Case Study on the Direct Filter Approach

Marcel Dettling and Marc Wildi

March 14, 2007

1 Purpose of the signalextraction-Package

In general, a time series may be viewed as an aggregate of various parts: for example trend plus seasonal plus noise component. Since users are often interested in particular components only (for example the trend or the seasonally adjusted time series), filters are used to 'remove' the undesirable ones.

In practice, signal extraction is based on finite samples $X_1, ..., X_T$ and, very often, *current* estimates of the interesting components ($t = T$) import: a so-called 'concurrent' or 'real-time' estimate of the trend or of its turning-points has a strong prospective content, since the future evolution of the time series is likely to be conditioned by this component. Whereas forecasting tools generally emphasize the one-step ahead perspective (at least this is true for model-based approaches such as ARIMA), real-time signal extraction may be used to infer future mid-term dynamics.

The following package proposes real-time signal extraction algorithms based on Wildi [**?**]. In the following, references to the book are marked by an asterisk * in the text. The main difference between traditional model-based approaches (such as ARIMA) and the direct filter approach (dfa) is that the latter optimization criteria emphasize specifically two important estimation problems: the estimation of the *current level* (of the interesting signal) and the *early detection of turning-points* (of a trend component). Since both criteria are incongruent, different algorithms are available which match user preferences (level or turning-points) and time series characteristics (trending or non-trending series). Also, important filter characteristics at frequency zero can be imposed or relaxed such as the *local level restriction*[1] or the *instantaneous level restriction*[2].

This document provides a 'step-by-step' introduction to the dfa real-time signal extraction package and demonstrates its functionalities based on a leading indicator application (the so-called KOF-Economic Barometer). More generally, the text emphasizes topics that are specific to real-time signal extraction.

[1] $A(0) = 1$, where $A(\omega)$ is the amplitude function.
[2] $A(0) = 1$ and the time delay of the filter vanishes in frequency zero.

2 Data Import

The starting point to the case study in this chapter is a time series with at least 61 monthly or 23 quarterly entries. A very common case is that the data are present in a column of a spreadsheet, as displayed in Figure 1. We will thus start our tutorial from there.

Figure 1: A time series with measurement time information and informative headers stored in a spreadsheet. Row A contains the headers *time* and *series*, column A information about when the measurements were made and column B is the series itself.

How can we get such data from a spreadsheet into R? We here recommend a straightforward way: first, save the data from the spreadsheet software as a tab-delimited text file named `series.txt`. Second, read them from the text file into R and process them accordingly. Only three simple commands are necessary.

```
> input <- read.delim("series.txt")
> x     <- as.numeric(input$series)
> names(x) <- input$time
```

Please note that the above commands only work if the configuration in the spreadsheet is exactly as displayed in Figure 1. In particular, the first line of code contains the text file name/path and needs to be set/chosen accordingly. The case where there is no header row can be dealt with by setting the argument `header=FALSE` in the `read.delim()` function. On the other hand, if column A with the times of measurement is not present, the last of the three lines of code in the above chunk gets obsolete. Further information about how to read external text files into R is available from the manual page of the import function `read.delim`, which can be called by

```
> help(read.delim)
```

3 Business Survey Data

In fact, the business survey dataset `x` that we just generated is already present as an example dataset in the `signalextraction`-package. Reading it from a text file is thus not strictly necessary (and was done here for illustrative purposes only), as it can conveniently be loaded into R via

```
> data(x)
```

This series is a monthly index of business survey data with 183 measurements between January 1991 and March 2006. It is used by the Swiss Institute for Business Cycle Research to anticipate movements of the GDP growth rate. For an analysis, we employ the direct filter approach that is implemented in the `signalextraction`-package by the `dfa()`-routine. The default call is

```
> dfa <- function(x, quart = FALSE, d = 0, pb = 1/14, sb = 1/7,
                  tpfilter = FALSE, lambda = 3, expweight = 1.5,
                  pbd = if (length(x)<100) 1.08 else 1.03,
                  limamp = if (!tpfilter) 1.5 else 3, i2 = TRUE,
                  n.loops = 10, verbose = 1)
```

The first three arguments `x`, `quart` and `d` describe the time series and their properties. We are going to work with the business survey data that we read/loaded into R before. Argument `quart` describes whether the series was recorded with monthly or quarterly frequency. As evident from Figure 1, we have monthly measurements for our series and thus use the default value `quart=FALSE`. **Please note: the present version of the package has been tested with monthly data only!**

Argument `d` is related to the determination of the optimization criterion and of filter constraints, see sections 6.1* and 6.2* (ref: Wildi [**?**]). Roughly, one can say that for stationary or asymptotically bounded time series (as is the case for the business survey data `x`) `d`=0 is the right choice whereas for trending series `d`=1 would be more appropriate.

Setting `d=1` imposes a local level restriction (see footnote **??**). A formal test verifying the necessity of imposing `d=1` is proposed in chapter 6*, where it is shown, also, that the proposed procedure can be interpreted as a generalization of traditional unit root tests. In particular, `d=1` should be set for integrated processes. It should be emphasized, however, that filter constraints and the optimal selection of optimization criteria determined by `d` address issues that extend the concept of integration, see for example sections 6.6* and 6.7*.

For `d=0`, the optimization is based on criterion 3.7* (periodogram) and the amplitude function $A(\omega)$ of the one-sided filter is not constrained to satisfy $A(0) = 1$[3], see for example the bottom left panels in Figures **??** and **??**. For `d=1`, the optimization is based on criterion 6.7* (pseudo-periodogram) and the amplitude function of the one-sided filter is constrained by requiring $A(0) = 1$[4].

The remaining arguments in the call of `dfa()` are described next.

4 Signal

Any signal can be approximated by optimal one-sided ZPC-filters[5] used in the package. For our purpose, we have found the so called 'ideal' trend to be a very

[3] See section 4.5* for a thorough discussion on these topics.

[4] Altough the constraint $A(0) = 1$ could be obtained by a simple scaling of the filter *after* optimization, it is better to impose it *during* the optimization process. As a consequence, this restriction entails filter performances more fundamentally by affecting the location of poles and zeroes directly.

[5] See section 3.2* for a formal definition of the latter filter class.

convenient target-signal. The transfer function of the ideal trend is defined by

$$\Gamma(\omega) := \begin{cases} 1 & 0 \leq |\omega| \leq \pi \cdot pb \\ \dfrac{\pi \cdot sb - |\omega|}{\pi \cdot sb - \pi \cdot pb} & \pi \cdot pb \leq |\omega| \leq \pi \cdot sb \\ 0 & \pi \cdot sb \leq |\omega| \leq \pi \end{cases} \tag{1}$$

where `sb` and `pb` (`pb` < `sb`) are parameters in the above function-call: `pb` determines the width of the pass-band and `sb` determines the width of the stop-band. They can take values in $(0, 1)$, corresponding to the frequency band $(0, \pi)$. For our own applications we set `pb`< `sb`< 1/6, which ensures that seasonal components are eliminated.

In-between `pb` and `sb`the transfer function decays linearly from pass- to stop-band. Strictly speaking, this signal cannot be computed exactly because it is the output of a doubly-infinite MA-filter. Moreover, at the current boundary $t = T$ of $X_1, ..., X_T$ a one-sided filter must be used necessarily.

> The function `dfa()` computes real-time (concurrent) one-sided filters whose output approximates the current level of the signal (see section **??** below) or for detecting turning-points of the trend (see section **??** below).

By choosing smaller `pb` and `sb`, the target (trend) signal becomes smoother implying that its real-time approximation, through `dfa()`, will as well become smoother in general. The default values `pb=1/14`, `sb=1/7` in the above function-call have been determined in the context of business-cycle analysis and are, to some extent, arbitrary[6].

It is up to the user to determine which signal best fits its particular purpose: an intra-day trader would prefer short-term trends while a climatologist would be interested in long-term variations. Ultimately, these parameters map user preferences to the (statistical) optimization algorithms which is impossible if signals are determined by automatic modeling procedures such as implemented in traditional model-based approaches.

Thus, the use of alternative target signals is possible, but this additional flexibility has not been implemented in the package yet. Nevertheless, the user can substitute the 'ideal' trend by a signal of its own by a simple source code manipulation. More precisely, one has to change a vector called `ttr` in `dfa()`: one can search for the following piece of code

```
## Symmetrical transfer function
ttr       <- as.numeric(((1:(npg2+1)) <= npg2*sb))
weli      <- floor((npg2*sb+1):(npg2*pb+1))
term1     <- npg2*sb+1-((npg2*pb+1):(npg2*sb+1))
ttr[weli[1]+weli[length(weli)]-weli] <- term1/(npg2*sb-npg2*pb)
```

[6] `sb=1/7` implies that components with frequency equal or greater to $\pi/7$ are eliminated by the ideal trend. Note that the first (fundamental) seasonal frequency is $\pi/6$.

and replace the values in `ttr`[7] by (almost) any other (transfer) function.

It is worth to emphasize that the present version of the package has been developed with trend-estimation in mind. It is best suited for approximating low-pass filters. *The optimization procedure is not optimally designed for high- or band-pass applications* (although this could be implemented of course) because of particular initial value settings and fixed filter constraints.

5 Level-Approximation

In analogy to `d`, the argument `tpfilter` is a very crucial input for the signal extraction, since it preselects the optimization criterion that is employed. For the default value, `tpfilter=FALSE`, we are in the case of level-estimation, see chapters 3^*, 4^* and 6^*. This means that the `dfa()`-routine fits a best level filter, i.e. tries to track the trend signal as closely as possible, without a special focus on turning points.

By setting `tpfilter=FALSE`, two level-specific optimization criteria can be accessed :

- For `d=0` criterion 3.7^* is used (best suited for stationary or asymptotically bounded time series).
- For `d=1` criterion 6.7^* is used (pseudo-periodogram) and $A(0) = 1$ is imposed (low-pass filter).

Figure 2: Graphical output after a signal extraction using the `dfa()` routine for best level estimation (with default arguments) on the business survey data `x`. The top two panels show the original and filtered time series, the middle two periodogram input and output, and the bottom two amplitude and time shift characteristics of the one-sided filter.

Let us now run the direct filter approach with default arguments (thus they are not visible in the call) which implies best level approximation. The command is as follows:

```
> fit <- dfa(x)
```

Note that the numerical optimization algorithm may need some time to complete: progress can be viewed in the R-window. Technically, we obtain a R-object named `fit`. It is an instance of class `dfa`, for which (among others) a plot method has been defined. It is called by

```
> plot(fit)
```

[7] A plot of ttr would reveal that it corresponds to the transfer function of the ideal trend (restricted to positive frequencies only). More precisely, ttr[i] corresponds to the value of the (intended) transfer function in frequency $\omega_i := (i-1) \cdot \pi/(L-1)$ whereby L is the length of ttr: therefore $\omega_1 = 0$ and $\omega_L = \pi$.

and yields a 3x2 panel of 6 diagnostic diagrams, see Figure 2. The top left plot shows an overlay of the business survey series `x` (black) with the filter output (red). The two panels in the middle show the periodogram input and output: this is a very convenient way to check whether the filter has removed (damped) undesirable components in the original time series or not. Finally, filter characteristics such as *amplitude* and *time shift*[8] functions can be viewed in the bottom panels, see chapter 3* for an overview on filter characteristics in the frequency-domain.

6 Turning Point Detection

In the following, turning-points are defined as local extrema of the ideal trend, see section 5.1* for a more comprehensive discussion about these topics. Therefore, altering the signal definition (through `pb` and/or `sb`) affects the turning-points, too.

If turning points are of interest, then argument `tpfilter` should be set to `tpfilter=TRUE` in the function call. As a consequence, the error criterion optimized by `dfa()` is the one in equation 5.4*. We also strongly recommend to set `d=0` and `i2=TRUE` which is the standard configuration (more on these below).

The filter output in Figure **??** has been obtained by using the following function call:

```
> fit <- dfa(x, quart = FALSE, d = 0, pb = 1/6.5, sb = 1/6,
             tpfilter = TRUE, lambda = 5, expweight = 1.5,
             pbd = 1.02, limamp = 3, i2 = TRUE, n.loops = 10,
             verbose = 1)
>
> plot(fit)
```

Figure 3: Graphical output after a signal extraction using the `dfa()` routine for turning point detection. The top two panels show the original and filtered time series, the middle two periodogram input and output, and the bottom two amplitude and time delay boundary filters.

Before motivating this particular choice (parameter-setting) we note that the output series is much smoother than for the previous level estimate. Level performances are generally worse but turning-points are detected very early and the enhanced smoothness ensures that the rate of 'false alarms' is small, see chapter 5*[9]. The smoothness is due to stronger damping of undesirable high-frequency components in the stop-band and the improved speed is obtained by very small

[8] The time shift function is simply the phase divided by frequency, see 2.9*.

[9] Empirical comparisons of turning-point filters, level filters and logit-models are shown in section 5.3.2*.

time delays in the pass-band, as confirmed by the bottom panels in Figure ??[10].

> It is worth to emphasize that filter-outputs obtained by `dfa()` are not subject to revisions (as would be the case for traditional model-based approaches) because causal (one-sided) filters are used only.

7 Other Arguments to dfa()

While it is far beyond the scope of this document to display or discuss the results with all possible combinations in the multi-dimensional space of the arguments `lambda`, `expweight`, `pbd`, `pb`, `sb`, `limamp` and `i2`, we will shed light on some of the most important aspects:

- `lambda` is a Lagrange parameter for phase restriction, where larger values induce smaller time delays in the pass-band of the filter, but noisier signal estimates. It emphasizes the filter error due to the time delay in the generalized criterion 5.4*.
- `expweight` is a parameter that determines the shape of the frequency weighting function $W(\omega) = |\omega|^{expweight}$ in criterion 5.4*. Larger values emphasize damping efforts in the stop-band of the filter. For `lambda`=1 and `expweight=0` the best level filter results. Emphasizing simultaneously the time delay (in the pass-band), through `lambda`, and the amount of high-frequency (noise) damping in the stop-band, through `expweight`, generates distortions of the amplitude function in the pass-band which are associated to poorer level performances, see section 5.3*, for a comprehensive discussion on these topics. In particular, it is shown that the detection of turning points and the approximation of the level (of the signal) are to some extent incongruent criteria which require specific solutions provided in `dfa()`.
- `pbd` implements a regularity constraint which must exceed one. Roughly speaking, the shorter the time series, the larger `pbd` should be set (see section 10.5* for details). More precisely, `pbd` is the value that the moduli of poles of the ARMA filter are constrained to exceed, see 3.20*, and therefore it ensures stability of the resulting one-sided filter. The larger it is, the smoother the transfer function usually is (which does not mean that the output signal will be smoother, of course). So far, we have found that `pbd`=1.08 is a good choice if $T < 100$ whereas if $T > 100$ ony may set `pbd`=1.03, see 3.20* (deviations are possible, of course).
- In the case of turning-point filters, the amplitude in the pass-band may be subject to more or less severe distortions. The parameter `limamp` enables to limit the extent of such distortions. More precisely, the parameter constrains the amplitude $A(\omega)$ to be smaller than $limamp * A(0)$ in the pass-band of the filter: this amounts to a particular (additional) regularity

[10] A comparison with filter characteristics of the level-filter in Figure ?? illustrates that the damping effect of the turning-point filter is much stronger in the stop-band and that the time delay is barely larger in the pass-band (in particular in the vicinity of the low-frequency spectral bulk).

constraint. The parameter should be larger than one. From a practical point of view, we have found that values between 1.5 and 2 are fine in the case of level approximation (`tpfilter=FALSE`) whereas values between 3 and 5 are reasonable constraints in the case of turning-point filters.

- `i2` is a logical which determines whether the time delay of the one-sided filter vanishes at frequency zero (default-value `i2=TRUE`) or not. If `i2=TRUE` and `d=1` then an *instantaneous level restriction* is imposed, see section 6.2.1*: it is shown that this constraint can be associated to processes whose trend-slope grows unboundedly in absolute value such as, for example, I(2)-processes. However, as for `d`, the scope of `i2` transcends 'integration orders'. If `i2=TRUE`, then the constraint is implemented formally by using an additional zero-pole pair satisfying 6.12*. In practice, we have found evidence that this constraint is often useful (whether series are integrated or not), in particular if turning points are of interest. Empirical results in chapter 4* suggest that level performances are enhanced also by imposing this restriction. Note that the analyzed series there are asymptotically bounded (and thus cannot be I(2)) which illustrates that methodological issues related to the values of `i2` or `d` (i.e. the choice of corresponding optimization criteria and/or filter restrictions) encompass model assumptions such as 'integration' (see the discussion in sections 6.6* and 6.7*).

In order to illustrate the effect of the above parameters we compare filter outputs obtained by alternative settings:

- Black line in Figure **??**: the series is based on the original setting, whose outcome was already plotted in Figure **??**.

- Red line in Figure **??**: the series is based on `expweight=0.5` (instead of 1.5), everything else being equal to the original setting. The smaller weight ($|\omega|^{0.5}$ instead of $|\omega|^{1.5}$ in the original setting) attributed to the 'undesirable' high-frequency components in criterion 5.4* implies that damping properties of the filter in the stop-band aren't emphasized as strongly as in the original setting. Therefore, the resulting filter output is contaminated by high-frequency 'noise' which makes an assessment of turning-points (in real-time) a more difficult task in practice.

- Blue line in Figure **??**: the series is based on `lambda=2` (instead of `lambda=5`), everything else being equal to the original setting. The filter output is close to the black-line. However, the smaller lambda-weight attributed to the time-delay in criterion 5.4* generates a filter which is slightly delayed (a closer look at the figure reveals that the delay with respect to the black line is approximately one month).

- Green line in Figure **??**: the series is based on `lambda=2` and `pb=1/15`, `sb=1/13`. The filter output is smoother and it is delayed. Note that the signal has changed by choosing `pb=1/15`, `sb=1/13`, recall section **??**. Therefore, turning-points have changed too. Strictly speaking, the green line should not be compared to the preceding ones because the optimal signal has been altered.

Figure 4: Effect of various parameter-settings on filter outputs

Regarding the influence of the filter arguments, one should note that `lambda` and `expweight` have no effect on the output for best level estimation, i.e. if `tpfilter=FALSE`. However, `d`, `pbd`, `pb`, `sb`, `limamp` and `i2` can be set.

8 Numerical Optimization Routine

Minimizing the error criteria given in 3.7^*, 5.4^* or 6.7^* is a difficult, highly non-linear problem that cannot be solved explicitly and requires a numerical approach[11]. As always in numerics, there is a trade-off between the accuracy of the solution and the required computing time. In `dfa()`, this is governed by argument `n.loops`, which corresponds to the number of initial Stochastic Annealing proposals that are drawn, and then further optimized by a variety of numerical optimization methods. The more of them we use, the smaller the chance of being trapped in local extrema becomes, but the computing gets more time consuming. To our experience, the default value `n.loops=10` balances computing load and precision reasonably well, and yields accurate solutions with an average running time of around 2-3 minutes.

Finally, argument `verbose` controls the amount of text output that is given during the numerical optimization. Three levels are available: 0 corresponds to no output at all, the default value 1 allows for a rough guess which percentage of the computing has been done, while a value of 2 yields full text output for every optimization step and is mainly useful for debugging and exact comparisons.

Since the numerical optimization partly relies on stochastic algorithms it is possible that the same initial parameter-setting may lead to slightly different solutions, depending on which draw is used during optimization. While we did not find significant departures for level-estimation problems, turning-point filters may be sensitive to such issues because the optimization problem is 'harder' to solve.

Storing and loading `dfa`-objects is possible with

```
> save(fit,file="myfile.rda")
> load("myfile.rda")
```

where `myfile.rda` is a file-name. This enables to circumvent laborious optimizations and to access more rapidly to 'old' results.

Every suggestion for improving the speed and/or the precision of the numerical optimization algorithm would be greatly appreciated! It should be noted, however, that the problem is technically more complex (less

[11] The standard optimization routines implemented in R are used: stochastic annealing, Nelder-Mead and BFGS.

'regular') than traditional 'amplitude fitting' common in engineering applications. In particular, optimal turning-point filters are likely to approach the frontier of regularity (they are prone to become nearly singular) because the amplitude function in the pass-band is to some extent 'sacrificed' at the benefits of smaller time delays (in the pass-band) and stronger damping of high-frequency 'noise' (in the stop-band).

9 Out-of-Sample Utilization of the Filter

Behind the scenes, the direct filter approach computes a one-sided ARMA-filter. The ARMA-coefficients are also stored in object `fit`, from where they can be accessed by

```
> fit$ar.coef
 [1]  1.0000e+00 -1.7595e+00  1.6305e+00 -2.1317e-01 -1.0193e+00
 [6]  1.5434e+00 -8.9463e-01  1.3691e-01  4.0717e-01 -2.8254e-01
[11]  8.0379e-02  5.9704e-03 -8.9685e-04 -3.9952e-04 -4.1361e-05
[16] -1.7538e-06
>
>
> fit$ma.coef
 [1]  0.12004985 -0.08212789 -0.02056278  0.16565752 -0.06297009
 [6]  0.04579233  0.03375345 -0.00358081  0.02689532  0.02600401
[11]  0.02158400 -0.02785126 -0.01723218  0.04432262  0.01160852
[16] -0.05509042
```

As an alternative, there is a method for the `dfa`-class that extracts the coefficients directly. It is called by

```
> coef(fit)
      AR-coefficients MA-coefficients
 [1,]    1.000000e+00     0.120049851
 [2,]   -1.759552e+00    -0.082127897
 [3,]    1.630578e+00    -0.020562783
 [4,]   -2.13...           ...
```

Figure 5: Level filter that was estimated on the business survey data (observations 1-183, filter values are in red) and then applied to a simulated series with similar mathematical properties (observations 184-283, filter values are in blue).

and yields a matrix with two columns, where the first one contains the AR- and the second one the MA-parameters. These coefficients can now in principle be used for filtering any arbitrary time series. However, it is important to note that this is reasonable only if the original and new series originate from a common probability distribution, or in colloquial language, if they have "the same mathematical properties".

The most convenient way to apply a previous filter to a new time series is with the `outsamp`-method that is associated to the `dfa`-class. It takes an arbitrary series as input and computes the filter value for every datapoint. Or in other words, for every datapoint in the new series, a nowcast of the trend signal is computed, based on parameters that were estimated from another time series. Such functionality also allows for easy performance comparisons among different filter methods.

For an illustration of the `outsamp`-method, we here simulate a time series that can be interpreted as a sequel to the business survey data `x`. We do this by expanding an AR(13)-model that was fitted to `x`. The commands are as follows.

```
> set.seed(21)
> fit.ar <- ar(x)
> p      <- fit.ar$order
> innov  <- rnorm(100, sd=sqrt(fit.ar$var.pred))
> xx     <- c(rev(x)[1:p], rep(0,100))
> for (i in 1:100)
+   {
+     xx[i+p] <- sum(xx[i:(i+p-1)]*rev(fit.ar$ar))+innov[i]
+   }
> newx <- xx[(p+1):length(xx)]+mean(x)
```

We employ the last 13 observations of `x` as starting values for the new series and then generate 100 new datapoints by using the AR(13)-model fitted on `x` with randomly drawn white noise innovations, whose variances are equal to the error variance on the business survey data. After doing so, we propose that the distributions of `x` and the new series `newx` are similar enough to justify an out-of-sample application of the filter.

```
> blf  <- outsamp.dfa(fit, newdata=newx, sequel=TRUE)
```

The resulting series (in the object `blf`) corresponds to the blue line in Figure **??**.

Note that there is an argument `sequel` that requires some explanation: since behind the scenes, `dfa()` fits an ARMA-ZPC-filter, we would in theory need an infinitely long series for an exact computation of the filter values. If the coefficients of the MA(∞)-representation of the filter decay reasonably quickly (which they do in most cases), this issue can be neglected. However, at the beginning of the new series `newx`, we need to know from which basis the filter values have to be computed or, in other terms, we need starting values for initializing the AR-part of the filter. In our case, where `newx` is a direct sequel to `x`, we set argument `sequel=TRUE`. In this case, the past values which are required to compute the nowcast are taken from the "old" series `x`. On the other hand, if the values from the old series cannot serve as initial values, the set argument `sequel=FALSE`. The nowcasts at the beginning of `newx` will then be determined on the basis of 'artificial' starting values, see section 10.7.3*.