Institut national de la statistique et des études économiques

Insee

Mesurer pour comprendre

R and JDemetra+ 3.0: A new toolbox around seasonal adjustment and time series analysis

Alain Quartier-la-Tente
Insee

Scientific Session: Time series and longitudinal data analysis

06/11/2022

Contents

JDemetra+ ???

- JDemetra+ is an open source software (build on ☕) officially recommended by Eurostat for seasonal adjustment (SA)

JDemetra+ ???

- JDemetra+ is an open source software (build on ☕) officially recommended by Eurostat for seasonal adjustment (SA)
- Implements the two leading SA methods X-13ARIMA and TRAMO-SEATS with a nice graphical interface

JDemetra+ ???

- JDemetra+ is an open source software (build on) officially recommended by Eurostat for seasonal adjustment (SA)
- Implements the two leading SA methods X-13ARIMA and TRAMO-SEATS with a nice graphical interface
- How to perform SA in ?
    - `stats::stl()` and `forecast::mstl()` for (M)STL: not recommended because they cannot perform calendar adjustment
    - `seasonal` and `x12`: interface to X-13-ARIMA-SEATS US Census Bureau binaries

Introduction (1)

- In March 2019, `RJDemetra` was published on CRAN:
  - only R package that enables to use TRAMO-SEATS
  - faster than existing R packages on seasonal adjustment
  - enables to interact with JDemetra+ “workspaces” used in production

Introduction (1)

- In March 2019, `RJDemetra` was published on CRAN:
  - only R package that enables to use TRAMO-SEATS
  - faster than existing R packages on seasonal adjustment
  - enables to interact with JDemetra+ “workspaces” used in production
- With the development of JDemetra+ 3.0, more than 13 R packages are being developped! Not only on seasonal adjustment!

Introduction (1)

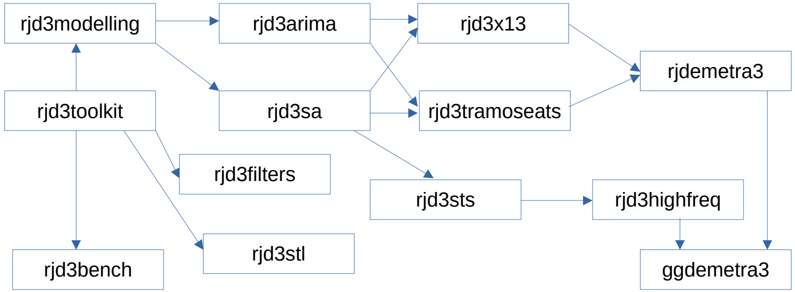
- In March 2019, `RJDemetra` was published on CRAN:
  - only R package that enables to use TRAMO-SEATS
  - faster than existing R packages on seasonal adjustment
  - enables to interact with JDemetra+ "workspaces" used in production
- With the development of JDemetra+ 3.0, more than 13 R packages are being developped! Not only on seasonal adjustment!
- They require Java ≥ 17 (see for example installation manual of `RJDemetra`: https://github.com/jdemetra/rjdemetra/wiki/Installation-manual)

Introduction (2)

They are all available in GitHub, currently:

```r
# install.packages("remotes")
remotes::install_github("palatej/rjd3toolkit")
remotes::install_github("palatej/rjd3modelling")
remotes::install_github("palatej/rjd3sa")
remotes::install_github("palatej/rjd3arima")
remotes::install_github("palatej/rjd3x13")
remotes::install_github("palatej/rjd3tramoseats")
remotes::install_github("palatej/rjdemetra3")
remotes::install_github("palatej/rjdfilters")
remotes::install_github("palatej/rjd3sts")
remotes::install_github("palatej/rjd3highfreq")
remotes::install_github("palatej/rjd3stl")
remotes::install_github("palatej/rjd3bench")
remotes::install_github("AQLT/ggdemetra3")
```

Introduction (3)



And it's just the begining! (might change in the future)

Contents

rjd3toolkit

Contains several utility functions used in other `rjd` packages and several functions to perform tests:

- Normality tests: Bowman-Shenton (`bowmanshenton()`), Doornik-Hansen (`doornikhansen()`), Jarque-Bera (`jarquebera()`)
- Runs tests (randomness of data): mean or the median (`testofruns()`) or up and down runs test (`testofupdownruns()`)
- autocorrelation functions (usual, inverse, partial)
- `aggregate()` to aggregate a time serie to a higher frequency

rjd3modelling

- 💡 create user-defined calendar and trading-days regressors: `calendar.new()` (create a new calendar), `calendar.holiday()` (add a specific holiday, e.g. christmas), `calendar.easter()` (easter related day) and `calendar.fixedday()`)
- 💡 create outliers regressors (AO, LS, TC, SO, Ramp, intervention variables), calendar related regressors (stock, leap year, periodic dummies and contrasts, trigonometric variables) -> to be added quadratic ramps
- 💡 Range-mean regression test (to choose log transformation), Canova-Hansen (`td.ch()`) and trading-days f-test (`td.f()`)
- manipulation of ARIMA models (generation, sum, decomposition, estimation)
- functions to stationarise your series `do.stationary()`, `differences()`, `differencing.fast()`
- specification functions for `rjd3x13` and `rjd3tramoseats`

Example of a specific calendar (1)

```r
library(rjd3modelling)
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 = "1953-03-20")
# 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")
```

Example of a specific calendar (2)

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 = TRUE)
```

```
##            [,1]
## 2020-12-24    0
## 2020-12-25    1
## 2020-12-26    0
## 2020-12-27    0
## 2020-12-28    0
## 2020-12-29    0
## 2020-12-30    0
## 2020-12-31    0
## 2021-01-01    1
## 2021-01-02    0
```

Example of a specific calendar (3)

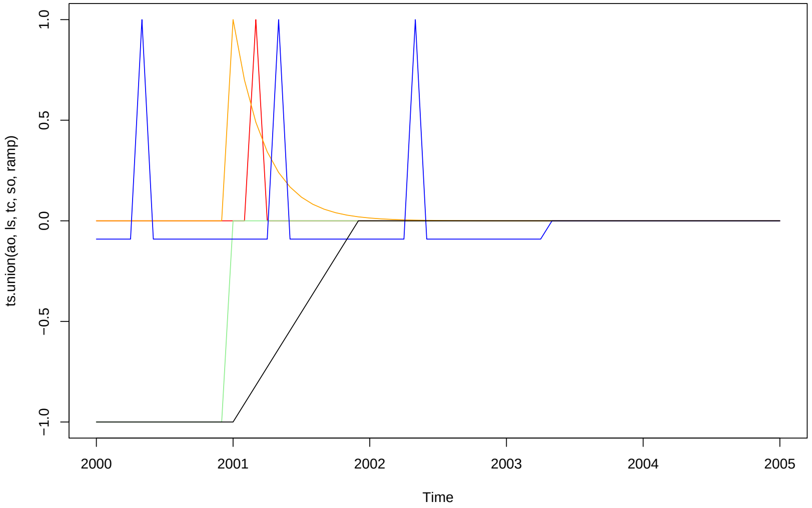
```
s <- ts(0, start = 2020, end = c(2020, 11), frequency = 12)
# Trading-days regressors (each day has a different effect, sunday as contrasts)
td_reg <- htd(fr_cal, s = s, groups = c(1, 2, 3, 4, 5, 6, 0))
# Working-days regressors (Monday = ... = Friday; Saturday = Sunday = contrasts)
wd_reg <- htd(fr_cal, 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
```

```
##              group-1    group-2
## Jan 2020  2.0000000  0.0000000
## Feb 2020  0.0000000  1.0000000
## Mar 2020 -1.7809251 -0.7968209
## Apr 2020  0.7809251 -0.2031791
## May 2020 -3.1554920  0.4740847
## Jun 2020  5.1554920  0.5259153
## Jul 2020  2.0000000  0.0000000
## Aug 2020 -4.0000000  0.0000000
## Sep 2020  2.0000000  0.0000000
## Oct 2020  2.0000000  1.0000000
## Nov 2020  0.0000000  0.0000000
```

Example of outliers (1)

```
s <- ts(0, start = 2000, end = 2005, frequency = 12)
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 = 0.7)
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"))
```

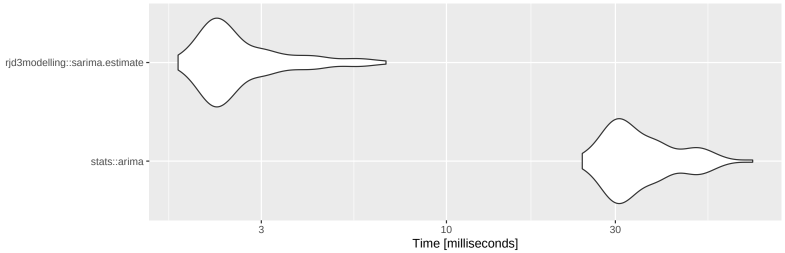
Example of outliers (2)



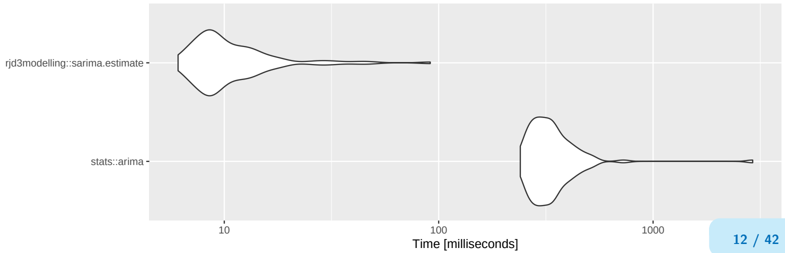
Benchmark of ARIMA estimations

More than 20 time faster in median!

ARIMA(0,1,1)(0,1,1)



ARIMA(3,1,1)(1,1,1)



rjd3sa (1)

Seasonality tests:

- Canova-Hansen (`seasonality.canovahansen()`)
- X-12 combined test (`seasonality.combined()`)
- F-test on seasonal dummies (`seasonality.f()`)
- Friedman Seasonality Test (`seasonality.friedman()`)
- Kruskall-Wallis Seasonality Test (`seasonality.kruskalwallis()`)
- Periodogram Seasonality Test (`seasonality.periodogram()`)
- QS Seasonality Test (`seasonality.qs()`)

rjd3sa (2)

⚠ Always correct the trend and remove the mean before seasonality tests:

```
library(rjd3sa)
y <- diff(rjd3toolkit::ABS$X0.2.09.10.M, 1); y <- y - mean(y)
# Or :
y <- rjd3modelling::differences(rjd3toolkit::ABS$X0.2.09.10.M)
seasonality.f(y)
```

```
## Value:  378.9234
## P-Value:  0.0000
```

```
seasonality.friedman(y)
```

```
## Value:  298.2529
## P-Value:  0.0000
```

```
seasonality.kruskalwallis(y)
```

```
## Value:  319.9801
## P-Value:  0.0000
```

```
seasonality.combined(y)
```

```
## $seasonality
## [1] "PRESENT"
##
```

Contents

rjd3arima

`rjd3arima` is devoted to formatting the output of Arima related results

Common functions

In `RJDemetra` you have one function to set the specification (`regarima_spec_x13()`, `regarima_spec_tramo()`, `x13_spec()` and `tramoseats_spec()`) now one function for each part of the specification

Common functions

In `RJDemetra` you have one function to set the specification (`regarima_spec_x13()`, `regarima_spec_tramo()`, `x13_spec()` and `tramoseats_spec()`) now one function for each part of the specification

Common functions (defined in `rjd3modelling`) to set the specification of the preprocessing:

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

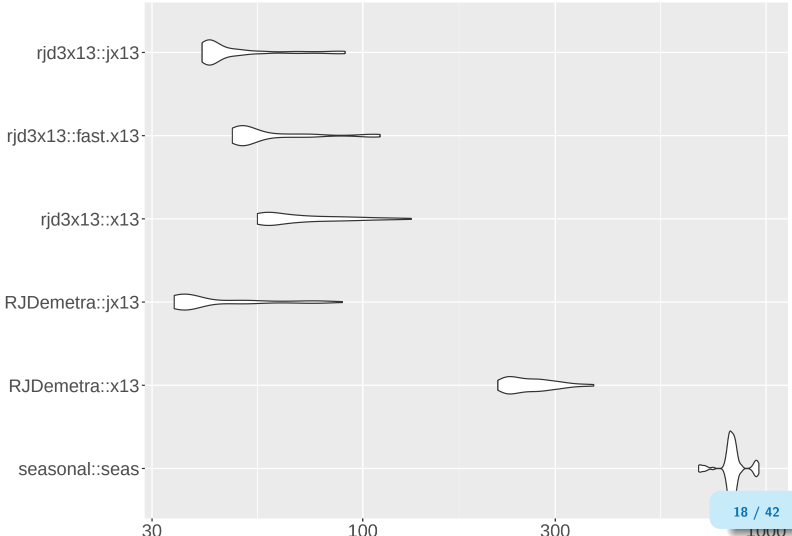
rjd3x13

Main functions:

- Specification: created with `spec_x11_default()`, `spec_x13_default()`, `spec_regarima_default()` and customized with `rjd3modelling` functions + `set_x11()`
- Apply SA model with `x11()`, `x13()`, `fast.x13()`
- ARIMA modelling with `regarima()`, `fast.regarima()`
- 💡 Refresh policies: `regarima.refresh()` and `x13.refresh()`

Performance

In median: `RJDemetra` more 3 time faster than `seasonal` and `rjdemetra3` more than 12 time faster than `seasonal`!



Exemple (1)

```
library(rjd3modelling);library(rjd3x13)
y <- rjd3toolkit::ABS$X0.2.09.10.M
spec <- spec_x13_default("rsa5c") |> set_easter(type = "unused") |>
  set_outlier(outliers.type = c("AO", "LS")) |>
  set_tradingdays(test = "None") |> set_x11(henderson.filter = 13) |>
  add_outlier(type = "TC", date = "2000-06-01",
              name = "My TC in 2000-06")
m = rjd3x13::x13(y, spec)
m$result$preprocessing
```

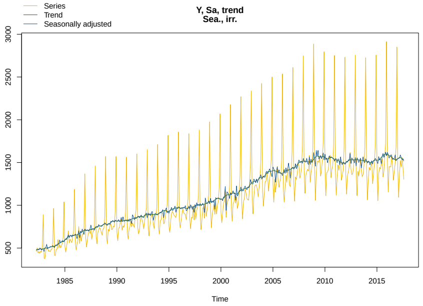
```
## Log-transformation: yes
## SARIMA model:  (0,1,2) (1,1,1)
##
## Coefficients
##            Estimate Std. Error  T-stat
## theta(1)  -1.01804    0.07639 -13.326
## theta(2)   0.20863    0.05378   3.879
## bphi(1)   -0.26680    0.05399  -4.942
## btheta(1) -0.77559    0.05384 -14.405
##
## Regression model:
##                    Estimate Std. Error T-stat
## monday            -0.011247   0.004004 -2.809
## tuesday            0.005870   0.004013  1.463
## wednesday         -0.002002   0.004003 -0.500
```

Exemple (2)

```
## thursday         0.014483   0.004021  3.602
## friday           0.001577   0.004023  0.392
## saturday         0.011465   0.003996  2.869
## lp               0.037501   0.010994  3.411
## easter           0.053486   0.008319  6.429
## My TC in 2000-06 0.022947   0.023666  0.970
## Number of observations:  425
## Number of effective observations:  412
## Number of parameters:  14
##
## Loglikelihood:  763.5143
## Adjusted loglikelihood:  -2104.113
##
## Standard error of the regression (ML estimate):  0.03757223
## AIC:  4236.225
## AICC:  4237.283
## BIC:  4292.519
```

```
# Also summary function
# summary(m)
plot(m)
```

Exemple (3)


Series
Trend
Seasonally adjusted
Y, Sa, trend
Sea., irr.
500
1000
1500
2000
2500
3000
1985
1990
1995
2000
2005
2010
2015
Time

rjd3tramoseats

Main functions:

- Specification: created with `spec_tramoseats_default()`, `spec_tramo_default()` and customized with `rjd3arima` functions + `set_seats()`
- Apply model with `tramoseats()`, `fast.tramoseats()`, `tramo()`, `fast.tramo()`
- Refresh policies: `tramo.refresh()` and `tramoseats.refresh()`

Exemple (1)

```r
spec <- spec_tramoseats_default("rsafull") |>
  set_easter(type = "IncludeEasterMonday") |>
  set_tradingdays(test = "Separate_T") |>
  set_seats(algorithm = "KalmanSmoother")
m <- rjd3tramoseats::tramoseats(y, spec)
```

rjdemetra3

Functions to manipulate JDemetra+ workspaces:

- Still in construction: you can load an existing workspace but not create a new one (use `jws.load()` for example)
- Will contain all the functionalities of `rjdworkspace` (more manipulation of workspaces)

rjd3highfreq and rjd3stl

Seasonal adjustment of high frequency data:

- fractional and multi airline decomposition
- Extension of X-11 decomposition with non integer periodicity

`rjd3stl` : STL, MSTL, ISTL, loess

See next presentation of Anna Smyk

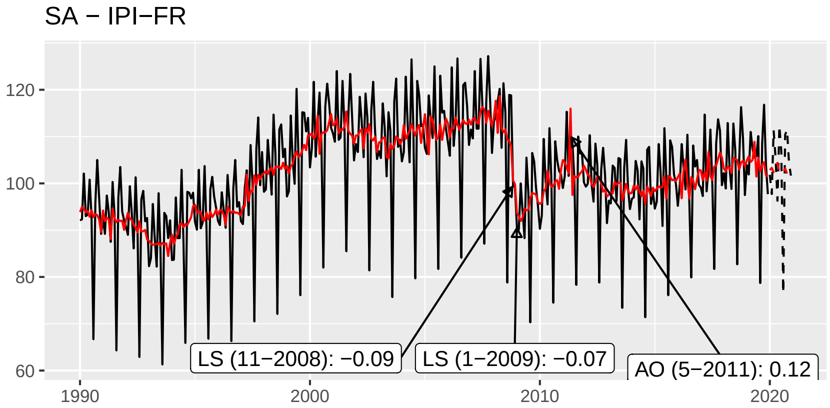
Contents

ggdemetra3 (1)

Like `ggdemetra` but compatible with `rjdemetra3`: ggplot2 to add seasonal adjustment statistics to your plot,`autoplot()` functions… Also compatible with high-frequency methods (WIP):

```
library(ggdemetra3)
spec <- spec_x13_default("rsa3") |> set_tradingdays(option = "WorkingDays")
ggplot(data = ipi_c_eu_df, mapping = aes(x = date, y = FR)) +
    geom_line() +
    labs(title = "SA - IPI-FR",
         x = NULL, y = NULL) +
  geom_sa(component = "y_f(12)", linetype = 2,
          spec = spec) +
  geom_sa(component = "sa", color = "red") +
  geom_sa(component = "sa_f", color = "red", linetype = 2) +
  geom_outlier(geom = "label_repel",
               coefficients = TRUE,
               ylim = c(NA, 65), force = 10,
               arrow = arrow(length = unit(0.03, "npc"),
                             type = "closed", ends = "last"),
               digits = 2)
```

ggdemetra3 (2)

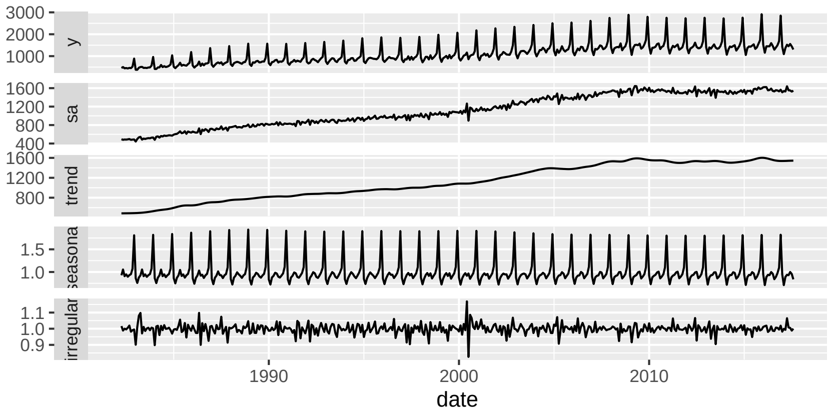


ggdemetra3 (1)

Or from an existing model:

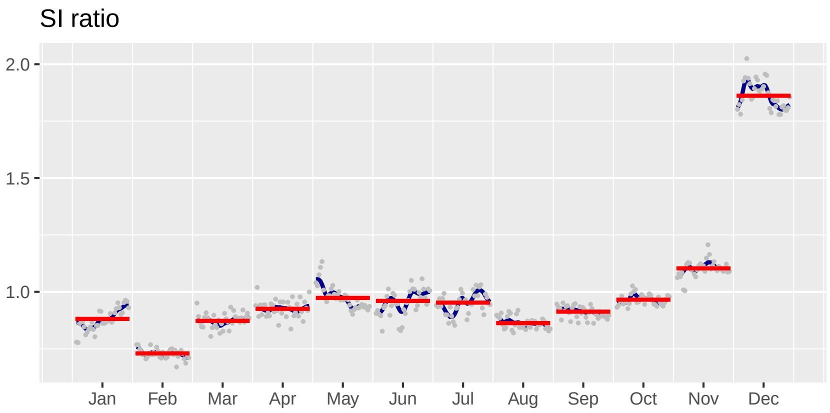
```
mod <- rjd3x13::x13(y, spec)
autoplot(mod)
```

ggdemetra3 (2)



```
ggsiratioplot(mod)
```

ggdemetra3 (3)



rjd3filters

- easily create/combine/apply moving averages `moving_average()` (much more general than `stats::filter()`) and study their properties: plot coefficients (`plot_coef()`), gain (`plot_gain()`), phase-shift (`plot_phase()`) and different statics (`diagnostic_matrix()`)

rjd3filters

- easily create/combine/apply moving averages `moving_average()` (much more general than `stats::filter()`) and study their properties: plot coefficients (`plot_coef()`), gain (`plot_gain()`), phase-shift (`plot_phase()`) and different statics (`diagnostic_matrix()`)

- trend-cycle extraction with different methods to treat endpoints:
  - `lp_filter()` local polynomial filters of Proietti and Luati (2008) (including Musgrave): Henderson, Uniform, biweight, Trapezoidal, Triweight, Tricube, "Gaussian", Triangular, Parabolic (= Epanechnikov)
  - `rkhs_filter()` Reproducing Kernel Hilbert Space (RKHS) of Dagum and Bianconcini (2008) with same kernels
  - `fst_filter()` FST approach of Grun-Rehomme, Guggemos, and Ladiray (2018)
  - `dfa_filter()` derivation of AST approach of Wildi and McElroy (2019)

rjd3filters

- easily create/combine/apply moving averages `moving_average()` (much more general than `stats::filter()`) and study their properties: plot coefficients (`plot_coef()`), gain (`plot_gain()`), phase-shift (`plot_phase()`) and different statics (`diagnostic_matrix()`)
- trend-cycle extraction with different methods to treat endpoints:
  - `lp_filter()` local polynomial filters of Proietti and Luati (2008) (including Musgrave): Henderson, Uniform, biweight, Trapezoidal, Triweight, Tricube, "Gaussian", Triangular, Parabolic (= Epanechnikov)
  - `rkhs_filter()` Reproducing Kernel Hilbert Space (RKHS) of Dagum and Bianconcini (2008) with same kernels
  - `fst_filter()` FST approach of Grun-Rehomme, Guggemos, and Ladiray (2018)
  - `dfa_filter()` derivation of AST approach of Wildi and McElroy (2019)
- change the filter used in X-11 for TC extraction

Create moving average `moving_average()`

(Recall: $B^i X_t = X_{t-p}$ and $F^i X_t = X_{t+p}$)

Goal : manipulate moving averages

$$M_\theta(X_t) = \sum_{k=-p}^{+f} \theta_k X_{t+k} = \left(\sum_{k=-p}^{+f} \theta_k F^k\right) X_t$$

Create moving average `moving_average()`

(Recall: $B^i X_t = X_{t-p}$ and $F^i X_t = X_{t+p}$)

Goal : manipulate moving averages

$$M_\theta(X_t) = \sum_{k=-p}^{+f} \theta_k X_{t+k} = \left(\sum_{k=-p}^{+f} \theta_k F^k \right) X_t$$

Currently in R, with `filter()` you can only manipulate symmetric filters ($p = f$, $\theta_{-k} = \theta_{-k}$) or real-time asymmetric filters:

$$M_\theta(X_t) = \left(\sum_{k=-p}^{0} \theta_k F^k \right) X_t = \theta_0 X_t + \cdots + \theta_{-p} X_{t-p}$$

Create moving average `moving_average()`

(Recall: $B^i X_t = X_{t-p}$ and $F^i X_t = X_{t+p}$)

Goal : manipulate moving averages

$$M_\theta(X_t) = \sum_{k=-p}^{+f} \theta_k X_{t+k} = \left(\sum_{k=-p}^{+f} \theta_k F^k\right) X_t$$

Currently in R, with `filter()` you can only manipulate symmetric filters ($p = f$, $\theta_{-k} = \theta_{-k}$) or real-time asymmetric filters:

$$M_\theta(X_t) = \left(\sum_{k=-p}^{0} \theta_k F^k\right) X_t = \theta_0 X_t + \cdots + \theta_{-p} X_{t-p}$$

In practice, you often need to combine filters

Create moving average moving_average() (1)

```
library(rjd3filters)
m1 = moving_average(rep(1,3), lags = 1); m1 # Forward MA
```

```
## [1] " F +  F^2 +  F^3"
```

```
m2 = moving_average(rep(1,3), lags = -1); m2 # centered MA
```

```
## [1] " B + 1,0000 +  F"
```

```
m1 + m2
```

```
## [1] " B + 1,0000 + 2,0000 F +  F^2 +  F^3"
```

```
m1 - m2
```

```
## [1] " -  B - 1,0000 +  F^2 +  F^3"
```

```
m1 * m2
```

```
## [1] "1,0000 + 2,0000 F + 3,0000 F^2 + 2,0000 F^3 +  F^4"
```

Create moving average moving_average() (2)

Can be used to create all the MA of X-11:

```
e1 <- moving_average(rep(1,12), lags = -6)
e1 <- e1/sum(e1)
e2 <- moving_average(rep(1/12, 12), lags = -5)
# used to have the 1rst estimate of the trend
tc_1 <- M2X12 <- (e1 + e2)/2
coef(M2X12) |> round(3)
```

```
##   t-6   t-5   t-4   t-3   t-2   t-1     t   t+1   t+2   t+3
## 0.042 0.083 0.083 0.083 0.083 0.083 0.083 0.083 0.083 0.083
##   t+4   t+5   t+6
## 0.083 0.083 0.042
```

```
si_1 <- 1 - tc_1
M3 <- moving_average(rep(1/3, 3), lags = -1)
M3X3 <- M3 * M3
# M3X3 moving average applied to each month
coef(M3X3) |> round(3)
```

```
##   t-2   t-1     t   t+1   t+2
## 0.111 0.222 0.333 0.222 0.111
```

Create moving average `moving_average()` (3)

```
M3X3_seasonal <- to_seasonal(M3X3, 12)
coef(M3X3_seasonal) |> round(3)
```

```
##  t-24  t-23  t-22  t-21  t-20  t-19  t-18  t-17  t-16  t-15
## 0.111 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##  t-14  t-13  t-12  t-11  t-10   t-9   t-8   t-7   t-6   t-5
## 0.000 0.000 0.222 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##   t-4   t-3   t-2   t-1     t   t+1   t+2   t+3   t+4   t+5
## 0.000 0.000 0.000 0.000 0.333 0.000 0.000 0.000 0.000 0.000
##   t+6   t+7   t+8   t+9  t+10  t+11  t+12  t+13  t+14  t+15
## 0.000 0.000 0.000 0.000 0.000 0.000 0.222 0.000 0.000 0.000
##  t+16  t+17  t+18  t+19  t+20  t+21  t+22  t+23  t+24
## 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.111
```

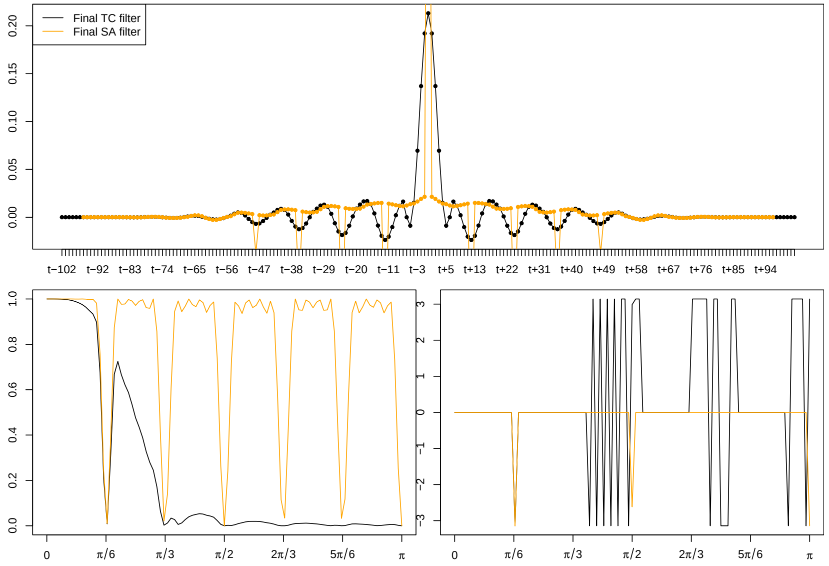
Create moving average moving_average() (4)

```
s_1 <- M3X3_seasonal * si_1
s_1_norm <- (1 - M2X12) * s_1
sa_1 <- 1 - s_1_norm
henderson_mm = moving_average(lp_filter(horizon = 6)$
                                  filters.coef[, "q=6"],
                              lags = -6)
tc_2 <- henderson_mm * sa_1
si_2 <- 1 - tc_2
M5 <- moving_average(rep(1/5, 5), lags = -2)
M5X5_seasonal <- to_seasonal(M5 * M5, 12)
s_2 <- M5X5_seasonal * si_2
s_2_norm <- (1 - M2X12) * s_2
sa_2 <- 1 - s_2_norm
tc_f <- henderson_mm * sa_2
```

Create moving average moving_average() (5)

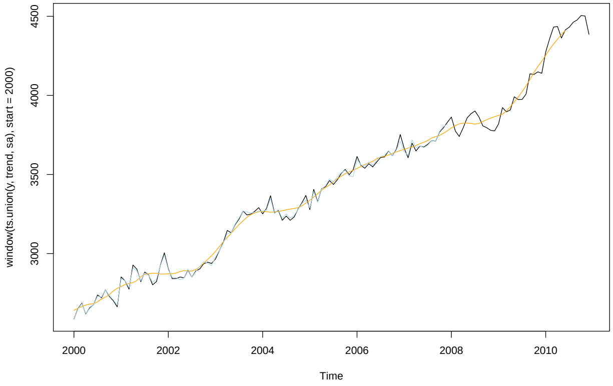
```r
par(mai = c(0.3, 0.3, 0.2, 0))
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))

plot_coef(tc_f);plot_coef(sa_2, col = "orange", add = TRUE)
legend("topleft",
       legend = c("Final TC filter", "Final SA filter"),
       col= c("black", "orange"), lty = 1)
plot_gain(tc_f);plot_gain(sa_2, col = "orange", add = TRUE)
plot_phase(tc_f);plot_phase(sa_2, col = "orange", add = TRUE)
```

Create moving average `moving_average()` (6)



Apply a moving average

```
y <- retailsa$AllOtherGenMerchandiseStores
trend <- y * tc_1
sa <- y * sa_1
plot(window(ts.union(y, trend, sa), start = 2000),
     plot.type = "single",
      col = c("black","orange", "lightblue"))
```



rjd3sts

Interface to structural time series and state space models

Several examples available here https://github.com/palatej/test_rjd3sts

rjd3bench

Benchmarking and temporal disaggregation

Several examples here: https://github.com/palatej/test_rjd3bench

Contents

Conclusion

With JDemetra+ 3.0, lots of new R packages are coming:

- On time series analysis and seasonal adjustment (much faster than standard packages)
- New developments on seasonal adjustment will be available (e.g. high-frequency data)
- Allow to create new trainings thanks to a deeper acces to all the functionalities of JDemetra+

Conclusion

With JDemetra+ 3.0, lots of new R packages are coming:

- On time series analysis and seasonal adjustment (much faster than standard packages)
- New developments on seasonal adjustment will be available (e.g. high-frequency data)
- Allow to create new trainings thanks to a deeper acces to all the functionalities of JDemetra+

Many ways to contribute:

- Testing it and reporting issues
- Developping new tools (other packages, new functions, etc.)

Thank you for your attention

Packages R:

- palatej/rjd3toolkit
- palatej/rjd3modelling
- palatej/rjd3sa
- palatej/rjd3arima
- palatej/rjd3x13
- palatej/rjd3tramoseats
- palatej/rjdemetra3
- palatej/rjdfilters
- palatej/rjd3sts
- palatej/rjd3stl
- palatej/rjd3highfreq
- palatej/rjd3bench
- AQLT/ggdemetra3