rjd3filters

Mon objectif était d'ajouter dans `rjd3filters` de quoi manipuler facilement les moyennes mobiles mais aussi les `finite_filters` qui sont composées d'une moyenne mobile utilisée pour l'estimation finale et d'un ensemble de moyennes mobiles asymétriques utilisées pour les estimations intermédiaires.
Toutes fonctions générant des moyennes mobiles renvoient maintenant les objets précédents.

Les moyennes mobiles

La fonction `moving_average()` permet de générer une moyenne mobile : l'objet stocké est un objet R mais une convertion java est faite pour les différentes opérations

```
library(rjd3filters)
M3 <- moving_average(rep(1/3, 3), lags = -1)
# Objet R :
unclass(M3)
```

```
<S4 Type Object>
attr(,"coefficients")
      t-1         t       t+1
0.3333333 0.3333333 0.3333333
attr(,"lower_bound")
[1] -1
attr(,"upper_bound")
[1] 1
```

```
M3 # Affichage "Java"
```

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

Une fois les objets créés, on peut ensuite facilement les manipuler

```
M3X3 <- M3 * M3 # ou M3 ^2
coef(M3X3) # pour récupérer les coefficients
```

```
      t-2       t-1         t       t+1       t+2
0.1111111 0.2222222 0.3333333 0.2222222 0.1111111
```

```
M3X3[c(1,3)] # donne encore une moyenne mobile
```

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

```
M3[3] <- 1
M3
```

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

```
1 - M3X3
```

```
[1] " - 0.1111 B^2 - 0.2222 B + 0.6667 - 0.2222 F - 0.1111 F^2"
```

```
sum(M3X3)
```

```
[1] 1
```

```
rev(moving_average(rep(1/3, 3), lags = -2)) # pour inverser
```

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

```
is_symmetric(M3X3)
```

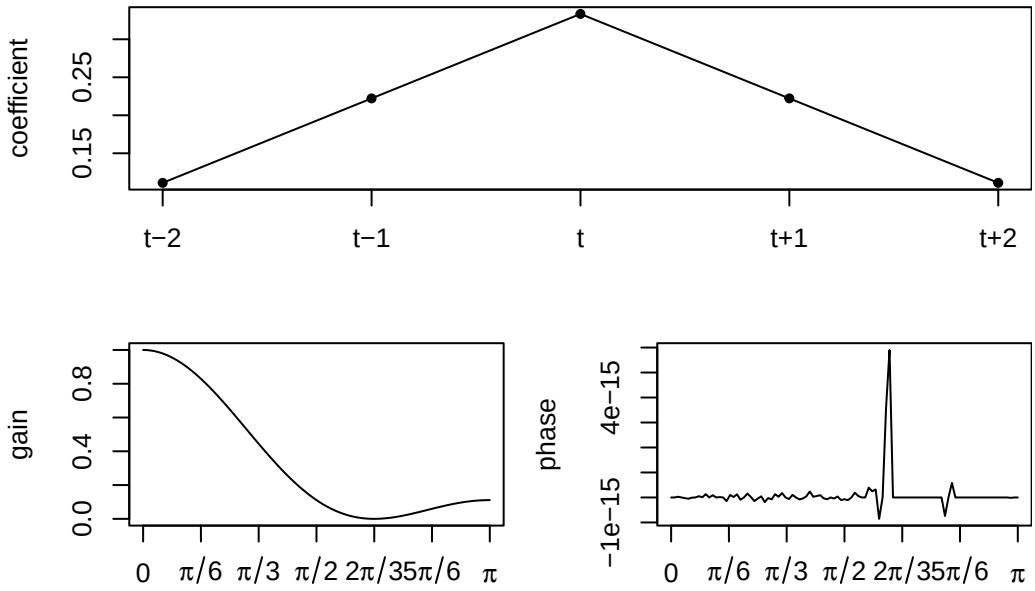
```
[1] TRUE
```

```r
# Pour avoir une MM que l'on applique à chaque période, ex: tous les trim
to_seasonal(M3X3, 4)
```

```
[1] "0.1111 B^8 + 0.2222 B^4 + 0.3333 + 0.2222 F^4 + 0.1111 F^8"
```

Il y a aussi quelques fonctions graphiques :

```r
def.par <- par(no.readonly = TRUE)
par(mai = c(0.5, 0.8, 0.3, 0))
layout(matrix(c(1, 1, 2, 3), nrow = 2,byrow = TRUE))
plot_coef(M3X3)
plot_gain(M3X3)
plot_phase(M3X3)
```



Et on peut directement récupérer les fonctions de transfert/gain/déphasage :

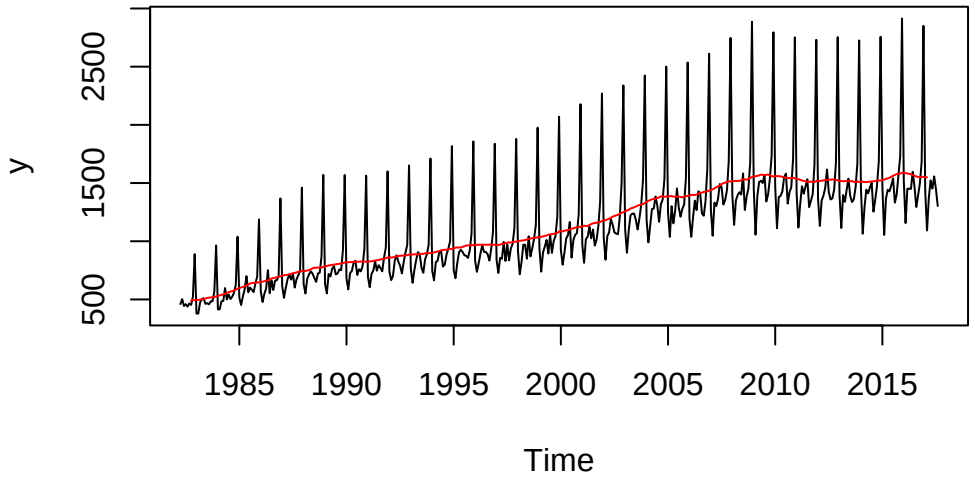
```r
gain <- get_properties_function(M3X3, "Symmetric Gain")
phase <- get_properties_function(M3X3, "Symmetric Phase")
frf <- get_properties_function(M3X3, "Symmetric transfer")
frf(c(0, pi/12, pi/4))
```

```
[1] 1.0000000+0i 0.9550838-0i 0.6476030-0i
```

À noter que pour le déphasage, c'est l'argument de la fonction de transfert (cf code Java) et non l'inverse de l'argument.

Pour appliquer les moyennes mobiles à une série il suffit d'utiliser la fonction `*` ou la fonction `filter` :

```
y <- rjd3toolkit::ABS$X0.2.09.10.M
e1 <- simple_ma(12, - 6)
e2 <- simple_ma(12, - 5)
M2X12 <- (e1 + e2)/2
plot(y)
lines(y * M2X12, col = "red")
```



finite_filters

```
h13 <- lp_filter()
# Ressemble à une matrice mais c'est en fait
# un ensemble de moyennes mobiles
h13
```

```
            q=6          q=5          q=4          q=3          q=2
t-6 -0.01934985 -0.016609040 -0.011623676 -0.009152423 -0.016139228
t-5 -0.02786378 -0.025914479 -0.022541271 -0.020981640 -0.024948087
t-4  0.00000000  0.001157790  0.002918842  0.003566851  0.002620762
t-3  0.06549178  0.065858066  0.066006963  0.065743350  0.067817618
t-2  0.14735651  0.146931288  0.145468029  0.144292794  0.149387420
```

```
t-1  0.21433675  0.213120014  0.210044599  0.207957742  0.216072726
t    0.24005716  0.238048915  0.233361346  0.230362866  0.241498208
t+1  0.21433675  0.211536998  0.205237273  0.201327171  0.215482871
t+2  0.14735651  0.143765257  0.135853376  0.131031652  0.148207710
t+3  0.06549178  0.061109020  0.051584983  0.045851637  0.000000000
t+4  0.00000000 -0.005174272 -0.016310464  0.000000000  0.000000000
t+5 -0.02786378 -0.033829557  0.000000000  0.000000000  0.000000000
t+6 -0.01934985  0.000000000  0.000000000  0.000000000  0.000000000
             q=1          q=0
t-6 -0.037925830 -0.07371504
t-5 -0.035216813 -0.04601336
t-4  0.003869912  0.01806602
t-3  0.080584644  0.11977342
t-2  0.173672322  0.23785375
t-1  0.251875504  0.34104960
t    0.288818862  0.40298562
t+1  0.274321400  0.00000000
t+2  0.000000000  0.00000000
t+3  0.000000000  0.00000000
t+4  0.000000000  0.00000000
t+5  0.000000000  0.00000000
t+6  0.000000000  0.00000000
```

```
h13[,"q=6"] # récupère la moyenne mobile d'Henderson
```

```
[1] " - 0.0193 B^6 - 0.0279 B^5 + 0.0655 B^3 + 0.1474 B^2 + 0.2143 B + 0.2401 + 0.2143 F + 0
```

```
as.matrix(h13)[,"q=6"] # ici on a vraiment la matrice
```

```
       t-6         t-5         t-4         t-3         t-2         t-1
-0.01934985 -0.02786378  0.00000000  0.06549178  0.14735651  0.21433675
         t         t+1         t+2         t+3         t+4         t+5
 0.24005716  0.21433675  0.14735651  0.06549178  0.00000000 -0.02786378
       t+6
-0.01934985
```

Ces objets peuvent également être combinés entre eux, ce qui permet notamment de reproduire tout X-11 :

```
x11_step <- rjd3filters::x11(y, trend.coefs = lp_filter(horizon = 6, ic = 3.5),
                             extreme.lsig = 300, extreme.usig = 400, mul = FALSE,
                             seas.s0 = "S3X3",
                             seas.s1 = "S3X5",
                             userdefined = sprintf("b%i", 1:11))
compare <- function(x, id, ...){
  res = cbind(na.omit(x), x11_step$user_defined[[id]])
  all.equal(res[,1], res[,2], ...)
}
compare(y, "b1")
```

```
[1] TRUE
```

```
e1 <- simple_ma(12, - 6)
e2 <- simple_ma(12, - 5)
# used to have the 1rst estimate of the trend
tc_1 <- M2X12 <- (e1 + e2)/2
# M2X12 * y # b2
compare(M2X12 * y , "b2")
```

```
[1] TRUE
```

```
si_1 <- 1 - tc_1
# si_1 * y # b3
compare(si_1 * y , "b3")
```

```
[1] TRUE
```

```
M3X3 <- macurves("S3x3")
M3X3_s <- to_seasonal(M3X3, 12)
s_1 <- M3X3_s * si_1
s_1_norm <- M2X12 * s_1
# Il faut une fonction pour gérer les derniers points :
# Ici les 6 premiers points non connus sont imputés en utilisé les précédentes valeurs
s_1_norm <- impute_last_obs(s_1_norm, n = 6, nperiod = 1)
s_1_demean <- s_1 - s_1_norm
s_1_f <- impute_last_obs(s_1_demean, n = 6, nperiod = 12)
```

```r
# s_1_f * y # b5
compare(s_1_f * y , "b5")
```

```
[1] TRUE
```

```r
sa_1 <- 1- s_1_f
# sa_1 * y # b6
compare(sa_1 * y , "b6")
```

```
[1] TRUE
```

```r
# Rmq : ic on devrait plutôt utiliser H23
select_trend_filter(sa_1 * y)
```

```
      icr    length
 7.634321 23.000000
```

```r
h13 <- lp_filter(horizon = 6, ic = 3.5)
tc_2 <- h13 * sa_1
# tc_2 * y # b7
compare(tc_2 * y , "b7")
```

```
[1] TRUE
```

```r
si_2 <- 1 - tc_2
# si_2 * y # b8
compare(si_2 * y , "b8")
```

```
[1] TRUE
```

```r
M3X5 <- macurves("S3x5")
M3X5_s <- to_seasonal(M3X5, 12)
s_2 <- M3X5_s * si_2
```

```
s_2_norm <- M2X12 * s_2
s_2_norm <- impute_last_obs(s_2_norm, n = 6, nperiod = 1)
s_2_demean <- s_2 - s_2_norm
s_2_f <- impute_last_obs(s_2_demean, n = 6, nperiod = 12)
# s_2_f * y # b10
compare(s_2_f * y , "b10")
```

```
[1] TRUE
```

```
sa_2 <- 1 - s_2_f
# sa_2 * y # b11
compare(sa_2 * y , "b11")
```

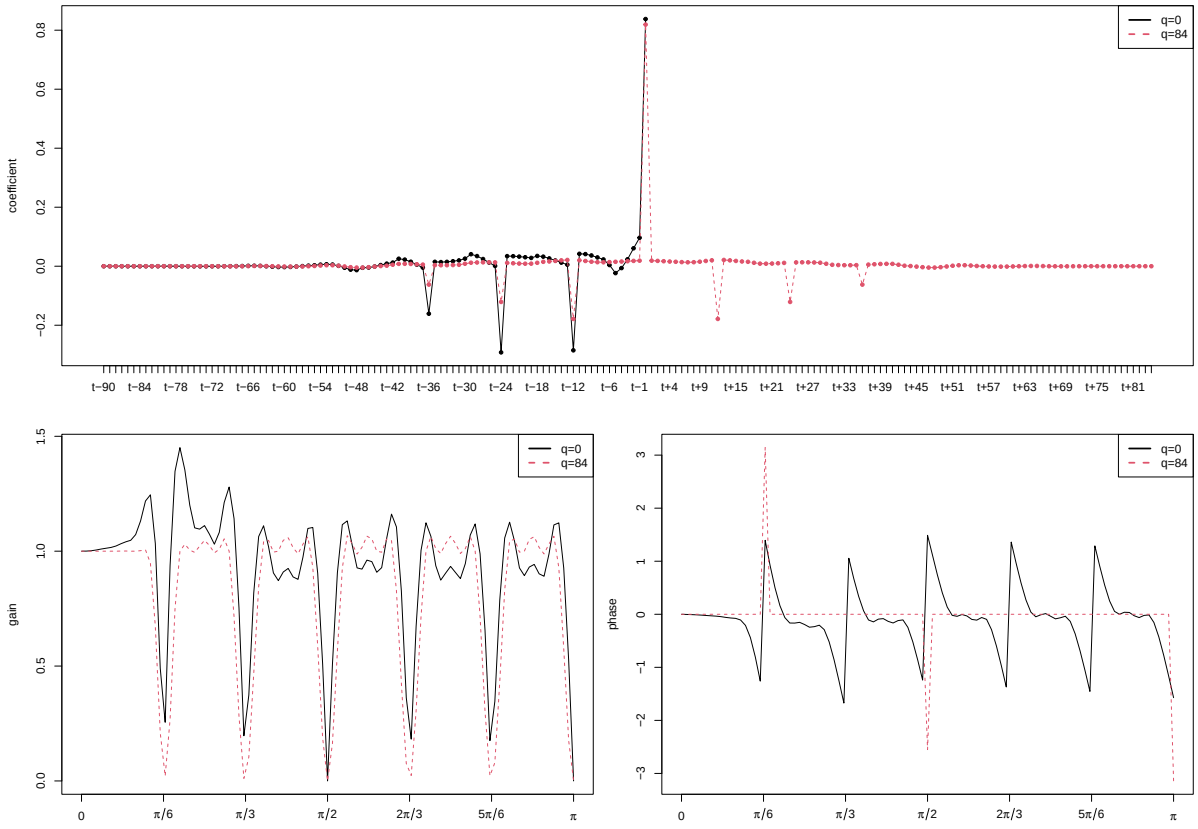
```
[1] TRUE
```

On maintenant également étudier les différentes fonctions (gain, déphasage, etc.) :

```
sa_2@sfilter@upper_bound
```

```
[1] 84
```

```
# On trace les graphiques pour q= 0 (filtre en temps-réel) et q=84 (filtre symétrique)
par(mai = c(0.5, 0.8, 0.3, 0))
layout(matrix(c(1, 1, 2, 3), nrow = 2,byrow = TRUE))
plot_coef(sa_2, q = c(0, 84))
plot_gain(sa_2, q = c(0, 84))
plot_phase(sa_2, q = c(0, 84))
```


coefficient
q=0
q=84
t−90 t−84 t−78 t−72 t−66 t−60 t−54 t−48 t−42 t−36 t−30 t−24 t−18 t−12 t−6 t−1 t+4 t+9 t+15 t+21 t+27 t+33 t+39 t+45 t+51 t+57 t+63 t+69 t+75 t+81
gain
phase
0 π/6 π/3 π/2 2π/3 5π/6 π