Multivariate Real-Time Signal Extraction

Marc Wildi and Tucker McElroy

August 14, 2020

Contents

Chapter 1

Introduction

1.1 Overview

1.1.1 Signals and Extraction

In the applications of time series analysis to macroeconomics, finance, and quality control it is essential to extract useful information about trends, turning points, and anomalies in real time. The practitioner does not have the luxury of sifting past data for structural breaks, indicators of regime change, or changes to volatility. Informative elections are contingent upon understanding the dynamics of various time series at time present. Because long-term movements, as well as aberrations, are defined in terms of the long-run behavior of a time series over past, present, and future, any analysis of the present state necessarily involves a degree of forecasting. This broad topic is referred to as real-time signal extraction.

A signal is any component of a time series that is deemed useful for a particular application. If long-term movements are of interest, the signal is a trend. If short-term fluctuations about a longer-term mean are of interest, the signal is a cycle. If shocks, due to rare terrorist events or natural disasters, are of interest, the signal consists of the extreme values. If regular patterns of an annual period, linked to cultural or meteorological patterns, are of interest, the signal is a seasonal component.

However, these signals are not directly observable at time present, because in each case their definition involves all the past and future values of a time series – but the future is unknown, and only part of the past is available to us. The statistical processes by which a signal is estimated from available data is referred to as extraction, and the residual from the signal extraction is referred to as the noise. Whereas signals can be estimated from historical, or past, sections of a time series, when effort is focused upon time present we refer to the analysis as real-time signal extraction.

Real-time signal extraction is considerably more challenging, and useful, than historical signal extraction. The difficulty lies in the uncertainty about the future, which is transmitted unto the signal extraction estimates themselves. One way to conceive of this difficulty is through the warring principles of timeliness and accuracy: should we procrastinate in providing our analysis of the present, we can increase the accuracy of signal extraction, but our answers become less

relevant, even as the present time rapidly drifts into the past. Conversely, extremely timely extractions suffer from greater future uncertainty, and are likely to exhibit inaccuracy.

There is a considerable body of literature addressing signal extraction, but this book focuses upon a particular methodology called Direct Filter Analysis (DFA). As the original development of DFA was univariate, the methodology's power was limited to the information content within a single time series. But because batches of time series can be closely linked, exhibiting correlated trends, common dynamics, or even predictive relationships, it is natural to expect that a multivariate extension of DFA to vector time series will more greatly facilitate informed decision making. The topic of this book is Multivariate Direct Filter Analysis (MDFA).

1.1.2 The Classic Model-Based Paradigm

Many signals can be formulated as weighted linear combinations of a time series, in which case the real-time signal extraction problem can be approached as a Linear Prediction Problem (LPP). In order to pose an LPP, a solution criterion is needed, and Mean Squared Error (MSE) is often used: one seeks a real-time signal extraction that has minimal MSE discrepancy with the actual target signal. Although an LPP can then be solved, the solution depends on knowing something about the dynamics in the time series process. The most venerable approach to understanding these dynamics is to posit a time series model, and fit this model to the observed data. This approach, which goes back to the work of Yule in the 1930s, is called the classic paradigm, being based upon a Model-Based Analysis (MBA).

An attractive feature of MBA is that analytical formulas for the LPP solutions can often be obtained, thereby facilitating computation. The philosophy underpinning the classic paradigm is that a Data Generation Process (DGP) exists – as a theoretical, or Platonic construct – to which the observed data closely adheres. Formally, the DGP is some stochastic process defined upon a probability space, and the observed data is a realization, or sample path, of the DGP. Statistical inference is involved with the science of identifying a model class for the DGP, narrowing down the class to a particular model (by eliminating contenders), and fitting that model via fixing values of the parameters. Successive applications of model diagnostics allow for refinements, and a process by which we can verify the validity of a postulated model. Of course, all of this is done on the basis of the single realization of the DGP.

While recognizing that any such model need not be correct, i.e., exactly match the DGP itself, such models can yet be useful to the extent to which they reflect important features in the data. Yet it is difficult to keep a model simple – which is necessay to its utility – and at the same time be sufficiently versatile to explain all the data's features. Moreover, the appellation of importance is subjective: a feature deemed important to one user may be irrelevant to another. This begs the question of customization: each user, with a distinct set of criteria and desired applications, could potentially stress the importance of a subset of features at the cost of de-emphasizing others. The classic paradigm ignores, or at least passes over, the issue of customization, and proposes a single all-purpose concept of utility: the minimization of one-step ahead forecast error MSE.

Another term for this classic conception of model utility is the Wold decomposition, which breaks a wide class of stochastic processes down in terms of a component that is completely

predictable from its own infinite past, and a second component fully describable in terms of one-step ahead forecast errors. Classical models can then be viewed as attempts to approximate the linear machinery in the Wold decomposition. However, were attention to focus upon an alternative utility, e.g., 10-step ahead forecasting, a different class of models would be suggested, with different apparatus for model selection, fitting, and evaluation.

However, customizing the modeling apparatus to allow for specific applications offers only a partial solution, because model mis-specification is the larger challenge. The full set of LPP solutions for a given time series is greatly constrained once a model is introduced, as only a particular subset of solutions can be obtained. If the model is badly mis-specified, the resulting LPP solution will be inadequate, even if the criteria for model selection are customized. This empirical disfunctionality motivated the genesis of DFA, which essentially provides access to a much wider pool of LPP solutions. Moreover, the basic DFA can be easily modified to allow for direct customization of real-time problems, according to whether users are concerned with timeliness, accuracy, or fidelity to the original signal (called smoothness).

1.1.3 The Scope of MDFA

Our critique of the classic paradigm has several facets. First, there is typically model mis-specification present. Second, the problem has typically not been structured properly, in the sense that the criteria used do not correspond to the relevant LPP, but rather to one-step ahead forecasting. Third, there is no specific customization of the model, in order to account for timeliness and accuracy. These weaknesses are actually linked together.

Model mis-specification is always present; the issue is whether it has a significant impact upon the objectives of analysis. For instance, a given model's mis-specification may have grave repercussions for certain problem structures, while being adequate for other LPPs. The given LPP of interest determines the gravity and impact of model mis-specification. Moreover, in the classic paradigm the one-step ahead forecasting LPP is solved, and it is merely hoped that timeliness and accuracy will be adequate for all users. Model parameters can be tweaked, or tuned, in order to indirectly modify timeliness and accuracy – but the relationships are indirect and often poorly understood. By building the timeliness-accuracy tradeoff directly into the DFA criterion, the optimality of an LPP solution for a customized application is assured.

These topics have been treated in Wildi and McElroy (2016, JTSE) in the case of univariate time series, which discusses at length the basic DFA (Sweave environment: replication). This book presents the generalized treatment of the multivariate LPP in Chapter **??**. But before discussing customization in Chapter **??**, we discuss the applications of forecasting and nowcasting, as well as the impact of data vintage, in Chapter **??**). Then the basic LPP treatment is extended to nonstationary processes in Chapter **??**, followed by a discussion of filter constraints (Chapter **??**). This treatment is extended to the case of co-integration in Chapter **??**. Applications to replicating and enhancing classical model-based approaches and HP/CF-filters are given in Chapter **??**, while more sophisticated gain/loss structures are discussed in Chapter **??**. Additional topics include inference (Chapter **??**), regularization (Chapter **??**), data revisions (Chapter **??**), mixed-frequency data (Chapter **??**), and adaptive filtering (Chapter **??**).

1.2 The Style of the Book

This book was generated using Sweave, in accordance with the philosophy of scientific replicability. Throughout the text are portions of R code that can be pasted into an R script and directly run, given that the user has certain packages already installed. This installation is described below.

1.2.1 Setting the Paths

Begin by clearing the workspace:

```
> #rm(list=ls())
```

The R code in various chapters of this book requires installation of the following R packages:

```
> # Load packages: time series and xts
> #library(tseries)
> library(xts)
> # State-space models (will be replicated by MDFA)
> library(dlm)
> # Classic filter designs (be replicated by MDFA)
> library(mFilter)
> # Numerical package
> library(numDeriv)
> # Graphical package for recession-shading (empirical examples based on US-GDP)
> library(tis)
> # Library for tables
> library(Hmisc)
> require(xtable)
> #install.packages("devtools")
> library(devtools)
> # Load MDFA package from github
> devtools::install_github("wiaidp/MDFA")
> # MDFA package
> library(MDFA)
```

US-GDP data for the empirical examples can be retrieved either directly from Quandl (requiring a preliminary user registration) or from a local data folder, which is the default-setting:

```
> # Load fresh data from quandl: T/F
> #   Default-setting is False: the data will be loaded from local data folder
> load_from_quandl <- F
```

Paths to MDFA code, as well as to the US-GDP data, must be provided. It is assumed that the MDFA package is saved to a main folder containing subfolders labeled as DFA, MDFA, model-based, and data. The R code in the book generates pdf graphs that are saved in a separate folder, whose path is specified by *path.out*.

```
> # Set main path
> path.main <- paste(getwd(),"/Sweave/",sep="")
> #path.main <- "C:\\Users\\Tucker\\Documents\\MDFAbook\\"
> # Set paths to subfolders
>   # Path to Latex-folder: all pdfs generated by the R code are filed there
> path.out <- paste(path.main,"Latex/",sep="")
>   # Path to data (US-GDP)
> path.dat <- paste(path.main,"Data/",sep="")
>   # Path to code that is part of MDFA-Legacy project but not part of MDFA package
> path.pgm <- paste(path.main,"R/",sep="")
```

The univariate DFA code is the same as in DFA; all empirical examples are and will be fully compatible.

1.2.2 DFA

We here briefly review the relevant facets of DFA, thereby providing an anchor for the MDFA discussion.

DFT and Periodogram

The Discrete Fourier Transform (DFT) and the periodogram are defined in Sections 2.2 and 2.3 of DFA. The following periodogram function – referred to as *per* below – in the MDFA package replicates these formulae. Note that frequency π is treated differently, depending on whether the sample size is odd or even; also, the value at frequency zero is scaled by $1/\sqrt{2}$, which is explained in later text.

```
> head(per,100)
```

```
1  function (x, plot_T)
2  {
3      len <- length(x)
4      per <- 0:(len/2)
5      DFT <- per
6      for (k in 0:(len/2)) {
7          cexp <- exp((0+1i) * (1:len) * 2 * pi * k/len)
8          DFT[k + 1] <- sum(cexp * x * sqrt(1/(2 * pi * len)))
9      }
10     if (abs(as.integer(len/2) - len/2) < 0.1)
11         DFT[k + 1] <- DFT[k + 1]/sqrt(2)
12     per <- abs(DFT)^2
13     if (plot_T) {
14         par(mfrow = c(2, 1))
15         plot(per, type = "l", axes = F, xlab = "Frequency", ylab = "Periodogram",
```

```
16             main = "Periodogram")
17         axis(1, at = 1 + 0:6 * len/12, labels = c("0", "pi/6",
18             "2pi/6", "3pi/6", "4pi/6", "5pi/6", "pi"))
19         axis(2)
20         box()
21         plot(log(per), type = "l", axes = F, xlab = "Frequency",
22             ylab = "Log-periodogram", main = "Log-periodogram")
23         axis(1, at = 1 + 0:6 * len/12, labels = c("0", "pi/6",
24             "2pi/6", "3pi/6", "4pi/6", "5pi/6", "pi"))
25         axis(2)
26         box()
27     }
28     return(list(DFT = DFT, per = per))
29 }
```

This function will be generalized in the new multivariate setting.

Basic DFA

A simple version of the DFA based on the MSE criterion alone – as proposed in Section 4.1 of DFA – is included in the MDFA package:

```
> # This function computes MSE DFA solutions
> # L is the length of the MA filter,
> # periodogram is the frequency weighting function in the DFA
> # Gamma is the transfer function of the symmetric filter (target) and
> # Lag is the lag-parameter: Lag=0 implies real-time filtering, Lag=L/2
> #     implies symmetric filter
> # The function returns optimal coefficients as well as the transfer
> #     function of the optimized real-time filter
> head(dfa_ms,100)
```

```
1  function (L, periodogram, Lag, Gamma)
2  {
3      periodogram[1] <- periodogram[1]/2
4      K <- length(periodogram) - 1
5      X <- exp(-(0+1i) * Lag * pi * (0:(K))/(K)) * rep(1, K + 1) *
6          sqrt(periodogram)
7      X_y <- exp(-(0+1i) * Lag * pi * (0:(K))/(K)) * rep(1, K +
8          1)
9      for (l in 2:L) {
10         X <- cbind(X, (cos((l - 1 - Lag) * pi * (0:(K))/(K)) +
11             (0+1i) * sin((l - 1 - Lag) * pi * (0:(K))/(K))) *
12             sqrt(periodogram))
```

```
13          X_y <- cbind(X_y, (cos((l - 1 - Lag) * pi * (0:(K))/(K)) +
14              (0+1i) * sin((l - 1 - Lag) * pi * (0:(K))/(K))))
15      }
16      xtx <- t(Re(X)) %*% Re(X) + t(Im(X)) %*% Im(X)
17      b <- as.vector(solve(xtx) %*% (t(Re(X_y)) %*% (Gamma * periodogram)))
18      trffkt <- 1:(K + 1)
19      trffkt[1] <- sum(b)
20      for (k in 1:(K)) {
21          trffkt[k + 1] <- (b %*% exp((0+1i) * k * (0:(length(b) -
22              1)) * pi/(K)))
23      }
24      return(list(b = b, trffkt = trffkt))
25 }
```

This function is nested in the multivariate MDFA, in the sense that the latter can replicate the former perfectly when suitably parametrized; see Section **??** below.

Customized DFA

A more general DFA function, called *dfa_analytic*, is proposed in Section 4.3.5 of DFA. Customization and the generic Accuracy-Timeliness-Smoothness (ATS) trilemma are presented in Sections 4.3 and 5 of DFA. This function is included in the MDFA package:

```
> head(dfa_analytic)
```

```
1 function (L, lambda, periodogram, Lag, Gamma, eta, cutoff, i1,
2     i2)
3 {
4     periodogram[1] <- periodogram[1]/2
5     lambda <- abs(lambda)
6     eta <- abs(eta)
```

The additional control parameters *lambda*, *eta* allow for customization of the filter, as discussed below in Chapter **??**. The Boolean *i1* and *i2* can enforce useful filter constraints; see Chapter **??**. This function is also encompassed by the MDFA.

1.2.3 MDFA

The R code for MDFA is more sophisticated than that of the DFA, and is correspondingly more complex and lengthy. As for the DFA package, the MDFA code can be sourced. We here briefly review the corresponding pieces.

Data Matrix

All time series are collected in a data-*matrix*, say X, which is organized as follows:

- the first column $X[,1]$ of X always corresponds to the target series: the target series $X[,1]$ is the time series to be forecasted, nowcasted or backcasted.

- Columns 2, 3, ... of X are allocated to the explanatory variables (more than one in a multivariate setting). If the target series is part of the set of explanatory variables (it does not have to be), then it must be assigned a specific column – by convention always the second one – in X, i.e., in this case the target series is entered twice, in the first column (target) and in the second column (explanatory data).

Example. Suppose we study a two-dimensional signal extraction problem, whereby the target series (first column) is part of the set of explanatory variables:

```
> set.seed(1)
> len <- 100
> target <- arima.sim(list(ar=0.9),n=len)
> explanatory_2 <- target+rnorm(len)
> explanatory <- cbind(target,explanatory_2)
> x <- cbind(target,explanatory)
> dimnames(x)[[2]] <- c("target","explanatory 1","explanatory 2")
> head(x)

       target explanatory 1 explanatory 2
[1,] 1.703613      1.703613     0.3191863
[2,] 1.398197      1.398197     3.2674879
[3,] 3.659995      3.659995     4.0850957
[4,] 3.254756      3.254756     3.0161086
[5,] 3.619020      3.619020     4.6775026
[6,] 3.285120      3.285120     4.1715424
```

For a one-step ahead forecast LPP, we might consider lagging both the explanatory variables:

```
> x<-cbind(x[,1],lag(x[,2:3],-1))
> dimnames(x)[[2]]<-c("target","lagged explanatory 1","lagged explanatory 2")
> head(x)

       target lagged explanatory 1 lagged explanatory 2
[1,] 1.703613                   NA                   NA
[2,] 1.398197             1.703613            0.3191863
[3,] 3.659995             1.398197            3.2674879
[4,] 3.254756             3.659995            4.0850957
[5,] 3.619020             3.254756            3.0161086
[6,] 3.285120             3.619020            4.6775026
```

By adopting the frequency-domain methods of this book, we can generalize this construction and avoid the introduction of missing values (denoted by NA in R). □

DFT

In contrast to the univariate DFA, where the LPP can be expressed in terms of the periodogram, the multivariate case requires the DFT of each time series in order to account for cross-sectional dependencies. These DFTs are complex-valued quantities, and the angular portion of the cross-spectrum provides information about the relative phase-shift of each explanatory time series. In the univariate case the relative phase-shift is irrelevant, because the target series and the explanatory series are identical. The scope of the method is extended in order to cover the mixed-frequency case, which is discussed in Chapter **??**. Another facet, is that we allow for the possibility of integrated processes; see Chapter **??**. In order to illustrate some of the new features we briefly look at the main DFT function called *spec_comp*:

```
> spec_comp

function (insamp, x, d)
{
    if (d == 1) {
        weight_func <- periodogram_bp(diff(x[1:insamp, 1]), 1,
            insamp - 1)$fourtrans
        if (length(weight_func) > 1) {
            for (j in 2:ncol(x)) {
                per <- periodogram_bp(diff(x[1:insamp, j]), 1,
                  insamp - 1)$fourtrans
                weight_func <- cbind(weight_func, per)
            }
        }
    }
    else {
        weight_func <- periodogram_bp(x[1:insamp, 1], 0, insamp)$fourtrans
        if (length(weight_func) > 1) {
            for (j in 2:ncol(x)) {
                per <- periodogram_bp(x[1:insamp, j], 0, insamp)$fourtrans
                weight_func <- cbind(weight_func, per)
            }
        }
    }
    colnames(weight_func) <- colnames(x)
    return(list(weight_func = weight_func))
}
<bytecode: 0x0000013c68e75600>
<environment: namespace:MDFA>
```

The inner loop tracks the columns of the data matrix X and the DFTs are stored in a matrix called *weight_func*, which is returned by the function. The matrix *weight_func* collects all DFTs;

the target series is always in the first column, whereas the DFTs of the explanatory series are in columns 2, 3, ... The function *periodogram_bp*, called in the above loop, is slightly more general than the DFA function *per* proposed in the previous section. In particular, it can handle various integration orders as well as seasonal peculiarities.

1.2.4 Using MDFA

A Versatile User Interface

MDFA is a generic forecast and signal extraction paradigm. Besides its capacity to replicate classical time series approaches, MDFA possesses unique features such as customization and regularization (Chapter **??**); it can treat data revisions (Chapter **??**), mixed-frequency problems (Chapter **??**), and non-stationarity (Chapters **??** and **??**. Accordingly, the user interface is more sophisticated than the precediing DFA package. Consider the head of the main estimation routine:

```
> head(mdfa_analytic)
```

```
1 function (L, lambda, weight_func, Lag, Gamma, eta, cutoff, i1,
2     i2, weight_constraint, lambda_cross, lambda_decay, lambda_smooth,
3     lin_eta, shift_constraint, grand_mean, b0_H0, c_eta, weight_structure,
4     white_noise, synchronicity, lag_mat, troikaner)
5 {
6     lambda <- abs(lambda)
```

Arguments such as *weight_func* (discussed above), the filter length (L), and the target specification *Gamma* are straightforward. But there are numerous additional control parameters: the relevance and the modus operandi of these will be discussed in this book.

Default Settings

For convenience, we store a so-called default setting of the parameters in a file called *control_default*. First we define the data (initialize the DFT matrix) and specify the filter length:

```
> weight_func <- matrix(rep(1:6,2),ncol=2)
> L <- 2
```

Given these two entries (DFT and filter length), the default-settings are as follows:

```
> d<-0
> lin_eta<-F
> lambda<-0
> Lag<-0
> eta<-0
> i1<-F
> i2<-F
> weight_constraint<-rep(1/(ncol(weight_func)-1),ncol(weight_func)-1)
```

```
> lambda_cross<-lambda_smooth<-0
> lambda_decay<-c(0,0)
> lin_expweight<-F
> shift_constraint<-rep(0,ncol(weight_func)-1)
> grand_mean<-F
> b0_H0<-NULL
> c_eta<-F
> weights_only<-F
> weight_structure<-c(0,0)
> white_noise<-F
> synchronicity<-F
> cutoff<-pi
> lag_mat<-matrix(rep(0:(L-1),ncol(weight_func)),nrow=L)
> troikaner<-F
```

This particular configuration will be used extensively in Chapter **??**; it corresponds to the basic MSE criterion (i.e., no customization) without regularization, without design constraints, and without any *a priori* knowledge. Also, this configuration presumes a common identical sampling frequency (i.e., no mixed frequency data) and the absence of data revisions. The default settings can be obtained by sourcing the corresponding R file:

```
> source(file=paste(path.pgm,"control_default.r",sep=""))
```

For later use we source a convenient plotting function:

```
> source(file=paste(path.pgm,"mplot_func.r",sep=""))
```

Selected Calls: Classic MSE, Customization and Regularization

Selected calls of the classic MSE criterion – as well as calls utilizing the customization or regularization features – are available through dedicated functions in the MDFA package:

```
> head(MDFA_mse)
```

```
1 function (L, weight_func, Lag, Gamma)
2 {
3     cutoff <- pi
4     lin_eta <- F
5     lambda <- 0
6     eta <- 0
```

```
> head(MDFA_mse_constraint)
```

```
1 function (L, weight_func, Lag, Gamma, i1, i2, weight_constraint,
2     shift_constraint)
3 {
```

```
4     cutoff <- pi
5     lin_eta <- F
6     lambda <- 0
```

```
> head(MDFA_cust)
```

```
1 function (L, weight_func, Lag, Gamma, cutoff, lambda, eta)
2 {
3     lin_eta <- F
4     weight_constraint <- rep(1/(ncol(weight_func) - 1), ncol(weight_func) -
5         1)
6     lambda_cross <- lambda_smooth <- 0
```

```
> head(MDFA_cust_constraint)
```

```
1 function (L, weight_func, Lag, Gamma, cutoff, lambda, eta, i1,
2     i2, weight_constraint, shift_constraint)
3 {
4     lin_eta <- F
5     lambda_cross <- lambda_smooth <- 0
6     lambda_decay <- c(0, 0)
```

```
> head(MDFA_reg)
```

```
1 function (L, weight_func, Lag, Gamma, cutoff, lambda, eta, lambda_cross,
2     lambda_decay, lambda_smooth, troikaner = F, b0_H0 = NULL)
3 {
4     lin_eta <- F
5     weight_constraint <- rep(1/(ncol(weight_func) - 1), ncol(weight_func) -
6         1)
```

```
> head(MDFA_reg_constraint)
```

```
1 function (L, weight_func, Lag, Gamma, cutoff, lambda, eta, lambda_cross,
2     lambda_decay, lambda_smooth, i1, i2, weight_constraint, shift_constraint,
3     troikaner = F, b0_H0 = NULL)
4 {
5     lin_eta <- F
6     lin_expweight <- F
```

The heads of the corresponding functions differ in the number of additional arguments available when going from specific (MSE) to generic (reg). The following chapters of the book provide an understanding of the use of these functions.