frame, matrix or mts object in R

Usage

```
create_vintages(x, ...)

## S3 method for class 'data.frame'
create_vintages(
  x,
  type = c("long", "horizontal", "vertical"),
  periodicity,
  date_format = "%Y-%m-%d",
  vintage_selection,
  ...
)

## S3 method for class 'mts'
create_vintages(
  x,
  type = c("long", "horizontal", "vertical"),
  periodicity,
  date_format = "%Y-%m-%d",
  vintage_selection,
  ...
)

## S3 method for class 'matrix'
create_vintages(
  x,
  type = c("long", "horizontal", "vertical"),
  periodicity,
  date_format = "%Y-%m-%d",
  vintage_selection,
  ...
)

## Default S3 method:
create_vintages(x, ...)
```

Arguments

x	a formatted object containing the input. It can be of type data.frame, matrix or mts and must represent one of the multiple vintage views (selected by the argument type.
...	Arguments to be passed to create_vintages according to the class of the object x
type	character specifying the type of representation of the input between "long", "horizontal" and "vertical" approach.
periodicity	Integer. Periodicity of the time period (12, 4 or 1 for resp. monthly, quarterly or annual data)

`date_format` `character` string corresponding to the format used in the input data.frame for the revision dates.

`vintage_selection`
`Date` vector (or a character vector with the same format as date_format) of length <= 2, specifying the range of revision dates to retain. As an example: c(start = "2022-02-02", end = "2022-08-05") or c(start = as.Date("2022-02-02"), end = as.Date("2022-08-05")) would keep all the vintages whose revision date is between 02 Feb. 2022 and 05 Aug. 2022. If missing (by default), the whole range is selected.

Details

From the input data.frame, the function displays vintages considering three different data structures or views: vertical, horizontal and diagonal. See the `details` section below for more information on the different views. The function returns an object of class `rjd3rev_vintages` that can be used as input in the main function `revision_analysis`.

The are four different vintage views:

1. The vertical view shows the observed values at each time period by the different vintages. This approach is robust to changes of base year and data redefinition. A drawback of this approach is that for comparing the same historical series for different vintages, we need to look at the smallest common number of observations and consequently the number of observations is in some circumstances very small. Moreover, it is often the the case that most of the revision is about the last few points of the series so that the number of observations is too small to test anything.
2. The horizontal view shows the observed values of the different vintages by the period. A quick analysis can be performed by rows in order to see how for the same data point (e.g. 2023Q1), figures are first estimated, then forecasted and finally revised. The main findings are usually obvious: in most cases the variance decreases, namely data converge towards the 'true value'. Horizontal tables are just a transpose of vertical tables and are not used in the tests in `revision_analysis`.
3. The diagonal view shows subsequent releases of a given time period, without regard for the date of publication. The advantage of the diagonal approach is that it gives a way to analyse the trade between the timing of the release and the accuracy of the published figures. It is particularly informative when regular estimation intervals exist for the data under study. However, this approach requires to be particularly vigilant in case there is a change in base year or data redefinition.
4. The long view is a representation of data that allows information to be grouped together in order to facilitate their manipulation. With 3 columns (1 column for the time period, 1 column for the publication / revision date and one column for the data), this representation allows for efficient and non-redundant storage of data.

Value

an object of class `rjd3rev_vintages` which contains the four different view of a revision

Examples

```
## creating the input

# Long format
long_view <- data.frame(
    rev_date = rep(x = c("2022-07-31", "2022-08-31", "2022-09-30", "2022-10-31",
                         "2022-11-30", "2022-12-31", "2023-01-31", "2023-02-28"),
                   each = 4L),
    time_period = rep(x = c("2022Q1", "2022Q2", "2022Q3", "2022Q4"), times = 8L),
    obs_values = c(
        .8, .2, NA, NA, .8, .1, NA, NA,
        .7, .1, NA, NA, .7, .2, .5, NA,
        .7, .2, .5, NA, .7, .3, .7, NA,
        .7, .2, .7, .4, .7, .3, .7, .3
    )
)

vintages_1 <- create_vintages(x = long_view, type = "long", periodicity = 4)

# Horizontal format
horizontal_view <- matrix(data = c(.8, .8, .7, .7, .7, .7, .7, .7, .2, .1,
                           .1, .2, .2, .3, .2, .3, NA, NA, NA, .5, .5, .7, .7,
                           .7, NA, NA, NA, NA, NA, NA, .4, .3),
                          ncol = 4)
colnames(horizontal_view) <- c("2022Q1", "2022Q2", "2022Q3", "2022Q4")
rownames(horizontal_view) <- c("2022-07-31", "2022-08-31", "2022-09-30", "2022-10-31",
                               "2022-11-30", "2022-12-31", "2023-01-31", "2023-02-28")

vintages_2 <- create_vintages(x = horizontal_view, type = "horizontal", periodicity = 4)

# Horizontal format
vertical_view <- matrix(data = c(.8, .2, NA, NA, .8, .1, NA, NA, .7, .1, NA,
                                 NA, .7, .2, .5, NA, .7, .2, .5, NA, .7, .3, .7, NA,
                                 .7, .2, .7, .4, .7, .3, .7, .3),
                          nrow = 4)
rownames(vertical_view) <- c("2022Q1", "2022Q2", "2022Q3", "2022Q4")
colnames(vertical_view) <- c("2022-07-31", "2022-08-31", "2022-09-30", "2022-10-31",
                               "2022-11-30", "2022-12-31", "2023-01-31", "2023-02-28")

vintages_3 <- create_vintages(x = vertical_view, type = "vertical", periodicity = 4)

## specifying the format of revision dates
vintages <- create_vintages(
    x = long_view,
    type ="long",
    periodicity = 4L,
    date_format= "%Y-%m-%d"
)

## including vintage selection
vintages <- create_vintages(
    x = long_view,
```

```
    type ="long",
    periodicity = 4L,
    date_format= "%Y-%m-%d",
    vintage_selection = c(start="2022-10-31", end="2023-01-31")
)
```

`create_vintages_from_csv` *Create vintages table from CSV or TXT files*

Description

Create vintages table from CSV or TXT files

Usage

```
create_vintages_from_csv(
  file,
  type = c("long", "horizontal", "vertical"),
  periodicity,
  date_format = "%Y-%m-%d",
  ...
)
```

Arguments

`file`	character containing the name of the file which the data are to be read from.
`type`	character specifying the type of representation of the input between `"long"`, `"horizontal"` and `"vertical"` approach.
`periodicity`	Integer. Periodicity of the time period (12, 4 or 1 for resp. monthly, quarterly or annual data)
`date_format`	`character` string corresponding to the format used in the input data.frame for the revision dates.
`...`	Arguments to be passed to `read.csv()`, for example: • `sep` the field separator character • `dec` the character used in the file for decimal points. • `row.names` a vector of row names • `skip` integer, the number of lines of the data file to skip before beginning to read data. • `...`

Value

an object of class `rjd3rev_vintages`

See Also

create_vintages_from_xlsx(), create_vintages() which this function wraps.

Examples

```
## Not run:
file_name <- "myinput.csv"
vintages <- create_vintages_from_csv(
    file = file_name,
    type = "vertical",
    periodicity = 12,
    date_format = "%Y-%m-%d",
    sep = ";"
)

## End(Not run)
```

---

create_vintages_from_xlsx

Create vintages table from XLSX files

Description

Create vintages table from XLSX files

Usage

```
create_vintages_from_xlsx(
  file,
  type = c("long", "horizontal", "vertical"),
  periodicity,
  ...
)
```

Arguments

file	character containing the name of the file which the data are to be read from.
type	character specifying the type of representation of the input between "long", "horizontal" and "vertical" approach.
periodicity	Integer. Periodicity of the time period (12, 4 or 1 for resp. monthly, quarterly or annual data)
...	Arguments to be passed to readxl::read_excel(), for example: • sheet character containing the sheet to read • range A cell range to read from • col_names a boolean to use the first row as column names • ...

Value

an object of class `rjd3rev_vintages`

See Also

`create_vintages_from_csv()`, `create_vintages()` which this function wraps.

Examples

```
## Not run:
file_name <- "myinput.xlsx"
sheet_name <- "Sheet1"
vintages <- create_vintages_from_xlsx(
    file = file_name,
    type = "horizontal",
    periodicity = 12L,
    sheet = sheet_name
)

## End(Not run)
```

---

descriptive_statistics *Descriptive statistics*

Description

Descriptive statistics

Usage

```
descriptive_statistics(revisions.view, rounding = 3)
```

Arguments

`revisions.view`	mts object. Vertical or diagonal view of the `get_revisions()` output
`rounding`	number of decimals to display

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)
```

```
## Create vintage and get descriptive statistics of revisions
vintages <- create_vintages(df_long, periodicity = 4)
revisions <- get_revisions(vintages, gap = 1)
descriptive_statistics(revisions$diagonal_view, rounding = 1)
```

`efficiencyModel1` *Efficiency Model 1*

Description

Linear regression model of the revisions (R) on a preliminary vintage (P)

Usage

```
efficiencyModel1(vintages.view, gap = 1, na.zero = FALSE)
```

Arguments

`vintages.view`	mts object. Vertical or diagonal view of the `create_vintages()` output
`gap`	Integer. Gap to consider between each vintages. Default is 1 which means that revisions are calculated and tested for each vintages consecutively.
`na.zero`	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

`revision_analysis()`, `render_report()`

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
efficiencyModel1(vintages$diagonal_view)
```

efficiencyModel2 *Efficiency Model 2*

Description

Linear regression model of R_v on R_{v-1}

Usage

```
efficiencyModel2(vintages.view, gap = 1, na.zero = FALSE)
```

Arguments

vintages.view	mts object. Vertical or diagonal view of the create_vintages() output
gap	Integer. Gap to consider between each vintages. Default is 1 which means that revisions are calculated and tested for each vintages consecutively.
na.zero	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

revision_analysis(), render_report()

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
efficiencyModel2(vintages$diagonal_view)
```

get_revisions *Calculate revisions from vintages*

Description

Calculate revisions from vintages

Usage

```
get_revisions(vintages, gap = 1)
```

Arguments

vintages	an object of class rjd3rev_vintages
gap	Integer. Gap to consider between each vintages to calculate revision. Default is 1 which means that revisions are calculated for each vintages consecutively.

Value

an object of class rjd3rev_revisions which contains the three different views of revisions

See Also

create_vintages()

Examples

```
df <- data.frame(rev_date = c(rep("2022-07-31",4), rep("2022-08-31",4),
                              rep("2022-09-30",4), rep("2022-10-31",4),
                              rep("2022-11-30",4), rep("2022-12-31",4),
                              rep("2023-01-31",4), rep("2023-02-28",4)),
                 time_period = c(rep(c("2022Q1","2022Q2","2022Q3","2022Q4"),8)),
                 obs_values = c(.8,.2,NA,NA, .8,.1,NA,NA,
                                .7,.1,NA,NA, .7,.2,.5,NA,
                                .7,.2,.5,NA, .7,.3,.7,NA,
                                .7,.2,.7,.4, .7,.3,.7,.3))
vintages <- create_vintages(df, periodicity = 4)
revisions <- get_revisions(vintages, gap = 1)
```

---

orthogonallyModel1 *Orthogonally Model 1*

Description

Linear regression model of R_v on R_{v-1},...,R_{v-p}. (p=nrevs)

Usage

```
orthogonallyModel1(revisions.view, nrevs = 1, na.zero = FALSE)
```

Arguments

revisions.view	mts object. Vertical or diagonal view of the get_revisions() output
nrevs	Integer. Number of lags to consider.
na.zero	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

revision_analysis(), render_report()

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
revisions <- get_revisions(vintages, gap = 1)
orthogonallyModel1(revisions$diagonal_view)
```

---

orthogonallyModel2 *Orthogonally Model 2*

Description

Linear regression model of R_v on R_{v-k} (k = reference)

Usage

```
orthogonallyModel2(revisions.view, reference = 1, na.zero = FALSE)
```

Arguments

revisions.view	mts object. Vertical or diagonal view of the get_revisions() output
reference	Integer. Number of lags to consider.
na.zero	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

revision_analysis(), render_report()

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
```

```
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
revisions <- get_revisions(vintages, gap = 1)
orthogonallyModel2(revisions$diagonal_view)
```

---

plot.rjd3rev_revisions

Plot function for objects of class "rjd3rev_revisions"

Description

Plot function for objects of class "rjd3rev_revisions"

Usage

```
## S3 method for class 'rjd3rev_revisions'
plot(x, view = c("vertical", "diagonal"), n_rev = 2, ...)
```

Arguments

x	an object of class "rjd3rev_revisions"
view	view to plot. By default, the vertical view is considered.
n_rev	number of revision dates to consider. For the vertical view, the lasts n_rev revisions are plotted. For the diagonal view, the revisions between the first n_rev releases are plotted. The maximum number of n_rev is 5.
...	further arguments passed to ts.plot().

---

plot.rjd3rev_vintages *Plot function for objects of class "rjd3rev_vintages"*

Description

Plot function for objects of class "rjd3rev_vintages"

Usage

```
## S3 method for class 'rjd3rev_vintages'
plot(x, col, ...)
```

Arguments

x	an object of class "rjd3rev_vintages".
col	a color vector of the same length as the number of releases
...	further arguments passed to or from other methods.

print.rjd3rev_revisions *Print function for objects of class* "rjd3rev_revisions"

Description

Print function for objects of class "rjd3rev_revisions"

Usage

```
## S3 method for class 'rjd3rev_revisions'
print(x, n_row = 12, n_col = 3, ...)
```

Arguments

x	an object of class "rjd3rev_revisions".
n_row	number of last rows to display. For the horizontal view, corresponds to the number of columns.
n_col	number of columns to display. Can be either the last n columns (verical view), the last n rows (horizontal view) or the first n columns (diagonal view).
...	further arguments passed to the print function.

print.rjd3rev_rslts *Print function for objects of class* rjd3rev_rslts

Description

Print function for objects of class rjd3rev_rslts

Usage

```
## S3 method for class 'rjd3rev_rslts'
print(x, ...)
```

Arguments

x	an object of class rjd3rev_rslts
...	further arguments passed to the print function.

print.rjd3rev_vintages *Print function for objects of class "rjd3rev_vintages"*

Description

Print function for objects of class "rjd3rev_vintages"

Usage

```
## S3 method for class 'rjd3rev_vintages'
print(x, n_row = 8, n_col = 3, ...)
```

Arguments

x	an object of class "rjd3rev_vintages".
n_row	number of last rows to display. For the horizontal view, corresponds to the number of columns.
n_col	number of columns to display. Can be either the last n columns (verical view), the last n rows (horizontal view) or the first n columns (diagonal view). This argument is not used for the long view.
...	further arguments passed to the print function.

render_report *Generate report on Revision Analysis*

Description

Generate report on Revision Analysis

Usage

```
render_report(
  rslt,
  output_file,
  output_dir,
  output_format = c("html_document", "pdf_document", "word_document"),
  plot_revisions = FALSE,
  open_report = TRUE,
  ...
)
```

Arguments

rslt	an object of class "rjd3rev_rslts" which is the output of the function revision_analysis()
output_file	path or name of the output file containing the report
output_dir	path of the dir containing the output file (Optional)
output_format	either an HTML document (default), a PDF document or a Word document
plot_revisions	Boolean. Default is FALSE meaning that a plot with the revisions will not be added to the report.
open_report	Boolean. Default is TRUE meaning that the report will open automatically after being generated.
...	Arguments to be passed to rmarkdown::render(), for example: • output_options List of output options that can override the options specified in metadata • ...

See Also

revision_analysis() to create the input object

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Make analysis and generate the report

vintages <- create_vintages(df_long, periodicity = 4L, type = "long")
rslt <- revision_analysis(vintages, view = "diagonal")

## Not run:
render_report(
    rslt,
    output_file = "my_report",
    output_dir = "C:/Users/xxx",
    output_format = "pdf_document",
    plot_revisions = TRUE
)

## End(Not run)
```

`revision_analysis` *Revision analysis through a battery of tests*

Description

The function perform parametric tests which enable the users to detect potential bias (both mean and regression bias) and sources of inefficiency in preliminary estimates. We would conclude to inefficiency in the preliminary estimates when revisions are predictable in some way. In the results, parametric tests are divided into 5 categories: relevancy (check whether preliminary estimates are even worth it), bias, efficiency, orthogonality (correlation at higher lags), and signalVSnoise. Descriptive statistics on revisions are also provided. For some of the parametric tests, prior transformation of the vintage data may be important to avoid misleading results. By default, the decision to differentiate the vintage data is performed automatically based on unit root and co-integration tests whose results can be found found in the results too (section 'varbased'). Finally, running the function `render_report()` on the output of `revision_analysis()` would give you both a formatted summary of the results and full explanations about each tests.

Usage

```
revision_analysis(
  vintages,
  gap = 1,
  view = c("vertical", "diagonal"),
  n.releases = 3,
  transf.diff = c("auto", "forced", "none"),
  transf.log = FALSE,
  descriptive.rounding = 3,
  nrevs = 1,
  ref = 1,
  na.zero = FALSE
)
```

Arguments

`vintages`	an object of class `"rjd3rev_vintages"` which is the output of the function `create_vintages()`
`gap`	Integer. Gap to consider between each vintages. Default is 1 which means that revisions are calculated and tested for each vintages consecutively.
`view`	Selected view. Can be "vertical" (the default) or "diagonal". Vertical view shows the observed values at each time period by the different vintages. Diagonal view shows subsequent releases of a given time period, without regard for the date of publication, which can be particularly informative when regular estimation intervals exist. See `?create_vintages()` for more information about interests and drawbacks of each view.
`n.releases`	only used when `view = "diagonal"`. Ignored otherwise. Allow the user to limit the number of releases under investigation). When `view = "vertical"`,

the user is invited to limit the number of vintages upstream through the parameter `vintage_selection` in `create_vintages()` whenever necessary.

`transf.diff` differentiation to apply to the data prior testing. Only used for regressions including vintage data as regressor and/or regressand. Regression including revision data only are never differentiated even if `transf.diff = "forced"`. Options are "automatic" (the default), "forced" and "none".

`transf.log` Boolean whether a log-transformation should first be applied to the data. Default is FALSE.

`descriptive.rounding`
Integer. Number of decimals to display for descriptive statistics. Default is 3.

`nrevs, ref` Integer. Number of lags to consider for orthogonality tests 1 and 2 respectively.

`na.zero` Boolean whether missing values should be considered as 0 or rather as data not yet available (the default).

Value

an object of class 'rjd3rev_rslts'

See Also

`create_vintages()` to create the input object, `render_report()` to get a summary and information the tests

Examples

```
## Simulated data

df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 10L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create a `"rjd3rev_vintages"` object with the input
vintages <- create_vintages(x = df_long, periodicity = 4L, date_format = "%Y-%m-%d")
# revisions <- get_revisions(vintages, gap = 1L) # just to get a first insight of the revisions

## Call using all default parameters
rslt1 <- revision_analysis(vintages)
# render_report(rslt1, output_file = "report1", output_dir = "C:/Users/xxx")
summary(rslt1) # formatted summary only
View(rslt1) # formatted tables in viewer panel

## Calls using diagonal view (suited in many situations such as to evaluate GDP estimates)
## Note: when input are not growth rates but the gross series, differentiation is
## performed automatically (if transf.diff is let to its default option) but `transf.log`
## must be set to TRUE manually whenever a log-transformation of the data is necessary
rslt2 <- revision_analysis(vintages, gap = 1, view = "diagonal", n.releases = 3)
# render_report(rslt2, output_file = "report2", output_dir = "C:/Users/xxx",
```

```
#               output_format = "word_document", plot_revisions = TRUE)
summary(rslt2)
View(rslt2)

## Call to evaluate revisions for a specific range of vintage periods
vintages <- create_vintages(
    x = df_long,
    periodicity = 4L,
    vintage_selection = c(start = "2012-12-31", end = "2018-06-30")
)
rslt3 <- revision_analysis(vintages, gap = 2, view = "vertical")
#render_report(rslt3, output_file = "report2", output_dir = "C:/Users/xxx", plot_revisions = TRUE)
summary(rslt3)
View(rslt3)

## Note that it is possible to change thresholds values for quality
## assessment using options (see vignette for details)
options(
    augmented_t_threshold = c(severe = 0.005, bad = 0.01, uncertain = 0.05),
    slope_and_drift_threshold = c(severe = 0.005, bad = 0.05, uncertain = 0.10),
    theil_u2_threshold = c(uncertain = .5, bad = .7, severe = 1)
)
rslt4 <- revision_analysis(vintages, gap = 1, view = "diagonal", n.releases = 3)
summary(rslt4)
View(rslt4)
```

`set_all_thresholds_to_default` *Set all test thresholds to their default values*

Description

Set all test thresholds to their default values

Usage

```
set_all_thresholds_to_default(diagnostic_tests = TRUE)
```

Arguments

`diagnostic_tests`	Boolean. Whether or not to reset thresholds for diagnostics tests on residuals as well in addition to parametric tests.

Examples

```
set_all_thresholds_to_default()
```

set_thresholds_to_default *Set thresholds of a given test to their default values*

Description

Set thresholds of a given test to their default values

Usage

```
set_thresholds_to_default(threshold_option_name)
```

Arguments

threshold_option_name	Boolean. Whether or not to reset thresholds for diagnostics tests on residuals as well in addition to parametric tests.

Examples

```
set_thresholds_to_default("t_threshold")
```

signalnoise *Signal VS Noise*

Description

Linear regression models to determine whether revisions are 'news' or 'noise'. For 'noise': R (revisions) on P (preliminary estimate). For 'news': R on L (latter estimate).

Usage

```
signalnoise(vintages.view, gap = 1, na.zero = FALSE)
```

Arguments

vintages.view	mts object. Vertical or diagonal view of the `create_vintages()` output
gap	Integer. Gap to consider between each vintages. Default is 1 which means that revisions are calculated and tested for each vintages consecutively.
na.zero	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

`revision_analysis()`, `render_report()`

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
signalnoise(vintages$diagonal_view)
```

`simulate_long` *Simulate long datasets with revisions*

Description

Simulate long datasets with revisions

Usage

```
simulate_long(
  n_period = 50,
  n_revision = 10,
  start_period = as.Date("2012-01-01"),
  periodicity = 12L
)
```

Arguments

`n_period`	Integer. Number of different time-period (length of the simulated series).
`n_revision`	Integer. Number of different revision dates.
`start_period`	Date. Start of the series.
`periodicity`	Integer. Periodicity of the time period (12, 4 or 1 for resp. monthly, quarterly or annual data).

Value

A dataset in the long format. See `create_vintages` for more information about the different data formats.

Examples

```
simulate_long(n_period = 100L, n_revision = 10L)
simulate_long(periodicity = 1L)
simulate_long(start_period = as.Date("2000-01-01"),
              n_period = 10L * 12L,
              periodicity = 12L)
simulate_long(periodicity = 4L, n_period = 5L * 4L)
```

`slope_and_drift` *Slope and Drift*

Description

Linear regression model of a latter vintage (L) on a preliminary vintage (P)

Usage

```
slope_and_drift(vintages.view, gap = 1, na.zero = FALSE)
```

Arguments

`vintages.view`	mts object. Vertical or diagonal view of the `create_vintages()` output
`gap`	Integer. Gap to consider between each vintages. Default is 1 which means that revisions are calculated and tested for each vintages consecutively.
`na.zero`	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

`revision_analysis()`, `render_report()`

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
slope_and_drift(vintages$diagonal_view)
```

summary.rjd3rev_revisions *Summary function for objects of class "rjd3rev_revisions"*

Description

Summary function for objects of class "rjd3rev_revisions"

Usage

```
## S3 method for class 'rjd3rev_revisions'
summary(object, ...)
```

Arguments

object	an object of class "rjd3rev_revisions".
...	further arguments passed to or from other methods.

summary.rjd3rev_rslts *Summary function for objects of class* rjd3rev_rslts

Description

Summary function for objects of class rjd3rev_rslts

Usage

```
## S3 method for class 'rjd3rev_rslts'
summary(object, ...)
```

Arguments

object	an object of class rjd3rev_rslts
...	further arguments passed to or from other methods.

summary.rjd3rev_vintages *Summary function for objects of class "rjd3rev_vintages"*

Description

Summary function for objects of class "rjd3rev_vintages"

Usage

```
## S3 method for class 'rjd3rev_vintages'
summary(object, ...)
```

Arguments

object	an object of class "rjd3rev_vintages".
...	further arguments passed to or from other methods.

theil *Theil's Inequality Coefficient U1*

Description

Theil's Inequality Coefficient U1

Usage

```
theil(vintages.view, gap = 1, na.zero = FALSE)
```

Arguments

vintages.view	mts object. Vertical or diagonal view of the create_vintages() output
gap	Integer. Gap to consider between each vintages. Default is 1 which means that revisions are calculated and tested for each vintages consecutively.
na.zero	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

revision_analysis(), render_report()

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4)
theil(vintages$diagonal_view)
```

`theil2` *Theil's Inequality Coefficient U2*

Description

Theil's Inequality Coefficient U2

Usage

```
theil2(vintages.view, gap = 1, na.zero = FALSE)
```

Arguments

`vintages.view`	mts object. Vertical or diagonal view of the `create_vintages()` output
`gap`	Integer. Gap to consider between each vintages. Default is 1 which means that revisions are calculated and tested for each vintages consecutively..
`na.zero`	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

`revision_analysis()`, `render_report()`

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
```

```
vintages <- create_vintages(df_long, periodicity = 4)
theil2(vintages$diagonal_view)
```

`unitroot` *Unit root test*

Description

Unit root test

Usage

```
unitroot(vintages.view, adfk = 1, na.zero = FALSE)
```

Arguments

`vintages.view`	mts object. Vertical or diagonal view of the `create_vintages()` output
`adfk`	Number of lags to consider for Augmented Dicky-Fuller (ADF) test
`na.zero`	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

See Also

`revision_analysis()`, `render_report()`

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
unitroot(vintages$diagonal_view)
```

vecm *Vector error correction model (VECM)*

Description

Can lead to a better understanding of the nature of any nonstationary process among the different component series.

Usage

```
vecm(
  vintages.view,
  lag = 2,
  model = c("none", "cnt", "trend"),
  na.zero = FALSE
)
```

Arguments

vintages.view	mts object. Vertical or diagonal view of the create_vintages() output
lag	Number of lags
model	Character. Must be "none" (the default), "cnt" or "trend".
na.zero	Boolean whether missing values should be considered as 0 or rather as data not (yet) available (the default).

Examples

```
## Simulated data
df_long <- simulate_long(
    n_period = 10L * 4L,
    n_revision = 5L,
    periodicity = 4L,
    start_period = as.Date("2010-01-01")
)

## Create vintage and test
vintages <- create_vintages(df_long, periodicity = 4L)
vecm(vintages$diagonal_view)
```

View *View function for objects of class "rjd3rev_vintages"*

Description

Display the different view in a different panel to visualize the data in a table / matrix format

Usage

```
View(x, ...)

## Default S3 method:
View(x, ...)

## S3 method for class 'rjd3rev_vintages'
View(x, type = c("all", "long", "horizontal", "vertical", "diagonal"), ...)
```

Arguments

x	an object of class "rjd3rev_vintages".
...	further arguments passed to the View method.
type	type of view to display

Details

Generate the view of the vintages in different format. With the type argument, you can choose the view to display. You can choose between the long, horizontal, vertical and diagonal view.

View.rjd3rev_rslts *View function for objects of class* rjd3rev_rslts

Description

View function for objects of class rjd3rev_rslts

Usage

```
## S3 method for class 'rjd3rev_rslts'
View(x, type = c("summary", "stats-desc", "revisions", "tests"), ...)
```

Arguments

x	an object of class rjd3rev_rslts
type	type of view to display
...	further arguments passed to View.

Index

bias, 3

check_date_month *(*check_date_year*)*, 3
check_date_quarter *(*check_date_year*)*, 3
check_date_year, 3
check_format_date *(*check_date_year*)*, 3
check_horizontal, 6
check_long, 7
check_vertical, 8
cointegration, 9
create_vintages, 9, *28*
create_vintages(), *14, 15*
create_vintages_from_csv, 13
create_vintages_from_csv(), *15*
create_vintages_from_xlsx, 14
create_vintages_from_xlsx(), *14*

descriptive_statistics, 15

efficiencyModel1, 16
efficiencyModel2, 17

get_revisions, 17

orthogonallyModel1, 18
orthogonallyModel2, 19

plot.rjd3rev_revisions, 20
plot.rjd3rev_vintages, 20
print, *21, 22*
print.rjd3rev_revisions, 21
print.rjd3rev_rslts, 21
print.rjd3rev_vintages, 22

render_report, 22
revision_analysis, 24

set_all_thresholds_to_default, 26
set_thresholds_to_default, 27
signalnoise, 27
simulate_long, 28
slope_and_drift, 29
summary.rjd3rev_revisions, 30
summary.rjd3rev_rslts, 30
summary.rjd3rev_vintages, 31

theil, 31
theil2, 32

unitroot, 33

vecm, 34
View, *35*, 35
View.rjd3rev_rslts, 35