5b - CVS en R avec les packages RJDemetra (v2) et rjd3x13 (v3)

Séries Temporelles avec R - Initiation

Anna Smyk, Tanguy Barthelemy

Insee - Département des Méthodes Statistiques

Section 1

Introduction

Désaisonnalisation : étapes usuelles

- test de saisonnalité
- prétraitement
- créer des variables personnalisées pour le prétraitement (par exemple, régresseurs calendaires)
- décomposition
- récupération des séries de sortie
- récupération des diagnostics
- personnaliser les paramètres
- rafraîchir les données
- ...
- répéter...

Cette présentation illustrera tous ces points dans X13-Arima via RJDemetra et rjd3x13 (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

Subsection 1

Lancement rapide avec les spécifications par défaut

Exécution d'un traitement de désaisonnalisation 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")
```

In version 3 (printed model identical to v2)

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

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")
```

In version 3 (not very different)

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

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

Running only decomposition

In version 2

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

In version 3

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

Subsection 2

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 de sortie
- des diagnostics

Le tout organisé d'une manière spécifique

```
# 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
```

= 90%}

"Model_sa" object structure in 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 de sortie

Les séries d'entrée et de 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 l'option de sortie définie par l'utilisateur.

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 une sortie définie par l'utilisateur,
- séries de prévisions accessibles uniquement via la sortie définie par l'utilisateur (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 pertinente de la structure de sortie ou d'imprimer l'ensemble 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 une sortie définie par l'utilisateur dans les options ### Récupération des sorties définies par l'utilisateur (1/2)

Retrieve user defined-output (2/2)

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

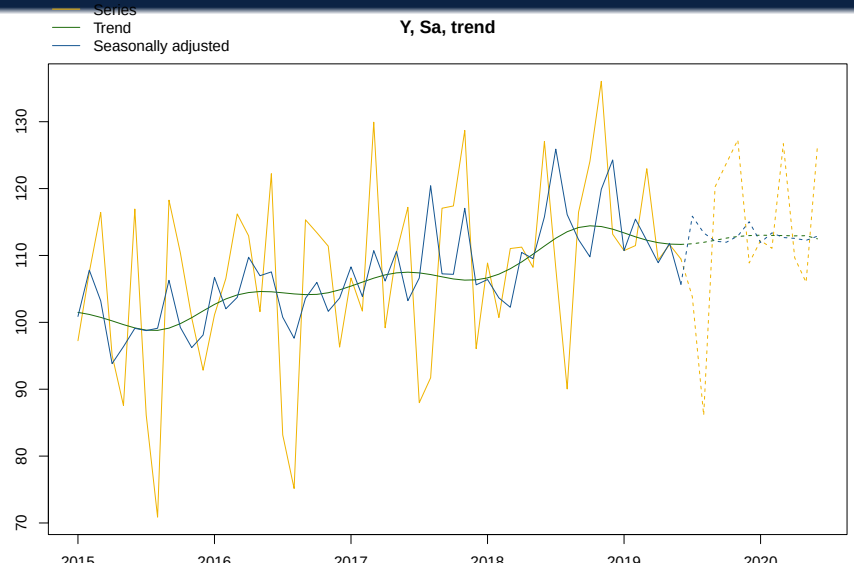
Dans la version 2, il existe trois types de graphiques :

- final (2 types : Plots identiques aux résultats principaux de l'interface graphique)
- résidus regarima (6 Plots)
- Rapports SI

Plots et visualisation des données dans la version 2 I

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

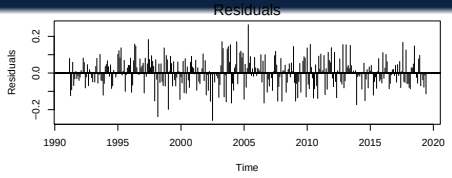
Plots et visualisation des données dans la version 2 II

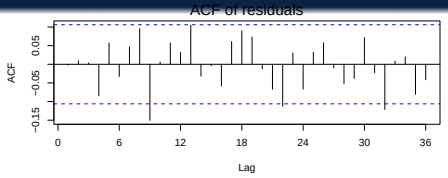


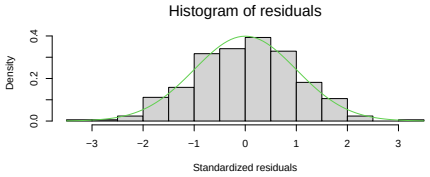
Plots et visualisation des données dans la version 2 I

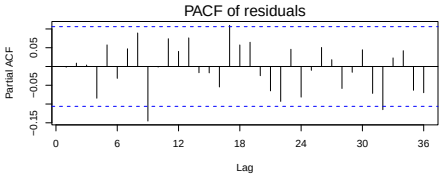
```
# regarima
layout(matrix(1:6, 3, 2))
plot(sa_x13_v2$regarima, ask = FALSE)
```

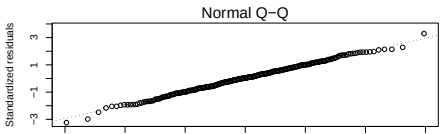
Plots et visualisation des données dans la version 2 II

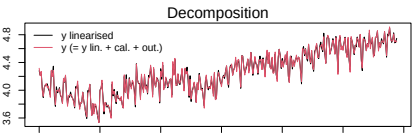








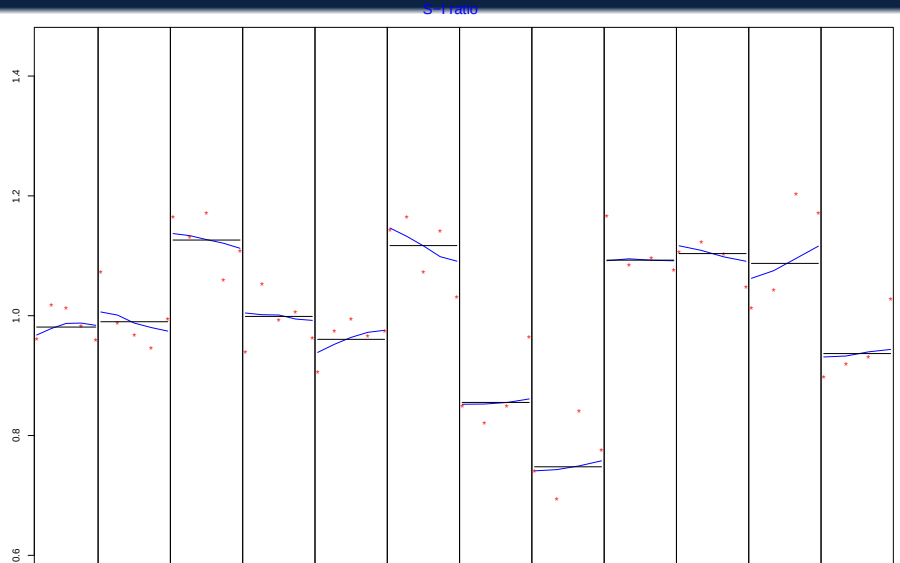




Plots et visualisation des données dans la version 2 I

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

Plots et visualisation des données dans la version 2 II



Plots et visualisation des données dans la version 3 I

- finale + NOUVELLE mise en page « autoplot
- regarima pas disponible (encore ?)
- ratios SI + NOUVELLE présentation ggplot

```
# version 3
# remotes::install_github("AQLT/ggdemetra3", INSTALL_opts = "--no-multiarch")
library("ggdemetra3")
ggdemetra3::siratioplot(sa_x13_v3)
```

Plots et visualisation des données dans la version 3 I

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

Plots et visualisation des données dans la version I

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

Personnalisation des spécifications

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
```

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

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)
```

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 (non standard, par exemple non aberrante).

V2 Ajout de régresseurs user-defined

```r
# 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)
# 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
```

V3 : Ajout d'un regresseur de calendrier user-defined ou d'autres régresseurs

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**.
- `rjd3toolkit::add_usrdefvar` est utilisé pour allocation à toute autre composante

Étape 1 : Création d'un calendrier

```
# 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")
))
```

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 = 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"
)
```

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)

```
# 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
```

Générer des variables auxiliaires user-defined

Outliers et variables d'intervention

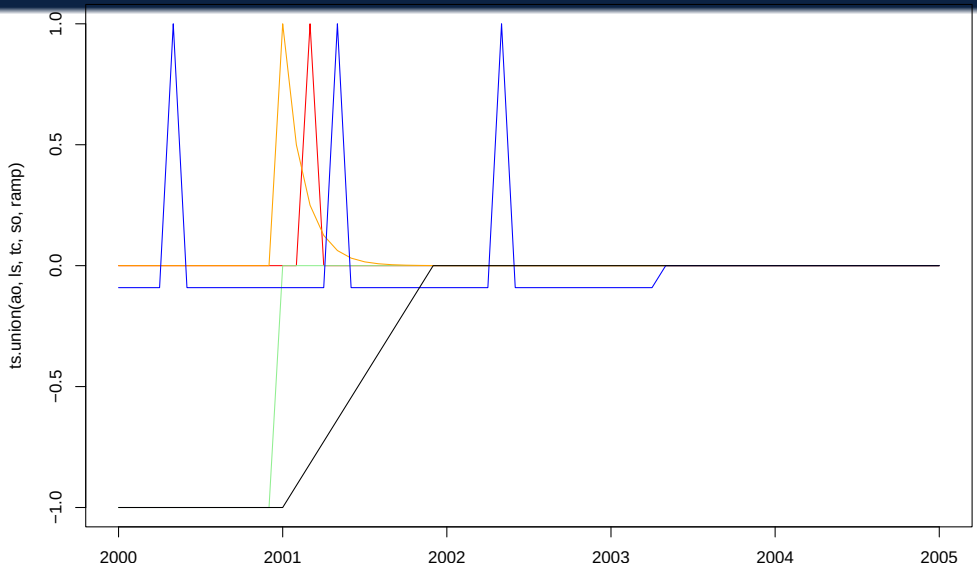
La nouvelle fonctionnalité de la version 3 permet de créer :

- des régresseurs outliers (AO, LS, TC, SO, Ramp (quadratique à ajouter)
- 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



Outils pour les séries temporelles

Outils de séries temporelles : NOUVELLES fonctionnalités dans la version 3

L'esprit de la version 3 est d'offrir plus d'outils issus des bibliothèques JDemetra+ tels que :

- tests (saisonnalité, normalité, aléa, effets résiduels des jours de bourse) dans rjd3toolkit.
- fonctions d'autocorrélation partielle et inverse
- estimation et décomposition du modèle arima (`rjd3toolkit::ucrima_estimate()`)
- agrégation à une fréquence plus élevée (`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.

Certaines d'entre elles peuvent également être disponibles dans d'autres packages R

Test 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
```

Section 3

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 de sortie, 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 calendaires...).
- 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)
  -0.7243   -0.5640
```

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 are policies not involving 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: 0
Ramps: No

Outliers
Is enabled: No

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

SARIMA coefficients:
```

Refreshed spec: un exemple III

phi(1)	phi(2)	phi(3)	theta(1)	btheta(1)
0.06758	-0.05262	-0.09236	-0.77257	-0.55676

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.**Not Available yet**

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: 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
```

The process would be identical using `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	Concurrent	complete /	complete

Section 4

Conclusion

Nouveautés de la version 3

Seasonal adjustment tools (including high-frequency data)

But also:

- Tests
- Arima estimation
- General and flexible definition of calendars and auxiliary variables
- Refresh Policies
- Direct setting of basic benchmarking