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
Session 10: Seasonal and Calendar Adjustment
Friday 23 September 2022

Contents

Introduction (1)

- In March 2019, `RJDemetra` was published on CRAN:
  - first 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:
  - first 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 12 R packages are being developped! Not only on seasonal adjustment!

Introduction (1)

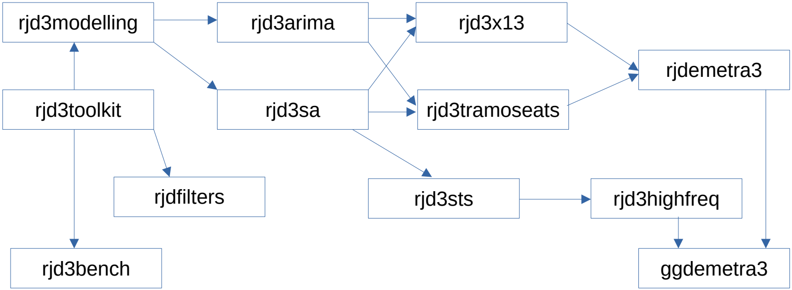
- In March 2019, `RJDemetra` was published on CRAN:
  - first R package that enables to use TRAMO-SEATS
  - faster than existing R packages on seasonal adjustment
  - enables to interact with JDemetra+ "workspaces" used in production
- With the development of JDemetra+ 3.0, more than 12 R packages are being developped! Not only on seasonal adjustment!
- They are require Java $\geq$ 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:

```
# 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/rjd3bench")
remotes::install_github("AQLT/ggdemetra3")
```

Introduction (3)



And it's just the beggining!

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()`)
- autocorrelations function (usual, inverse, partial)
- `aggregate` to aggregate a time serie to a higher frequency

Examples (1)

```
library(rjd3toolkit)
set.seed(100)
x = rnorm(1000);y = rlnorm(1000)
bowmanshenton(x) # normal distribution
```

```
## Value:  0.3117551
## P-Value:  0.8557
```

```
bowmanshenton(y) # log-normal distribution
```

```
## Value:  33551.78
## P-Value:  0.0000
```

```
testofruns(x) # random data
```

```
## Value:  1.396856
## P-Value:  0.1625
```

```
testofruns(y) # random data
```

```
## Value:  -0.1150397
## P-Value:  0.9084
```

Examples (2)

```
testofruns(1:1000) # non-random data
```

```
## Value:  -31.57534
## P-Value:  0.0000
```

```
autocorrelations(x)
```

```
##            1            2            3            4
## -0.039797636 -0.028616535  0.038409192  0.012282902
##            5            6            7            8
## -0.035815187 -0.008406605  0.010077238  0.037414192
##            9           10           11           12
## -0.063957619 -0.015995017 -0.003748914  0.016326224
##           13           14           15
## -0.051273264 -0.015552059  0.035965008
```

```
autocorrelations.inverse(x)
```

Examples (3)

```
##           1           2           3           4
## -0.038225207 -0.030030005  0.034985887  0.014697477
##           5           6           7           8
## -0.032164035 -0.012375939  0.005587471  0.039725092
##           9          10          11          12
## -0.057199640 -0.020771981 -0.011968366  0.019437797
##          13          14          15
## -0.043170872 -0.021167341  0.027156206
```

```
autocorrelations.partial(x)
```

```
##           1           2           3           4
## -0.039797636 -0.030248296  0.036122272  0.014485158
##           5           6           7           8
## -0.032734128 -0.011864534  0.006444671  0.040137674
##           9          10          11          12
## -0.059177846 -0.020600211 -0.012229212  0.019298100
##          13          14          15
## -0.045255005 -0.021485597  0.028314840
```

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()`)

Example of a specific calendar (1)

```
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 = T)
```

```
##             [,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)

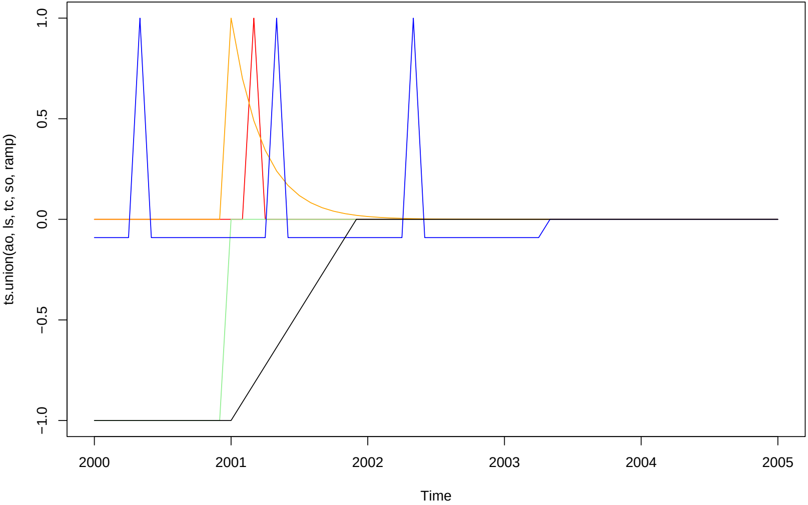
```
s = ts(0, start = 2020, end = c(2020, 11), frequency = 12)
# Trading-days regressors (each day has a different effect, sunday as contrasts)
td_reg <- htd(fr_cal, s = s, groups = c(1, 2, 3, 4, 5, 6, 0))
# Working-days regressors (Monday = ... = Friday; Saturday = Sunday = contrasts)
wd_reg <- htd(fr_cal,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

```r
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)



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)
seasonality.f(y, 12)
```

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

```
seasonality.friedman(y, 12)
```

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

```
seasonality.kruskalwallis(y, 12)
```

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

```
seasonality.combined(y, 12)
```

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

Contents

rjd3arima

Utility functions used in `rjd3x13` and `rjd3tramoseats` 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()` and `remove_ramp()`‘

`add_usrdefvar()` not yet available

rjd3arima

Utility functions used in `rjd3x13` and `rjd3tramoseats` 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()` and remove_ramp()‘

`add_usrdefvar()` not yet available

Functions commons to RegARIMA and TRAMO

rjd3arima

Utility functions used in `rjd3x13` and `rjd3tramoseats` 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()` and remove_ramp()‘

`add_usrdefvar()` not yet available

Functions commons to RegARIMA and TRAMO

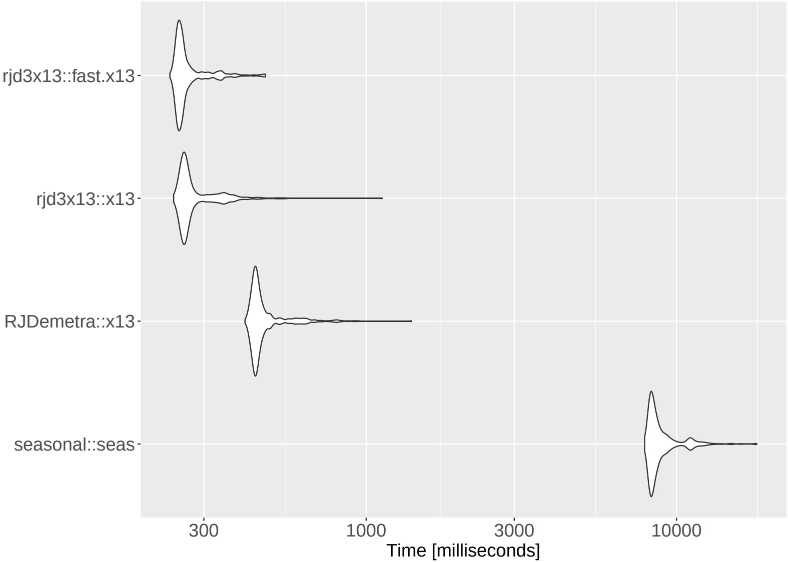
In `RJDemetra` you have one function to set the specification (`regarima_spec_x13()`, `regarima_spec_tramo()`, `x13_spec()` and `tramoseats_spec()`) now one function for each part of the specification

rjd3x13

Main functions:

- Specification: created with `spec_x11_default()`, `spec_x13_default()`, `spec_regarima_default()` and customized with `rjd3arima` functions + `set_x11()`
- Apply model with `x11()`, `x13()`, `fast.x13()`, `regarima()`, `fast.regarima()`
- Refresh policies: `regarima.refresh()` and `x13.refresh()`

Performance



Exemple (1)

```r
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::fast.x13(y, spec)
# m is a list with several outputs:
names(m)
```

```
## [1] "preprocessing" "preadjust"     "decomposition"
## [4] "final"         "mstats"        "diagnostics"
## [7] "user_defined"
```

```r
m
```

Exemple (2)

```
## RegARIMA
## 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
## 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
```

Exemple (3)

```
## 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
##
##
## Decomposition
## Monitoring and Quality Assessment Statistics:
##     M stats
## m1    0.045
## m2    0.043
## m3    1.778
## m4    0.403
```

Exemple (4)

```
## m5     1.419
## m6     0.020
## m7     0.052
## m8     0.155
## m9     0.049
## m10    0.116
## m11    0.112
## q      0.410
## qm2    0.455
##
## Final filters:
## Seasonal filter:
## Trend filter: 13 terms Henderson moving average
##
## Diagnostics
## Relative contribution of the components to the stationary
## portion of the variance in the original series,
## after the removal of the long term trend (in %)
##
##              Component
##  cycle           13.508
```

Exemple (5)

```
##   seasonal      86.645
##   irregular      0.429
##   calendar       0.688
##   others         0.004
##   total        101.274
##
## Residual seasonality tests
##                      P.value
##   seas.ftest.i         0.975
##   seas.ftest.sa        0.999
##   seas.qstest.i        0.984
##   seas.qstest.sa       1.000
##   td.ftest.i           0.982
##   td.ftest.sa          0.982
##
##
## Final
## Last values
##            series       sa    trend seas       irr
## Sep 2016 1393.5 1537.129 1537.064    1 1.0000420
## Oct 2016 1497.4 1588.929 1531.988    1 1.0371684
```

Exemple (6)

```
## Nov 2016 1684.3 1520.076 1532.076    1 0.9921677
## Dec 2016 2850.4 1535.647 1537.080    1 0.9990677
## Jan 2017 1428.5 1547.286 1544.701    1 1.0016735
## Feb 2017 1092.4 1547.740 1552.749    1 0.9967744
## Mar 2017 1370.3 1554.062 1557.995    1 0.9974762
## Apr 2017 1522.6 1588.035 1557.819    1 1.0193965
## May 2017 1452.4 1556.976 1553.193    1 1.0024353
## Jun 2017 1557.2 1533.334 1546.419    1 0.9915389
## Jul 2017 1445.5 1535.987 1540.819    1 0.9968643
## Aug 2017 1303.1 1518.261 1537.522    1 0.9874725
```

```
summary(m$preprocessing)
```

```
## Log-transformation: yes
## SARIMA model:  (0,1,2) (1,1,1)
##
## Coefficients
##           Estimate Std. Error  T-stat Pr(>|t|)
## theta(1)  -1.01804    0.07639 -13.326  < 2e-16 ***
## theta(2)   0.20863    0.05378   3.879 0.000123 ***
## bphi(1)   -0.26680    0.05399  -4.942 1.14e-06 ***
## btheta(1) -0.77559    0.05384 -14.405  < 2e-16 ***
```

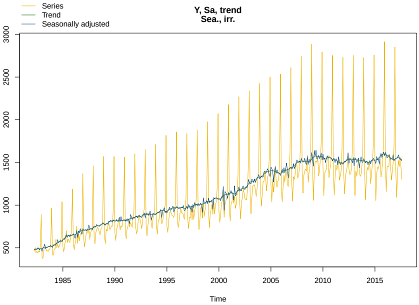
Exemple (7)

```
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Regression model:
##                    Estimate Std. Error T-stat Pr(>|t|)
## monday            -0.011247   0.004004 -2.809 0.005219 **
## tuesday            0.005870   0.004013  1.463 0.144306
## wednesday         -0.002002   0.004003 -0.500 0.617304
## thursday           0.014483   0.004021  3.602 0.000356 ***
## friday             0.001577   0.004023  0.392 0.695391
## saturday           0.011465   0.003996  2.869 0.004333 **
## lp                 0.037501   0.010994  3.411 0.000713 ***
## easter             0.053486   0.008319  6.429 3.67e-10 ***
## My TC in 2000-06   0.022947   0.023666  0.970 0.332814
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Number of observations:  425 , Number of effective observations:  412 , Numbe
## Loglikelihood:  763.5143, Adjusted loglikelihood:  -2104.113
## Standard error of the regression (ML estimate):  0.03757223
```

Exemple (8)

```
## AIC:  4236.225 , AICc:  4237.283 , BIC:  4292.519
```

```r
plot(m)
```



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)
# More informations:
names(m)
```

```
## [1] "result"          "estimation_spec" "result_spec"
## [4] "user_defined"
```

```r
m$result
```

```
## TRAMO
## Log-transformation: yes
## SARIMA model:  (2,1,2) (0,1,1)
##
## Coefficients
##           Estimate Std. Error T-stat
## phi(1)    -0.14639    0.43584 -0.336
## phi(2)     0.10790    0.09393  1.149
## theta(1)  -1.08360    0.43778 -2.475
```

Exemple (2)

```
## theta(2)    0.29094    0.34667  0.839
## btheta(1) -0.44535    0.06267 -7.107
##
## Regression model:
##                  Estimate Std. Error T-stat
## monday          -0.012187   0.003628 -3.359
## tuesday          0.005855   0.003667  1.597
## wednesday        0.000611   0.003632  0.168
## thursday         0.012270   0.003685  3.330
## friday          -0.001877   0.003670 -0.511
## saturday         0.014919   0.003655  4.082
## lp               0.038721   0.010019  3.865
## easter           0.053208   0.008117  6.556
## AO (2000-07-01) -0.182202   0.029404 -6.197
## AO (2000-06-01)  0.173258   0.029500  5.873
## Number of observations:  425
## Number of effective observations:  412
## Number of parameters:  16
##
## Loglikelihood:  785.0729
## Adjusted loglikelihood:  -2082.554
```

Exemple (3)

```
##
## Standard error of the regression (ML estimate):  0.03582226
## AIC:  4197.108
## AICC:  4198.485
## BIC:  4261.444
##
##
## Decomposition
## model
##
## AR:  1 -0.1463901 0.1079007
## DIF:  1 -1 0 0 0 0 0 0 0 0 0 0 -1 1
## MA:  1 -1.083604 0.2909389 0 0 0 0 0 0 0 0 0 -0.4453507 0.4825838 -0.1295699
## var:  1
##
## trend
##
## DIF:  1 -2 1
## MA:  1 0.06428983 -0.9357102
## var:  0.006008503
##
```

Exemple (4)

```
## seasonal
##
## DIF:  1 1 1 1 1 1 1 1 1 1 1 1
## MA:  1 0.4148092 0.07471926 -0.02314904 -0.09472672 -0.1666228 -0.2176017 -0.
## var:  0.1403324
##
## transitory
##
## AR:  1 -0.1463901 0.1079007
## MA:  1 -0.9028412 -0.09715885
## var:  0.1324809
##
## irregular
##
## var:  0.1411435
##
##
## Diagnostics
## Relative contribution of the components to the stationary
## portion of the variance in the original series,
## after the removal of the long term trend (in %)
```

Exemple (5)

```
##
##               Component
##   cycle           0.342
##   seasonal       97.001
##   irregular       0.558
##   calendar        0.755
##   others          0.306
##   total          98.962
##
## Residual seasonality tests
##                    P.value
##   seas.ftest.i       1.000
##   seas.ftest.sa      1.000
##   seas.qstest.i      1.000
##   seas.qstest.sa     1.000
##   td.ftest.i         0.999
##   td.ftest.sa        0.999
##
##
## Final
## Last values
```

Exemple (6)

```
##             series       sa    trend      seas       irr
## Sep 2016 1393.5 1550.895 1558.077 0.8985132 0.9953904
## Oct 2016 1497.4 1568.003 1555.153 0.9549727 1.0082629
## Nov 2016 1684.3 1528.301 1552.937 1.1020733 0.9841360
## Dec 2016 2850.4 1543.909 1551.947 1.8462222 0.9948212
## Jan 2017 1428.5 1546.610 1552.150 0.9236331 0.9964306
## Feb 2017 1092.4 1550.336 1553.025 0.7046215 0.9982684
## Mar 2017 1370.3 1553.185 1554.073 0.8822515 0.9994289
## Apr 2017 1522.6 1582.383 1554.508 0.9622198 1.0179317
## May 2017 1452.4 1555.526 1553.761 0.9337034 1.0011358
## Jun 2017 1557.2 1552.133 1552.228 1.0032648 0.9999388
## Jul 2017 1445.5 1545.388 1550.589 0.9353637 0.9966456
## Aug 2017 1303.1 1534.518 1549.372 0.8491916 0.9904127
```

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`

rjd3highfreq

Seasonal adjustment of high frequency data:

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

See Session 3: High Frequency Data and
https://github.com/palatej/test_rjd3hf

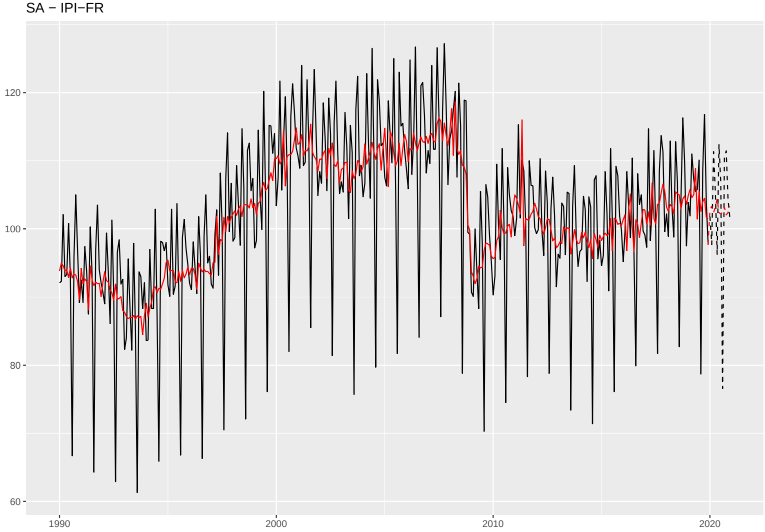
Contents

ggdemetra3 (1)

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

```
library(ggdemetra3)
spec <- spec_x13_default("rsa3") |> set_tradingdays(option = "WorkingDays")
p_ipi_fr <- ggplot(data = ipi_c_eu_df, mapping = aes(x = date, y = FR)) +
    geom_line() +
    labs(title = "SA - IPI-FR",
         x = NULL, y = NULL)
p_sa <- p_ipi_fr +
    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)
p_sa
```

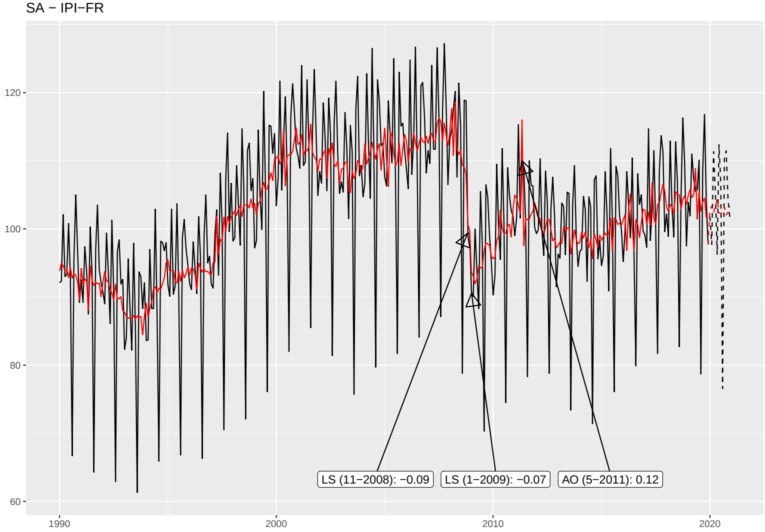
ggdemetra3 (2)



ggdemetra3 (3)

```r
p_sa +
    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 (4)



rjdfilters (1)

- 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()`)

rjdfilters (1)

- 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)

`rjdfilters` (1)

- 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()` (1)

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

```
library(rjdfilters)
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
```

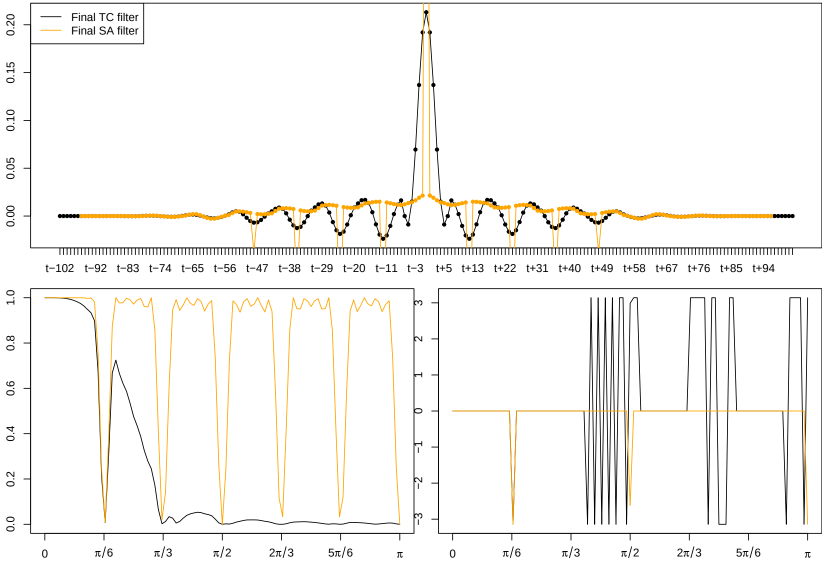
Create moving average moving_average() (4)

```
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)

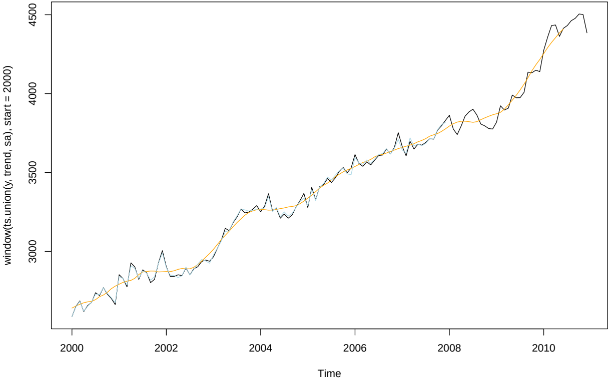
```
par(mai = c(0.3, 0.3, 0.2, 0))
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))

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/rjd3highfreq
- palatej/rjd3bench
- AQLT/ggdemetra3