R packages around JDemetra+ - Part 1

A versatile toolbox for time series analysis

Anna Smyk and Tanguy Barthelemy (Insee, France)

UROS Conference, Athens (GR), Nov 27th 2024

Section 1

rjd3 ecosystem overview

JDemetra+: a library of algorithms for time series analysis

JDemetra+ a library of algorithms (written in Java) on:

- Seasonal Adjustment (Historical domain)
- Trend and cycle estimation
- Benchmarking and temporal disaggregation
- Revision Analysis
- Nowcasting

They can be accessed via Graphical user-interface (GUI) and/or R packages.

JDemetra+ is an open source software, officially recommended by Eurostat since 2015 for Seasonal Adjustment to Eurosystem members.

JDemetra + on Github

- Repository dedicated to Java algorithms and Graphical User interface (+ extensions) : https://github.com/jdemetra
- Repository dedicated to R packages: https://github.com/rjdverse

For each R package:

- Readme files
- Documentation of (almost) all functions in (almost) all R packages
- GitHub pages (linked in JD+ on-line documentation, https://jdemetra-new-documentation.netlify.app/)

JDemetra+ algorithms in R (1/3)

By domain of use:

- Seasonal adjustment of low frequency data
  - rjd3x13 (Reg-Arima + x11 based decomposition)
  - rjd3tramoseats (Tramo+ AMB decomposition)
  - rjd3sts (Basic structural models, SA)
  - rjd3stl (SA with Local regression)
- Seasonal adjustment of high frequency data
  - rjd3highfreq (extended airline model + extended AMB decomposition)
  - rjd3x11plus (extended X11)
  - rjd3sts (basic structural models, SA)
  - rjd3stl (SA with local regression)

JDemetra+ algorithms in R (2/3)

By domain of use:

- Filtering and trend estimation
  - rjd3filters
  - rjd3x11plus (local polynomials)
- General purpose tools
  - rjd3toolkit (specifications, tests, regressors)
  - rjd3sts (state space framework)
  - rjd3filters (generating moving averages)

JDemetra+ algorithms in R (3/3)

By domain of use:

- Non Seasonal Adjustment related tools
  - rjd3bench (benchmarking and temporal disaggregation)
  - rjd3revisions (revision analysis)
  - rjd3nowcasting (...nowcasting !)
- Tools related to GUI (workspaces)
  - rjd3providers (input data)
  - rjd3workspace (workspace wrangling)

Installing rjd3 packages: Java configuration

You can find here : https://jdemetra-new-documentation.netlify.app/#Rconfig a tutorial to configure Java on your computer.

Installing rjd3 packages: latest release

```
# install.packages("remotes")
remotes::install_github("rjdverse/rjd3toolkit@*release")
remotes::install_github("rjdverse/rjd3x13@*release")
remotes::install_github("rjdverse/rjd3tramoseats@*release")
remotes::install_github("rjdverse/rjd3providers@*release")
remotes::install_github("rjdverse/rjd3filters@*release")
remotes::install_github("rjdverse/rjd3sts@*release")
remotes::install_github("rjdverse/rjd3highfreq@*release")
remotes::install_github("rjdverse/rjd3x11plus@*release")
remotes::install_github("rjdverse/rjd3stl@*release")
remotes::install_github("rjdverse/rjd3workspace@*release")
remotes::install_github("rjdverse/rjd3revisions@*release")
remotes::install_github("rjdverse/rjd3bench@*release")
remotes::install_github("rjdverse/rjd3nowcasting@*release")
remotes::install_github("AQLT/ggdemetra3@*release") # additional graphics
```

Installing rjd3 packages: runiverse

```
# install.packages("remotes")
options(repos = c(runiverse = "https://rjdverse.r-universe.dev",
                  CRAN = "https://cloud.r-project.org"))

install.packages("rjd3toolkit")
install.packages("rjd3x13")
install.packages("rjd3tramoseats")
install.packages("rjd3providers")
install.packages("rjd3filters")
install.packages("rjd3sts")
install.packages("rjd3highfreq")
install.packages("rjd3x11plus")
install.packages("rjd3stl")
install.packages("rjd3workspace")
install.packages("rjd3revisions")
install.packages("rjd3bench")
install.packages("rjd3nowcasting")
install.packages("ggdemetra3", repos = c("https://aqlt.r-universe.dev",
                                         "https://cloud.r-project.org"))
```

Installing rjd3 packages (develop version)

Installing (the develop version) from the GitHub home repo

```
# install.packages("remotes")
remotes::install_github("rjdverse/rjd3toolkit")
remotes::install_github("rjdverse/rjd3x13")
remotes::install_github("rjdverse/rjd3tramoseats")
remotes::install_github("rjdverse/rjd3providers")
remotes::install_github("rjdverse/rjd3filters")
remotes::install_github("rjdverse/rjd3sts")
remotes::install_github("rjdverse/rjd3highfreq")
remotes::install_github("rjdverse/rjd3x11plus")
remotes::install_github("rjdverse/rjd3stl")
remotes::install_github("rjdverse/rjd3workspace")
remotes::install_github("rjdverse/rjd3revisions")
remotes::install_github("rjdverse/rjd3bench")
remotes::install_github("rjdverse/rjd3nowcasting")
remotes::install_github("AQLT/ggdemetra3") # additional graphics
```

Loading packages

```
library("rjd3toolkit")
library("rjd3x13")
library("rjd3tramoseats")
library("rjd3filters")
library("rjd3highfreq")
library("rjd3x11plus")
library("rjd3revisions")
library("rjd3bench")
library("rjd3nowcasting")
library("rjd3sts")
library("rjd3stl")
library("ggdemetra3")
```

Section 2

Time series tools

Time series tools

JDemetra+ 3.x offers stand alone tools (mainly in `rjd3toolkit`)

- Tests (seasonality, auto-correlation, normality, randomness...)
- (Fast) Arima Modelling
- Flexible Calendar regressors generation
- Auxiliary variables for pre-adjustment in seasonal adjustment
- Spectral analysis
- Detection of multiple seasonal patterns (Canova-Hansen test)
- State space frame work as a toolbox (`rjd3sts`)

Testing for seasonality

In **{rjd3toolkit}**:

- Canova-Hansen (`rjd3toolkit::seasonality_canovahansen_trigs()`)
- X-12 combined test (`rjd3toolkit::seasonality_combined()`)
- F-test on seasonal dummies (`rjd3toolkit::seasonality_f()`)
- Friedman Seasonality Test (`rjd3toolkit::seasonality_friedman()`)
- Kruskall-Wallis Seasonality Test (`rjd3toolkit::seasonality_kruskalwallis()`)
- Periodogram Seasonality Test (`rjd3toolkit::seasonality_periodogram()`)
- QS Seasonality Test (`rjd3toolkit::seasonality_qs()`)

Subsection 1

Sarima modelling in rjdverse

Sarima modelling in rjd3verse

- (Reg)-Sarima model estimation: `rjd3toolkit::sarima_estimate`
- (Reg)-Sarima model identification and estimation pre-adjustment part of Seasonal Adjustment Processes, available in **rjd3x13** and **rjd3tramoseats** packages

example `rjd3x13::regarima`

Sarima estimation

```
library("microbenchmark")

y_example <- log(rjd3toolkit::ABS$X0.2.09.10.M)
microbenchmark(
    JD_arima = rjd3toolkit::sarima_estimate(
        x = y_example,
        order = c(2, 1, 1), seasonal = list(order = c(0, 1, 1), period = 12)
    ),
    built_in_arima = stats::arima(
        x = y_example,
        order = c(2, 1, 1), seasonal = list(order = c(0, 1, 1), period = 12)
    ), times = 10
)
```

```
Unit: milliseconds
           expr      min       lq      mean    median        uq       max neval
       JD_arima  15.5575  20.4269  55.96686  23.97535  31.8216 334.6192    10
 built_in_arima 259.8594 287.0958 300.39110 300.84330 304.4804 366.1775    10
```

Generating User-defined auxilary variables

What is useful for sa and examples below

- Calendar correction : specific regressors, see below in SA part

Outliers and intervention variables

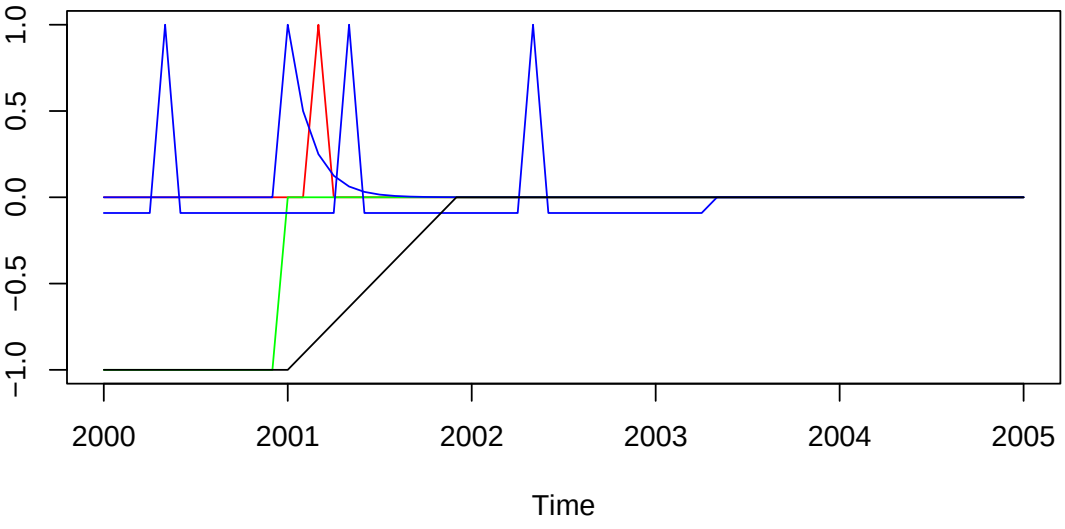
- Outliers regressors (AO, LS, TC, SO, Ramp (quadratic to be added)
- Trigonometric variables
- Seasonal dummies

Example of outliers I

```
library("rjd3toolkit")

# ts for initialization
s <- ts(0, start = 2000, end = 2005, frequency = 12)
# You can use an initialization ts or provide frequency, start and length
# Creating outliers
ao <- ao_variable(s = s, date = "2001-03-01")
ls <- ls_variable(s = s, date = "2001-01-01")
tc <- tc_variable(s = s, date = "2001-01-01", rate = 0.5)
# Customizable rate
so <- so_variable(s = s, date = "2003-05-01")
ramp <- ramp_variable(s = s, range = c("2001-01-01", "2001-12-01"))
ts.plot(ts.union(ao, ls, tc, so, ramp),
        col = c("red", "green", "Blue", "blue", "black"))
```

Example of outliers II



Section 3

Seasonal adjustment using JDemetra+ in R

Subsection 1

Low frequency data

Seasonal Adjustment Algorithms in JDemetra+

Algorithm	Version 2.x		Version 3.x	
	Access in GUI	Access in R	Access in GUI	Access in R
X-13 Arima	yes	RJDemetra	yes	`rjd3x13`
Tramo-Seats	yes	RJDemetra	yes	`rjd3tramoseats`
X12plus			yes	`rjd3x11plus`
STL			yes	`rjd3stl`
BSM			yes	`rjd3sts`
SEATS+			upcoming	upcoming

Two categories of algorithms for low frequency data:

- Historical core (main): X-13-Arima and Tramo-Seats
- Version 3 (recent) additional algorithms

Additional algorithms

Tramo-Seats and X-13-Arima share a very similar and sophisticated pre-adjustment process for the Arima model selection phase.

For additional algorithms, the new philosophy is to offer

- A simplified pre-adjustment on the arima modelling side, reduced to airline model
- Several enhanced decomposition options
  - **stl+** ("+" stands for airline based pre-adjustment)
  - **x12+**: airline based pre-adjustment + new trend estimation filters (Local Polynomials)
  - **seats+** (to come in the target v3 version): airline based pre-adjustment + AMB decomposition

Subsection 2

Quick Launch with default specifications

Seasonal Adjustment: common steps

- Testing for seasonality (identify seasonal patterns for HF data)
- Pre-treatment
- Create customised variables for pre-treatment (e.g calendar regressors)
- Decomposition
- Retrieve output series
- Retrieve diagnostics
- Customize parameters
- Refresh data (bonus)
- ...
- Repeat...

Acceptable frequencies: data with p in $2, 3, 4, 6, 12$ is admissible in all algorithms.

Importing data

Here we import the data from the french industrial production index:

We create a ts object with one of the series:

```
library("readr")
library("dplyr")

ipi <- read_delim("Data/IPI_nace4.csv", delim = ";") ▷
    mutate(date = as.Date(date, format = "%d/%m/%Y"),
           across(!date, as.numeric))
y_raw <- ts(data = ipi[, "RF3030"], frequency = 12, start = c(1990, 1))
```

Subsection 3

Quick launch with default specifications

Quick launch with default specifications

In this section, we will use the packages **{rjd3x13}** and **{rjd3tramoseats}**:

```
library("rjd3x13")
library("rjd3tramoseats")
```

Different processing I

Running a full Seasonal Adjustment processing

```
# X13
sa_x13_v3 <- rjd3x13::x13(y_raw, spec = "RSA5")

# Tramo seats
sa_ts_v3 <- rjd3tramoseats::tramoseats(y_raw, spec = "RSAfull")
```

Running only pre-adjustment

Different processing II

```
# X13
sa_regarima_v3 <- rjd3x13::regarima(y_raw, spec = "RG5c")

# Tramo seats
sa_tramo_v3 <- rjd3tramoseats::tramo(y_raw, spec = "TRfull")

# "fast_XXX" versions ... (just results, cf output structure)
```

Running only decomposition

```
# X11 is a specific function
x11_v3 <- rjd3x13::x11(y_raw)
```

Subsection 4

Retrieving output and data visualization

"Model_sa" object structure I

Results vs specification...and then by domain

```
# Model_sa = sa_x13_v3
sa_x13_v3 <- rjd3x13::x13(y_raw, spec = "RSA5")
```

```
sa_x13_v3$result
sa_x13_v3$estimation_spec
sa_x13_v3$result_spec
sa_x13_v3$user_defined
```

Retrieve output series

Input and output series are TS objects in R

- final series

```
# final seasonally adjusted series
sa_x13_v3$result$final$d11final
```

```
           Jan       Feb       Mar       Apr       May       Jun       Jul
2015 101.00943 107.77175 103.00131  94.23577  96.69935  98.80666  98.23143
2016 107.16594 102.22538 104.12974 110.10259 106.94256 108.05816  99.38677
2017 108.45470 104.31209 111.38200 107.67890 111.45404 103.49818 104.07304
2018 107.40292 104.69368 103.66255 112.65495 110.93944 116.46396 120.32502
2019 112.03770 117.71382 113.65766 112.74112 115.12816 106.17979 112.27642
2020 105.12123 107.97116  79.92035  65.67288  59.22572  70.01258  67.38306
2021  72.91616  63.38023  70.46020  70.87969  71.13369  72.27524  76.98557
           Aug       Sep       Oct       Nov       Dec
2015 100.19901 106.04675  99.31709  95.83252  97.33066
2016  98.46974 103.05138 106.20451 100.80206 102.89267
2017 121.14561 106.80214 106.75247 115.60535 104.16300
2018 116.86920 111.76512 109.73788 116.70961 121.76784
```

Series from decomposition

check output names

```
# tables from D1 to D13
sa_x13_v3$result$decomposition$d5
```

	Jan	Feb	Mar	Apr	May	Jun	Jul
2015	0.9671969	1.0259171	1.1433522	0.9977630	0.9344235	1.1512369	0.8502556
2016	0.9912076	0.9998156	1.1413886	1.0114885	0.9559634	1.1345715	0.8512917
2017	0.9948506	0.9799265	1.1293967	1.0007843	0.9685890	1.1052400	0.8712689
2018	0.9792026	0.9614345	1.1029869	0.9836683	0.9660983	1.0922782	0.9003009
2019	0.9633474	0.9593604	1.0900038	0.9493774	0.9514231	1.0739355	0.9251931
2020	0.9547157	0.9482438	1.0868284	0.9268561	0.9374488	1.0770919	0.9406608
2021	0.9571768	0.9371914	1.0926323	0.9134289	0.9290156	1.0763504	0.9494751
2022	0.9602994	0.9263622	1.0973870	0.9134289	0.9290156	1.0763504	0.9494751

	Aug	Sep	Oct	Nov	Dec
2015	0.7250049	1.1026940	1.1232296	1.0430244	0.9232794
2016	0.7254213	1.0950852	1.1088890	1.0696157	0.9356648
2017	0.7433354	1.0981051	1.0941449	1.1166228	0.9472417
2018	0.7569311	1.0944032	1.0821739	1.1554418	0.9759946
2019	0.7689645	1.1015712	1.0853035	1.1802980	0.9852067

Retrieving Diagnostics

Just fetch the needed objects in the relevant part of the output structure or print the whole "model"

```
sa_x13_v3$result$diagnostics$td.ftest.i
```

```
Value: 0.03063328
P-Value: 0.9999
```

What is missing (series or diagnostics) can be retrieved adding user-defined output in the options

Retrieving user defined-output (1/2) I

First define the vector of objects you wish to add

Lists of available diagnostics or series

```
rjd3x13::userdefined_variables_x13("regarima") # restriction
rjd3x13::userdefined_variables_x13()

rjd3tramoseats::userdefined_variables_tramoseats("tramo") # restriction
rjd3tramoseats::userdefined_variables_tramoseats("tramoseats")
```

Retrieve user defined-output (2/2)

Select the objects and customize estimation function

```
ud <- rjd3x13::userdefined_variables_x13()[15:17] # b series
ud
```

```
[1] "cal"      "cal_b"    "cal_b(?)"
```

```
sa_x13_v3_ud <- rjd3x13::x13(y_raw, "RSA5c", userdefined = ud)

# Retrieve the object
sa_x13_v3_ud$user_defined$cal
```

	Jan	Feb	Mar	Apr	May	Jun	Jul
1990	1.0363890	0.9911504	1.0153134	0.9652130	1.0190605	0.9971963	1.0012305
1991	1.0190605	0.9911504	0.9364198	1.0662135	1.0208211	0.9479590	1.0363890
1992	1.0208211	1.0043748	1.0207756	0.9824016	0.9661709	1.0333817	1.0208211
1993	0.9661709	0.9911504	1.0566204	0.9983698	0.9627990	1.0204147	0.9958729
1994	0.9627990	0.9911504	1.0002567	1.0159425	1.0012305	1.0015791	0.9661709
1995	1.0012305	0.9911504	1.0407486	0.9298082	1.0363890	1.0178591	0.9627990
1996	1.0363890	1.0251863	0.9726470	1.0265012	1.0208211	0.9479590	1.0363890

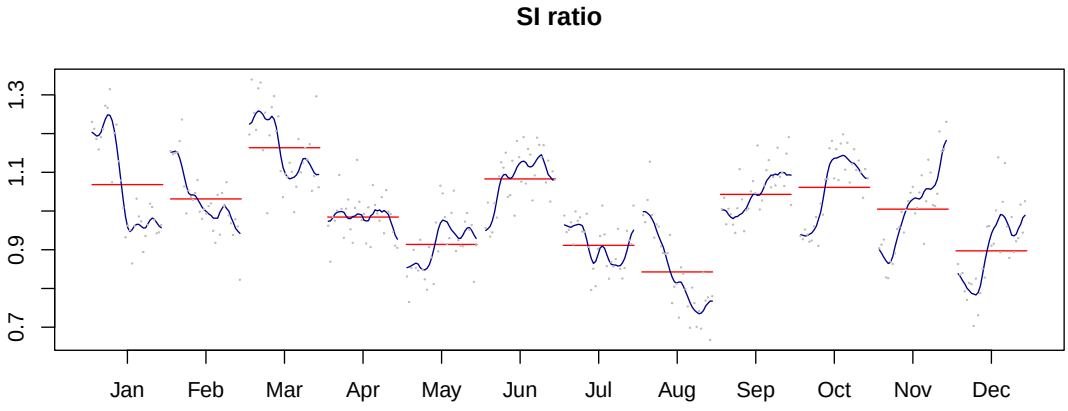
Plots and data visualisation I

EXAMPLES

- final + "autoplot" layout
- regarima not available (yet ?) !!!
- SI ratios (ggplot layout)

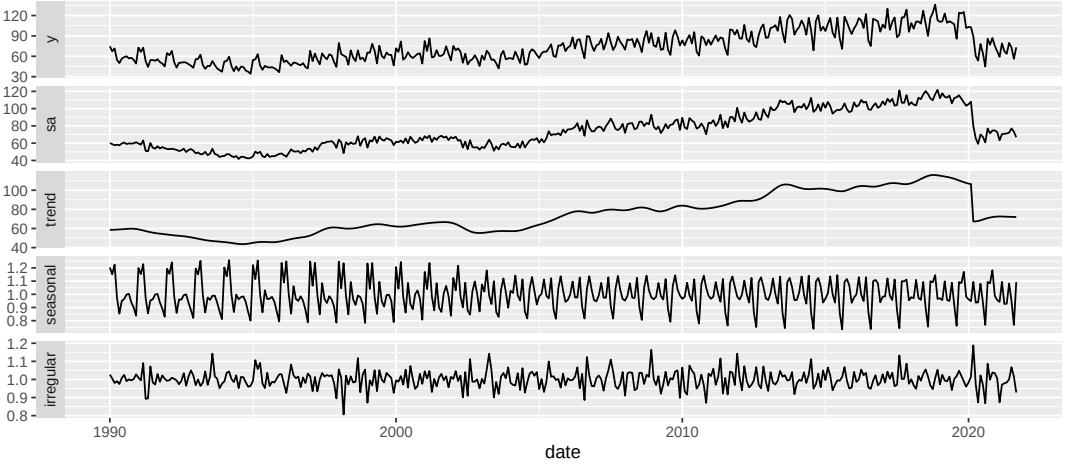
```
library("ggdemetra3")
siratioplot(sa_x13_v3)
```

Plots and data visualisation II




```
library("ggplot2")
autoplot(sa_x13_v3)
```

Plots and data visualisation III



Subsection 5

Customizing specifications

Customising specifications: general steps

To customise a specification:

- start with a valid specification, usually one of the default specs (equivalent to cloning a spec in GUI)
- create a new specification
- apply the new specification to raw series

Customising specifications: local functions

Use of specific `set_` functions

- for the pre-processing step (functions defined in `rjd3toolkit`):

`set_arima()`, `set_automodel()`, `set_basic()`, `set_easter()`, `set_estimate()`, `set_outlier()`, `set_tradingdays()`, `set_transform()`, `add_outlier()` and `remove_outlier()`, `add_ramp()` and `remove_ramp()`, `add_usrdefvar()`

- for the decomposition step with X11 (function defined in `rjd3x13`): `set_x11()`
- for the decomposition step with Tramo-Seats (function defined in `rjd3tramoseats`): `set_seats()`
- for the benchmarking step (function defined in `rjd3toolkit`): `set_benchmarking()`

Simple examples I

```
# start with default spec
spec_1 <- spec_x13("RSA3")
# or start with existing spec (no extraction function needed)
# spec_1 <- sa_x13_v3_UD$estimation_spec
```

```
# set a new spec
## add outliers
spec_2 <- rjd3toolkit::add_outlier(spec_1,
    type = "AO", c("2015-01-01", "2010-01-01")
)
```

```
## set trading days
spec_3 <- rjd3toolkit::set_tradingdays(spec_2,
    option = "workingdays"
) # JD+ regressors
```

Simple examples II

```
# set x11 options
spec_4 <- set_x11(spec_3, henderson.filter = 13)
# apply with `fast.x13` (results only)
fast_x13(y_raw, spec_4)
```

```
Model: X-13
Log-transformation: yes
SARIMA model: (0,1,1) (0,1,1)

SARIMA coefficients:
 theta(1) btheta(1)
  -0.7440   -0.5815

Regression model:
```

Simple examples III

```
          td AO (2010-01-01) AO (2015-01-01) LS (2020-03-01)
    0.009482        0.128942       -0.004723       -0.468454

Seasonal filter: S3X3; Trend filter: H-13 terms
M-Statistics: q Good (0.792); q-m2 Good (0.794)
QS test on SA: Good (1.000); F-test on SA: Good (0.995)

For a more detailed output, use the 'summary()' function.
```

Adding user-defined calendar or other regressors

When adding regressors which are not predefined (like outliers or ramps):

- `rjd3toolkit::set_tradingdays` to be used when allocating a regressor to the **calendar** component
- `rjd3toolkit::add_usrdefvar` is used for any other component

Step 1: Creating regressors (1/2)

```
# create national (or other) calendar if needed
frenchCalendar <- national_calendar(days = list(
    fixed_day(7, 14), # Bastille Day
    fixed_day(5, 8, validity = list(start = "1982-05-08")), # End of 2nd WW
    special_day("NEWYEAR"),
    special_day("CHRISTMAS"),
    special_day("MAYDAY"),
    special_day("EASTERMONDAY"),
    special_day("ASCENSION"),
    special_day("WHITMONDAY"),
    special_day("ASSUMPTION"),
    special_day("ALLSAINTSDAY"),
    special_day("ARMISTICE")
))
```

Step 1: Creating regressors (2/2)

```
# create set of 6 regressors every day is different, contrast with Sunday, based on french nat
regs_td <- rjd3toolkit::calendar_td(
    calendar = frenchCalendar,
    # formats the regressor like your raw series (length, frequency..)
    s = y_raw,
    groups = c(1, 2, 3, 4, 5, 6, 0),
    contrasts = TRUE
)

# create an intervention variable (to be allocated to "trend")
iv1 <- intervention_variable(
    s = y_raw,
    starts = "2015-01-01",
    ends = "2015-12-01"
)
```

regressors can be any TS object

Step 2: Creating a modelling context

Modelling context is necessary for any external regressor (new v3 set up)

```
# Gather regressors into a list
my_regressors <- list(
    Monday = regs_td[, 1],
    Tuesday = regs_td[, 2],
    Wednesday = regs_td[, 3],
    Thursday = regs_td[, 4],
    Friday = regs_td[, 5],
    Saturday = regs_td[, 6],
    reg1 = iv1
)

# create modelling context
my_context <- modelling_context(variables = my_regressors)
# check variables present in modelling context
rjd3toolkit::.r2jd_modellingcontext(my_context)$getTsVariableDictionary()
```

Step 3: Adding regressors to specification (calendar)

```r
# Add calendar regressors to spec
x13_spec <- rjd3x13::x13_spec("rsa3")
x13_spec_user_defined <- rjd3toolkit::set_tradingdays(
    x = x13_spec,
    option = "UserDefined",
    uservariable = c(
        "r.Monday", "r.Tuesday", "r.Wednesday",
        "r.Thursday", "r.Friday", "r.Saturday"
    ),
    test = "None"
)
```

Step 3: Adding regressors to specification (trend)

```
# Add intervention variable to spec, choosing the component to allocate the e
x13_spec_user_defined <- add_usrdefvar(
    x = x13_spec_user_defined,
    group = "r",
    name = "reg1",
    label = "iv1",
    regeffect = "Trend"
)
```

```
x13_spec_user_defined$regarima$regression$users
```

Step 4: Estimating with context

Applying full user-defined specification

```
sa_x13_ud <- rjd3x13::x13(y_raw, x13_spec_user_defined, context = my_context)
sa_x13_ud$result$preprocessing
```

```
Log-transformation: yes
SARIMA model: (0,1,1) (0,1,1)

SARIMA coefficients:
 theta(1) btheta(1)
  -0.7393   -0.5659

Regression model:
         iv1       r.Monday      r.Tuesday    r.Wednesday     r.Thursday
   -0.045028       0.007131       0.016860       0.005819       0.003516
      r.Friday     r.Saturday LS (2020-03-01)
```

Seasonal adjustment of unusual frequencies with rjd3x11 plus I

Example of periodicity p: decomposition with **rjd3x11plus**

Production prices in agriculture : some products are not available several months.

But the data are not missing, they are just not available and never will be : it's structural !

```
fl <- read.csv("Data/fruits_legumes_base_2015_F4.csv", sep = ";")
fl[fl[] == 0] <- NA
strawberry <- fl$FL6
```

Seasonal adjustment of unusual frequencies with rjd3x11 plus II

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2018	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
2019	NA	NA	1.210	0.875	0.775	0.788	0.836	NA	NA	NA	NA	NA
2020	NA	NA	0.925	1.029	1.012	1.018	0.915	NA	NA	NA	NA	NA
2021	NA	NA	1.441	1.130	1.041	0.745	1.036	NA	NA	NA	NA	NA
2022	NA	NA	1.557	1.036	0.731	0.908	1.017	NA	NA	NA	NA	NA
2023	NA	NA	1.292	1.074	0.983	0.980	1.055	NA	NA	NA	NA	NA

Data treatment :

1. Remove the useless months

```
strawberry <- matrix(fl$FL6, nrow = 12)[, 10:14]
strawberry_cut <- as.numeric(strawberry[-c(1:2, 8:12), ])
```

2. Use the function `x11plus` from the package **{rjd3x11plus}**

Seasonal adjustment of unusual frequencies with rjd3x11 plus III

```
library("rjd3x11plus")
mod_strawberry <- x11plus(
    y = strawberry_cut,
    period = 5,
    mul = TRUE,
    trend.horizon = 5 + 2,  # 1/2 Filter length : not too long vs p
    trend.degree = 3,                          # Polynomial degree
    trend.kernel = "Henderson",                # Kernel function
    trend.asymmetric = "CutAndNormalize",      # Truncation method
    seas.s0 = "S3X1", seas.s1 = "S3X1",        # Seasonal filters
    extreme.lsig = 1.5, extreme.usig = 2.5   # Sigma-limits
)
```

3. Re-assemble the data

Seasonal adjustment of unusual frequencies with rjd3x11 plus IV

```
strawberry_sa <- mod_strawberry$decomposition$sa |>
    matrix(nrow = 5)
strawberry_sa <- as.numeric(rbind(NA, NA, strawberry_sa, NA, NA, NA, NA, NA))

y <- as.numeric(strawberry)
sa <- strawberry_sa
```

4. Plot the output

```
plot(y, type = "l")
lines(sa, col = "blue")
```

Seasonal adjustment of unusual frequencies with rjd3x11 plus V