4th Seasonal Adjustment Practitioners Workshop

Trend-cycle extraction and moving average manipulations in R with the rjdfilters package

Alain Quartier-la-Tente

June 8th - June 9th

INSEE, LEMNA (French Statistician)

Contents

Introduction

Moving average are ubiquitous in trend-cycle extraction and seasonal adjustment (e.g. : X-13ARIMA) :

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

Introduction

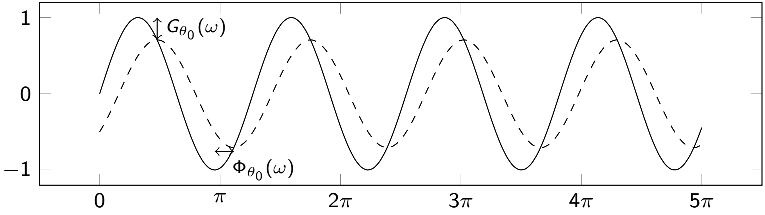
Moving average are ubiquitous in trend-cycle extraction and seasonal adjustment (e.g. : X-13ARIMA) :

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

Applying M_θ to $X_t = \mathrm{e}^{-i\omega t}$ will have two effects:

$$M_\theta X_t = \sum_{k=-p}^{+f} \theta_k \, \mathrm{e}^{-i\omega(t+k)} = \left(\sum_{k=-p}^{+f} \theta_k \, \mathrm{e}^{-i\omega k}\right) \cdot X_t = G_\theta(\omega) \, \mathrm{e}^{-i\Phi_\theta(\omega)} \, X_t$$

1. Multiply the level by $G_\theta\left(\omega\right)$ (*gain*)
2. *phase-shift* $\Phi_\theta(\omega)/\omega$ which directly affects the detection of turning points



Local polynomial filters

Hypothesis : $y_t = \mu_t + \varepsilon_t$ with $\varepsilon_t \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$

μ_t locally approximated by a polynomial of degree d:

$$\forall j \in [\![-h, h]\!] : y_{t+j} = m_{t+j} + \varepsilon_{t+j}, \quad m_{t+j} = \sum_{i=0}^{d} \beta_i j^i$$

Local polynomial filters

Hypothesis : $y_t = \mu_t + \varepsilon_t$ with $\varepsilon_t \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$

μ_t locally approximated by a polynomial of degree d:

$$\forall j \in [\![-h, h]\!] : y_{t+j} = m_{t+j} + \varepsilon_{t+j}, \quad m_{t+j} = \sum_{i=0}^{d} \beta_i j^i$$

Estimation with WLS (K=weights=*kernels*): $\hat{\beta} = (X'KX)^1 X'Ky$ and

$\hat{m}_t = \hat{\beta}_0 = w'y = \sum_{j=-h}^{h} w_j y_{t-j}$ ➔ equivalent to a symmetric moving average

➔ Arithmetic mean with $K = 1$ and $d = 0$ or 1.

➔ Henderson filter with $d = 2$ or 3 and specific kernel

What already exists in R? (1)

In terms of smoothing, in R you can:

1. `stats::filter(., method= "recursive", sides = 2)`: symmetric MA (p even)

$$y_t = f_1 x_{t+\lceil (p-1)/2 \rceil} + \cdots + f_p x_{t+\lceil (p-1)/2 \rceil - (p-1)}$$

or `stats::filter(., method= "recursive", sides = 1)`: real-time asymmetric MA

$$y_t = f_1 x_t + \cdots + f_p x_{t-(p-1)}$$

What already exists in R? (1)

In terms of smoothing, in R you can:

1. `stats::filter(., method= "recursive", sides = 2)`: symmetric MA (p even)

$$y_t = f_1 x_{t+\lceil (p-1)/2 \rceil} + \cdots + f_p x_{t+\lceil (p-1)/2 \rceil - (p-1)}$$

   or `stats::filter(., method= "recursive", sides = 1)`: real-time asymmetric MA

$$y_t = f_1 x_t + \cdots + f_p x_{t-(p-1)}$$

➔ possible to add 0 to create general asymmetric filters but endpoints won't be well treated.

What already exists in R? (2)

Local polynomial models

4. `KernSmooth::locpoly()` local polynomial with gaussian kernel
5. `locfit::locfit()` local polynomial with tricube, rectangular, triweight, triangular, epanechnikov, bisquare, gaussian
6. `stats::loess()` tricube kernel

What already exists in R? (2)

Local polynomial models

4. `KernSmooth::locpoly()` local polynomial with gaussian kernel
5. `locfit::locfit()` local polynomial with tricube, rectangular, triweight, triangular, epanechnikov, bisquare, gaussian
6. `stats::loess()` tricube kernel

Seasonal adjustment: `seasonal`, `RJDemetra`, `x12`: run X-13 but tricky to isolate one MA

What already exists in R? (2)

Local polynomial models

4. `KernSmooth::locpoly()` local polynomial with gaussian kernel
5. `locfit::locfit()` local polynomial with tricube, rectangular, triweight, triangular, epanechnikov, bisquare, gaussian
6. `stats::loess()` tricube kernel

Seasonal adjustment: `seasonal`, `RJDemetra`, `x12`: run X-13 but tricky to isolate one MA

➔ No way to easily manipulate asymmetric moving averages, analyse their properties (gain, phase-shift)

➔ No way to create SA MA: Henderson, Musgrave, Macurves, etc.

rjdfilters (1)

rjdfilters: R package based on the Java libraries of JDemetra+ 3.0

Allows to:

- easily create/combine/apply moving averages moving_average()
- study the properties of the MA: plot coefficients (plot_coef()), gain (plot_gain()), phase-shift (plot_phase()) and different statics (diagnostic_matrix())

rjdfilters (1)

rjdfilters: R package based on the Java libraries of JDemetra+ 3.0

Allows to:

- easily create/combine/apply moving averages moving_average()
- study the properties of the MA: 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)

rjdfilters: R package based on the Java libraries of JDemetra+ 3.0

Allows to:

- easily create/combine/apply moving averages moving_average()
- study the properties of the MA: 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

rjdfilters (2)

Available at palatej/rjdfilters

Development version AQLT/rjdfilters

```
# rjdfilters depends on rjd3toolkit
remotes::install_github("palatej/rjd3toolkit")
remotes::install_github("AQLT/rjdfilters")
```

Contents

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
```

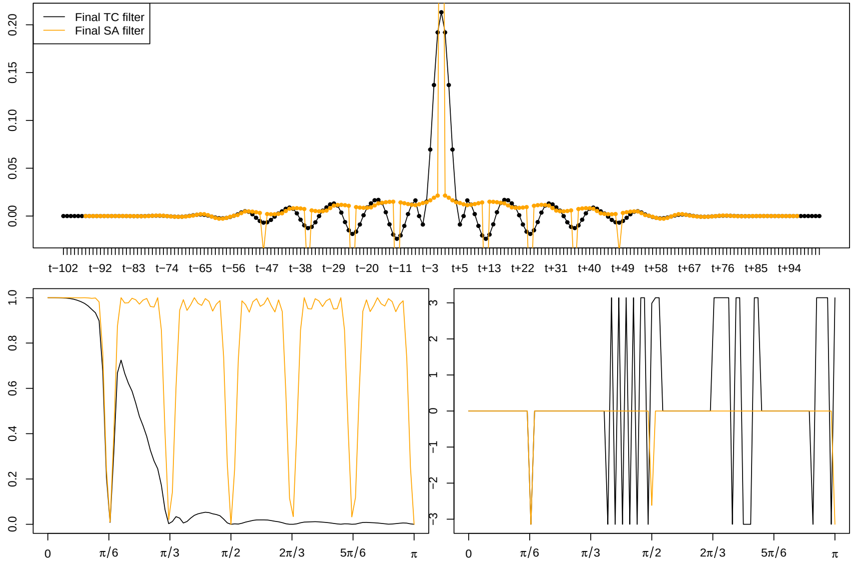
Create moving average `moving_average()` (4)

```
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)

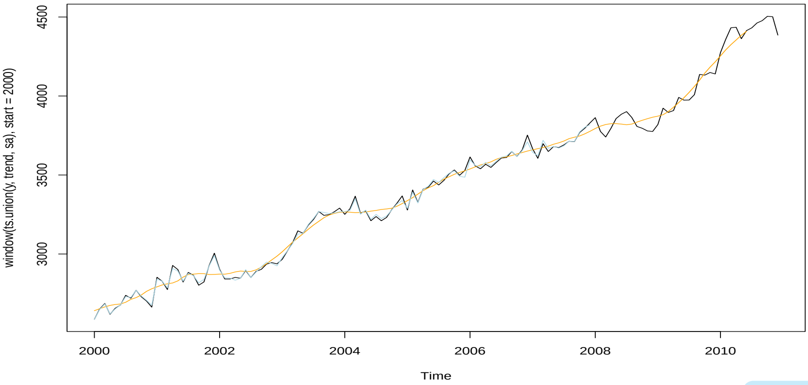
```
par(mai = c(0.3, 0.3, 0.2, 0))
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))

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"))
```



Contents

Trend-cycle extraction in X-13ARIMA

Idea closed to the following:

1. Series extended by an ARIMA model
2. Trend-cycle extraction with the Henderson filter:
   a. Selection of bandwidth with the I-C ratio
   b. Musgrave filter for the last points of the extended series

Trend-cycle extraction in X-13ARIMA

Idea closed to the following:

1. Series extended by an ARIMA model
2. Trend-cycle extraction with the Henderson filter:
   a. Selection of bandwidth with the I-C ratio
   b. Musgrave filter for the last points of the extended series

➔ Equivalent to the use of asymmetric MA with coefficients optimized to minimize the one-step-ahead forecast error

➔ Different technics could be used

Local polynomial filters

Hypothesis : $y_t = \mu_t + \varepsilon_t$ with $\varepsilon_t \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$

μ_t locally approximated by a polynomial of degree d:

$$\forall j \in \llbracket -h, h \rrbracket : y_{t+j} = m_{t+j} + \varepsilon_{t+j}, \quad m_{t+j} = \sum_{i=0}^{d} \beta_i j^i$$

Estimation with WLS (K=weights=*kernels*): $\hat{\beta} = (X'KX)^1 X'Ky$ and

$\hat{m}_t = \hat{\beta}_0 = w'y = \sum_{j=-h}^{h} w_j y_{t-j}$ ➔ equivalent to a symmetric moving average

- ➔ Arithmetic mean with $K = 1$ and $p = 0/1$.
- ➔ Henderson filter with $d = 3$ and specific kernel

Asymmetric filters: R `rjdfilters::lp_filter()`

Several solutions:

1. Same method with less data (DAF) $\iff$ Minimize revisions under same polynomial constraints (reproduce cubic trend)

➔ **no bias** but **lots of variance**

Asymmetric filters: R `rjdfilters::lp_filter()`

Several solutions:

1. Same method with less data (DAF) $\iff$ Minimize revisions under same polynomial constraints (reproduce cubic trend)

➔ **no bias** but **lots of variance**

2. Minimization of revisions filter under polynomial constraints:

   2.1 *Linear-Constant* (LC): y_t linear and v reproduce constant trends (*Musgrave* filter)

   2.2 *Quadratic-Linear* (QL): y_t quadratic and v reproduce linear trends

   2.3 *Cubic-Quadratic* (CQ): y_t cubic and v reproduce quadratic trends

   ➔ Asymmetric filters v linked to "IC-Ratio" `rjdfilters::ic_ratio()`

Asymmetric filters: R `rjdfilters::lp_filter()`

Several solutions:

1. Same method with less data (DAF) $\iff$ Minimize revisions under same polynomial constraints (reproduce cubic trend)

➔ **no bias** but **lots of variance**

2. Minimization of revisions filter under polynomial constraints:

   2.1 *Linear-Constant* (LC): y_t linear and v reproduce constant trends (*Musgrave* filter)

   2.2 *Quadratic-Linear* (QL): y_t quadratic and v reproduce linear trends

   2.3 *Cubic-Quadratic* (CQ): y_t cubic and v reproduce quadratic trends

   ➔ Asymmetric filters v linked to "IC-Ratio" `rjdfilters::ic_ratio()`

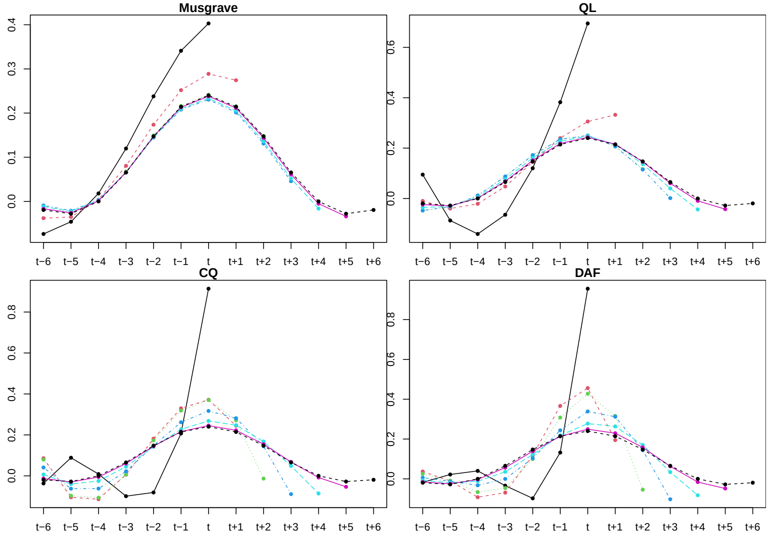
🙂 simple models with easy interpretation

🙁 Timeliness not controlled ➔ method extended in `rjdfilters::lp_filter()`

Example (1)

```r
par(mai = c(0.3, 0.3, 0.2, 0))
layout(matrix(c(1,2,3,4), 2, byrow = TRUE))

lp_filter(endpoints = "LC") |>
  plot_coef(q = 0:6, main = "Musgrave", zeroAsNa = TRUE)
lp_filter(endpoints = "QL") |>
  plot_coef(q = 0:6, main = "QL", zeroAsNa = TRUE)
lp_filter(endpoints = "CQ") |>
  plot_coef(q = 0:6, main = "CQ", zeroAsNa = TRUE)
lp_filter(endpoints = "DAF") |>
  plot_coef(q = 0:6, main = "DAF", zeroAsNa = TRUE)
```

Example (2)


Musgrave
QL
CQ
DAF
0.0
0.1
0.2
0.3
0.4
0.6
0.8
t−6
t−5
t−4
t−3
t−2
t−1
t
t+1
t+2
t+3
t+4
t+5
t+6

RKHS filters: R `rjdfilters::rkhs_filter()`

- RKHS theory used to approximate Henderson filter
- With K_p the **kernel function**, the symmetric filter:

$$\forall j \in [\![-h, h]\!] : w_j = \frac{K_p(j/b)}{\sum_{i=-h}^{h} K_p(i/b)}$$

RKHS filters: R `rjdfilters::rkhs_filter()`

- RKHS theory used to approximate Henderson filter
- With K_p the **kernel function**, the symmetric filter:

$$\forall j \in [\![-h, h]\!] : w_j = \frac{K_p(j/b)}{\sum_{i=-h}^{h} K_p(i/b)}$$

- For asymmetric filters:

$$\forall j \in [\![-h, q]\!] : w_{a,j} = \frac{K_p(j/b)}{\sum_{i=-h}^{q} K_p(i/b)}$$

RKHS filters: R `rjdfilters::rkhs_filter()`

- RKHS theory used to approximate Henderson filter
- With K_p the **kernel function**, the symmetric filter:

$$\forall j \in \llbracket -h, h \rrbracket : w_j = \frac{K_p(j/b)}{\sum_{i=-h}^{h} K_p(i/b)}$$

➔ with $b = h + 1$ and a specific K_p you have the Henderson filter

- For asymmetric filters:

$$\forall j \in \llbracket -h, q \rrbracket : w_{a,j} = \frac{K_p(j/b)}{\sum_{i=-h}^{q} K_p(i/b)}$$

RKHS filters: R `rjdfilters::rkhs_filter()`

- RKHS theory used to approximate Henderson filter
- With K_p the **kernel function**, the symmetric filter:

$$\forall j \in [\![-h, h]\!] : w_j = \frac{K_p(j/b)}{\sum_{i=-h}^{h} K_p(i/b)}$$

➔ with $b = h + 1$ and a specific K_p you have the Henderson filter

- For asymmetric filters:

$$\forall j \in [\![-h, q]\!] : w_{a,j} = \frac{K_p(j/b)}{\sum_{i=-h}^{q} K_p(i/b)}$$

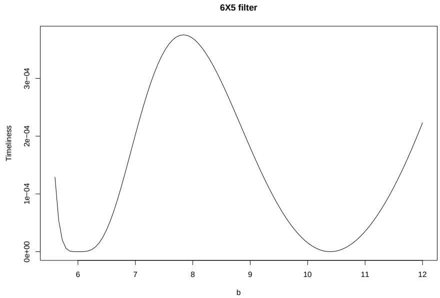
➔ b chosen by optimization, e.g. minimizing revisions linked to phase-shift:

$$b_{q,\varphi} = \min_{b_q} \int_0^{2\pi/12} \rho_s(\lambda)\rho_\theta(\lambda) \sin^2\left(\frac{\varphi_\theta(\omega)}{2}\right) \mathrm{d}\omega$$

Asymmetric filters

☹ several local extremum

```
fun <- rkhs_optimization_fun(horizon = 6,
          leads = 5, degree = 3,
          asymmetricCriterion = "Timeliness")
plot(fun, 5.6, 12, xlab = "b",
     ylab = "Timeliness", main = "6X5 filter")
```




```
rkhs_optimal_bw()
```

```
##    q=0    q=1    q=2    q=3    q=4    q=5
## 6.0000 6.0000 6.3875 8.1500 9.3500 6.0000
```

☺ Generalizable to create filters that could be applied to irregular frequency series

FST approach: R `rjdfilters::fst_filter()`

Minimization of a weighted sum of 3 criteria under polynomial constraints:

$$\begin{cases} \min_{\theta} & J(\theta) = \alpha F_g(\theta) + \beta S_g(\theta) + \gamma T_g(\theta) \\ s.c. & C\theta = a \end{cases}$$

$$\begin{cases} F_g(\theta) = \sum_{k=-p}^{+f} \theta_k^2 \text{ Fidelity (variance reduction ratio, Bongard)} \\ S_g(\theta) = \sum_j (\nabla^d \theta_j)^2 \; d = 3 \text{ Smoothness (Henderson criterion)} \\ T_g(\theta) = \int_0^{\omega_2} \rho_\theta(\omega)^2 \sin(\varphi_\theta(\omega))^2 \,\mathrm{d}\omega \text{ Timeliness (phase-shift)} \end{cases}$$

FST approach: rjdfilters::fst_filter()

Minimization of a weighted sum of 3 criteria under polynomial constraints:

$$\begin{cases} \min_{\theta} & J(\theta) = \alpha F_g(\theta) + \beta S_g(\theta) + \gamma T_g(\theta) \\ s.c. & C\theta = a \end{cases}$$

$$\begin{cases} F_g(\theta) = \sum_{k=-p}^{+f} \theta_k^2 \text{ Fidelity (variance reduction ratio, Bongard)} \\ S_g(\theta) = \sum_j (\nabla^d \theta_j)^2 \; d = 3 \text{ Smoothness (Henderson criterion)} \\ T_g(\theta) = \int_0^{\omega_2} \rho_\theta(\omega)^2 \sin(\varphi_\theta(\omega))^2 \,\mathrm{d}\omega \text{ Timeliness (phase-shift)} \end{cases}$$

- 🙂 Unique solution
- 🙂 Asymmetric filters independent of data and symmetric filter
- ☹ Non-normalized weights

How to apply a filter (1)

`rjdfilters::jfilter()` to apply a filter with asymmetric MA

```
y <- retailsa$AllOtherGenMerchandiseStores
sc <- henderson(y, length = 13, musgrave = FALSE)
icr <- ic_ratio(y, sc)
icr
```

```
## [1] 2.414569
```

```
daf <- lp_filter(horizon = 6, ic = icr, endpoints = "DAF")$filters.coef
round(daf, 3)
```

```
##         q=0    q=1    q=2    q=3    q=4    q=5    q=6
## t-6 -0.017  0.037  0.025  0.006 -0.008 -0.016 -0.019
## t-5  0.022 -0.011 -0.009 -0.013 -0.020 -0.025 -0.028
## t-4  0.040 -0.092 -0.066 -0.032 -0.012 -0.003  0.000
## t-3 -0.034 -0.069 -0.047  0.000  0.036  0.056  0.065
## t-2 -0.098  0.118  0.100  0.104  0.123  0.139  0.147
## t-1  0.132  0.366  0.308  0.244  0.218  0.213  0.214
## t    0.955  0.456  0.428  0.339  0.278  0.249  0.240
## t+1  0.000  0.195  0.316  0.312  0.263  0.229  0.214
## t+2  0.000  0.000 -0.054  0.144  0.170  0.158  0.147
```

How to apply a filter (2)

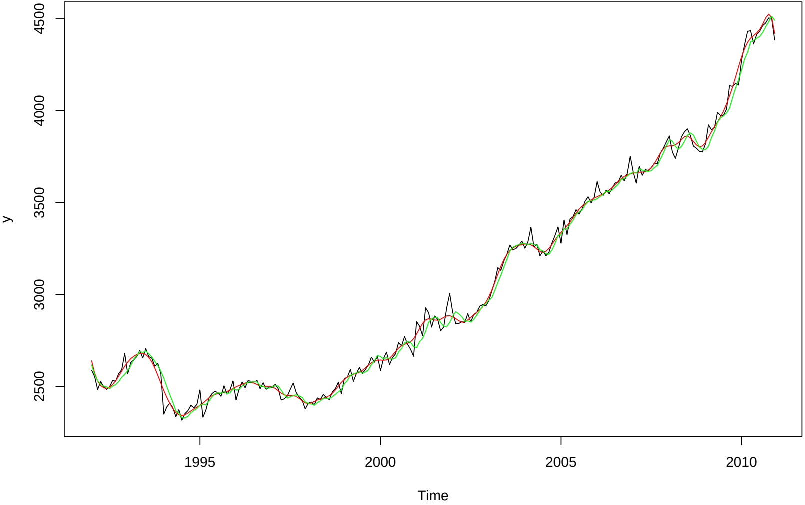
```
## t+3  0.000  0.000  0.000 -0.102  0.034  0.063  0.065
## t+4  0.000  0.000  0.000  0.000 -0.082 -0.016  0.000
## t+5  0.000  0.000  0.000  0.000  0.000 -0.048 -0.028
## t+6  0.000  0.000  0.000  0.000  0.000  0.000 -0.019
```

```
trend <- jfilter(y, daf)
```

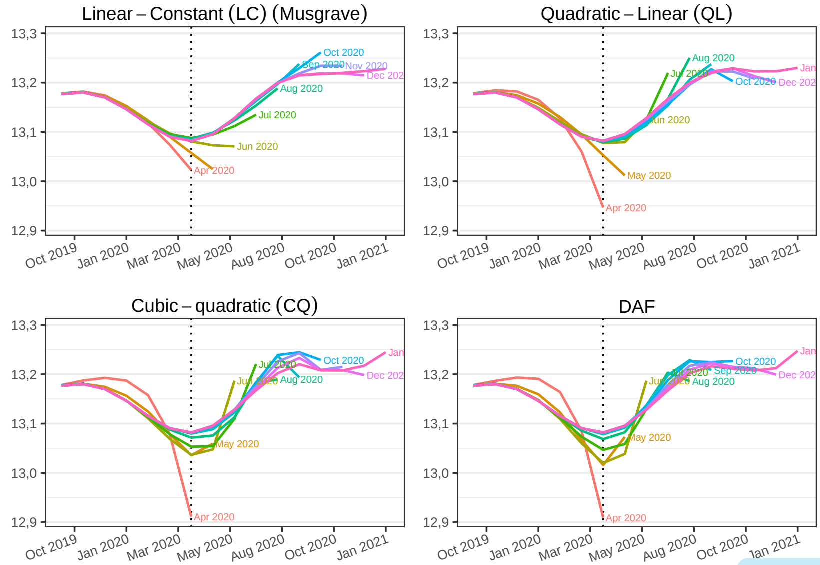
`rjdfilters::x11()` to change the filters in X-11

```
decomp_daf = x11(y, trend.coefs = daf)
daf_modif <- daf
# We change the final filter to a asymmetric filter
daf_modif[,"q=6"] <- lp_filter(endpoints = "LC")$filters.coef[,"q=0"]
decomp_daf_modif = x11(y, trend.coefs = daf_modif)

plot(y)
lines(decomp_daf$decomposition[,"t"], col = "red")
lines(decomp_daf_modif$decomposition[,"t"], col = "green")
```

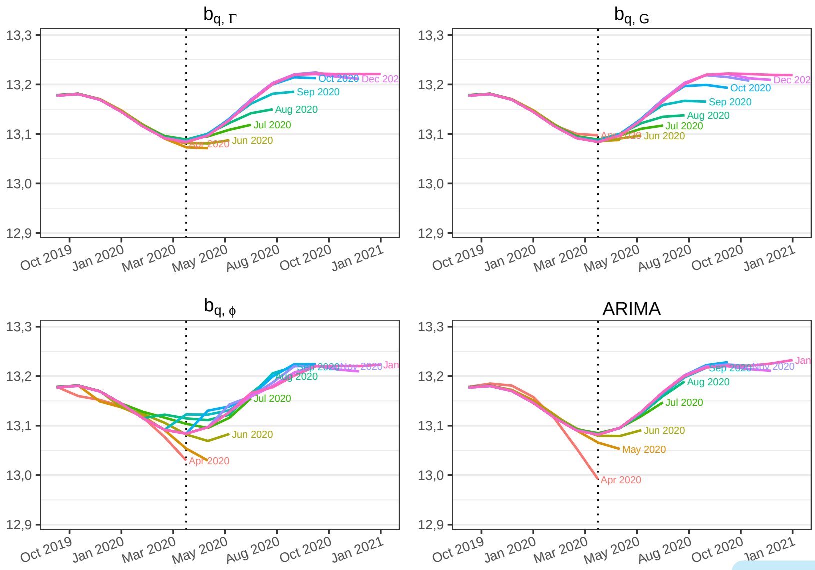
How to apply a filter (3)



Successive trend-cycle estimates (1)



Successive trend-cycle estimates (2)



Implicit forecast: `rjdfilters::implicit_forecast()`

w^q used when q future values are known and $\forall i > q, w_i^q = 0$:

$$\forall q, \quad \underbrace{\sum_{i=-h}^{0} v_i y_i + \sum_{i=1}^{h} v_i y_i*}_{\text{smoothing by } v \text{ of the extended data}} = \underbrace{\sum_{i=-h}^{0} w_i^q y_i + \sum_{i=1}^{h} w_i^q y_i*}_{\text{smoothing by } w^q \text{ of the extended data}}$$

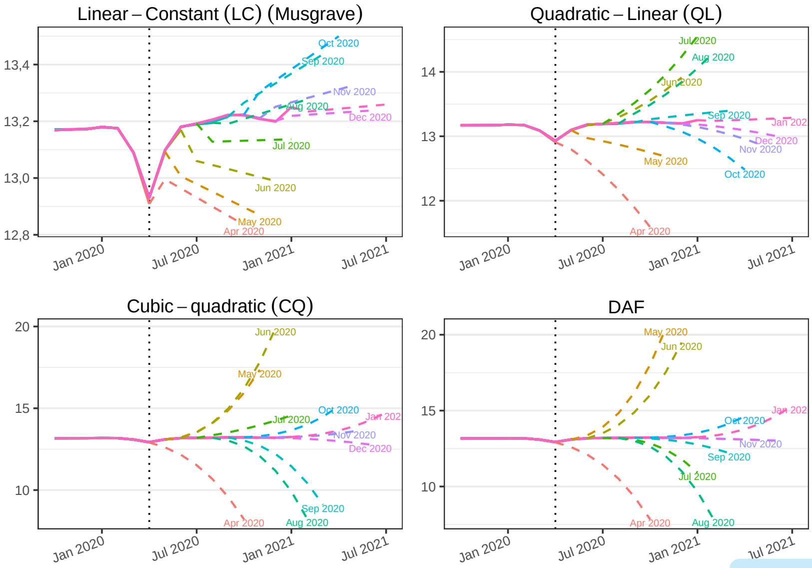
Which is equivalent to:

$$\forall q, \quad \sum_{i=1}^{h} (v_i - w_i^q) y_i^* = \sum_{i=-h}^{0} (w_i^q - v_i) y_i.$$

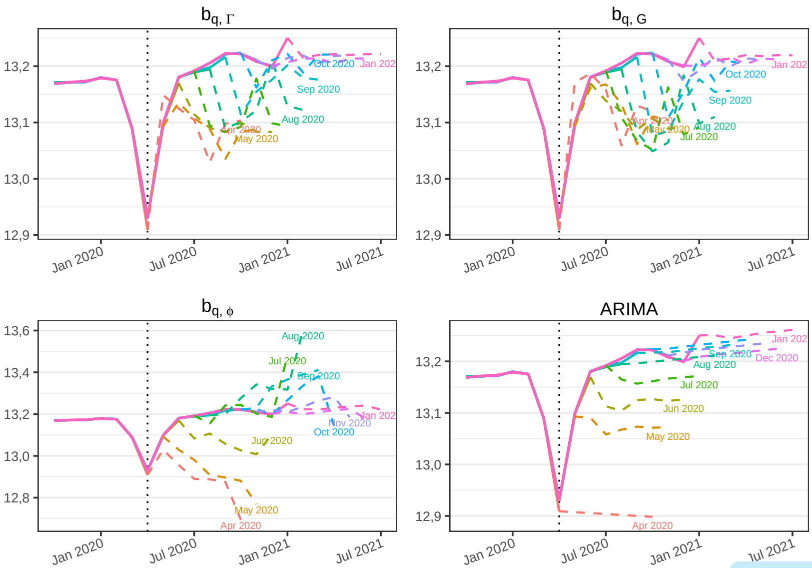
In matrix:

$$\begin{pmatrix} v_1 & v_2 & \cdots & v_h \\ v_1 - w_1^1 & v_2 & \cdots & v_h \\ \vdots & \vdots & \cdots & \vdots \\ v_1 - w_1^{h-1} & v_2 - w_2^{h-1} & \cdots & v_h \end{pmatrix} \begin{pmatrix} y_1^* \\ \vdots \\ y_h^* \end{pmatrix} = \begin{pmatrix} w_{-h}^0 - v_{-h} & w_{-(h-1)}^0 - v_{-(h-1)} & \cdots & w_0^0 - v_0 \\ w_{-h}^1 - v_{-h} & w_{-(h-1)}^1 - v_{-(h-1)} & \cdots & w_0^1 - v_0 \\ \vdots & \vdots & \cdots & \vdots \\ w_{-h}^{h-1} - v_{-h} & w_{-(h-1)}^{h-1} - v_{-(h-1)} & \cdots & w_0^{h-1} - v_0 \end{pmatrix} \begin{pmatrix} y_{-h} \\ \vdots \\ y_0 \end{pmatrix} \tag{1}$$

Implicit forecasts (1)



Implicit forecasts (2)



Contents

Conclusion

With `rjdfilters` you can already:

- create and manipulate moving averages
- used different technics for real-time trend-cycle estimates
- custom X-11 filters

Conclusion

With `rjdfilters` you can already:

- create and manipulate moving averages
- used different technics for real-time trend-cycle estimates
- custom X-11 filters

What might be implemented in the future:

- Class on finiteFilters (central filter with asymmetric filters) to easily combined them
- Bandwidth selection methods: IC ratios, CV, AIC, etc.
- More ideas?

Thank you for your attention

Package R:

palatej/rjdfilters

Development version AQLT/rjdfilters