ESTP: Introduction to seasonal adjustment

The RJDemetra package and the R ecosystem

Anna Smyk (Insee) - Daniel Ollech (Deutsche Bundesbank)

Table of Contents

RJDemetra

RJDemetra is a package which allows to

- run (some) JDemetra+ SA algorithms (X13-Arima and Tramoseats) directly in R (without knowing what is a workspace or that GUI even exists..)
- read existing workspaces (created in GUI) : series, parameters and diagnostics directly in R

: https://github.com/jdemetra/rjdemetra

The RJDemetra package website : https://jdemetra.github.io/rjdemetra/

To install it :

```
install.packages("RJDemetra")
```

Functionalities

- SA algorithms : TRAMO-SEATS and x13-ARIMA
  - predefined and customized specifications
  - graphics
  - diagnostics
  - fine-tune parameters tuning

- Handling JD+ workspaces
  - Import workspaces with seasonal adjustment specifications
  - Xml export of the models created in RJDemetra into a workspace (compatible with JDemetra+ but not “crunchable”)

- Contains a database : the Industrial Production Indexes for the manufacturing industry for the EU countries and GB.

RegARIMA : examples (1/4)

```
library(RJDemetra)
ipi_fr <- ipi_c_eu[,"FR"]
regarima_model <- regarima_x13(ipi_fr, spec = "RG4c")
regarima_model
```

```
## y = regression model + arima (2, 1, 1, 0, 1, 1)
## Log-transformation: no
## Coefficients:
##             Estimate Std. Error
## Phi(1)       0.05291      0.108
## Phi(2)       0.18672      0.074
## Theta(1)    -0.52137      0.103
## BTheta(1)   -0.66132      0.042
##
##                Estimate Std. Error
## Week days        0.6927      0.031
## Leap year        2.0903      0.694
## Easter [1]      -2.5476      0.442
## TC (4-2020)    -35.6481      2.092
## AO (3-2020)    -21.1492      2.122
## AO (5-2011)     13.1869      1.810
## LS (11-2008)    -9.2744      1.758
##
```

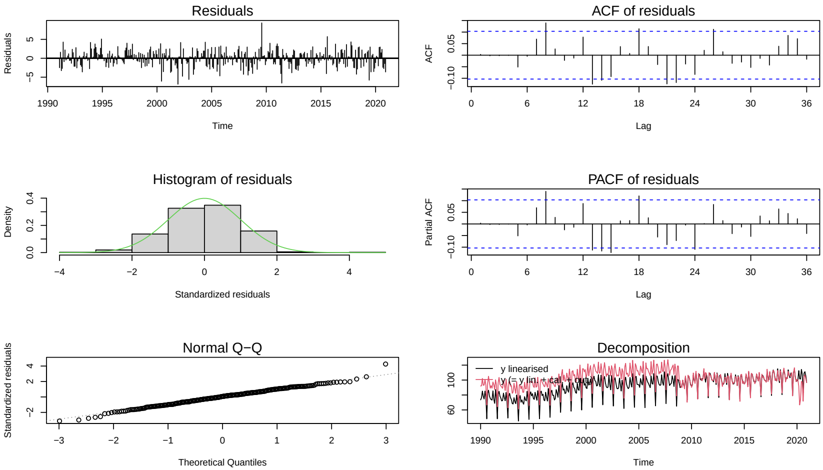
RegARIMA : examples (2/4)

```
summary(regarima_model)
```

```
## y = regression model + arima (2, 1, 1, 0, 1, 1)
##
## Model: RegARIMA - X13
## Estimation span: from 1-1990 to 12-2020
## Log-transformation: no
## Regression model: no mean, trading days effect(2), leap year effect, Easter e
##
## Coefficients:
## ARIMA:
##           Estimate Std. Error  T-stat Pr(>|t|)
## Phi(1)     0.05291    0.10751   0.492    0.623
## Phi(2)     0.18672    0.07397   2.524    0.012 *
## Theta(1)  -0.52137    0.10270  -5.076 6.19e-07 ***
## BTheta(1) -0.66132    0.04222 -15.665  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Regression model:
##               Estimate Std. Error  T-stat Pr(>|t|)
## Week days      0.69265    0.03143  22.039  < 2e-16 ***
## Easter [1]    -2.54756    0.44179  -5.766 1.75e-08 ***
```

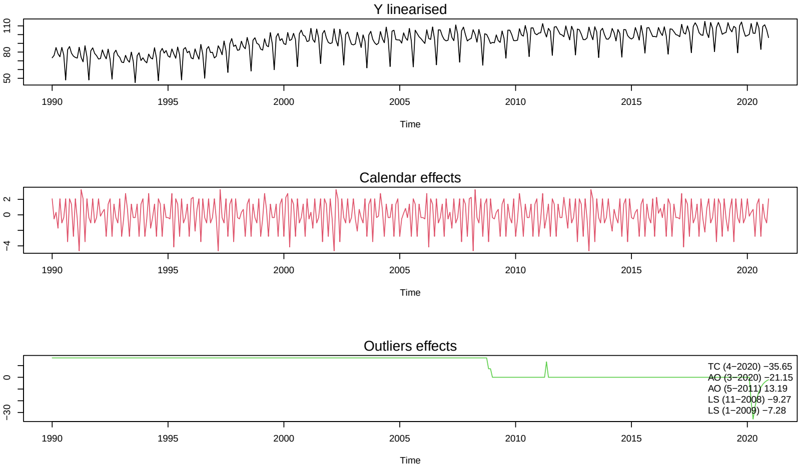
RegARIMA : examples (3/4)

```
layout(matrix(1:6, 3, 2));plot(regarima_model, ask = FALSE)
```



RegARIMA : examples (4/4)

```
plot(regarima_model, which = 7)
```



Seasonal adjustment : examples (1/8)

In R, an adjusted series (`SA`) is a `list()` of 5 elements :

```
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
```

Seasonal adjustment : examples (2/8)

Specifications can be the same by default as in JD+ or customized just like in the graphical interface :

```
x13_usr_spec <- x13_spec(spec = c("RSA5c"),
                         usrdef.outliersEnabled = TRUE,
                         usrdef.outliersType = c("LS", "AO"),
                         usrdef.outliersDate = c("2008-10-01",
                                                 "2002-01-01"),
                         usrdef.outliersCoef = c(36, 14),
                         transform.function = "None")
x13_mod <- x13(ipi_fr, x13_usr_spec)
ts_mod <- tramoseats(ipi_fr, spec = "RSAfull")
```

Seasonal adjustment : examples (3/8) : decomposition

```
x13_mod$decomposition
```

```
##  Monitoring and Quality Assessment Statistics:
##        M stats
## M(1)    0.151
## M(2)    0.097
## M(3)    1.206
## M(4)    0.558
## M(5)    1.041
## M(6)    0.037
## M(7)    0.082
## M(8)    0.242
## M(9)    0.062
## M(10)   0.267
## M(11)   0.252
## Q       0.366
## Q-M2    0.399
##
## Final filters:
## Seasonal filter:  3x5
```

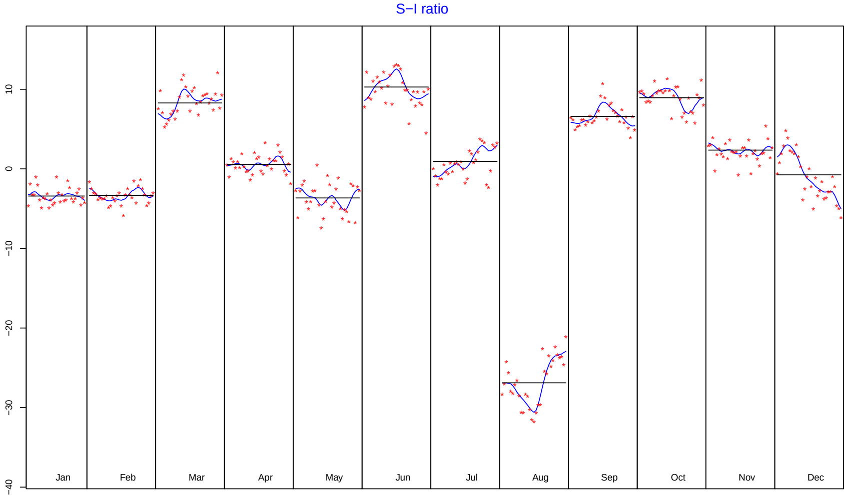
Seasonal adjustment : examples (4/8) : decomposition

```r
ts_mod$decomposition
```

```
## Model
## AR :  1 + 0.403230 B + 0.288342 B^2
## D :  1 - B - B^12 + B^13
## MA :  1 - 0.664088 B^12
##
##
## SA
## AR :  1 + 0.403230 B + 0.288342 B^2
## D :  1 - 2.000000 B + B^2
## MA :  1 - 0.970348 B + 0.005940 B^2 - 0.005813 B^3 + 0.003576 B^4
## Innovation variance:  0.7043507
##
## Trend
## D :  1 - 2.000000 B + B^2
## MA :  1 + 0.033519 B - 0.966481 B^2
## Innovation variance:  0.06093642
##
## Seasonal
```

Seasonal adjustment : examples (5/8)

```
plot(x13_mod$decomposition)
```



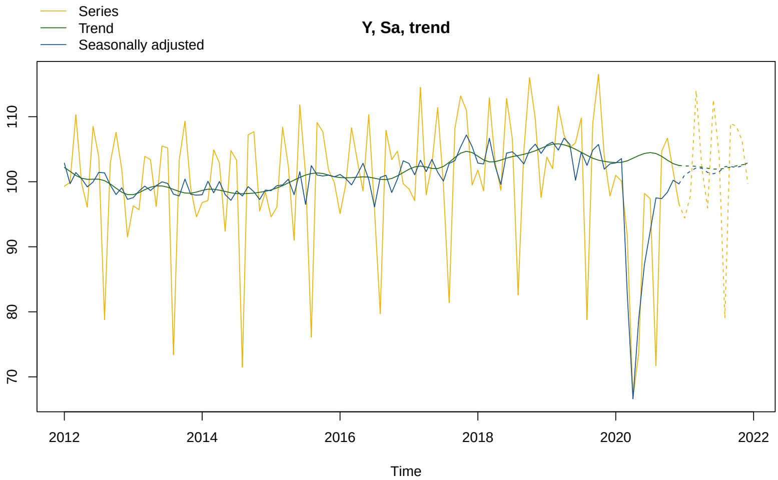
Seasonal adjustment : examples (6/8)

```
x13_mod$final
```

```
## Last observed values
##                  y        sa        t           s          i
## Jan 2020 101.0 102.89447 102.9447  -1.89446776  -0.0502488
## Feb 2020 100.1 103.56224 102.9860  -3.46224124   0.5762734
## Mar 2020  91.8  82.81896 103.2071   8.98103618 -20.3881828
## Apr 2020  66.7  66.62390 103.6164   0.07610348 -36.9925073
## May 2020  73.7  78.88976 104.0255  -5.18976181 -25.1357871
## Jun 2020  98.2  87.30845 104.3450  10.89154932 -17.0365408
## Jul 2020  97.4  92.39390 104.4861   5.00609785 -12.0921816
## Aug 2020  71.7  97.51560 104.3380 -25.81559971  -6.8224392
## Sep 2020 104.7  97.40102 103.9044   7.29897634  -6.5033820
## Oct 2020 106.7  98.39408 103.3109   8.30592464  -4.9168409
## Nov 2020 101.6 100.23574 102.7824   1.36426365  -2.5467131
## Dec 2020  96.6  99.67219 102.4984  -3.07218537  -2.8261840
##
## Forecasts:
##                     y_f      sa_f      t_f          s_f         i_f
## Jan 2021  94.41766 101.0272 102.4220  -6.60952495 -1.39481900
## Feb 2021  97.82331 101.6172 102.4196  -3.79385040 -0.80247216
## Mar 2021 114.01485 102.1273 102.3712  11.88751670 -0.24388469
```

Seasonal adjustment : examples (7/8)

```
plot(x13_mod$final, first_date = 2012, type_chart = "sa-trend")
```



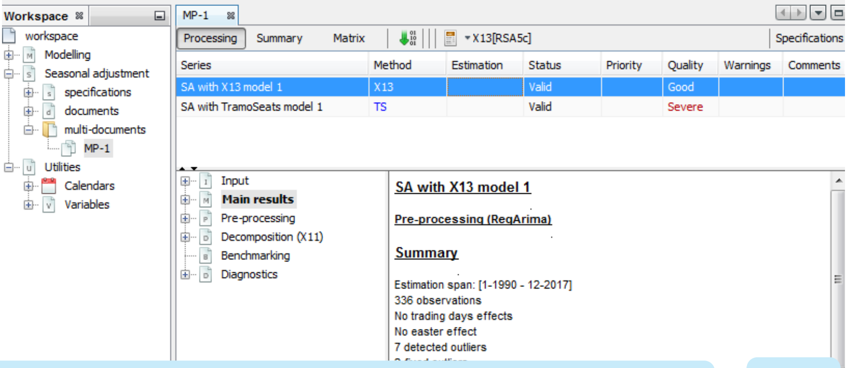
Seasonal adjustment : examples (8/8)

```
x13_mod$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
##  Trend computed by Hodrick-Prescott filter (cycle length = 8.0 years)
##            Component
##  Cycle         1.625
##  Seasonal     41.918
##  Irregular     0.727
##  TD & Hol.     1.851
##  Others       55.678
##  Total       101.800
##
##  Combined test in the entire series
##  Non parametric tests for stable seasonality
##                                                               P.value
##    Kruskall-Wallis test                                         0.000
##    Test for the presence of seasonality assuming stability      0.000
##    Evolutive seasonality test                                   0.042
##
##  Identifiable seasonality present
##
##  Residual seasonality tests
```

Workspace export

```
wk <- new_workspace()
new_multiprocessing(wk, name = "MP-1")
add_sa_item(wk, multiprocessing = "MP-1",
            sa_obj = x13_mod, name =  "SA with X13 model 1 ")
add_sa_item(wk, multiprocessing =  "MP-1",
            sa_obj = ts_mod, name = "SA with TramoSeats model 1")
save_workspace(wk, "workspace.xml")
```



Workspace import

```
wk <- load_workspace("workspace.xml")
compute(wk) # Important to get the Sa model
models <- get_model(wk) # A progress bar is printed by default
```

```
## Multiprocessing 1 on 1:
##    |
```

```
# To extract only one model
mp <- get_object(wk, 1)
count(mp)
```

```
## [1] 2
```

```
sa2 <- get_object(mp,2)
get_name(sa2)
```

```
## [1] "SA with TramoSeats model 1"
```

Analysis of the JDemetra+ output in R

- R enables the user to import and read output files generated when tuning series in the graphical interface
- it makes comparing different versions of the same series easier : raw and adjusted, adjusted with different parameters, from different workspaces, etc., without having to generate a custom JD+ output. Plus, comparison tables are refreshable as the analysis progresses, “in real time”.

Detailed example in the program “Retrieving_ONE_series_in_N_WS_with_RJD.R”

The other packages built on RJDemetra

- (**rjdqa** : qa = "quality assessment", produces a dashboard per series)
- **rjdmarkdown** : automatic report generation (parameters and diagnostics)
- **ggdemetra** : enriched graphics
- **rjdworkspace** : workspace handling and merging

RJDemetra doesn't provide refresh policies. (Has been fixed in rjd3 family)

Detailed worked examples

- To use the JDemetra+ SA algorithms without having to open the graphical interface or the cruncher :

CVS_in_R_with_RJD.R

- To access the output generated in JDemetra+ and make comparisons easily :

Retrieve_ONE_series_in_N_WS_with_RJD.R

- Working paper : "R tools for JDemetra+ : Seasonal adjustment made easier" see. Biblio repository

Table of Contents

JDemetra+ : a (wide) library of algorithms for time series analysis

JDemetra+ is a library of algorithms on :

- Seasonal Adjustment (GUI and R)
- Trend and cycle estimation (R only)
- Benchmarking and temporal disaggregation (GUI and R)
- Nowcasting (R and Plug-in)

They can be accessed via graphical user-interface (GUI) and/or R and/or plug-ins.

The available output, functions or tools are in general not identical using R or GUI for example, it is often fruitful to combine them.

JDemetra+ also offers general utilities for time series analysis : tests, auto-correlation functions, Arima modelling, spectral analysis tools . . .(in GUI and R)

A bit of History

Before 2019 : access only through GUI and plug-ins.

Why add R packages ?

Allows to immerse JD+ algorithms in the R universe, with all its pre-existing statistical functions and user-community.

In March 2019, `RJDemetra` (containing X-13 Arima and Tramo-Seats) was published on CRAN :

- first R package that enables to use Tramo-Seats
- faster than existing R packages on seasonal adjustment

Ever-growing R ecosystem

Since, many more packages have been developed as JDemetra+ Core was upgraded from version 2 to version 3

Extension of scope :

- High-frequency data (extended)
- STL
- refresh policies for SA
- new tools

Modifications of existing functions and output organisation

Table of Contents

From V2 to v3 : changing gears

Version 2 :

- RJDemetra (X13-Arima, Tramo-Seats, functions for workspace wrangling)
- additional packages based on RJDemetra : rjdworkspace, ggdemetra, rjdqa (see paper on V2)
- more specific packages like rjdhighfreq, rjdsts

(corresponding to the version 2.x family of JD+ Core)

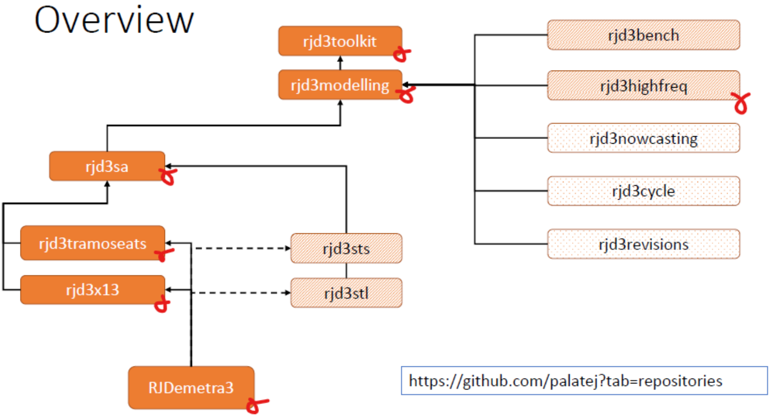
The mindset of version 3 is :

- modular organisation : independent more specific functions
- more "stand alone" tools (not only retrieving results from SA processing)

Version 3 : A suite of packages (see below)

- corresponding to the version 3.x family of JD+ Core
- still under construction, moving perimeters

Organisation of the rjd3 packages



Seasonal adjustment algorithms in JDemetra+ version 3

For seasonal adjustment, specifically, JDemetra+ contains :

- X13-Arima (GUI and R)
- Tramo-Seats (GUI and R)
- STL (R only)
- Structural Time Series (SSF framework) (R only)

All of these algorithms can be used with HF data.

For X13-Arima and Tramo-Seats some illustrations are available only in GUI. (auxiliary tools like : spectral analysis, sliding spans, revision history...)

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

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

Ressources on R packages (version 2 and 3)

Material from a webinar on the subject

- slides (pdf)
- slides with code (rmd)
- video
- additional documentation and links

https://github.com/annasmyk/Tsace_RJD_Webinar_Dec22