# JDemetra+ v3.x R ecosystem

## Seasonal adjustment using rjd3: selected issues

Anna Smyk - Insee

February 28th 2024

# Table of Contents I

Section 1

# Selected issues

# Selected issues

Last webinar highlighted differences v2 vs v3 on all the steps of an SA process

Here we will just revisit the parts where the functions scope, names and arguments have changed

SA of low frequency data

- customising a specification: adding calendar and other external regressors
- refreshing data (focus on spec object, test and show in GUI ?)

SA of high frequency data

- changes in decomposition packages
- new prints and plots in `rjd3highfreq`

Section 2

# Customising 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 (new v3 set up)

- 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()`

Benchmarking New v3 feature, same options available as in GUI (already in v 2.x).

# Simple examples

```
# 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" ) #JD+ regressors
# 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 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
library("rjd3toolkit")
# French ca
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
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

```
### 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"
)
```

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

x13_spec_d$regarima$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
```

The process would be identical using `rjd3tramoseats::tramoseats`

Section 3

# Refreshing data

# Refreshing data: Estimation_spec vs result_spec

Possibility of refreshing data is a **new** feature of version 3

Convenient option if production process fully in R with TS objects (no workspace structure)

In the "sa_model" object generated by the estimation process:

- specification is separated from results

```
# Model_sa = sa_x13_v3
sa_x13_v3 <- rjd3x13::x13(y_raw, spec = "RSA3")
sa_x13_v3$result
sa_x13_v3$estimation_spec
sa_x13_v3$result_spec
sa_x13_v3$user_defined
```

# Refreshing data: estimation_spec vs result_spec

In the output object, the specification is split in two:

- "estimation_spec" (domain spec): set constraints defining the estimation process, can be a default spec ("RSA3") or a user defined-spec (e.g RSA3 + calendar regressors...)
- "result_spec" (point spec): result of the estimation process, contains selected model, estimated coefficients...enough information so that if applied to raw series would allow to retrieve all output (sa, s, cal...)
- in v3.x possibility to re-estimate the "result_spec" inside a domain of constraints (estimation spec), freeing only restrictions on selected parameters (just like in GUI, or Cruncher in v2.x)

# Estimation_spec vs result_spec: an example (1/2)

- estimation spec

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

```
SARIMA model:  (0,1,1) (0,1,1)

Coefficients
          Estimate      Type
theta(1)         0 UNDEFINED
btheta(1)        0 UNDEFINED
```

# Estimation_spec vs result_spec: an example (2/2)

- result spec (or point spec)

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

```
SARIMA model:  (3,1,1) (0,1,1)

Coefficients
             Estimate      Type
phi(1)     0.07759472 ESTIMATED
phi(2)    -0.05319340 ESTIMATED
phi(3)    -0.09894132 ESTIMATED
theta(1)  -0.77199495 ESTIMATED
btheta(1) -0.55342125 ESTIMATED
```

# Refresh Policies (1/2)

**Fixed**: applying the current pre-adjustment reg-arima model and replacing forecasts by new raw data points.

**FixedParameters**: pre-adjustment reg-arima model is partially modified: regression coefficients will be re-estimated but regression variables, Arima orders and coefficients are unchanged.

**FixedAutoRegressiveParameters**: same as FixedParameters but Arima Moving Average coefficients (MA) are also re-estimated, Auto-regressive (AR) coefficients are kept fixed. When using Seats for decomposition it avoids a possible re-allocation of roots between the trend and seasonal components, which might have led to strong revisions (cf INE at NTTS 2023).

**FreeParameters**: all regression and Arima model coefficients are re-estimated, regression variables and Arima orders are kept fixed.

*Those are policies not involving a data span.*

# Refresh Policies (1/2): an example

```
sa_x13_v3<-rjd3x13::x13(y_raw,spec="rsa3")
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 = "Fixed")

# apply the new spec on new data : y_new = y_raw + 6 months
sa_x13_v3_refreshed <- rjd3x13::x13(y_new, refreshed_spec)
```

# Refreshed spec: an example I

```
# refreshed spec
refreshed_spec$regarima
```

```
Specification

Series
Serie span: All
Preliminary Check: Yes

Estimate
Model span: All

Tolerance: 1e-07

Transformation
Function: LOG
AIC difference: -2
Adjust: NONE
```

# Refreshed spec: an example II

```
Regression
No calendar regressor

Easter: No

Pre-specified outliers: 0
Ramps: No

Outliers
Is enabled: No

ARIMA
SARIMA model:  (3,1,1) (0,1,1)

Coefficients
             Estimate    Type
phi(1)     0.07759472 INITIAL
```

# Refreshed spec: an example III

```
phi(2)     -0.05319340 INITIAL
phi(3)     -0.09894132 INITIAL
theta(1)   -0.77199495 INITIAL
btheta(1) -0.55342125 INITIAL
```

# Refresh Policies (2/2)

*Policies involving a data span.*

**Outliers**: regression variables and Arima orders are kept fixed, but outliers will be re-detected, from a given *end*, thus all regression and Arima model coefficients are re-estimated

**Outliers_StochasticComponent**: same as "Outliers" but Arima model orders (p,d,q)(P,D,Q) can also be re-identified.

**Current**: Not Available yet, behaves like "Fixed". Will be: applying the current pre-adjustment reg-arima model and adding the new raw data points as Additive Outliers (defined as new intervention variables)

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

# Refresh Policies (2/2): an example

```
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",
                                       period= 12,
                                       start = c(1990,1),
                                       # start of series to refresh
                                       end = c(2019,6))
# date from which outliers will be re-detected

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

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

# Refreshed spec: an example I

```
# refreshed spec
refreshed_spec$regarima
```

```
Specification

Series
Serie span: All
Preliminary Check: Yes

Estimate
Model span: All

Tolerance: 1e-07

Transformation
Function: LOG
AIC difference: -2
Adjust: NONE
```

# Refreshed spec: an example II

```
Regression
No calendar regressor

Easter: No

Pre-specified outliers: 0
Ramps: No

Outliers
Detection span: From 2019-06-01
Outliers type:
    - AO, critical value : 0 (Auto)
    - LS, critical value : 0 (Auto)
    - TC, critical value : 0 (Auto)
TC rate: 0.7 (Auto)
Method: ADDONE (Auto)
```

# Refreshed spec: an example III

```
ARIMA
SARIMA model:  (3,1,1) (0,1,1)

Coefficients
          Estimate      Type
phi(1)           0 UNDEFINED
phi(2)           0 UNDEFINED
phi(3)           0 UNDEFINED
theta(1)         0 UNDEFINED
btheta(1)        0 UNDEFINED
```

The process would be identical using `rjd3tramoseats::refresh`

# Refresh Policies names

Towards a harmonisation ?

| Revision Policy | JDemetra+ Interface (GUI) | Cruncher (via R) | Rjd3x13 / rjd3tramoseats |
|---|---|---|---|
| Applying the current model (unchanged) adding the new raw points as AO | Current adjustment (AO approach) | current (n) | current |
| Applying the current model (unchanged) replacing forecasts by new raw points | Fixed model | fixed(f) | fixed |
| Regression variables, Arima orders and coefficients are unchanged, only regression coefficents are re-estimated | Estimate regression coefficients | fixedparameters (fp) | FixedParameters |
| ...previous + Arima model MA coefficents also re-estimated | + Moving average parameters | FixedAutoRegressiveParameters | FixedAutoRegressiveParameters |
| ...previous + Arima model coefficents also re-estimated | + Arima parameters | parameters (p) | FreeParameters |
| ...previous + outliers re-identified for the last year | + Last outliers | lastoutliers (l) | Outliers (+span) |
| ...previous + outliers re-identified for the whole series | + All outliers | outliers (o) | Outliers |
| ...previous + orders of the Arima model are re-identified | + Arima model | stochastic (s) | Outliers_StochasticComponent |

Section 4

# Seasonal adjustment of High-Frequency data

# New package structure

`rjd3highfreq` was split in two parts: AMB and filter based

- X-11 related functions have been removed from **rjd3highfreq**, which now contains only model based algorithms.
- **rjd3x11plus** contains all the extended X11 functions for any (high) frequency data, and new trend estimation filters (weighted polynomials)
- **rjd3x11plus** depends on rjd3filters. Possibility de to reproduce X-11 algorithm fully, including correction for extreme values. Example in related vignette https://github.com/rjdemetra/rjd3x11plus/blob/develop/vignettes/X11.Rmd
- `rjd3stl` (Loess based) and `rjd3sts` (ssf based) are the two other ways to decompose high (any)- periodicity data.

# Decomposition with extended X-11: code template

Will tackle any periodicity, iterative decomposition starting with the highest frequency (same function as previously in `rjd3highfreq`)

```
#step 1: p = 7
x11.dow <- rjd3x11plus::x11(
    ts = exp(pre.mult$model$linearized),
    # result from preadjustment part
    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 <- rjd3x11plus::x11(x11.dow$decomposition$sa,  # previous sa
                            period = 365.2425,          # DOY pattern
                            mul = TRUE) #other parameters skipped here
```

Section 5

# Conclusion

# More info in talks about JDemetra+

More information about v3 and potential applications

Talks from OECD December 2023 workshop

- JDemetra+: from version 2 to version 3 how has the software evolved?, by Anna Smyk (Insee)
- Introducing the potential of rjd3sts, the R tool from JDemetra+ dedicated to State Space models with some case studies, by Corentin Lemasson (National Bank of Belgium).
- R-package tvCoef, implementing time-varying coefficients models has never been so easy, by Alain Quartier-la-Tente (Insee)

All available on our Blog: https://jdemetra-universe-blog.netlify.app/

# Overview of rjd3 tools for SA (and modelling)

- General time series tools (in `rjd3toolkit`, `rjd3sts`,`rjd3filters`) - tests - arima estimation - generation of regressors (calendar, trigonometric...)
- More algorithms and HF extensions
  - extended airline and seats
  - extended x11, stl
  - sts
- Refresh Policies
- Direct setting of basic benchmarking in SA
- All the previous when using TS objects directly in R..but rjd3 also provides R tools when using the Graphical User Interface and Workspaces...see next presentation !