13b - CVS en R avec les packages RJDemetra (v2) et rjd3x13 ou rjd3tramoseats (v3): Part 2

Anna Smyk & Tanguy Barthélémy (Insee)

Insee
Mesurer pour comprendre

JD+
Time Series Software for Official Statistics

Sommaire I

1. Ajout de regresseurs externes
2. Rafraîchissement des données

Section 1

Ajout de regresseurs externes

Désaisonnalisation simple I

Dans la 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")
```

Dans la version 3 (printed model identical to v2)

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

Ajout de régresseurs user-defined

Différences :

Dans la version 2 : régresseurs ajoutés directement à la spécification

Dans la version 3 : nouvelle notion de « contexte » : un concept supplémentaire conçu pour ajouter toute variable définie par l'utilisateur.

(ajout autre que spécification d'un outlier, d'une rampe ou de regresseurs de calendrier qui seront directement générés dans l'algoritme)

V2 Ajout de régresseurs user-defined (1/2) I

```
# 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)
```

V2 Ajout de régresseurs user-defined (2/2) I

```
# 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)
```

V3 : Ajout d'un regresseur de calendrier

Lors de l'ajout de régresseurs qui ne sont pas prédéfinis (comme les outliers ou les rampes) :

- `rjd3toolkit::set_tradingdays` à utiliser lors de l'allocation d'un régresseur à la composante **calendrier**.

(statut special des effets de calendrier)

- `rjd3toolkit::add_usrdefvar` est utilisé pour l'allocation à toute autre composante

Étape 1 : Création d'un calendrier

```
# create national (or other) calendar if needed
library("rjd3toolkit")
# French ca
french_calendar <- 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")
))
```

Etape 2: Création de regresseurs

```
# create set of 6 regressors every day is different, contrast with Sunday, based on t
regs_td <- rjd3toolkit::calendar_td(
    calendar = french_calendar,
    # 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"
)
```

les regresseurs peuvent être n'importe quel objet de classe TS

Etape 3: Création d'un "modelling context"

Un "modelling context" est nécessaire pour l'ajout de n'importe quel regresseur (nouveauté v3)

```
# 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()
```

Etape 4: Ajouter les regresseurs à la specification (calendrier)

```
### 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"
)
```

Etape 4b: Ajouter les regresseurs à la specification (autre que calendrier)

```
# add intervention variable to spec, choosing the component to allocate the effects to TREND
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
```

Etape 5: Estimation avec contexte

Utiliser la specification “user-defined” complète

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

Outliers et variables d'intervention

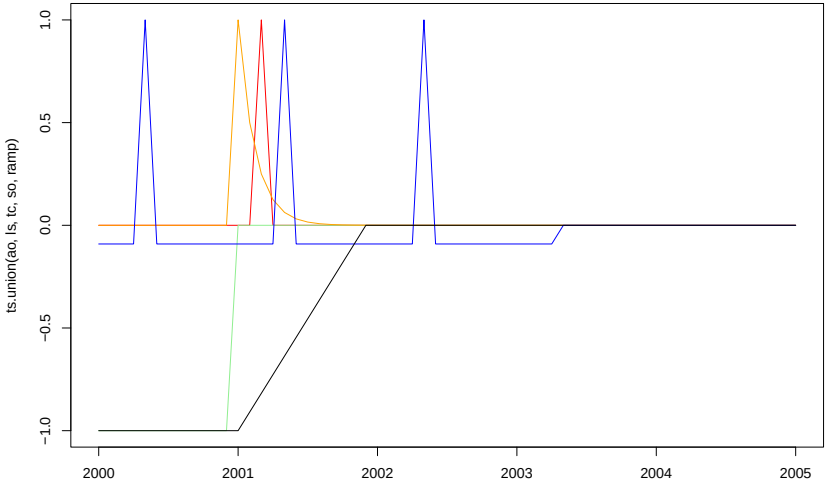
Une nouvelle fonctionnalité de la version 3 permet de créer :

- des régresseurs représentant des outliers (AO, LS, TC, SO) ou des rampes (lineaire seulement, quadratique à ajouter)
- des variables trigonométriques

Exemple d'outliers I

```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")
)
```

Exemple d'outliers II



Section 2

Rafraîchissement des données

Rafraîchissement des données : Estimation_spec vs result_spec

La possibilité de rafraîchir les données est une **nouveauté** de la version 3

Option pratique si le processus de production est entièrement en R avec des objets TS (pas de structure workspace)

Dans l'objet « sa_model » généré par le processus d'estimation :

- la spécification est séparée des résultats

```
# 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
```

Actualisation des données : estimation_spec vs result_spec

Dans l'objet « sa_model » généré par le processus d'estimation, la spécification est divisée en deux :

- « estimation_spec » (domain spec) : ensemble de contraintes définissant le processus d'estimation ; il peut s'agir d'une spécification par défaut (« RSA3 ») ou d'une spécification définie par l'utilisateur (par exemple RSA3 + régresseurs de calendrier…).
- result_spec » (point spec) : résultat du processus d'estimation, contient le modèle sélectionné, les coefficients estimés… suffisamment d'informations pour que, si elles sont appliquées à des séries brutes, elles permettent de récupérer tous les résultats (CVS, s, cal…).
- dans la v3.x possibilité de ré-estimer le « result_spec » à l'intérieur d'un domaine de contraintes (estimation spec), en ne libérant que les restrictions sur les paramètres sélectionnés (comme dans la GUI, ou Cruncher dans la v2.x)

Estimation_spec vs result_spec : un exemple (1/2)

- estimation spec

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

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

SARIMA coefficients:
 theta(1) btheta(1)
        0         0
```

Estimation_spec vs result_spec: un exemple (2/2)

- result spec (or point spec)

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

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

SARIMA coefficients:
 theta(1) btheta(1)
  -0.7460   -0.4781
```

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.

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

Those policies do not involve a data span.

Refresh Policies (1/2): un exemple

```
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: un exemple 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
```

Refreshed spec: un exemple II

```
Adjust: NONE

Regression
No calendar regressor

Easter: No

Pre-specified outliers: 1
    - LS (2020-04-01), coefficient: -0.485805683501112 (FIXED)
Ramps: No

Outliers
Is enabled: No

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

Refreshed spec: un exemple III

```
SARIMA coefficients:
   phi(1)    phi(2)    phi(3) btheta(1)
  -0.1652   -0.3345   -0.3090   -0.4393
```

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 **start**, thus all regression and Arima model coefficients are re-estimated (modifications under way, here code ok, but `rjd3x13::x13_refresh` function documentation in help pages not up to date)

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

Current: applying the current pre-adjustment reg-arima model and handling the new raw data points, or any sub-span of the series as Additive Outliers (defined as new intervention variables)

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

Refresh Policies: un exemple

```
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, # periodicity of the series
    start = 2022
)
# 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: un exemple I

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

Le processus serait identique en utilisant `rjd3tramoseats::refresh`

Noms des Refresh Policies

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
All the parameters are re-identified and re-estimated (note : any user defined variable or constraint is kept)	Concurrent	complete / concurrent (c)	complete

Conclusion

Nouvelles fonctionnalités $=>$ faisons tourner du code