TSACE Webinar updated in March 2023

Using JDemetra+ in R: from version 2 to version 3
Presentation 2: Seasonal adjustment in R

Anna Smyk and Tanguy Barthelemy
With the collaboration of Alain Quartier-la-tente

Contents

1. **Introduction**

2. **X13 (. . . and some Tramo-Seats)**

3. **SA of High-Frequency data**

4. **Generating User-defined auxilary variables**

5. **Time series tools**

6. **Conclusion**

Seasonal adjustment: common steps

- testing for seasonality (identify seasonal patterns for HF data)
- pre-treatment
- create customisezd variables for pre-treatment (e.g calendar regressors)
- decomposition
- retrieve output series
- retrieve diagnostics
- customize parameters
- refresh data
- . . .
- repeat..

This presentation will illustrate all this points, mainly in X13-Arima.

Context of use

Producing Seasonally adjusted series in R (with parameters customized according to needs and previous diagnostics)

- not being aware of JD+ GUI existence
- no workspace structure of data
- time series objects in R
- use exclusively JD+ algorithms and no other SA R packages (Seasonal, TBATS...)

All the examples are related to ONE series. For an entire data set you can of course use loops or `lapply()` type of functions

Contents

1. Introduction

2. X13 (. . . and some Tramo-Seats)

2.1 Quick Launch with default specifications

2.2 Retrieving output and data visualization

2.3 Customizing specifications

2.4 Refreshing data

3. SA of High-Frequency data

4. Generating User-defined auxilary variables

5. Time series tools

Running a Seasonal Adjustment processing (1)

In version 2

```
# X13
sa_x13_v2 <- RJDemetra::x13(y_raw, spec = "RSA5c")
# see help pages for default spec names, identical in v2 and v3
#Tramo-Seats
sa_ts_v2 <- RJDemetra::tramoseats(y_raw, spec = "RSAfull")
```

In version 3 (printed model identical to v2)

```
#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

In version 2

```
# Reg-Arima part from X13 only (different default spec names, cf help pages)
regA_v2 <- RJDemetra::regarima_x13(y_raw, spec = "RG5c")

# Tramo only
tramo_v2 <- RJDemetra::regarima_tramoseats(y_raw,spec = "TRfull")
```

In version 3 (not very different)

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

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

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

Running only decomposition

In version 2

```
# X11 (spec option)
X11_v2 <- RJDemetra::x13(y_raw, spec = "X11")

#Tramo-Seats ? you
#sa_ts_v2 <- RJDemetra::tramoseats(y_raw, spec = "RSAfull")
```

In version 3

```
#X11 is a specific function
x11_v3 <- rjd3x13::x11(y_raw) # specific function
#Seats: you need an arima model
```

"Model_sa" object structure in version 2 (1/2)

"Model_sa" is the resulting object of the estimation, it contains

- raw series
- parameters (specification)
- output series
- diagnostics

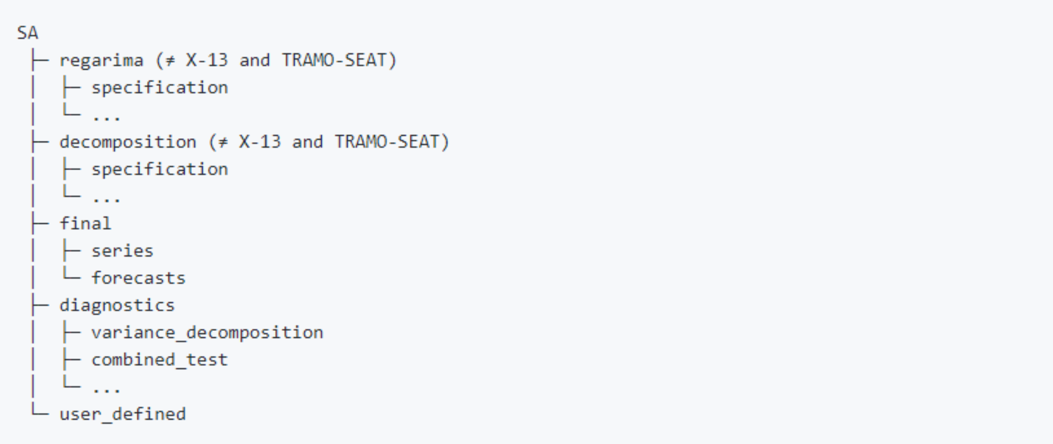
All arranged in a specific way

```r
# v2 "output"
Model_sa <- RJDemetra::x13(y_raw, spec = "RSA5")

Model_sa$regarima
Model_sa$decomposition
#...
```

"Model_sa" object structure in version 2

Organised by domain:



```
SA
 ├─ regarima (≠ X-13 and TRAMO-SEAT)
 │   ├─ specification
 │   └─ ...
 ├─ decomposition (≠ X-13 and TRAMO-SEAT)
 │   ├─ specification
 │   └─ ...
 ├─ final
 │   ├─ series
 │   └─ forecasts
 ├─ diagnostics
 │   ├─ variance_decomposition
 │   ├─ combined_test
 │   └─ ...
 └─ user_defined
```

{width = 90%}

"Model_sa" object structure in version 3

Results vs specification. . . and then by domain

```r
# 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
```

Differences from version 2 to version 3

In version 3

- specification is separated from results
- results are more specific (“X11” like series names in X13-Arima)
- specifications are directly available (no extraction function needed like in v2)
- two concepts of spec: estimation spec (domain) and result spec (point) in v3
- in v2 only only result spec (more about this in refresh section)

Retrieve output series

Input and output series are TS objects in R (not when using specific extensions for HF data)

- final series: different names and layout from v2 to v3

```
# Version 2 : display of Main Results table (from GUI)
sa_x13_v2$final$series #y, sa,t,s,i
sa_x13_v2$final$forecasts

# Version 3
# final seasonally adjusted series
sa_x13_v3$result$final$d11final
```

In version 3 much more series are available without using the user-defined output option.

Series from preadjustment

```
# Version 2
sa_x13_v2$regarima$model$effects #MTS object

# forecast accessible only via user defined output (cf below)

# Version 3: "x11 names" : pre-adjustment effects as stored in the A table
# add doc on names
sa_x13_v3$result$preadjust$a6
```

Series from decomposition

In version 2 - D tables accessible via user-defined output, - forecast series accessible only via user defined output (cf below)

In Version 3: “x11 names”

```
# Version 3
sa_x13_v3$result$decomposition$d5 # tables from D1 to D13
```

Retrieving Diagnostics

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

```
# Version 2
print(sa_x13_v2)
sa_x13_v2$decomposition$mstats
sa_x13_v2$decomposition$s_filter
sa_x13_v2$decomposition$t_filter

# version 3 (more diagnostics available by default)
print(sa_x13_v2)
sa_x13_v3$result$diagnostics$td.ftest.i
```

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

Retrieving user defined-output (1/2)

In version 2 or version 3: first define the vector of objects you wish to add

Lists of available diagnostics or series

```
# Version 2
RJDemetra::user_defined_variables("X13-ARIMA")
RJDemetra::user_defined_variables("TRAMO-SEATS")

# Version 3: more specific functions
rjd3tramoseats::userdefined_variables_tramoseats("tramoseats")
rjd3tramoseats::userdefined_variables_tramoseats("tramo") # restriction

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

Retrieve user defined-output (2/2)

Select the objects and customize estimation function (identical in v2 and v3)

```
# version 3
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)
sa_x13_v3_UD$user_defined # remainder of the names
```

```
## Names of additional variables (3):
## cal, cal_b, cal_b(?)
```

```
# retrieve the object
sa_x13_v3_UD$user_defined$decomposition.b1
```

```
## NULL
```

Plots and data visualisation in version 2 (1)

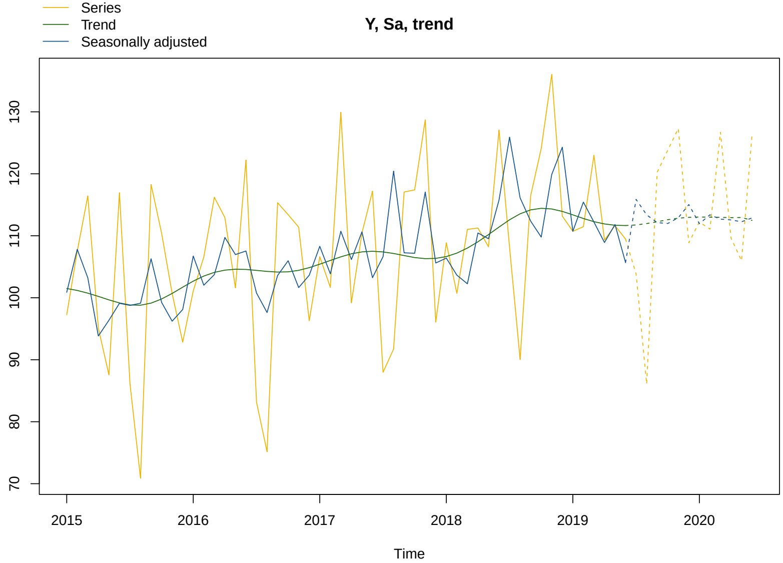
In version 2 three kinds of plots:

- final (2 types: plots identical to GUI main results)
- regarima residuals (6 plots)
- SI ratios

Plots and data visualisation in version 2 (1)

```
# Version 2
# for class 'final' : 2 types
plot(sa_x13_v2, type_chart = "sa-trend", first_date = c(2015, 1))
```

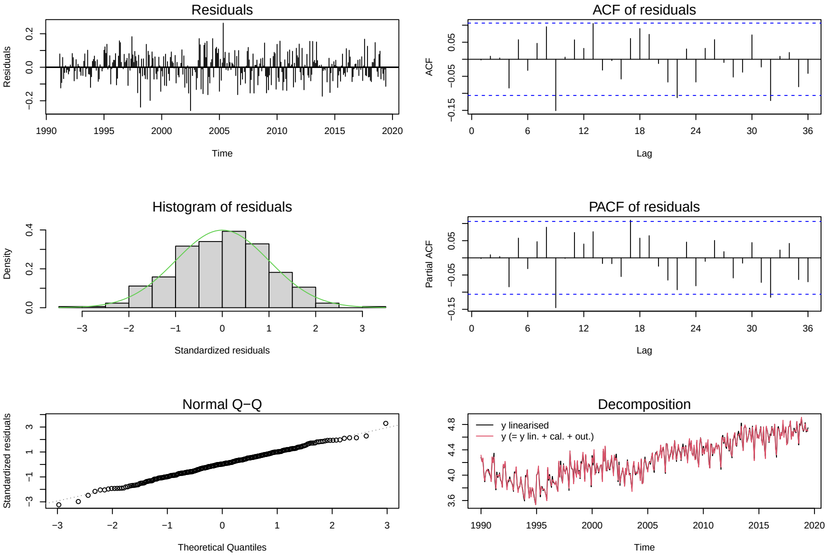
Plots and data visualisation in version 2 (2)



Plots and data visualisation in version 2 (1)

```
# regarima
layout(matrix(1:6, 3, 2))
plot(sa_x13_v2$regarima, ask = FALSE)
```

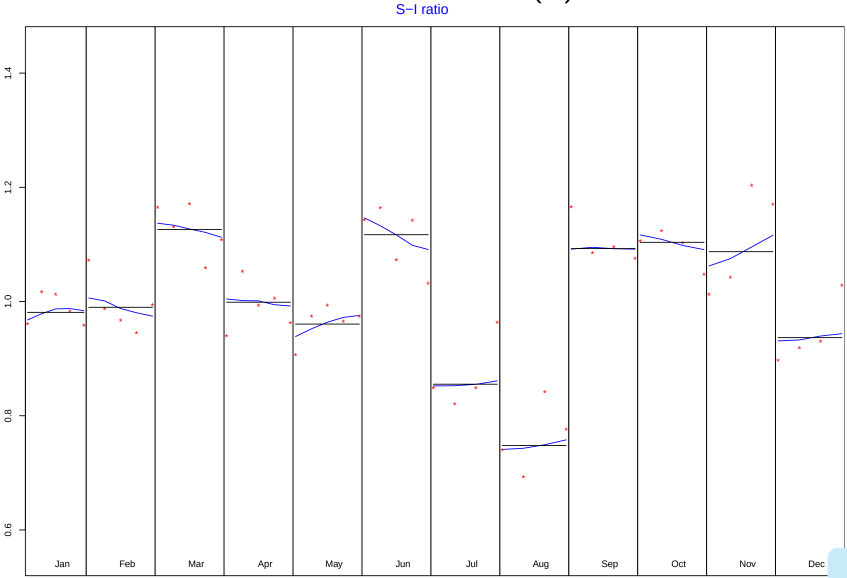
Plots and data visualisation in version 2 (2)



Plots and data visualisation in version 2 (1)

```r
# Plotting SI ratios
plot(sa_x13_v2$decomposition, first_date = c(2015, 1))
```

Plots and data visualisation in version 2 (2)



Plots and data visualisation in version 3 (1)

In version 3

- final + NEW “autoplot” layout
- regarima not available (yet ?)
- SI ratios + NEW ggplot layout

```
# version 3
# remotes::install_github("AQLT/ggdemetra3", INSTALL_opts = "--no-multiarch")
library("ggdemetra3")
ggdemetra3::siratioplot(sa_x13_v3)
```

Plots and data visualisation in version 3 (1)

```
# version 3
ggdemetra3::ggsiratioplot(sa_x13_v3)
```

Plots and data visualisation in version 3 (1)

```
# version 3
library("ggplot2")
ggplot2::autoplot(sa_x13_v3)
```

Customizing specifications: general steps

To customize a specification you must

- 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 your raw series

Some differences between v2 and v3

Customizing specifications in version 2

Direct parameter modification as arguments of the specification function

```
# version 2
# changing estimation span, imposing additive model and
#adding user defined outliers
# first create a new spec modifying the previous one
spec_1 <- x13_spec(sa_x13_v2) #extraction from the full model
spec_2 <- x13_spec(spec_1, estimate.from = "2004-01-01",
                   usrdef.outliersEnabled = TRUE,
                   usrdef.outliersType = c("LS", "AO"),
                   usrdef.outliersDate = c("2008-10-01", "2018-01-01"),
                   transform.function = "None") # additive model
# here the reg-arima model will be estimated from  "2004-01-01"
# the decomposition will be run on the whole span

# new sa processing
sa_x13_v2_2 <- RJDemetra::x13(y_raw, spec_2)
sa_x13_v2_2$final$series
```

Customizing specifications in version 3

Use direct and 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 in X13 (function defined in `rjd3x13`): `set_x11()`
- for the decomposition step in Tramo-Seats (function defined in `rjd3tramoseats`): `set_seats()`
- for the benchmarking step (function defined in `rjd3toolkit`): `set_benchmarking()`

Benchmarking New v3 feature, same options available as in GUI.

Customizing specifications in version 3: example

```
# 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 = c("AO"), c("2015-01-01", "2010-01-01"))
## set trading days
spec_3 <- rjd3toolkit::set_tradingdays(spec_2,
                                       option = "workingdays" )
# set x11 options
spec_4 <- set_x11(spec_3, henderson.filter = 13)
# apply with `fast.x13` (results only)
fast_x13(y_raw, spec_4)
```

Adding user-defined regressors

Differences:

In version 2: regressors added directly to the specification

In version 3: new notion of “context”: an additional concept designed to add any user defined (non standard, e.g non outlier”) variable

Adding user-defined regressors in v2

```
# defining user defined trading days
spec_td <- RJDemetra::x13_spec(
    spec = "RSA3",
    tradingdays.option = "UserDefined",
    tradingdays.test = "None",
    usrdef.varEnabled = TRUE,
    # the user defined variable will be assigned to the calendar component
    usrdef.varType = "Calendar",
    usrdef.var = td_regs ) # regressors have to be a single or multiple TS
# new sa processing
sa_x13_v2_4 <- RJDemetra::x13(y_raw, spec_td)
# user defined intervention variable
spec_int <- RJDemetra::x13_spec(
    spec = "RSA3",
    usrdef.varEnabled = TRUE,
    # the user defined variable will be assigned to the trend component
    usrdef.varType = "Trend",
    usrdef.var = x) # x has to to be a single or multiple TS
# new sa processing
sa_x13_v2_5 <- RJDemetra::x13(y_raw, spec_int)
```

Adding user-defined CALENDAR regressors in version 3

function rjd3toolkit::set_tradingdays is used when allocating a regressor to the calendar component, whereas rjd3toolkit::add_usrdefvar is used for any other component

```r
# step 1: define a user defined trading days regressor
td_reg1 <- rjd3toolkit::td(
    frequency = 12, start = start(y_raw),
    length = length(y_raw), groups = c(1, 1, 1, 1, 1, 0, 0))
# step 2: build new specification to customize or take an existing one
spec <- rjd3x13::spec_x13("RSA3")
# step 3: customize default specification
spec_ud_TD <- set_tradingdays(spec, option = "UserDefined",
                              uservariable = "regs.td_reg1")
# "regs.td_reg1": "group_name.variable_name: has to be the same as in context below

# NEW in V3: define a context (to add regressors)
# define a context
## step 1: create a list of regressors, and name the group
## here : regs = group name, td_reg1 = variable name
vars <- list(regs = list(td_reg1 = td_reg1))
## step 2: create context
my_context <- rjd3toolkit::modelling_context(variables = vars)

# New X13 estimation with user defined spec and corresponding context
sa_x13_v3_td <- rjd3x13::x13(y_raw, spec_ud_TD, context = my_context)
```

Adding user-defined regressors (Not Calendar) in version 3

```
# step 1: define a regressor, for example
x <- rjd3toolkit::intervention_variable(
    frequency = 12, start(y_raw), length(y_raw),
    starts = "2001-01-01", ends = "2001-12-01")
# step 2: build new specification to customize or take an existing one
spec <- rjd3x13::spec_x13("RSA3")
# step 3: customize default specification
spec_T <- add_usrdefvar(spec, name = "regs.x", regeffect = "Trend")

# "regs.x": "group_name.variable_name: has to be the same as in context below
# NEW in V3: define a context (to add regressors)
vars <- list(regs = list(x = x))
## step 2: create context
my_context_2 <- rjd3toolkit::modelling_context(variables = vars)

# New X13 estimation with user defined spec and corresponding context
sa_x13_v3_t <- rjd3x13::x13(y_raw, spec_T, context = my_context_2)
# to check results
sa_x13_v3_t$result$preprocessing
```

Refreshing data: Estimation_spec vs result_spec (1/2)

Possibility of refreshing data is a NEW feature of version 3.

In the “sa_model” object generated by the estimation process:

- specification is separated from results
- split in “estimation_spec” (domain spec): set of customizable constraints
- and “result_spec” (point spec)

```
sa_x13_v3$estimation_spec$regarima$arima
```

- result spec (or point spec)

```
sa_x13_v3$result_spec$regarima$arima
```

Estimation_spec vs result_spec

- in v2 could only retrieve a (point) result_spec (extracted with `x13_spec()` for example)
- in v3 your are able to re-estimate the "result_spec" inside a domain of constraints (estimation spec), freeing restrictions on selected parameters: just like in GUI, or Cruncher.

Steps for refreshing data

```r
current_result_spec <- sa_x13_v3$result_spec
current_domain_spec <- sa_x13_v3$estimation_spec

# generate NEW spec for refresh
refreshed_spec <- rjd3x13::x13_refresh(current_result_spec, # point spec to be refreshed
                                       current_domain_spec, #domain spec (set of constraints)
                                       policy = "Outliers",
                                       start = "2017-01-01",
                                       end = NULL)

# apply the new spec on new data : y_new = y_raw + 1 month

sa_x13_v3_refreshed <- rjd3x13::x13(y_new, refreshed_spec)
```

Outliers identification : more flexible than “last outliers” or “all outliers” in v2, here the span can be customized .

(Warning: x13_refresh hasn’t been thoroughly tested yet)

Refresh Policies

- "Complete": all reset to default but user defined parameters are stored ("Concurrent" in GUI)
- "Outliers_StochasticComponent" ("Arima Model" in GUI)
- "Outliers" ("Last Oultliers in GUI", but with flexible span for "last")
- "FreeParameters" ("ArimaParameters in GUI")
- "FixedParameters" ("Estimate Regression Coefficients" in GUI)
- "FixedAutoRegressiveParameters" (for Seats, NEW, like "ArimaParameters" bur AR coeffs fixed)
- "Fixed" ("Fixed Model" in GUI)

(see JDemetra+ documentation for complete description of the policies: https://jdemetra-new-documentation.netlify.app/t-rev-policies-production)

Contents

1. Introduction
2. X13 (. . . and some Tramo-Seats)
3. **SA of High-Frequency data**
4. Generating User-defined auxilary variables
5. Time series tools
6. Conclusion

SA of High-Frequency data (1/2)

Specificity: high-frequency data can display multiple and non integer periodicities:

For example a daily series might display 3 periodicities: - weekly ($p = 7$): Mondays are alike and different from Sundays (DOW) - intra-monthly ($p = 30.44$): the last days of each month are different from the first ones (DOM) - yearly ($p = 365.25$): from on year to another the 15th of June are alike, summer days are alike (DOY)

Two classes of solutions: - round periodicities (might involve imputing data) (extended STL,..) - use approximations for fractional backshift powers (extended X13-Arima and Tramo-Seats)

SA of High-Frequency data (2/2)

- Specific tools:
  rjd3highfreq and rjd3stl (version 3) (version 2 : rjdhighfreq)

Different data format: numeric vectors (and NOT TS objects)

- linerarization with **fractional airline model** (correction for calendar effects and outlier detection)
- iterative decomposition (extended X-11 and Seats) starting with the highest frequency

(See presentation about rjd3highfreq in Webinar GitHub Repo)

data initialization

```
df_daily <- read.csv2("../Data/TS_daily_births_franceM_1968_2020.csv") |>
    dplyr::mutate(log_births = log(births))
```

Linearization: code template

```r
# calendar regressors can be defined with the rjd3toolkit package
# see below how to generate the calendar (here frenchCalendar) first
q <- rjd3toolkit::holidays(
    calendar = frenchCalendar,
    "1968-01-01", length = 200000, type = "All", nonworking = 7L)
# pre-adjustment
rjd3highfreq::fractionalAirlineEstimation(
    y = df_daily$log_births, # here a daily series in log
    x = q, # q = calendar
    periods = 7, # approx  c(7,365.25)
    ndiff = 2, ar = FALSE, mean = FALSE,
    outliers = c("ao","wo","LS"),
    # WO compensation
    criticalValue = 0, # computed in the algorithm
    precision = 1e-9, approximateHessian = TRUE)
```

See {rjd3highfreq} help pages

Decomposition with extended X-11: code template

```
#step 1: p = 7
x11.dow <- rjd3highfreq::x11(
    ts = exp(pre.mult$model$linearized),
    period = 7,                 # DOW pattern
    mul = TRUE,
    trend.horizon = 9,  # 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 = "S3X9", seas.s1 = "S3X9",       # Seasonal filters
    extreme.lsig = 1.5, extreme.usig = 2.5)   # Sigma-limits
#step 2: p = 365.25
x11.doy <- rjd3highfreq::x11(x11.dow$decomposition$sa,  # previous sa
                             period = 365.2425,         # DOY pattern
                             mul = TRUE) #other parameters skipped here
```

Decomposition with extended Seats: code template

```
#step 1: p = 7
#step 2: p = 365.25
amb.doy <- rjd3highfreq::fractionalAirlineDecomposition(
    amb.dow$decomposition$sa,  # DOW-adjusted linearised data
    period = 365.2425,         # DOY pattern
    sn = FALSE,                # Signal (SA)-noise decomposition
    stde = FALSE,              # Compute standard deviations
    nbcasts = 0, nfcasts = 0)  # Numbers of back- and forecasts
```

Contents

1. Introduction

2. X13 (. . . and some Tramo-Seats)

3. SA of High-Frequency data

4. **Generating User-defined auxilary variables**

4.1 Calendars

4.2 Outliers and intervention variables

5. Time series tools

6. Conclusion

Calendars

New features of version 3:

- generating calendars in R (see GUI function in v2)
- generating calendar regressors
- raw number of days or contrasts
- long term mean correction or not
- user-defined groups of days
- user-defined contrast days (associated with holidays)

Can be done with rjd3toolkit package

Creation of a specific calendar

Example: French Calendar

```
library("rjd3toolkit")
# French
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'))
)
```

Creation of a associated regressors (1)

- For daily data: Use `holidays()` to get the days of the holidays, dummy variables

```
q <- rjd3tookit::holidays(frenchCalendar, start = "1968-01-01", length = 200000,
                          type = "All", nonworking = 7L)
```

- For monthly or quarterly data, aggregation by groups

In v3 flexible definition of groups and reference day

```
td_regs <- calendar_td(frenchCalendar,12, start = c(2000,1), length = 100,
                       groups = c(1, 1, 2, 2, 0, 3, 4),
                       # 1: Mondays = Tuesdays, 2 :Wednesdays = Thursdays
                       # 0: Fridays = reference for contrasts
                       # 3: Saturdays, 4: Sundays
                       holiday = 5, #day for aggregating holidays with (here Fridays)
                       contrasts = TRUE,
                       meanCorrection = contrasts
)
```

Outliers and intervention variables

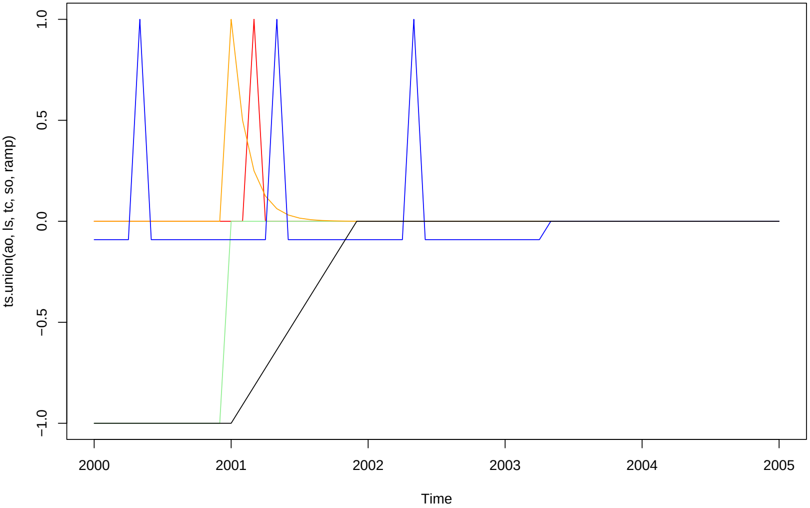
New feature of version 3 allows to create:

- outliers regressors (AO, LS, TC, SO, Ramp (quadratic to be added)
- trigonometric variables

Example of outliers (1)

```r
# 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"))
plot(ts.union(ao, ls, tc, so, ramp), plot.type = "single",
     col = c("red","lightgreen","orange","blue","black"))
```

Example of outliers (2)



Contents

1. Introduction
2. X13 (. . . and some Tramo-Seats)
3. SA of High-Frequency data
4. Generating User-defined auxilary variables
5. **Time series tools**
6. Conclusion

Time series tools: NEW features in version 3

The spirit of version 3 is to offer more tools from JDemetra+ libraries such as:

- tests (seasonality, normality, randomness, residual trading days effects) in rjd3toolkit
- autocorrelation functions partial and inverse
- arima model estimation and decomposition (`rjd3toolkit::ucrima_estimate()`)
- aggregation to higher frequency (`rjd3toolkit::aggregate()`)

More flexibility for the user as they can be applied any time not just as part of an SA processing.

Some of might also be available in other R packages. Arima model estimation is notoriously faster than other R functions.

Testing for seasonality

In rjd3toolkit:

- Canova-Hansen (`rjd3toolkit::seasonality.canovahansen()`) spectral
- 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()`)

Arima estimation

```
# JD+
print(system.time(
    for (i in 1:1000) {
        j <- rjd3toolkit::sarima_estimate(log(rjd3toolkit::ABS$X0.2.09.10.M),
                                          order = c(2, 1, 1), seasonal = list(order = c(0, 1, 1), pe
    }))
#       user    system        elapsed (in seconds)
#      4.98        0.37        4.63

#R-native
print(system.time(
    for (i in 1:1000) {
        r <- stats::arima(
            x = log(rjd3toolkit::ABS$X0.2.09.10.M),
            order = c(2, 1, 1), seasonal = list(order = c(0, 1, 1), period = 12))
    }))
#       user    system        elapsed (in seconds)
#     158.74       0.23      160.49

print(j$likelihood )
print(r)
```

Contents

1. Introduction
2. X13 (. . . and some Tramo-Seats)
3. SA of High-Frequency data
4. Generating User-defined auxilary variables
5. Time series tools
6. Conclusion

SA in R: What's new in v3 ?

Tests and time series tools

General and flexible definition of

- calendars
- auxilary variables

Refresh Policies

Direct setting of basic benchmarking