TSACE Webinar, Wednesday December 14th 2022

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

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

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")
sa_x13_v3
```

Running a Seasonal Adjustment processing (2)

```
## RegARIMA
## Log-transformation: yes
## SARIMA model:  (0,1,1) (0,1,1)
##
## Coefficients
##           Estimate Std. Error T-stat
## theta(1)  -0.72466    0.03740 -19.38
## btheta(1) -0.56372    0.04992 -11.29
##
## Regression model:
##            Estimate Std. Error T-stat
## monday     0.016430   0.008647  1.900
## tuesday    0.012493   0.008603  1.452
## wednesday  0.006496   0.008621  0.754
## thursday  -0.003046   0.008598 -0.354
## friday     0.019581   0.008638  2.267
## saturday  -0.020445   0.008608 -2.375
## easter    -0.045446   0.017158 -2.649
## Number of observations:  354
## Number of effective observations:  341
## Number of parameters:  10
```

Running a Seasonal Adjustment processing (3)

```
##
## Loglikelihood:  374.7681
## Adjusted loglikelihood:  -1077.716
##
## Standard error of the regression (ML estimate):  0.07999264
## AIC:  2175.432
## AICC:  2176.099
## BIC:  2213.751
##
##
## Decomposition
## Monitoring and Quality Assessment Statistics:
##      M stats
## m1    0.986
## m2    0.660
## m3    1.888
## m4    0.262
## m5    1.877
## m6    0.140
## m7    0.374
## m8    0.823
```

Running a Seasonal Adjustment processing (4)

```
## m9    0.441
## m10   0.557
## m11   0.484
## q     0.799
## qm2   0.816
##
## Final filters:
## Seasonal filter:
## Trend filter: 23 terms Henderson moving average
##
## Diagnostics
## Relative contribution of the components to the stationary
## portion of the variance in the original series,
## after the removal of the long term trend (in %)
##
##            Component
##  cycle        35.745
##  seasonal     49.917
##  irregular     6.601
##  calendar      2.726
##  others        0.000
```

Running a Seasonal Adjustment processing (5)

```
##  total        94.989
##
## Residual seasonality tests
##                  P.value
##  seas.ftest.i     0.977
##  seas.ftest.sa    0.992
##  seas.qstest.i    1.000
##  seas.qstest.sa   1.000
##  td.ftest.i       0.999
##  td.ftest.sa      0.999
##
##
## Final
## Last values
##             series       sa    trend seas       irr
## Jul 2018 108.12963 125.7729 112.5273    1 1.1177102
## Aug 2018  90.03625 116.6883 113.3824    1 1.0291574
## Sep 2018 116.46355 112.2071 113.8818    1 0.9852950
## Oct 2018 124.07923 109.5869 113.9926    1 0.9613510
## Nov 2018 136.04300 119.6826 113.7513    1 1.0521420
## Dec 2018 113.17850 124.2718 113.2503    1 1.0973202
```

Running a Seasonal Adjustment processing (6)

```
## Jan 2019 108.28574 108.9028 112.6145    1 0.9670404
## Feb 2019 110.21151 114.3220 111.9838    1 1.0208800
## Mar 2019 122.43580 111.6401 111.4757    1 1.0014743
## Apr 2019 108.64593 108.2331 111.1456    1 0.9737949
## May 2019 111.25296 111.2931 110.9967    1 1.0026702
## Jun 2019 109.35264 105.3926 111.0227    1 0.9492889
```

```
#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
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

```
# 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
```

Figure 1: V2 structure

“Model_sa” object structure in version 3

Results vs specification. . . and then by domain

```
# Model_sa = sa_x13_v3
sa_x13_v3<-RJDemetra::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 (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" : preadjustement effets 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 avaible diagnostics or series

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

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

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

Retrieve user defined-output (2/2)

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

```
# version 3
ud<-userdefined_variables_x13()[15:17] # b series
ud
```

```
## [1] "decomposition.b1"  "decomposition.b10"
## [3] "decomposition.b11"
```

```
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):
## decomposition.b1, decomposition.b10, decomposition.b11
```

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

```
##           Jan      Feb      Mar      Apr      May
## 1990 72.32302 67.87415 70.64560 56.56822 49.22295
## 1991 71.73786 67.08462 77.20924 50.20607 43.31947
## 1992 63.44092 61.27638 66.91835 51.81981 44.79343
## 1993 57.50439 56.72361 59.12162 47.06855 43.00137
```

Plots and data visualisation in version 2 (1)

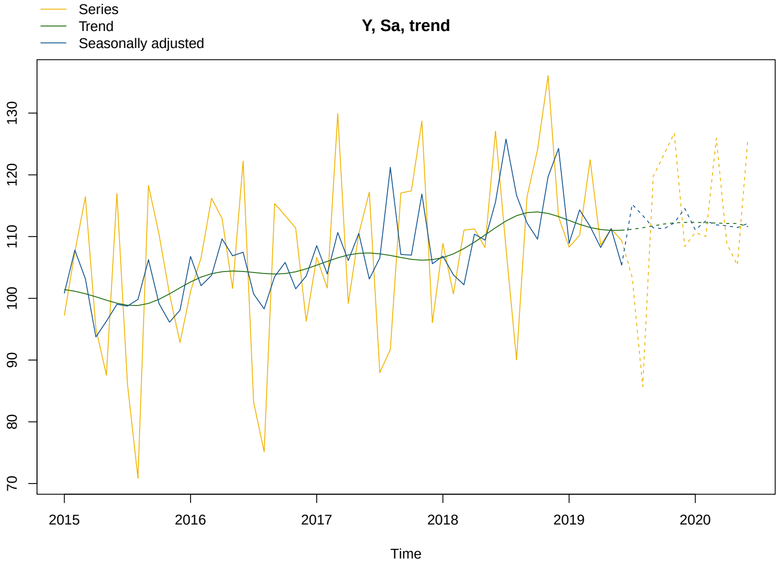
In version 2 three kinds of plots:

- 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)

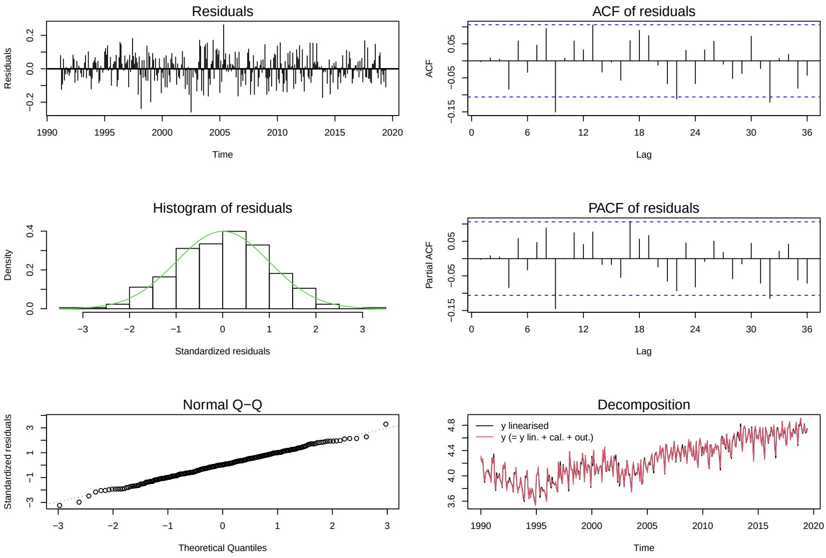



```
#plot(sa_x13_v2, type= "cal-seas-irr",first_date = c(2015, 1))
```

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)

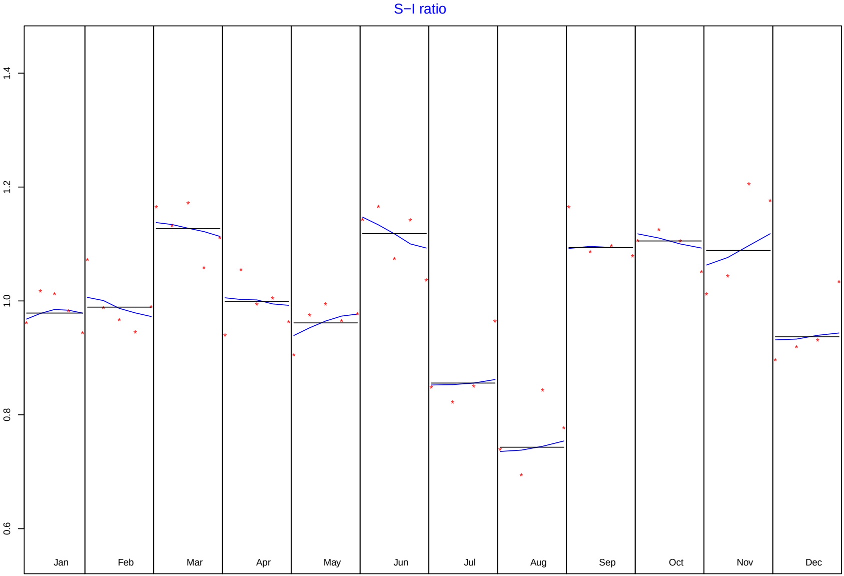



```
# Plotting SI ratios
```

Plots and data visualisation in version 2 (1)

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

Plots and data visualisation in version 2 (2)



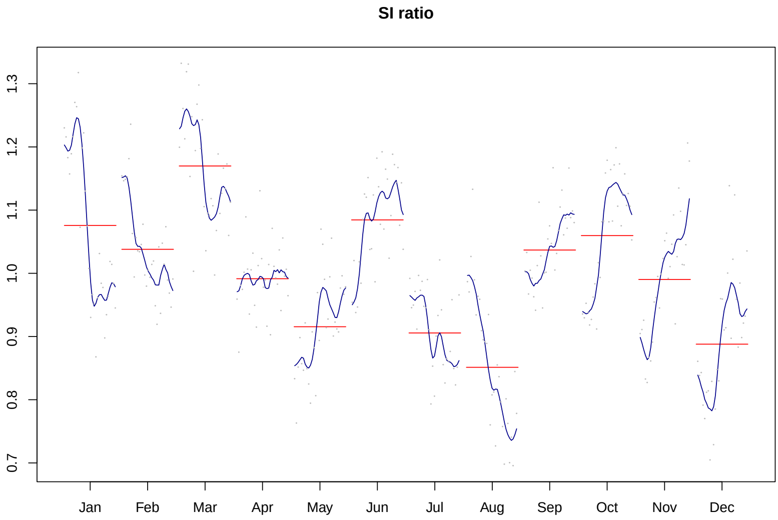
Plots and data visualisation in version 3 (1)

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)
```

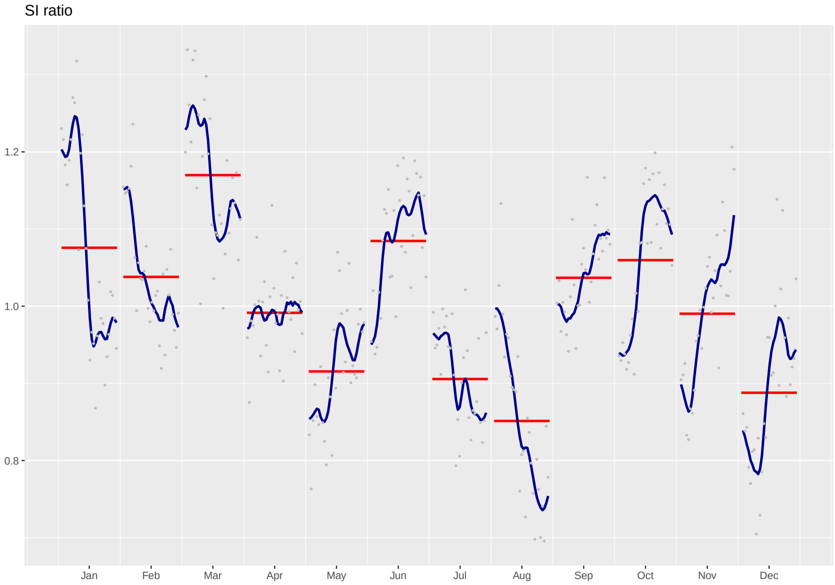
Plots and data visualisation in version 3 (2)



Plots and data visualisation in version 3 (1)

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

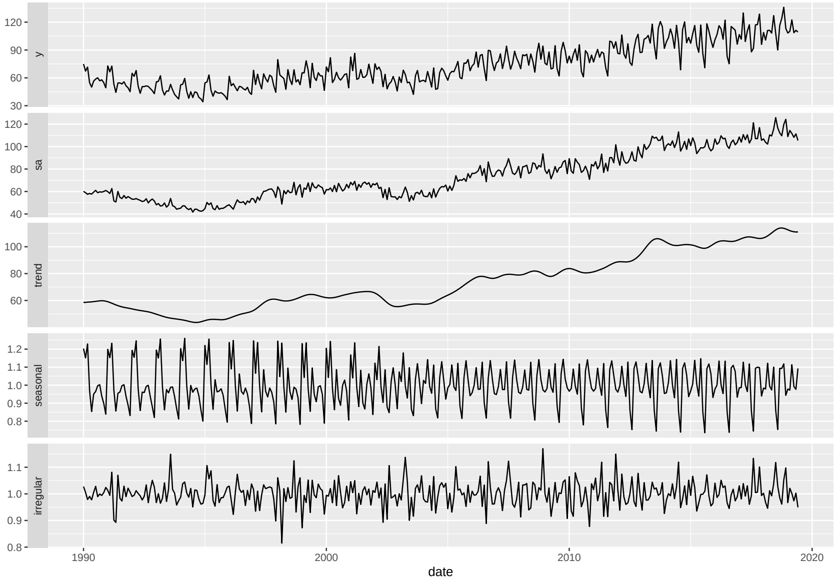
Plots and data visualisation in version 3 (2)



Plots and data visualisation in version 3 (1)

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

Plots and data visualisation in version 3 (2)



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 ouliers
# first create a new spec modifying the previous one
spec_1<- x13_spec(sa_x13_v2)
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 preprocessing step (functions defined in `rjd3modelling`):

`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 `rjd3modelling`): `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_default("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 = rjd3modelling::add_outlier(spec_1,
                  type = c("AO"), c("2015-01-01", "2010-01-01"))
## set trading days
spec_2<-rjd3modelling::set_tradingdays(spec_2,
    option = "workingdays" )
# set x11 options
spec_2<-set_x11(spec_2,henderson.filter = 13)
# apply with `fast.x13` (results only)
fast.x13(y,spec_2)
```

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 <- x13_spec(spec_1,
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<-x13(y_raw,spec_td)
# user defined intervention variable
spec_int <- x13_spec(spec_1,
                   usrdef.varEnabled = TRUE,
                   # the user defined variable will be assigned to the trend comp
                   usrdef.varType="Trend",
                   usrdef.var=x ) # x has to to be a single or multiple TS
# new sa processing
sa_x13_v2_5<-x13(y_raw,spec_int)
```

Adding user-defined regressors in version 3

```
# define a user defined trading days regressor
td_reg1<- rjd3modelling::td(12, start=start(y_raw),length = length(y_raw),groups

# define a context
my_context<-rjd3modelling::modelling_context(variables=list(a=xvar))

# set a new specification from a default specification
spec_td<- rjd3x13::spec_regarima_default(name = "rg3") |>
  rjd3modelling::add_usrdefvar(id = "r.a")

# new reg-arima estimation
reg_a_estimation<-rjd3x13::regarima(window(ts, start=1985, end=2013),spec_td,con
```

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

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

# generate NEW spec for refresh
refreshed_spec<-x13.refresh(current_result_spec, # point spec to be refreshed
            current_domain_spec, #domain spec (set of constraints)
            policy = "Outliers",
            period = 12, # monthly series
            start = "2017-01-01",
            end = NULL)

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

sa_x13_v3_refresh<-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

- “FreeParameters” : all reset to default
- “Complete”: all reset to default but user defined stored
- “Outliers_StochasticComponent”
- “Outliers”
- “FixedParameters”
- “FixedAutoRegressiveParameters” (for Seats)
- “Fixed”

Contents

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)

Linearization: code template

```
rjd3highfreq::fractionalAirlineEstimation
                        (df_daily$log_births, # here 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)

# calendar regressors can be defined with the rjd3modelling package
```

See {rjd3highfreq} help pages

Decomposition with extended X-11: code template

```
#step 1: p=7
x11.dow<-rjd3highfreq::x11(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

```r
#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

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 rjd3modelling package

Creation of a specific calendar

```
library(rjd3modelling)
# French
fr_cal <- calendar.new()
calendar.holiday(fr_cal, "NEWYEAR")
calendar.holiday(fr_cal, "EASTERMONDAY")
calendar.holiday(fr_cal, "MAYDAY")
calendar.fixedday(fr_cal, month = 5, day = 8,
                  start = "1982-01-01")
# calendar.holiday(fr_cal, "WHITMONDAY") # Equivalent to:
calendar.easter(fr_cal, offset = 61)

calendar.fixedday(fr_cal, month = 7, day = 14)
# calendar.holiday(fr_cal, "ASSUMPTION")
calendar.easter(fr_cal, offset = 61)
calendar.holiday(fr_cal, "ALLSAINTSDAY")
calendar.holiday(fr_cal, "ARMISTICE")
calendar.holiday(fr_cal, "CHRISTMAS")
```

Creation of a associated regressors (1)

Use `holidays()` to get the days of the holidays and `htd()` to get the trading days regressors

```
holidays(fr_cal, "2020-12-24", 10,single = T)
```

```
##              [,1]
## 2020-12-24    0
## 2020-12-25    1
## 2020-12-26    0
## 2020-12-27    0
## 2020-12-28    0
## 2020-12-29    0
## 2020-12-30    0
## 2020-12-31    0
## 2021-01-01    1
## 2021-01-02    0
```

Creation of a associated regressors (2)

```
s = ts(0, start = 2020, end = c(2020, 11), frequency = 12)
# Trading-days regressors (each day has a different effect, sunday as contrasts)
td_reg <- htd(fr_cal, s = s, groups = c(1, 2, 3, 4, 5, 6, 0))
# Working-days regressors (Monday = ... = Friday; Saturday = Sunday = contrasts)
wd_reg <- htd(fr_cal, s = s, groups = c(1, 1, 1, 1, 1, 0, 0))
# Monday = ... = Friday; Saturday; Sunday = contrasts
wd_reg <- htd(fr_cal, s = s, groups = c(1, 1, 1, 1, 1, 2, 0))
wd_reg
```

```
##             group-1    group-2
## Jan 2020  2.0000000  0.0000000
## Feb 2020  0.0000000  1.0000000
## Mar 2020 -1.7809251 -0.7968209
## Apr 2020  0.7809251 -0.2031791
## May 2020 -3.1554920  0.4740847
## Jun 2020  5.1554920  0.5259153
## Jul 2020  2.0000000  0.0000000
## Aug 2020 -4.0000000  0.0000000
## Sep 2020  2.0000000  0.0000000
## Oct 2020  2.0000000  1.0000000
```

Creation of a associated regressors (3)

```
## Nov 2020  0.0000000  0.0000000
```

```
# Monday = ... = Wednesday; Thursday; Friday = contrasts
wd_reg2<- htd(fr_cal, s = s, groups = c(1, 1, 1, 2, 0, 1, 1))
wd_reg2
```

```
##            group-1    group-2
## Jan 2020 -5.000000  0.0000000
## Feb 2020  1.000000  0.0000000
## Mar 2020  3.000000  0.0000000
## Apr 2020  1.000000  1.0000000
## May 2020  4.155492  0.5259153
## Jun 2020 -1.155492 -0.5259153
## Jul 2020 -5.000000  0.0000000
## Aug 2020  3.000000  0.0000000
## Sep 2020  2.000000  0.0000000
## Oct 2020 -4.000000  0.0000000
## Nov 2020  0.000000  0.0000000
```

Outliers and intervention variables

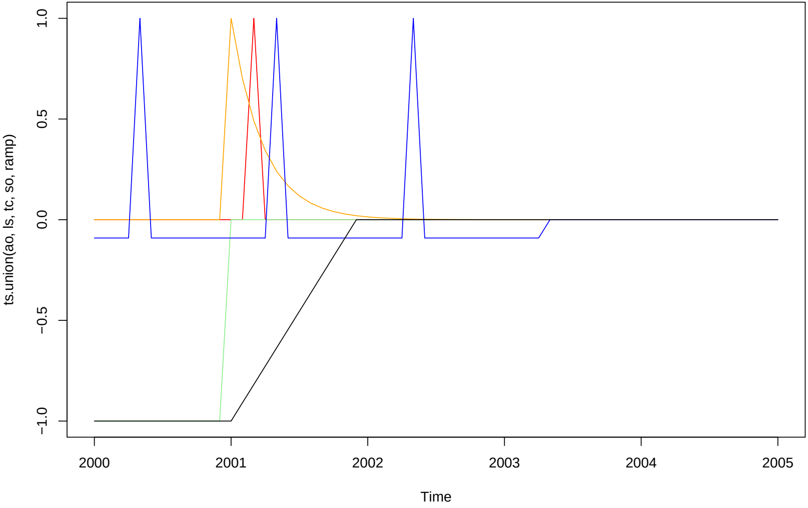
New feature of version 3 allows to create:

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

Example of outliers (1)

```
s = ts(0, start = 2000, end = 2005, frequency = 12)
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 = 0.7) # 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

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 dayseffects) in rjd3tookit, rjd3modelling and rjd3sa packages
- autocorrelation functions (in rjd3toolkit), incl partial and inverse
- arima model estimation and decomposition (rjd3modelling)
- 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 rjd3sa:

- Canova-Hansen (`seasonality.canovahansen()`) spctral, allows identifying patterns in HF data
- X-12 combined test (`seasonality.combined()`)
- F-test on seasonal dummies (`seasonality.f()`)
- Friedman Seasonality Test (`seasonality.friedman()`)
- Kruskall-Wallis Seasonality Test (`seasonality.kruskalwallis()`)
- Periodogram Seasonality Test (`seasonality.periodogram()`)
- QS Seasonality Test (`seasonality.qs()`)

Arima estimation

```
# JD+
print(system.time(for (i in 1:1000) {  j<-rjd3modelling::sarima.estimate(log(rjd3

#        user     system          elapsed (in seconds)
#       4.98         0.37          4.63

#R-native
print(system.time(for (i in 1:1000) {  r<-arima(log(rjd3toolkit::ABS$X0.2.09.10.I
print(j$likelihood )
print(r)

#        user     system          elapsed (in seconds)
#      158.74         0.23       160.49
```

Contents

SA in R: What's new in v3 ?

Tests and time series tools

General and flexible defintion of

```
- calendars

- auxilary variables
```

Refresh Policies

Direct setting of basic benchmarking