13a - CVS en R avec les packages RJDemetra (v2) et rjd3x13 ou rjd3tramoseats (v3), Partie 1

Anna Smyk & Tanguy Barthélémy (Insee)

Insee
Mesurer pour comprendre

JD+
Time Series Software for Official Statistics

Sommaire I

Section 1

Introduction

Désaisonnalisation : étapes usuelles

- tests de saisonnalité
- prétraitement
- création des variables personnalisées pour le prétraitement (par exemple, régresseurs de calendrier)
- décomposition
- récupération des séries estimées (sa, s, t, i...)
- récupération des diagnostics
- personnalisation les paramètres..au vu des diagnostics
- rafraîchissement des données (production infra-annuelle)
- ...
- répétition du processus...

Cette présentation illustrera tous ces points dans X13-Arima via RJDemetra et rjd3x13 ou rjd3tramoseats (et rjd3toolkit) .

Contexte d'utilisation

Produire des séries désaisonnalisées avec R

(avec des paramètres personnalisés en fonction des besoins et des diagnostics précédents)

- ne pas être au courant de l'existence de l'interface graphique de JD+.
- pas de structure de workspace
- objets séries temporelles dans R
- utiliser exclusivement les algorithmes de JD+ et aucun autre package R de SA (Seasonal, TBATS...)

Tous les exemples sont liés à UNE série. Pour un ensemble de données complet, vous pouvez bien sûr utiliser des boucles ou des fonctions du type `lapply()`.

Section 2

X13 : Lancement rapide avec les spécifications par défaut

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")
```

Pre-ajustement seul

En version 2

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

En version 3 (pas très différent)

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

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

Décomposition seule

En version 2

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

En version 3

```
# X11 is a specific function
x11_v3 <- rjd3x13::x11(y_raw) # specific function
```

Subsection 1

Récupération des résultats et visualisation des données

Structure de l'objet « Model_sa » dans la version 2 (1/2)

« Model_sa » est l'objet résultant de l'estimation, il contient

- la série brute
- Les paramètres (spécification)
- Les séries en sortie
- des diagnostics

Le tout est structuré par domaines

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

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

“Model_sa” object structure en 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 en version 3

Results vs specification…and then by domain

```
# 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
```

Différences entre la version 2 et la version 3

Dans la version 3

- Les spécifications sont séparées des résultats
- Les résultats sont plus spécifiques (« X11 » comme les noms de séries dans X13-Arima)
- Les spécifications sont directement disponibles (aucune fonction d'extraction n'est nécessaire comme dans la version 2)
- deux concepts de spécification : spécification d'estimation (domaine) et spécification de résultat (point) dans la v3
- dans v2 seulement la spécification de résultat (plus d'informations à ce sujet dans la section refresh)

Récupérer les séries en sortie

Les séries en entrée et en sortie sont des objets TS dans R (pas lors de l'utilisation d'extensions spécifiques pour les données HF)

- séries finales : noms et disposition différents de v2 à 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
```

Dans la version 3, beaucoup plus de séries sont disponibles sans utiliser le user-defined output.

Séries issues du préajustement

```
# 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
```

Séries issues de la décomposition

Dans la version 2

- Tableaux D accessibles via un user-defined output
- prévision pour chaque série accessibles uniquement via un user-defined output (cf. ci-dessous)

Dans la version 3 : « x11 names »

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

Récupération des diagnostics

Il suffit de récupérer les objets nécessaires dans la partie corresponde du modèle (liste de listes) ou de faire un summary du modèle.

```
# 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
```

Ce qui manque (séries ou diagnostics) peut être récupéré en ajoutant un user-defined output dans les options

User-defined output (1/2)

En version 2 ou version 3 : définissez d'abord le vecteur d'objets que vous souhaitez ajouter

Listes des diagnostics ou séries disponibles

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

# Version 3: more specific functions

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

User-defined output (2/2)

Sélectionner les objets et personnaliser la fonction d'estimation (identique dans les v2 et 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 et visualisation des données dans la version 2 I

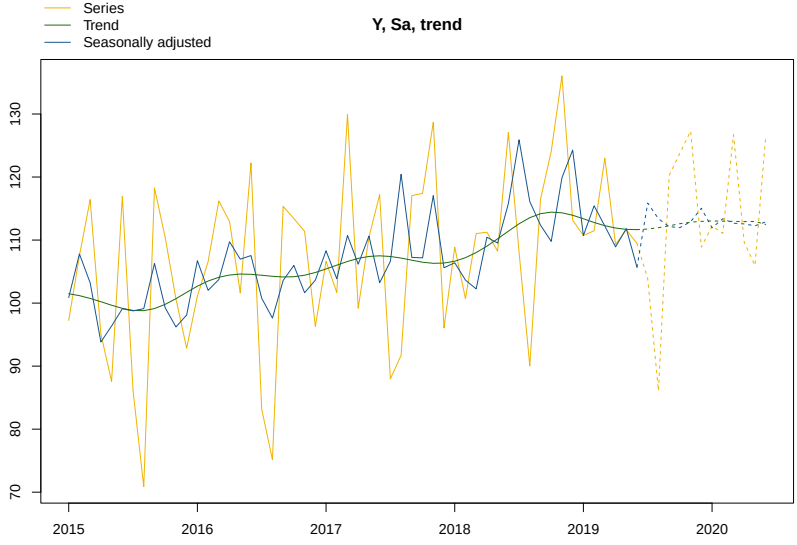
Dans la version 2, il existe trois types de graphiques :

- final (2 types : Plots identiques aux “Main Results” de l’interface graphique)
- résidus regarima (6 Plots)
- SI ratios

Plots en version 2 I

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

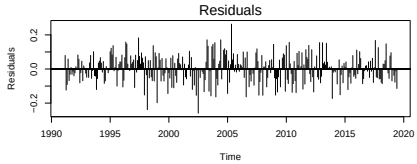
Plots en version 2 II

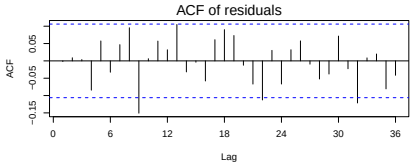


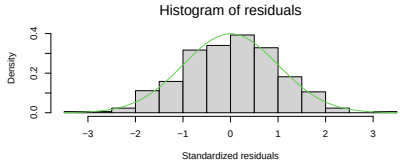
Plots en version 2 I

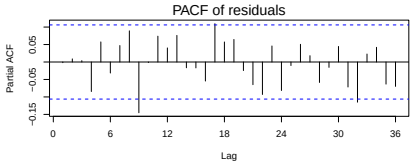
```
# regarima
layout(matrix(1:6, 3, 2))
plot(sa_x13_v2$regarima, ask = FALSE)
```

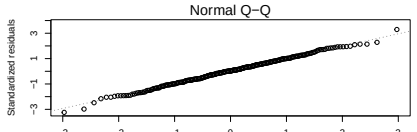
Plots en version 2 II

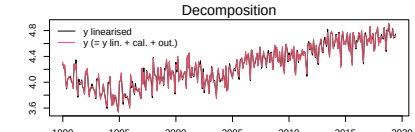








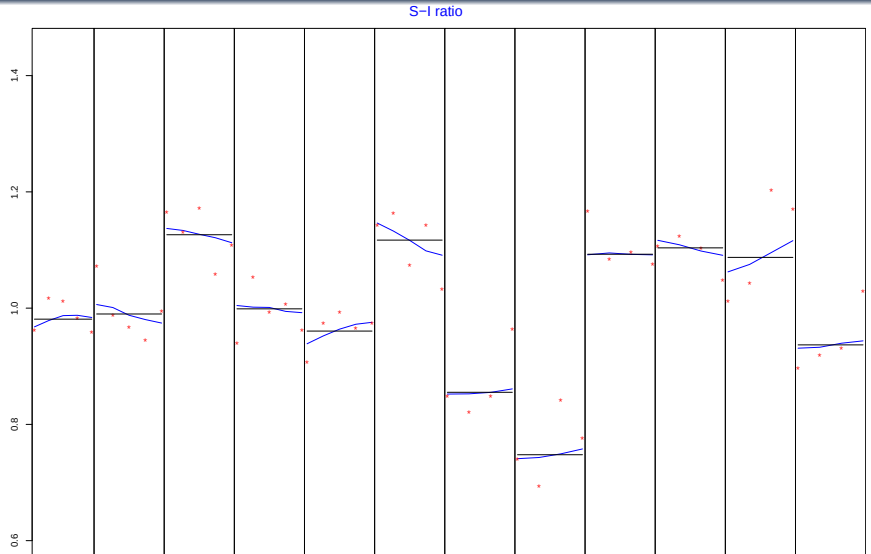




Plots en version 2 I

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

Plots en version 2 II



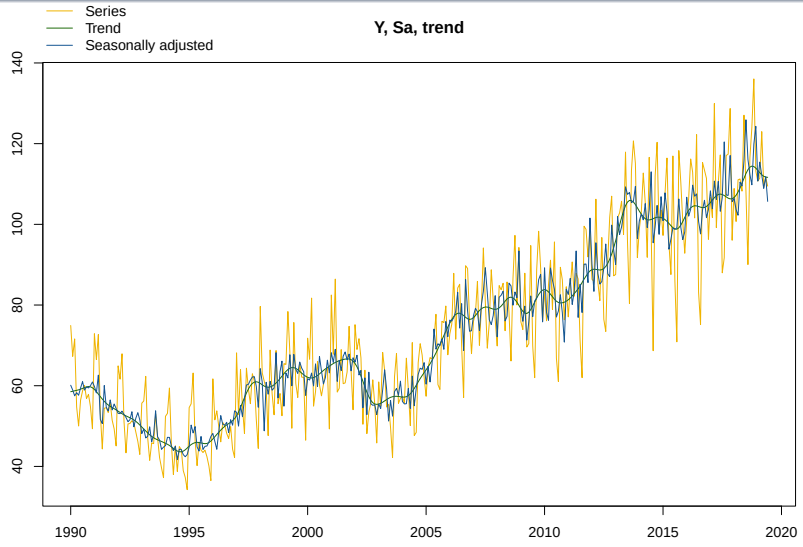
Plot de la decomposition en version 3 I

```
# version 3
### ggdemetra3 for enhanced plots

# remotes::install_github("AQLT/ggdemetra3", INSTALL_opts = "--no-multiarch")
library("ggdemetra3")

# plot final decomposition
plot(sa_x13_v3)
```

Plot de la decomposition en version 3 II



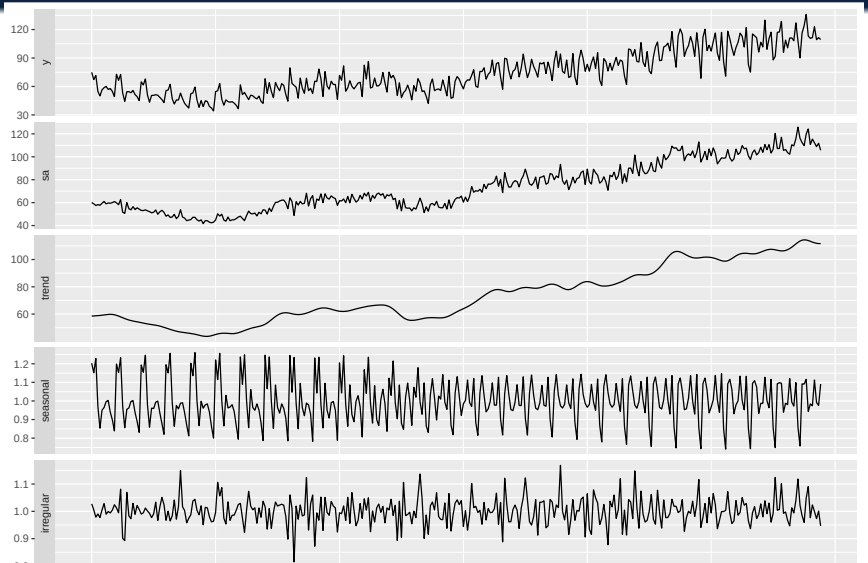
Plot de la decomposition en version 3 I

New autoplot formats

```
# version 3


# in autopolot
library("ggplot2")
autoplot(sa_x13_v3)
```

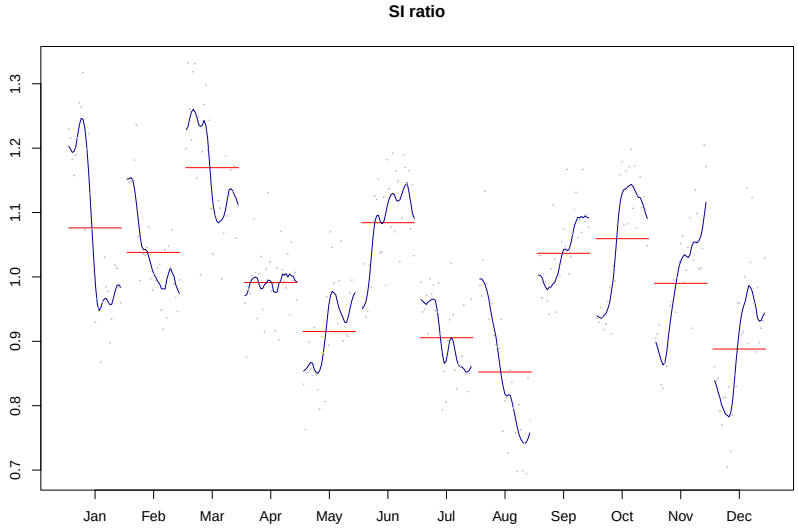
Plot de la decomposition en version 3 II



Plots des SI ratios I

```
# version 3
#Plot SI ratios
siratioplot(sa_x13_v3)
```

Plots des SI ratios II

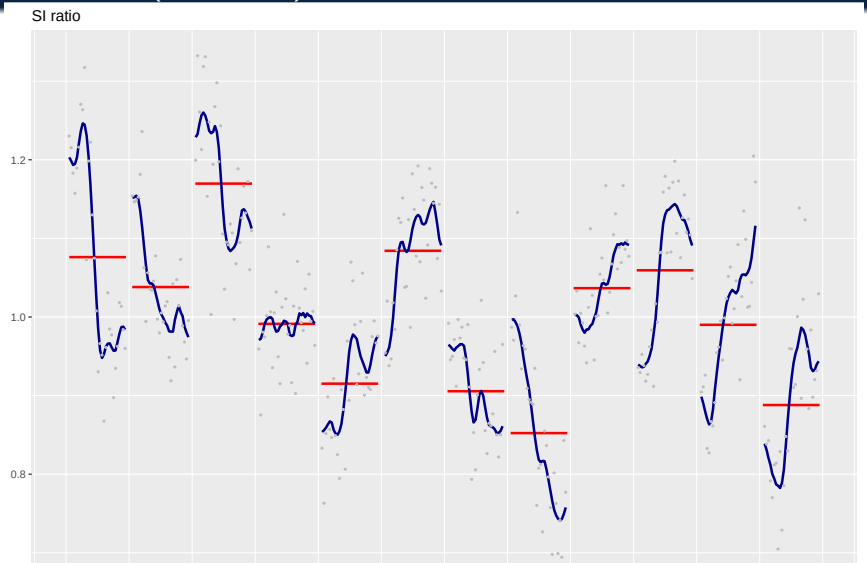


Plots des SI ratios (autoplot) I

```
# version 3
# avec le format autoplot

ggsiratioplot(sa_x13_v3)
```

Plots des SI ratios (autoplot) II



Personnalisation des spécifications : étapes générales

Pour personnaliser une spécification, vous devez

- commencer par une spécification valide, généralement l'une des spécifications par défaut (ce qui équivaut à cloner une spécification dans la GUI)
- créer une nouvelle spécification
- appliquer la nouvelle spécification à votre série de données brutes.

Quelques différences entre la v2 et la v3

Personnalisation des spécifications dans la version 2

Modification directe des paramètres en tant qu'arguments de la fonction de spécification

```
# 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)
```

Personnaliser les spécifications dans la version 3

Utiliser des fonctions `set_` directes et spécifiques - pour l'étape de pré-traitement (fonctions définies dans `rjd3toolkit`) :

`set_arima()`, `set_automodel()`, `set_basic()`, `set_easter()`, `set_estimate()`, `set_outlier()`, `set_tradingdays()`, `set_transform()`, `add_outlier()` et `remove_outlier()`, `add_ramp()` et `remove_ramp()`, `add_usrdefvar()`.

- pour l'étape de décomposition en X13 (fonction définie dans `rjd3x13`) : `set_x11()`
- pour l'étape de décomposition en Tramo-Seats (fonction définie dans `rjd3tramoseats`) : `set_seats()`
- pour l'étape de Benchmarking (fonction définie dans `rjd3toolkit`) : `set_benchmarking()`

Benchmarking Nouvelle fonctionnalité de la v3, mêmes options disponibles que dans la GUI.

Personnalisation des spécifications dans la version 3 : exemple

```
# 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 = "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)
```

Outils pour les séries temporelles

Nouvelles fonctionnalités dans la version 3

L'esprit de la version 3 est d'offrir plus d'outils indépendants :

- tests (saisonnalité, résidus (normalité, independance), effets de calendrier résiduels) dans rjd3toolkit.
- fonctions d'autocorrélation partielle et inverse
- estimation et décomposition du modèle arima (`rjd3toolkit::ucrima_estimate()`)
- agrégation à une fréquence plus basse (`rjd3toolkit::aggregate()`)

Plus de flexibilité pour l'utilisateur car elles peuvent être appliquées à tout moment et pas seulement dans le cadre d'un traitement CVS.

L'estimation du modèle Arima est notoirement plus rapide que d'autres fonctions disponibles en R.

Tests de saisonnalité

Dans rjd3toolkit :

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

Estimation Arima

```
# 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), period = 12)
    )
}))
#       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
```