R packages around JDemetra+ - Part 2

A versatile toolbox for time series analysis

Anna Smyk and Tanguy Barthelemy (Insee, France)

UROS Conference, Athens (GR), Nov 27th 2024

Insee
Mesurer pour comprendre

Time Series Software for Official Statistics

Section 1

rjd3 packages part 2

Subsection 1

SA of High-Frequency data

High-Frequency data specificities

Specificity: High-frequency data can display multiple and non integer periodicities:

For example a daily series might display 3 periodicities:

- **weekly** ($p = 7$): Mondays are alike and different from Sundays (DOW)
- **intra-monthly** ($p = 30.44$): the last days of each month are different from the first ones (DOM)
- **yearly** ($p = 365.25$): from on year to another the 15th of June are alike, summer days are alike (DOY)

Classic algorithms not directly applicable

Two classes of solutions:

- round periodicities (might involve imputing data) (extended STL,..)
- use approximations for fractional backshift powers (extended X13-Arima and Tramo-Seats)

For methodologiocal details see JOS Paper, Webel and Smyk (2024)

High-Frequency data in rjd3 packages

In packages for HF data:

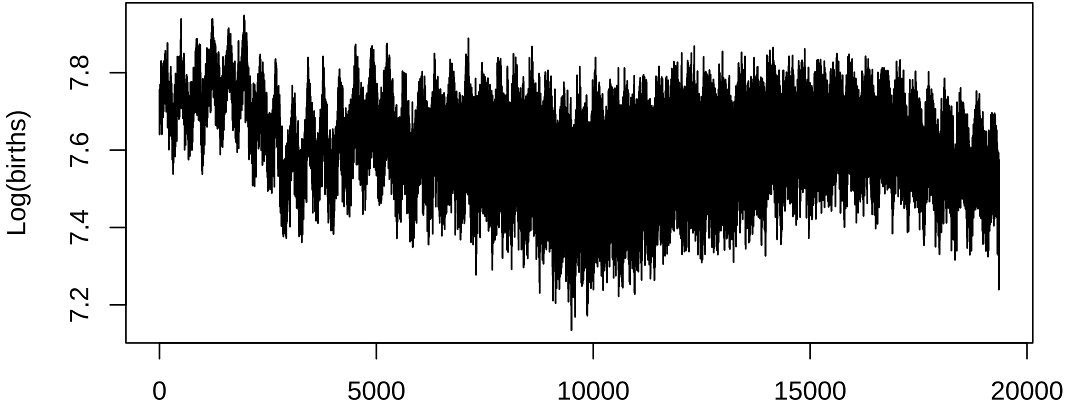
- No constraint on data input as no TS structure (numeric vector)
- Any seasonal patters, positive numbers
- Linearisation with **fractional airline model** (correction for calendar effects and outlier detection)
- Iterative decomposition (extended X-11 and Seats) starting with the highest frequency

Packages perimeters

- **{rjd3highfreq}**: Extended airline model, AMB decomposition (extended SEATS)
- **{rjd3x11plus}** contains all the Extended X11 functions for any (high) frequency data, and new trend estimation filters (weighted polynomials), depends on **{rjd3filters}**
- **{rjd3stl}** (Loess based) and **{rjd3sts}** (SSF based) are the two other tools to decompose high (any)- periodicity data.

Data initialization

```
df_daily <- read_csv2(file.path("Data", "TS_daily_births_franceM_1968_2020.cs
    mutate(log_births = log(births))
plot(df_daily$log_births, type = "l", ylab = "Log(births)")
```



Canova-Hansen test to identify (multiple) seasonal patterns

```
rjd3toolkit::seasonality_canovahansen_trigs(
    data = df_daily$births,
    periods = seq(from = 1 / 367, to = 1 / 2, by = 0.001)
)
```

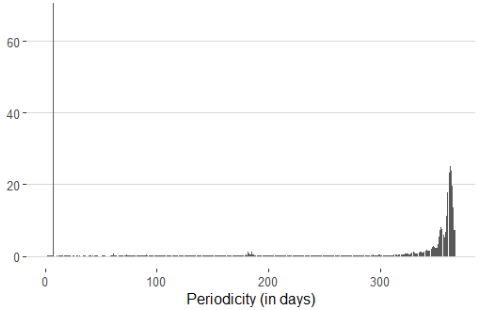


Figure 1: Canova Hansen seasonality test

Linearization

```
# calendar regressors can be defined with the rjd3toolkit package
# see below how to generate the calendar (here french_calendar) first
q ← rjd3toolkit::holidays(
    calendar = french_calendar,
    "1968-01-01", length = 200000, type = "All", nonworking = 7L
)
# pre-adjustment
rjd3highfreq::fractionalAirlineEstimation(
    y = df_daily$log_births, # here a daily series in log
    x = q, # q = calendar
    periods = 7, # approx  c(7,365.25)
    ndiff = 2, ar = FALSE, mean = FALSE,
    outliers = c("ao", "wo", "ls"),
    # WO compensation
    criticalValue = 0, # computed in the algorithm
    precision = 1e-9, approximateHessian = TRUE
)
```

See **{rjd3highfreq}** help pages

Decomposition with extended X-11

```r
# step 1: p = 7
x11.dow <- rjd3x11plus::x11(
    ts = exp(pre.mult$model$linearized),
    period = 7, # DOW pattern
    mul = TRUE,
    trend.horizon = 9, # 1/2 Filter length : not too long vs p
    trend.degree = 3, # Polynomial degree
    trend.kernel = "Henderson", # Kernel function
    trend.asymmetric = "CutAndNormalize", # Truncation method
    seas.s0 = "S3X9", seas.s1 = "S3X9", # Seasonal filters
    extreme.lsig = 1.5, extreme.usig = 2.5
) # Sigma-limits
# step 2: p = 365.25
x11.doy <- rjd3highfreq::x11(x11.dow$decomposition$sa, # previous sa
    period = 365.2425, # DOY pattern
    mul = TRUE
) # other parameters skipped here
```

Decomposition with extended Seats

```
# step 1: p = 7
# step 2: p = 365.25
amb.doy ← rjd3highfreq::fractionalAirlineDecomposition(
    amb.dow$decomposition$sa, # DOW-adjusted linearised data
    period = 365.2425, # DOY pattern
    sn = FALSE, # Signal (SA)-noise decomposition
    stde = FALSE, # Compute standard deviations
    nbcasts = 0,
    nfcasts = 0
) # Numbers of back- and forecasts
```

Section 2

Revision Analysis

Revision Analysis

```r
library("rjd3revisions")
```

The package **{rjd3revisions}** performs **revision analysis**.

It offers a battery of relevant tests on revisions and submit a visual report including both the main results and their interpretation. The tool can perform analysis on different types of revision intervals and on different vintage views.

The vignette is here.

What is revision analysis?

Revision analysis is composed on a selection of **parametric tests** which enable the users to detect potential bias (both mean and regression bias) and other sources of inefficiency in preliminary estimates.

Data structure

Your input data must be in a specific format: long, vertical or horizontal.

There are 2 types of period in the study of revisions:

- the `time_period`, which designates the reference period to which the value refers
- the `revision_date`, which designates the date on which the value was published

For example, for a series, the September 2023 point may be published for the first time in October 2023, then revised in November 2023 and even in September 2024.

Vertical format

Here we imagine a series in which each point is published from the 1st of the month.

	2012-01-31	2012-02-09	2012-05-27
Jan 2012	12.3	13.2	12.8
Feb 2012	NA	16.4	16.8
Mar 2012	NA	NA	19.3
Apr 2012	NA	NA	15.0

Long format

```
   revdate       time obs_values
1  2012-01-31 2012-01-01       12.3
2  2012-01-31 2012-02-01         NA
3  2012-01-31 2012-03-01         NA
4  2012-01-31 2012-04-01         NA
5  2012-02-09 2012-01-01       13.2
6  2012-02-09 2012-02-01       16.4
7  2012-02-09 2012-03-01         NA
8  2012-02-09 2012-04-01         NA
9  2012-05-27 2012-01-01       12.8
10 2012-05-27 2012-02-01       16.8
11 2012-05-27 2012-03-01       19.3
12 2012-05-27 2012-04-01       15.0
```

Horizontal format

	2012-01-01	2012-02-01	2012-03-01	2012-04-01
2012-01-31	12.3	NA	NA	NA
2012-02-09	13.2	16.4	NA	NA
2012-05-27	12.8	16.8	19.3	15

Diagonal format

	Release[1]	Release[2]	Release[3]
Jan 2012	12.3	13.2	12.8
Feb 2012	16.4	16.8	NA
Mar 2012	19.3	NA	NA
Apr 2012	15.0	NA	NA

Data simulation

The package **{rjd3revisions}** also lets you simulate data sets. You can choose :

- the periodicity,
- the number of revision periods,
- the number of study periods
- the start date of the period

```
long_format <- simulate_long(
    n_period = 12L * 5L,
    n_revision = 10L,
    periodicity = 12L
)
```

Creation of vintages

Then you can create your vintages with the function `create_vintages()`

```
vintages <- create_vintages(long_format, periodicity = 12L)
```

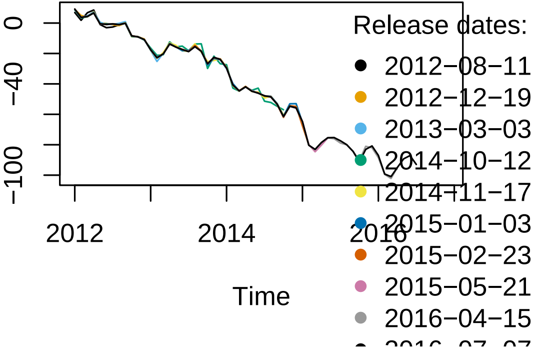
The function `get_revisions()` allows you to compute the revisions and observe the evolutions:

```
revisions <- get_revisions(vintages, gap = 2L)
```

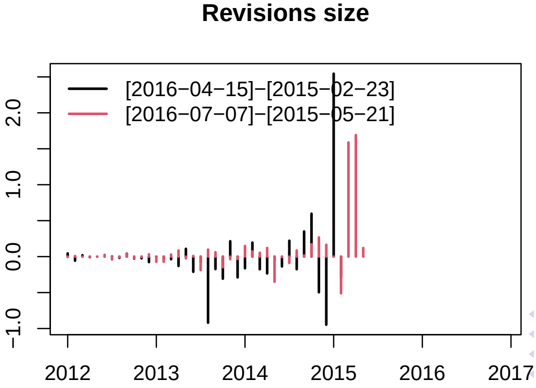
Plot

You can plot your vintages and the revisions :

```r
plot(vintages)
```




```r
plot(revisions)
```



Make the anaylsis of the revisions

Finally, you can make the analysis of the revisions with the function `revision_analysis()` :

```
rslt <- revision_analysis(vintages, gap = 1, view = "diagonal", n.releases =
```

Creating report

Additionnaly, to create a report and get a summary of the results, you can use

```
render_report(
    rslt,
    output_file = "my_report",
    output_dir = tempdir(),
    output_format = "pdf_document"
)
```

Section 3

Trend estimation

Local polynomial methods

Using **{rjd3filters}** for trend estimation

- Note detailed in this tutorial
- See examples in README file
- For more details see JOS Paper, Quartier-La-Tente (2024)

Section 4

Filtering data

Moving Averages

Moving Averages are used for smoothing and decomposition of time series:

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

{rjd3filters} offers features to

- Perform operations on MAs and build more complex ones
- Study their properties (plot, gain, phase...)

Using rjd3filters to wrangle Moving Averages I

(Notation: $B^i X_t = X_{t-i}$ et $F^i X_t = X_{t+i}$)

```r
library("rjd3filters")
m1 <- moving_average(rep(1, 4), lags = -2) / 4
m1
```

```
[1] "0.2500 B^2 + 0.2500 B + 0.2500 + 0.2500 F"
```

```r
m2 <- moving_average(rep(1, 3), lags = -1) / 3
m2
```

```
[1] "0.3333 B + 0.3333 + 0.3333 F"
```

```r
m1 + m2
```

```
[1] "0.2500 B^2 + 0.5833 B + 0.5833 + 0.5833 F"
```

Using rjd3filters to wrangle Moving Averages II

```r
m1 - m2
```

```
[1] "0.2500 B^2 - 0.0833 B - 0.0833 - 0.0833 F"
```

```r
m1 * m2
```

```
[1] "0.0833 B^3 + 0.1667 B^2 + 0.2500 B + 0.2500 + 0.1667 F + 0.0833 F^2"
```

```r
m1^2
```

```
[1] "0.0625 B^4 + 0.1250 B^3 + 0.1875 B^2 + 0.2500 B + 0.1875 + 0.1250 F + 0.0625 F
```

```r
rev(m1)
```

```
[1] "0.2500 B + 0.2500 + 0.2500 F + 0.2500 F^2"
```

Seasonality suppression I

For quarterly data M2*4

```
library("rjd3filters")
e1 <- simple_ma(4, lags = -2)
e1
```

```
[1] "0.2500 B^2 + 0.2500 B + 0.2500 + 0.2500 F"
```

```
e2 <- simple_ma(4, lags = -1)
e2
```

```
[1] "0.2500 B + 0.2500 + 0.2500 F + 0.2500 F^2"
```

```
# averaging MA's
M2X4 <- (e1 + e2) / 2
M2X4
```

Seasonality suppression II

```
[1] "0.1250 B^2 + 0.2500 B + 0.2500 + 0.2500 F + 0.1250 F^2"
```

```
# or convolution 1
m ← simple_ma(2, lags = 0)
m
```

```
[1] "0.5000 + 0.5000 F"
```

```
M2X4_2 ← m * e1
M2X4_2
```

```
[1] "0.1250 B^2 + 0.2500 B + 0.2500 + 0.2500 F + 0.1250 F^2"
```

```
# or convolution 2
m ← simple_ma(2, lags = -1)
m
```

Seasonality suppression III

```
[1] "0.5000 B + 0.5000"
```

```
M2X4_3 <- m * e2
M2X4_3
```

```
[1] "0.1250 B^2 + 0.2500 B + 0.2500 + 0.2500 F + 0.1250 F^2"
```

```
M2X4 - M2X4_2
```

```
[1] ""
```

```
M2X4 - M2X4_3
```

```
[1] ""
```

Seasonality extraction I

M3*3 filter

```
m3_1 <- moving_average(rep(1, 3), lags = -1) / 3
m3_1
```

```
[1] "0.3333 B + 0.3333 + 0.3333 F"
```

```
m3_2 <- moving_average(rep(1, 3), lags = -2) / 3
m3_2
```

```
[1] "0.3333 B^2 + 0.3333 B + 0.3333"
```

```
m3_3 <- moving_average(rep(1, 3), lags = 0) / 3
m3_3
```

```
[1] "0.3333 + 0.3333 F + 0.3333 F^2"
```

Seasonality extraction II

```
# averaging MA's
M3X3 <- (m3_1 + m3_2 + m3_3) / 3
M3X3
```

```
[1] "0.1111 B^2 + 0.2222 B + 0.3333 + 0.2222 F + 0.1111 F^2"
```

```
# Or convolution
M3X3_2 <- m3_1 * m3_1
```

```
M3X3 - M3X3_2
```

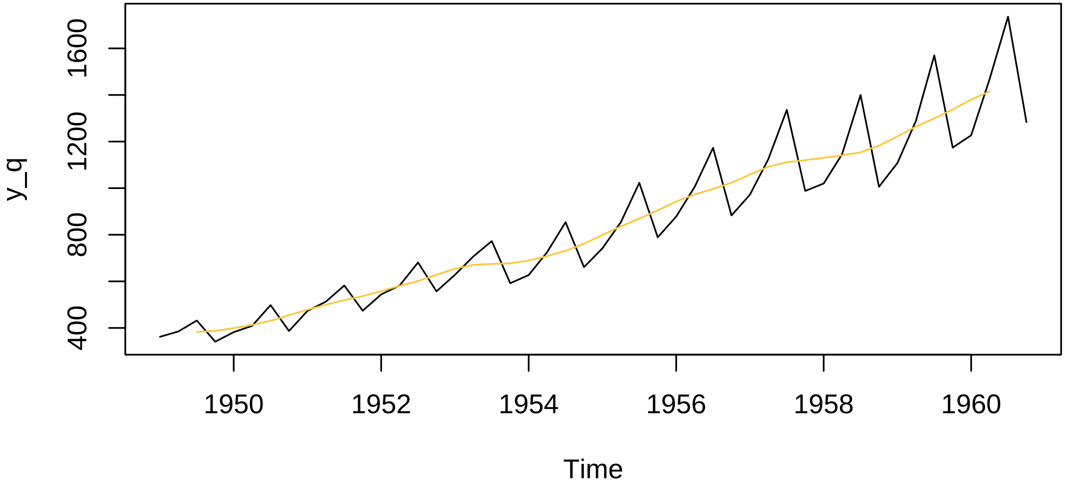
```
[1] ""
```

Seasonality extraction III

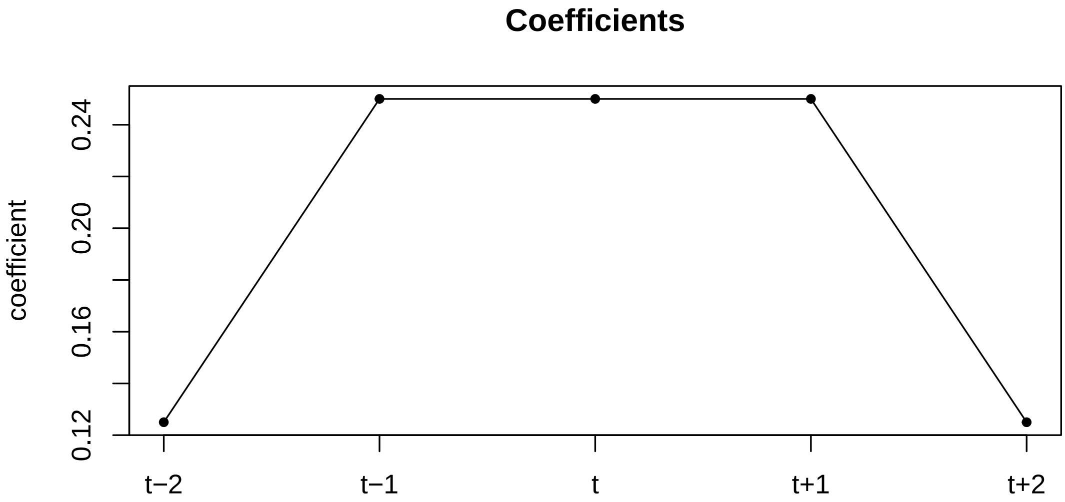
```
# Seasonal format
# q: horizon, q=0 : last data point

M3X3_s ← M3X3 * to_seasonal(M3X3, 4)
M3X3_s
```

```
[1] "0.0123 B^10 + 0.0247 B^9 + 0.0370 B^8 + 0.0247 B^7 + 0.0370 B^6 + 0.0494 B^5 +
```

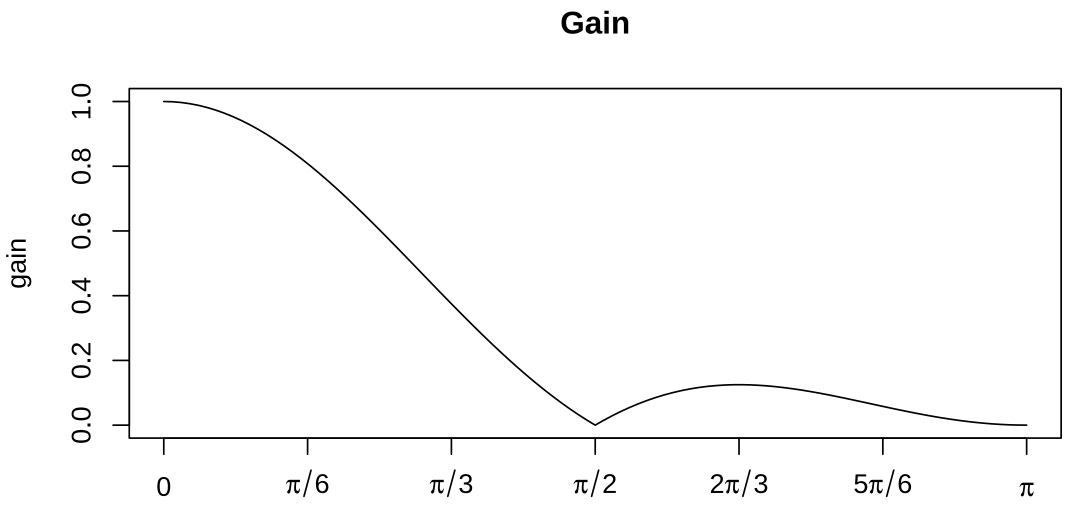
Using rjd3filters to wrangle Moving Averages I



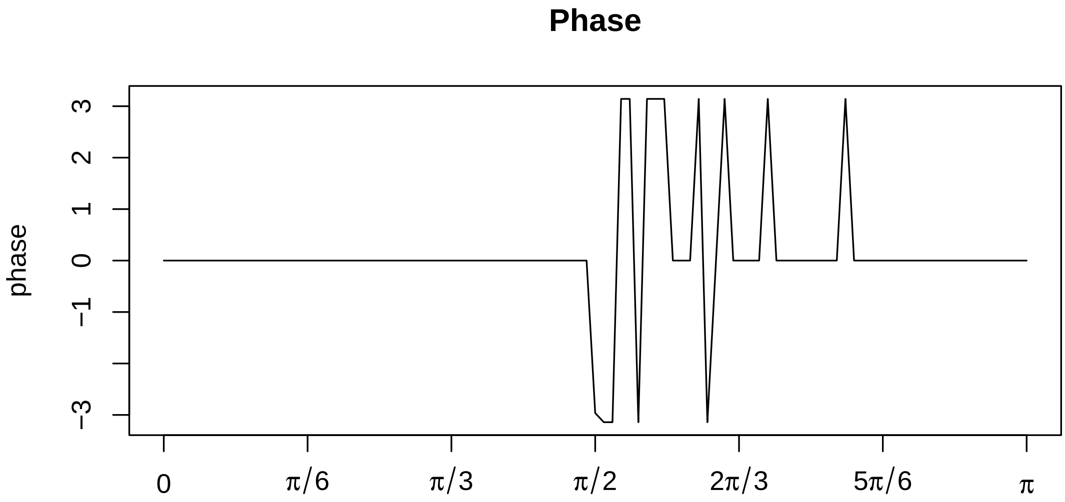
Using rjd3filters to wrangle Moving Averages II



Using rjd3filters to wrangle Moving Averages III



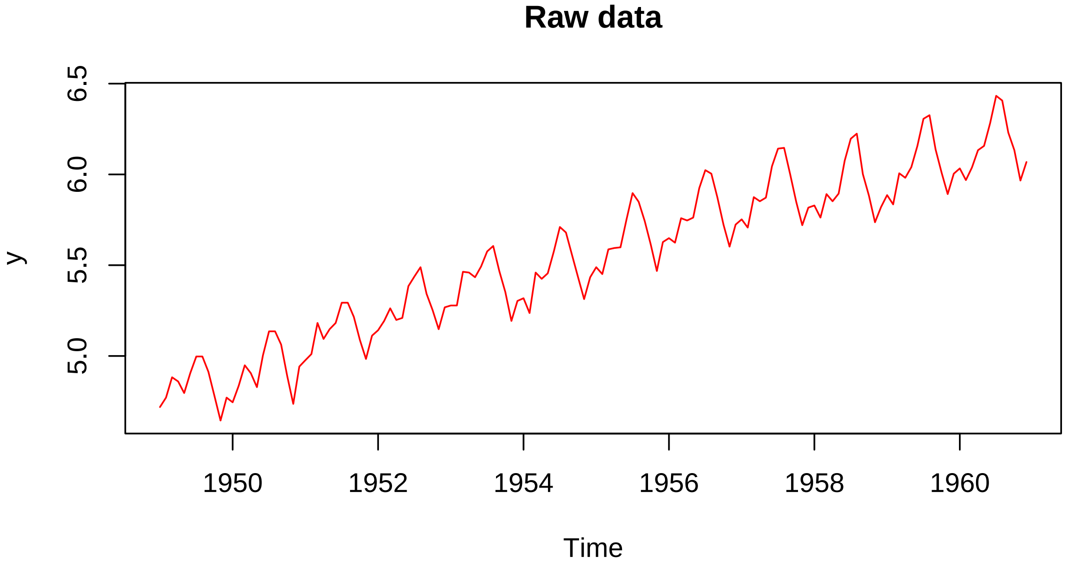
Using rjd3filters to wrangle Moving Averages IV



Simplified X-11 steps I

```
          q=2       q=1       q=0
t-2 0.1111111 0.1111111 0.1851852
t-1 0.2222222 0.2592593 0.4074074
t   0.3333333 0.3703704 0.4074074
t+1 0.2222222 0.2592593 0.0000000
t+2 0.1111111 0.0000000 0.0000000

len  ub
145  72
```

Simplified X-11 steps (2)



(Really) Reproducing X11 steps with `rjd3 filters`

Possibility de to reproduce X-11 algorithm fully, including correction for extreme values.

Example in related vignette
https://github.com/rjdverse/rjd3x11plus/blob/develop/vignettes/X11.Rmd

Section 5

Nowcasting

Different model types

```r
library("rjd3nowcasting")
```

{rjd3nowcasting} proposes the implementation of Dynamic Factor model (DFM). These are factor models using a state-space modeling structure to provide consistent forecasts.

The vignette is here.

DFM - equation

This is the state-space representation.

The underlying idea here is that factors f_t generate and predict variables y_t.

$$y_t = Zf_t + \epsilon_t, \quad \epsilon_t \sim N(0, R_t)$$
$$f_t = A_1 f_{t-1} + ... + A_p f_{t-p} + \eta_t, \quad \eta_t \sim N(0, Q_t)$$

But what data? I

Here we will take the data provided by the package. It comes from the French national statistical institute: Insee.

```r
data("data0", "data1")

tail(data0)
```

```
          date     FR_PVI FR_TURN FR_B1g_BCDE FR_BS FR_BS_prdexp EA_PVI EA_TURN
145 2024-01-01 -1.0000000    -1.8          NA  -8.4          6.0   -2.0    -3.7
146 2024-02-01  0.3988041     2.4          NA  -5.4          6.4    0.1     1.7
147 2024-03-01 -0.1000000    -1.2   0.9711108  -3.4          9.0    0.4    -0.6
148 2024-04-01  0.5000000      NA          NA  -7.7          6.5   -0.1      NA
149 2024-05-01         NA      NA          NA  -8.9          0.5     NA      NA
150       <NA>         NA      NA          NA    NA           NA     NA      NA
    EA_BS EA_BS_prdexp EA_PMI_manuf
```

But what data? II

```
145  -9.3           0.7          46.6
146  -9.5           0.8          46.5
147  -8.8           0.5          46.1
148 -10.3           0.6          45.7
149  -9.8           0.3          47.3
150    NA            NA            NA
```

```
tail(data1)
```

```
          date     FR_PVI   FR_TURN FR_B1g_BCDE FR_BS FR_BS_prdexp     EA_PVI
145 2024-01-01 -1.1916725 -1.689229          NA  -8.4          6.0 -2.2358655
146 2024-02-01  0.3988041  2.606285          NA  -5.4          6.4  0.0000000
147 2024-03-01 -0.1992033 -1.252626   0.9711108  -3.4          9.0  0.5125588
148 2024-04-01  0.5964232  1.501279          NA  -7.8          6.5  0.0000000
149 2024-05-01 -2.1032323        NA          NA  -8.9          0.5 -0.6153866
```

But what data? III

```
150 2024-06-01        NA        NA          NA  -7.9         2.3        NA
       EA_TURN EA_BS EA_BS_prdexp EA_PMI_manuf
145 -3.8781249  -9.3          0.7         46.6
146  1.5693435  -9.5          0.8         46.5
147 -0.4334641  -8.8          0.5         46.1
148  0.6063249 -10.4          0.6         45.7
149         NA  -9.9          0.3         47.3
150         NA -10.1          0.5         45.8
```

But what data?

These two datasets contain data on:

- Monthly industrial production index (PVI),
- Turnover (TURN),
- Quarterly GDP,
- Business survey data (BS)
- Other survey data (PMI) for both France and the Eurozone.

We will use these datasets to illustrate how one of these variable can be nowcasted using the others using a Dynamic Factor model.

Transforming our data

First we have to transform our data into ts object:

```
data0_ts ← data0 ▷
    select(-date) ▷
    ts(start = c(2012, 1), frequency = 12)
data1_ts ← data1 ▷
    select(-date) ▷
    ts(start = c(2012, 1), frequency = 12)
```

Here the `date` column will not be useful in the forecasting.

Creation of our first model

Our model here will initially be agnostic of our data, i.e. it does not depend on the values of our series but on the model we want to give to our forecasts. Nevertheless, it is important to know the structure of our data in order to structure our model properly.

```
dfm_model <- create_model(
    nfactors = 2,
    nlags = 2,
    factors_type = c("M", "M", "Q", "YoY", "YoY",
                     "M", "M", "YoY", "YoY", "YoY"),
    factors_loading = matrix(data = TRUE, nrow = ncol(data0_ts), ncol = 2),
    var_init = "Unconditional"
)
```

See `?create_model` to get the documentation of the argument.

Estimate the model

Then you can estimate your model with your initial data.

Parameters can be estimated using different algorithms:

- The function `estimate_pca()` estimates the model parameters using only **Principal Component Analysis (PCA)**. Although this is fast, this approach is not recommended, especially if some series are *quarterly series* or series associated to year-on-year growth rates
- The function `estimate_em()` estimates the model parameters using the **EM** algorithm;
- The function `estimate_ml()` estimates the model parameters by **Maximum Likelihood**.

```
dfm_estimated <- estimate_ml(dfm_model, data0_ts)
# dfm_estimated <- estimate_em(dfm_model, data0_ts)
# dfm_estimated <- estimate_pca(dfm_model, data0_ts)
```

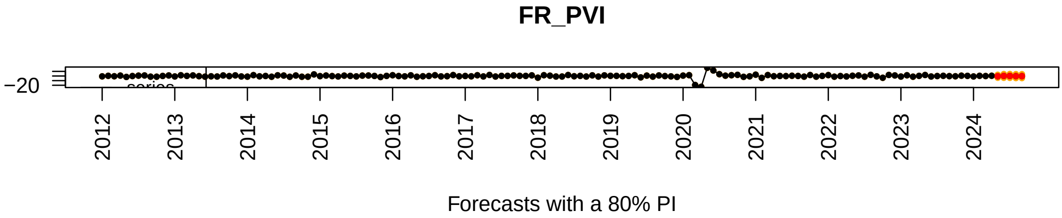
Get and analyse our results

Finally, you can get your results with the functions `get_results()` and `get_forecasts()`.

```
dfm_results ← get_results(dfm_estimated)
dfm_forcast ← get_forecasts(dfm_estimated, n_fcst = 3)
```

Plot the results results

```
plot(dfm_forcast, series_name = "FR_PVI")
```



Study of news I

If you want to compare your forecasts with the actual results, the function get_news():

```
dfm_news <- get_news(dfm_estimates = dfm_estimated,
                     new_data = data1_ts,
                     target_series = "FR_PVI", n_fcst = 3)
dfm_news$impacts
```

```
        series period expected_value observed_value   news impacts(6-2024)
1       FR_PVI 5-2024         -0.552         -2.103 -1.552           0.063
2      FR_TURN 4-2024         -0.159          1.501  1.660          -0.103
3        FR_BS 6-2024         -9.164         -7.900  1.264          -0.032
4 FR_BS_prdexp 6-2024          3.892          2.300 -1.592          -0.013
5       EA_PVI 5-2024         -0.396         -0.615 -0.220           0.010
6      EA_TURN 4-2024         -0.021          0.606  0.628          -0.085
  impacts(7-2024) impacts(8-2024)
```

Study of news II

1	0.251	0.035
2	-0.021	0.032
3	-0.004	0.008
4	-0.003	0.005
5	0.040	0.006
6	-0.018	0.025

Section 6

Conclusion and useful links

rjdverse family of packages

Versatile toolbox as multiple algorithms and tools for

- Seasonal Adjustment, including High-Frequency data
- Building filters
- Revision Analysis
- Nowcasting

And also (not covered today..):

- Trend and cycle estimation
- Benchmarking and temporal disaggregation

Useful Links

To get the Software:

- R Packages giving access to JDemetra+: https://github.com/rjdverse
- Graphical User Interface: https://github.com/jdemetra

Documentation and news:

- Online documentation: https://jdemetra-new-documentation.netlify.app/
- Blog: https://jdemetra-universe-blog.netlify.app/
- YouTube channel (Tutorials, Webinars): https://www.youtube.com/@TSwithJDemetraandR

After the tutorial

Assistance with JDemetra+ use and SA production process set up can be provided

If you have any questions, just email us

- anna.smyk@insee.fr
- tanguy.barthelemy@insee.fr

THANK YOU FOR YOUR ATTENTION