Multivariate Real-Time Signal Extraction

Marc Wildi and Tucker McElroy

May 23, 2019

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: 0x000000f41a8d4948>
<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)
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)
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.

Chapter 2

Linear Prediction Problems

2.1 Background on Stationary Vector Time Series

The reader should have a basic familiarity with multivariate time series analysis, such as that provided by Lütkepohl (2007). Our focus is on discrete-time stochastic processes taking values in $\mathbb{R}^n$, and such will be denoted $\{X_t\}$, i.e., a vector time series. Each X_t for a particular $t \in \mathbb{Z}$ is a random vector with n components, and the jth component will be denoted $X_{t,j}$ for $1 \leq j \leq n$. This can also be written as $X_{t,j} = e_j' X_t$, where e_j is the jth unit vector in $\mathbb{R}^n$. The union of these unit vectors is the $n \times n$ identity matrix, denoted 1_n.

In this book we are focused upon square integrable random variables, so that the classic Hilbert space projection theory (see, for example, Brockwell and Davis (1991)) can be applied. Occasionally, we consider vector time series $\{Y_t\}$ or $\{Z_t\}$, in which case the same conventions apply.When $\{X_t\}$ is weakly stationary, its autocovariance function (acf) is defined for $h \in \mathbb{Z}$ via

$$\Gamma(h) = \mathrm{Cov}[X_{t+h}, X_t],$$

which does not depend upon t by the stationarity assumption. Recall that $\Gamma(-h) = \Gamma(h)'$, and clearly

$$\Gamma_{jk}(h) = \mathrm{Cov}[X_{t+h,j}, X_{t,k}]$$

for $1 \leq j, k \leq n$. The spectral density is a complex matrix-valued function of $\omega \in [-\pi, \pi]$, defined as the Fourier Transform (FT) of the acf:

$$F(\omega) = \sum_{h \in \mathbb{Z}} \Gamma(h)\, z^h,$$

where we use the shorthand $z = e^{-i\omega}$. Clearly,

$$F(-\omega) = \sum_{h \in \mathbb{Z}} \Gamma(h)\, z^{-h} = \sum_{h \in \mathbb{Z}} \Gamma(-h)\, z^h = \sum_{h \in \mathbb{Z}} \Gamma(h)'\, z^h = F(\omega)',$$

which shows that the spectral density function (sdf) is Hermitian. In addition, its eigenvalues (for each ω) are real and non-negative. Given a bounded sdf (i.e., each F_{jk} has bounded modulus as

a function of ω), the acf can be recovered via inverse FT:

$$\Gamma(h) = \langle F \rangle_h = \frac{1}{2\pi} \int_{-\pi}^{\pi} F(\omega)\, e^{i\omega h}\, d\omega, \tag{2.1}$$

which uses the bracket notation to define the average integral of a function (of ω) multiplied by $e^{i\omega h} = z^{-h}$.

The lag operator on a time series is denoted L, and is defined via the action

$$LX_t = X_{t-1}.$$

Powers of L are defined analogously, with $L^0 = 1$ (an operator identity) and negative powers yielding forward time shifts, i.e., leads. Matrix polynomials of L yield new operators that act upon a time series using the linearity principal. Thus, if $A(L) = \sum_{j=0}^{a} A_j\, L^j$ for $n \times n$ matrices A_j, then

$$A(L)\, X_t = \sum_{j=0}^{a} A_j\, X_{t-j}.$$

For many applications in this book, the polynomials are actually scalar, and can be interpreted as having coefficients A_j given by an identity matrix 1_n multiplied by a scalar coefficient a_j.

The spectral representation of a stationary square integrable vector time series is particularly useful (see Brockwell and Davis (1991) for further details). Assuming that $\mathbb{E}X_t = 0$ (the zero vector in $\mathbb{R}^n$) so that no fixed effects are present, we describe the stochastic process via

$$X_t = \int_{-\pi}^{\pi} e^{i\omega t}\, \mathcal{Z}(d\omega), \tag{2.2}$$

which is a stochastic integral computed with a Stieltjes measure $\mathcal{Z}(d\omega)$. This is an orthogonal increments process, which mean $\mathcal{Z}$ is a random measure defined on $[-\pi, \pi]^n$ that maps disjoint sets to independent random variables. The actual distribution of the random measure is not our concern, but the particular orthogonal increments process associated with $\{X_t\}$ has the property that

$$\mathrm{Cov}[\mathcal{Z}(d\omega), \mathcal{Z}(d\xi)] = (2\pi)^{-1}\, F(\omega)\, d\omega\, 1_{\{\omega=\xi\}}$$

where 1_A is the indicator for the set A. (Also recall that for complex variables, a covariance involves conjugation of the second argument.) This validates the expression

$$\mathrm{Cov}[X_{t+h}, X_t] = \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} e^{i\omega(t+h)}\, e^{-i\xi t}\, \mathrm{Cov}[\mathcal{Z}(d\omega), \mathcal{Z}(d\xi)] = \frac{1}{2\pi} \int_{-\pi}^{\pi} F(\omega)\, e^{i\omega h}\, d\omega = \Gamma(h).$$

As an example – that furnishes a basic building block for subsequent processes – we have *white noise*, which refers to a mean zero $\{X_t\}$ where F is constant, i.e., $F(\omega) = \Sigma$ for all ω, where Σ is real and symmetric and non-negative definite. Clearly, $\Gamma(h) = 0$ for $h \neq 0$ and $\Gamma(0) = \Sigma$. We denote this type of process with the notation $\mathrm{WN}(\Sigma)$.

The advantage of the spectral representation is that it quickly facillitates the understanding of linear filtering in the frequency domain. A multivariate filter maps one vector time series to another, and for now we suppose that both input and output belong to $\mathbb{R}^n$. Linear filters can be expressed as matrix Laurent series in L:

$$\Psi(L) = \sum_{\ell \in \mathbb{Z}} \psi(\ell)\, L^{\ell},$$

where each $\psi(\ell)$ is an $n \times n$ matrix. Individual entries of the matrix are denoted $\psi_{jk}(\ell)$, for $1 \le j,k \le n$. We also use the notation $[\Psi(L)]_r^s$ to denote $\sum_{\ell=r}^s \psi(\ell)\, L^\ell$. Another quantity of interest is the derivative of a filter, defined via $\partial\Psi(L) = \sum_{j\in\mathbb{Z}} j\,\psi(j)\, L^{j-1}$.

In the special case that $\Psi(L)$ is a power series in L, the only nonzero coefficients are for $\ell \ge 0$, so that no negative powers of L are featured, i.e., the filter only utilizes present and past data. Such a filter is called a *concurrent* filter. The action of a linear filter on a weakly stationary time series, expressed in terms of the spectral representation, is

$$Y_t = \Psi(L)\, X_t = \sum_{\ell\in\mathbb{Z}} \psi(\ell)\, X_{t-\ell} = \sum_{\ell\in\mathbb{Z}} \psi(\ell) \int_{-\pi}^{\pi} e^{i\omega(t-\ell)}\, \mathcal{Z}(d\omega) = \int_{-\pi}^{\pi} e^{i\omega t}\, \Psi(e^{-i\omega})\, \mathcal{Z}(d\omega).$$

So the output time series $\{Y_t\}$ has orthogonal increments process $\Psi(z)\,\mathcal{Z}(d\omega)$, and in particular its sdf is

$$\Psi(z)\, F(\omega)\, \Psi(z)^*,$$

where $*$ denotes the conjugate transpose. Thus, it is very natural to analyze a filter in terms of the function $\Psi(e^{-i\omega})$, which is called the *frequency response function* (frf). In the scalar case, an frf can be further dissected via the polar decomposition of a complex number, yielding its *gain function* (the modulus) and the *phase function* (its angular portion). Note that the coefficients are recovered from the frf via the inverse FT:

$$\psi(\ell) = \langle \Psi(e^{-i\cdot}) \rangle_\ell.$$

It is well-known from Fourier theory that the degree of smoothness of a function at $\omega = 0$ corresponds to the degree of decay in the coefficients of its inverse FT. In particular, when the frf is smooth and flat in a neighborhood of the origin then the matrix norm of the coefficients $\psi(\ell)$ decays rapidly as $|\ell| \to \infty$. Conversely, discontinuities in the frf indicates slowly decaying coefficients.

Datasets are typically available as a finite set of contiguous regular measurements, denoted $\{x_1, x_2, \ldots, x_T\}$, where T is the length of sample. The data is viewed as a realization of the corresponding random vectors $\{X_1, X_2, \ldots, X_T\}$, or alternatively as a time window of the sample path $\{x_t\}$ corresponding to times $1, 2, \ldots, T$. Applying the *vec* operator to such a sample yields the full vector $\underline{X}$, which is given by

$$\underline{X} = \mathrm{vec}[X_1, X_2, \ldots, X_T].$$

The covariance matrix of this nT-dimensional random vector, in the stationary case, is block Toeplitz, Each block is $n \times n$, and the stth such block, for $1 \le s,t \le T$, is given by $\Gamma(s-t)$. Also, from the sample we can compute the *Discrete Fourier Transform* (DFT) via

$$\widetilde{X}(\omega) = T^{-1/2} \sum_{t=1}^{T} z^t\, X_t. \tag{2.3}$$

This can be computed for any $\omega \in [-\pi, \pi]$, though if we restrict to Fourier frequencies – of the form $2\pi j/T$ for integer j – then the various real and imaginary components of the DFT will be asymptotically uncorrelated, and also asymptotically normal. The multivariate *periodogram* is defined to be the rank one Hermitian matrix

$$\widehat{F}(\omega) = \widetilde{X}(\omega)\, \widetilde{X}(\omega)^*. \tag{2.4}$$

The periodogram furnishes a basic estimate of the spectral density F of the process. There is an empirical version of (2.1), where the periodogram is mapped to the sample autocovariance:

$$\widehat{\Gamma}(h) = \langle \widehat{F} \rangle_h = \frac{1}{2\pi} \int_{-\pi}^{\pi} \widehat{F}(\omega)\, e^{i\omega h}\, d\omega. \tag{2.5}$$

This is easily verified using the definition of sample autocovariance

$$\widehat{\Gamma}(h) = T^{-1} \sum_{t=1}^{T-h} X_{t+h}\, X_t'$$

for $h \geq 0$, and with $\widehat{\Gamma}(h) = \widehat{\Gamma}(-h)'$ for $h < 0$. Conversely, the periodogram is the FT of the sample autocovariances:

$$\widehat{F}(\omega) = \sum_{|h|<T} \widehat{\Gamma}(h)\, e^{-i\omega h}. \tag{2.6}$$

2.2 MSE Optimal Prediction Problems

2.2.1 The Linear Prediction Problem

We define the class of real-time estimation problems considered in this book. This chapter focuses upon the case of weakly stationary vector time series, but Chapter 4 makes extensions to difference stationary processes.

Definition 1 A **target** is defined to be the output of any known linear filter acting on the data process, i.e., $\{Y_t\}$ is a target time series corresponding to a given filter $\Psi(L)$ acting on a given observed time series $\{X_t\}$ if and only if we can write for all integers t

$$Y_t = \Psi(L) X_t.$$

We say that $\{Y_t\}$ is a **scalar target** if $\Psi(L)$ is a $1 \times n$-dimensional filter.

We are only interested in scalar targets. The reason is that if $\{Y_t\}$ is multivariate, we can treat each component series $\{Y_{t,j}\}$ for $1 \leq j \leq n$ in turn, so that without loss of generality we can just give the treatment for the scalar case.

Example 1 Multi-step Ahead Forecasting. Suppose that our goal is to forecast one of the component series h steps ahead, where $h \geq 1$ is the given *forecast lead*. Here, suppose that the series of interest is the first component, so that

$$Y_t = X_{t+h,1}$$

for all $t \in \mathbb{Z}$. This is indeed a scalar target, setting $\Psi(L) = L^{-h}\, e_1'$. That is, each $\psi(\ell)$ is a $1 \times n$ row vector, each of which are zero except $\psi(-h)$, which is given by e_1'.

Example 2 Ideal Low-Pass. In order to estimate a trend from a given series, conceptually we wish to screen out all the higher frequency components in the data. With reference to the spectral representation, if $\Psi(z)$ is zero for all ω in a band of the higher frequencies, then $\{Y_t\}$ will only be

composed of low frequency stochastic sinusoids. The simplest way to achieve such an output is to design the frf as an indicator function, involving a steep cutoff of noise frequencies; see Baxter and King (1999). This is viewed by some as the best possible definition of trend, and hence the filter is called the ideal low-pass. For scalar target, we have

$$\Psi(z) = 1_{[-\mu,\mu]}(\omega)\, e_1'$$

for some cutoff $\mu \in (0, \pi)$ that separates the pass-band from the stop-band. To understand this terminology of pass-band and stop-band, observe that the spectral representation of the scalar target is

$$Y_t = \int_{[-\mu,\mu]} e^{i\omega t}\, e_1' \mathcal{Z}(d\omega).$$

Here, the stochastic integration only includes frequencies in the pass-band $[-\mu, \mu]$, and all content belonging to the stop-band has been eliminated. The coefficients are given by

$$\psi(\ell) = \frac{\sin(\ell\mu)}{\pi\ell}\, e_1'$$

for $\ell \neq 0$ and $\psi(0) = \mu/\pi\, e_1'$.

Example 3 HP Low-pass. The Hodrick-Prescott (HP) filter (Hodrick and Prescott, 1997) is a low-pass filter appropriate for producing trends. A multivariate version of the HP low-pass (or just HP), associated with trend-irregular structural models, was proposed in McElroy and Trimbur (2015); the frf is given by

$$\Psi(z) = \Sigma_\mu \left(\Sigma_\mu + |1-z|^4\, \Sigma_\iota \right)^{-1}$$

in the case that the matrices Σ_μ and Σ_ι have full rank. When Σ_μ has reduced rank, an alternative expression is available, but note that the frf is a continuous matrix-valued function of ω. The matrices Σ_μ and Σ_ι have an interpretation in terms of the econometric concept of trend co-integration, which is further explored in Chapter **??**. It is always assumed that Σ_ι has full rank, and hence we can rewrite as

$$\Psi(z) = Q \left(Q + |1-z|^4\, 1_n \right)^{-1}$$

with $Q = \Sigma_\mu\, \Sigma_\iota^{-1}$ representing a matrix *signal-to-noise ratio*, or snr. This formula generalizes the univariate HP filter, which has frf

$$\Psi(z) = \frac{q}{q + |1-z|^4}$$

for snr parameter $q > 0$. Small values of q correspond to trends that are buried in volatile white noise, and thus require much smoothing to recover. The filter perfectly reflects this need, because a small q indicates a steep drop in the frf (which takes value one at $\omega = 0$) as ω is increased from zero, and hence the filter coefficients decay slowly. Conversely, higher values of q – corresponding to highly salient trends – yield an frf that equals unity in a large neighborhood of the origin, with coefficients that decay swiftly, indicating that little smoothing is needed to discover the trend. These observations carry over to the multivariate case, though we judge the size of the snr via a matrix norm (such as the maximum eigenvalue) of Q. Some of these eigenvalues can be zero, corresponding to the case that Σ_μ has reduced rank – this has the effect of generating trends that

are collinear. In the case of a scalar target, where we seek a trend for the first input series $\{X_{t,1}\}$, we have

$$\Psi(z) = e_1' \, Q \left(Q + |1 - z|^4 \, 1_n \right)^{-1}.$$

There are no known analytical formulas for the coefficients in the multivariate case, although in the univariate case they are available in McElroy (2008).

Example 4 HP High-pass. While the HP filter is used to extract trends, the residual is thought to measure the business cycle along with higher frequency oscillations in the data. Thus, taking the identity minus the HP low-pass yields the HP high-pass filter:

$$\Psi(z) = \left(Q + |1 - z|^4 \, 1_n \right)^{-1} |1 - z|^4.$$

The presence of the term $|1 - z|^4$ indicates differencing by the $(1 - L)^2$ and $(1 - L^{-1})^2$; thus the HP high-pass will annihilate cubic polynomials, and generally reduces high order stochastic trends to stationarity.

As we see from these examples, the targets of real-time signal extraction are features of the stochastic process that are of interest to a particular user. Some scalar targets depend upon only a single component of the time series (Examples 1 and 2), whereas others may be defined in terms of all the components (Examples 3 and 4). However, these targets represent an ideal feature of the time series that typically we cannot compute in real-time.

Real-time refers to time present, wherein we have access to present and past information, but have great uncertainty about the future. This is an essential feature of human existence. Time series methodology provides tools to model and understand the flow of information from past to present to future, with the implicit viewpoint that whereas causality is to some degree present – past events have a causative impact on future events, but not vice versa – there are other facets governing present and future outcomes that are not traceable to a particular variable's past. In other words, knowing the past values of a component time series $\{X_{t,1}\}$ is not sufficient to flawlessly determine its future values. However, having other explanatory variables in play can reduce the uncertainty of the future; taking n higher, we may be able to reduce the errors in forecasts.

The concept of *Granger causality* can be used to parse these notions mathematically. We may consider other component series $\{X_{t,j}\}$ for $j \geq 2$ useful for determining the future of $\{X_{t,1}\}$ if the one-step ahead forecast mean square error (MSE) is reduced, in which case we say that Granger causality is present. In such a scenario it can be proved that the one-step ahead forecast MSE arising from utilizing $\{X_{t,1}\}$ alone is greater than that obtained using the additional series. Hence, there is benefit to increasing n with additional ancillary series so long as they are helpful for forecasting. For real-time estimation problems, we seek to determine the best possible estimates of a target given a relevant collection of ancillary series.

More formally, the real-time estimation problem is concerned with projecting the target Y_t onto the available data $X_{t:} = \{X_t, X_{t-1}, \ldots\}$, i.e., the semi-infinite past. This formulation presumes that we have access to relevant ancillary series, and that we have access to all present and past values. In practice, databases only extend back a few decades, and the infinitely remote past

represents merely an idyll useful for mathematical simplicity. The linear estimation problem seeks a linear estimate of the form

$$\widehat{Y}_t = \sum_{\ell \geq 0} \widehat{\psi}(\ell)\, X_{t-\ell} = \widehat{\Psi}(L)\, X_t,$$

which shows that we seek a linear (time-invariant) concurrent filter $\widehat{\Psi}(L)$, applied to $\{X_t\}$. We desire that the error in approximating the target with the available data be small with respect to MSE. If $\{X_t\}$ were Gaussian, we could view our estimate as the conditional expectation $\widehat{Y}_t = \mathbb{E}[Y_t | X_{t:}]$, with the coefficients $\{\widehat{\psi}(\ell)\}$ selected to minimize the MSE of the approximation error $Y_t - \widehat{Y}_t$. However, in our treatment in this book we do not presume Gaussian structure, and are not concerned with conditional expectations *per se*; rather, we seek linear solutions with minimal MSE.

Definition 2 The **Linear Prediction Problem** (LPP) seeks the minimal MSE linear estimate that solves the real-time estimation problem arising from a scalar target. That is, the LPP involves determining causal $\widehat{\Psi}(L)$ such that the prediction error

$$Y_t - \widehat{Y}_t = \left[\Psi(L) - \widehat{\Psi}(L)\right]\, X_t$$

has mean zero and minimal MSE.

Example 5 Multi-step Ahead Forecasting. The LPP corresponds to optimal h-step forecasting, and the forecast error is $[L^{-h}\, e_1' - \widehat{\Psi}(L)]\, X_t$. Note that although $\Psi(L)$ only involves one component series $\{X_{t,1}\}$, the real-time concurrent filter $\widehat{\Psi}(L)$ can involve all n component series.

Example 6 HP Low-pass. The LPP attempts to determine an optimal real-time trend estimate, where the target trend – sometimes called the historical trend – is defined through the HP low-pass filter. Here, both the target filter $\Psi(L)$ and the real-time concurrent filter $\widehat{\Psi}(L)$ involve all n component series.

2.2.2 Solution to the Linear Prediction Problem

When the data process is itself causal and linear, it is possible to give an explicit solution to the LPP in terms of the Wold decomposition (Brockwell and Davis, 1991). All purely nondeterministic weakly stationary (mean zero) processes have a Wold decomposition $X_t = \Theta(L)\epsilon_t$, where $\{\epsilon_t\}$ is WN(Σ) and $\Theta(L) = \sum_{\ell \in \mathbb{Z}} \theta(\ell)\, L^\ell$. When $\theta(\ell) = 0$ for all $\ell < 0$, the process is called *causal*. First, the error in the LPP is denoted $E_t = Y_t - \widehat{Y}_t$, which is clearly mean zero and covariance stationary, in fact having spectral representation

$$E_t = \int_{-\pi}^{\pi} e^{i\omega t}\, \left[\Psi(z) - \widehat{\Psi}(z)\right]\, \mathcal{Z}(d\omega). \tag{2.7}$$

With these preliminaries, we can state the solution to the LPP.

Proposition 1 *Suppose that $\{X_t\}$ is mean zero and weakly stationary with spectral representation (2.2), and moreover is causal, expressed as $X_t = \Theta(L)\, \epsilon_t$. Then the solution to the LPP posed by a scalar target $Y_t = \Psi(L)\, X_t$ is given by*

$$\widehat{\Psi}(L) = \sum_{\ell \geq 0} \psi(\ell)\, L^\ell + \sum_{\ell < 0} \psi(\ell)\, [\Theta(L)]_{-\ell}^{\infty}\, L^\ell\, \Theta(L)^{-1}. \tag{2.8}$$

Moreover, the minimal MSE is given by

$$\frac{1}{2\pi}\int_{-\pi}^{\pi}\sum_{\ell>0}\psi(-\ell)\,z^{-\ell}[\Theta(z)]_0^{\ell-1}\,\Sigma\,[\Theta(z)]_0^{\ell-1^*}\sum_{\ell>0}\psi(-\ell)\,z^{\ell}\,d\omega. \tag{2.9}$$

Proof of Proposition 1. In order for a linear solution to be MSE optimal, it is sufficient that the resulting error process be uncorrelated with the data $X_{t:}$. If we can show that the real-time signal extraction error process $\{E_t\}$ depends only on future innovations, then by the causality of $\{X_t\}$ the error process must be uncorrelated with $X_{t:}$, establishing optimality. The filter error of the putative solution is given by

$$\begin{aligned}\Psi(L)-\widehat{\Psi}(L) &= \sum_{\ell<0}\psi(\ell)\,L^{\ell}\,\left(1-[\Theta(L)]_{-\ell}^{\infty}\,\Theta(L)^{-1}\right)\\ &= \sum_{\ell<0}\psi(\ell)\,L^{\ell}\,[\Theta(L)]_0^{-(\ell+1)}\,\Theta(L)^{-1}.\end{aligned}$$

Applying this to $\{X_t\}$ yields

$$E_t = \sum_{\ell=1}^{\infty}\psi(-\ell)\,[\Theta(L)]_0^{\ell-1}\,\epsilon_{t+\ell}.$$

Noting that $[\Theta(L)]_0^{\ell-1}$ is an order $\ell-1$ polynomial in L, and is applied to $\epsilon_{t+\ell}$, it is apparent that E_t is a linear function of future innovations $\{\epsilon_{t+1},\epsilon_{t+2},\ldots\}$. Computing the variance of E_t yields the expression for the minimal MSE. □

Remark 1 The formula (2.9) gives us a lower bound on the MSE when we use sub-optimal proxies for $\widehat{\Psi}(L)$.

As indicated by Remark 1, the result of Proposition 1 is chiefly useful when we know $\Theta(L)$. However, this is rarely the case in practice: a classical parametric approach involves formulating a time series model, fitted using the Gaussian likelihood, and finally computing the LPP solution in terms of the fitted model. Alternatively, one might consider fitting a specified model such that the LPP MSE is minimized. A more broad nonparametric approach involves considering classes of concurrent filters and directly minimizing the LPP MSE over this class – this is the methodology of Direct Filter Analysis (DFA).

Illustration 1 VAR(1). Consider an LPP where the true process $\{X_t\}$ is a Vector Autoregression (VAR) of order 1. This process can be described via

$$X_t = \Phi\,X_{t-1}+\epsilon_t$$

for a matrix Φ that is stable, i.e., has all eigenvalues bounded by one in modulus (Lütkepohl, 2007). It is known that the VAR(1) has the causal representation $\Theta(L)=(1-\Phi\,L)^{-1}$. Because for $\ell<0$

$$[\Theta(L)]_{-\ell}^{\infty} = \sum_{j=-\ell}^{\infty}\Phi^j\,L^j = \Phi^{-\ell}\,L^{-\ell}\,(1-\Phi\,L)^{-1},$$

we find that (2.8) reduces to

$$\widehat{\Psi}(L) = \sum_{\ell\geq 0}\psi(\ell)\,L^{\ell}+\sum_{\ell<0}\psi(\ell)\,\Phi^{-\ell}.$$

The second term in this expression we denote by $A_\Psi(\Phi)$. Hence, the optimal concurrent filter is determined by applying the filter to past data and modifying the present weight $\psi(0)$ by adding the quantity $A_\Psi(\Phi)$. In the case of h-step ahead forecasting of the first time series (Example 1), $\widehat{\Psi}(L) = A_\Psi(\Phi) = e_1' \, \Phi^h$. This formula demonstrates that it is essential that Φ be stable, and if fitting a VAR(1) we must parametrize Φ such that stability is guaranteed. The following parametrization (Roy, McElroy, and Linton (2019) provides a bijection from $\mathbb{R}^{n^2} \times \{\pm 1\}$ to the space of stable $n \times n$ matrices:

$$
\begin{aligned}
&(\phi, \delta) \to (V, Q) \\
&\quad V = L \, D \, L' \\
&\quad Q = E \, (1_n - S) \, (1_n + S)^{-1} \\
&\quad \Phi = V^{1/2} \, Q \, (1_n + V)^{-1/2}
\end{aligned}
$$

provides the map from the parameters to Φ. Here V is a positive definite matrix with Cholesky decomposition in terms of a unit lower triangular matrix L and a positive diagonal matrix D; any such matrix can be described by $\binom{n}{2}$ arbitrary real entries in the lower portion of L, and by the exponential of n arbitrary real numbers for D. The matrix Q is orthogonal, given by the Cayley representation, where S is anti-symmetric – and hence can be described by $\binom{n}{2}$ arbitrary real numbers – and E is diagonal, with the first entry given by δ (which equals ± 1) and the remaining $n-1$ entries are equal to one. The inverse transformation is given by

$$
\begin{aligned}
&\Phi \to (V, Q) \\
&V = \sum_{j \geq 1} \Phi^j \, \Phi^{j\prime} \\
&Q = V^{-1/2} \, \Phi \, (1_n + V)^{1/2} \\
&S = (1_n + E \, Q)^{-1} \, (1_n - E \, Q) \\
&\delta = \det Q,
\end{aligned}
$$

where ϕ can be recovered from the entries of L and D in the Cholesky decomposition of V, along with the lower left entries of S.

2.3 Model Fitting via LPP MSE Minimization

Here we study the mechanics of fitting a parametric model such that the LPP MSE is minimized. In the case of the one-step ahead forecasting MSE, this is related to Whittle estimation of vector time series models (c.f., Taniguchi and Kakizawa (2000)). We will focus on the class of separable causal linear models, wherein the innovation variance Σ is governed by a separate set of parameters from those describing the power series $\Theta(L)$. The model is essentially described through a particular class of power series $\Theta_\vartheta(L)$, parameterized by a vector ϑ belonging to some model parameter manifold. Hence the model sdf is

$$
F_\vartheta(\omega) = \Theta_\vartheta(z) \, \Sigma \, \Theta_\vartheta(z)^* .
$$

However, the model may be misspecified: the process' sdf is denoted $\widetilde{F}$, and may not belong to the model class. The goal of model fitting is to determine ϑ such that F_ϑ is a good approximation to $\widetilde{F}$. Clearly, knowing ϑ does not fully determine F_ϑ because Σ remains unknown; however, the methods described below provide for estimates of Σ in terms of ϑ and the process. From the proof of Proposition 1 we know that the filter error satisfies

$$E_t = \sum_{\ell=1}^{\infty} \psi(-\ell)\, L^{-\ell}\, [\Theta(L)]_0^{\ell-1}\, \Theta_\vartheta(L)^{-1}\, X_t.$$

In other words, we have an error filter $\Xi_\vartheta(L)\,\Theta_\vartheta(L)^{-1}$, where

$$\Xi_\vartheta(L) = \sum_{\ell=1}^{\infty} \psi(-\ell)\, L^{-\ell}\, [\Theta(L)]_0^{\ell-1},$$

such that for any choice of ϑ we can compute filter errors $\{E_t\}$. Note that these are not in general to be interpreted as residuals, and they need not be white noise. But we can seek to minimize their variance. In practice, the calculation of such filter errors may require a truncation of the error filter, because the finite sample $X_1, X_2, \ldots, X_T$ is available, not the entire infinite past. The error filter is $1 \times n$, and for any ϑ and any Hermitian function G we can compute

$$\begin{aligned} J_\Psi(\vartheta, G) &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \Xi(z)\, \Theta_\vartheta(z)^{-1}\, G(\omega)\, \Theta_\vartheta(z)^{-1^*}\, \Xi(z)^*\, d\omega = \mathrm{tr}\{\langle G\, K_\vartheta \rangle_0\} \\ K_\vartheta(z) &= \Theta_\vartheta(z)^{-1^*}\, \Xi(z)^*\, \Xi(z)\, \Theta_\vartheta(z)^{-1}. \end{aligned}$$

Then $\mathrm{Var}[E_t] = J_\Psi(\vartheta, \widetilde{F})$. As we seek to minimize the variability in the filter errors, we can take $J_\Psi(\vartheta, \widetilde{F})$ as our criterion function. However, this will only determine the proximity of the model to the true sdf, which is unavailable to us – in order to compute actual parameter estimates, we must utilize the data to approximate the true sdf. There are basic results giving asymptotic normality for simple functionals of the periodogram, and therefore this crude estimator of the true sdf is sufficient for our purposes. We propose $J_\Psi(\vartheta, \widehat{F})$ as an estimator of $J_\Psi(\vartheta, \widetilde{F})$, and intend that the respective minimizers have the same relationship. Namely, if $\vartheta(\widetilde{F})$ is the unique minimizer of $J_\Psi(\vartheta, \widetilde{F})$ and $\vartheta(\widehat{F})$ is the unique minimizer of $J_\Psi(\vartheta, \widehat{F})$, then $\vartheta(\widehat{F})$ is an estimator of $\vartheta(\widetilde{F})$, which is called the *pseudo-true value* (PTV). From the PTV and estimator, we can also compute the innovation covariance matrix by the formulas

$$\begin{aligned} \Sigma(\widetilde{F}) &= \langle \Theta_{\vartheta(\widetilde{F})}(z)^{-1}\, \widetilde{F}\, \Theta_{\vartheta(\widetilde{F})}(z)^{-1^*} \rangle_0 \\ \Sigma(\widehat{F}) &= \langle \Theta_{\vartheta(\widehat{F})}(z)^{-1}\, \widehat{F}\, \Theta_{\vartheta(\widehat{F})}(z)^{-1^*} \rangle_0. \end{aligned}$$

In the special case that the model is correctly specified, there exists some $\widetilde{\vartheta}$ and $\widetilde{\Sigma}$ such that $\widetilde{F}(\omega) = \Theta_{\widetilde{\vartheta}}(z)\, \widetilde{\Sigma}\, \Theta_{\widetilde{\vartheta}}(z)^*$; it is shown below that the PTV matches the truth.

Proposition 2 *Given an LPP Ψ and the criterion function $J_\Psi(\vartheta, \widetilde{F})$, if the model is correctly specified and the minimizer $\vartheta(\widetilde{F})$ is unique then it equals the true parameter $\widetilde{\vartheta}$, and $\Sigma(\widetilde{F}) = \widetilde{\Sigma}$.*

Proof of Proposition 2. Because the model is correct, the criterion function becomes

$$J_\Psi(\vartheta, \widetilde{F}) = \langle \Xi(z)\, \Theta_\vartheta(z)^{-1}\, \Theta_{\widetilde{\vartheta}}(z)\, \widetilde{\Sigma}\, \Theta_{\widetilde{\vartheta}}(z)^*\, \Theta_\vartheta(z)^{-1^*}\, \Xi(z)^* \rangle_0,$$

which for $\vartheta = \widetilde{\vartheta}$ achieves the minimal value:

$$J_\Psi(\widetilde{\vartheta}, \widetilde{F}) = \langle \Xi(z)\, \widetilde{\Sigma}\, \Xi(z)^* \rangle_0.$$

Because the minimizer is unique by assumption, $\vartheta(\widetilde{F}) = \widetilde{\vartheta}$. Plugging this into the formula for $\Sigma(\widetilde{F})$, we see that it equals $\widetilde{\Sigma}$. $\square$

Remark 2 One of the conditions of Proposition 2 is that the PTV is unique. Non-uniqueness can arise in practice, but any PTV that minimizes the LPP criterion is suitable for generating an optimal solution. Hence, even though in this situation a particular PTV does not equal the true parameter, yet it generates the same minimal value of J_Ψ as the true parameter, and is "just as good" as the true parameter for purposes of the particular LPP. We do not care about correct model fitting *per se*.

Example 7 Multi-step Ahead Forecasting. For h-step ahead forecasting, only $\psi(-h)$ is nonzero, so that $\Xi(L) = L^{-h}\, e_1' \, [\Theta(L)]_0^{h-1}$. Hence, the criterion function J_Ψ fits models so as to minimize (in the frequency domain) h-step ahead forecast error of the first series. In the special case that $h = 1$, the criterion function is

$$J_\Psi(\vartheta, G) = e_1' \, \langle \Theta_\vartheta(z)^{-1}\, G\, \Theta_\vartheta(z)^{-1^*} \rangle_0 \, e_1.$$

If we were to compute such a measure for all n series, and sum over the n criteria, we would obtain the concentrated Whittle likelihood, namely

$$\mathrm{tr}\, \{ \langle \Theta_\vartheta(z)^{-1}\, G\, \Theta_\vartheta(z)^{-1^*} \rangle_0 \}.$$

See the discussion in McElroy and Findley (2015). This connection justifies viewing J_Ψ as a generalization of the Whittle likelihood from one-step ahead forecasting to more general real-time LPPs.

It is possible to conduct inference from the PTVs on the basis of the estimates $\vartheta(\widehat{F})$, and thereby assess model fit. In order to formulate our result, we assume that the PTVs are not on the boundary of the parameter set (otherwise the limit theory is non-standard; c.f., Self and Liang (1987)), and that they are unique. We also assume that the Hessian $H(\vartheta) = \nabla \nabla' J_\Psi(\vartheta, \widetilde{F})$ of J_Ψ is positive definite at the PTV. The so-called Hosoya-Taniguchi (HT) conditions of Hosoya and Taniguchi (1982) impose sufficient regularity on the process $\{X_t\}$ for our purposes; these conditions require that $\{X_t\}$ is a causal filter of a higher-order martingale difference. A simpler limiting variance expression is available if the fourth order cumulant function of $\{X_t\}$ is zero.

Theorem 1 *Suppose that $\vartheta(\widetilde{F})$ exists uniquely in the interior of the model parameter space, and that $H(\vartheta(\widetilde{F}))$ is positive definite. Suppose that $\{X_t\}$ has finite fourth moments, conditions (HT1)-(HT6) of Taniguchi and Kakizawa (2000, pp.55-56) hold, and that the fourth order cumulant function of $\{X_t\}$ is zero. Then the estimator is consistent for the PTV, and*

$$\sqrt{T}\, \left(\vartheta(\widehat{F}) - \vartheta(\widetilde{F}) \right) \stackrel{\mathcal{L}}{\Longrightarrow} \mathcal{N}\left(0, H(\vartheta(\widetilde{F}))^{-1}\, V(\vartheta(\widetilde{F}))\, H(\vartheta(\widetilde{F}))^{-1} \right)$$

as $T \to \infty$, where

$$V_{jk}(\vartheta) = tr\{ \langle \partial_j K_\vartheta(z)\, \widetilde{F}\, \partial_k K_\vartheta(z)\, \widetilde{F} \rangle_0 \}.$$

Proof of Theorem 1. A Taylor series exapansion of the gradient of $J_\Psi(\vartheta, \widehat{F})$ and $J_\Psi(\vartheta, \widetilde{F})$ yields the asymptotic expression

$$\sqrt{T}\,\left(\vartheta(\widehat{F}) - \vartheta(\widetilde{F})\right) = o_P(1) - H(\vartheta(\widetilde{F}))^{-1}\,\mathrm{tr}\,\{\langle(\widehat{F} - \widetilde{F})\,\nabla K_\vartheta\rangle_0\},$$

where the trace operator acts upon the spectral matrices, for each component of the gradient operator. Our assumptions allow us to apply Lemma 3.1.1 of Taniguchi and Kakizawa (2000) to the right hand expression, yielding the stated central limit theorem. □

Illustration 2 VAR(1). When the model is a VAR(1), the parameter vector ϑ describes the entries of Φ in such a way that the matrix is stable, as described previously. The error filter can be expressed

$$\begin{aligned}\Xi(L) &= \sum_{\ell=1}^{\infty} \psi(-\ell)\,L^{-\ell} \sum_{k=0}^{\ell-1} \Phi^k\,L^k \\ &= \sum_{\ell=1}^{\infty} \psi(-\ell)\,L^{-\ell}\,\left(1 - \Phi^\ell\,L^\ell\right)\,(1 - \Phi\,L)^{-1} \\ &= \left(\sum_{\ell=1}^{\infty} \psi(-\ell)\,L^{-\ell} - A_\Psi(\Phi)\right)\,(1 - \Phi\,L)^{-1}.\end{aligned}$$

It follows that

$$\begin{aligned}J_\Psi(\vartheta, G) = &\sum_{\ell,k>0} \psi(-\ell)\,\langle G\rangle_{\ell-k}\,\psi(-k)' - A_\Psi(\Phi) \sum_{k>0} \langle G\rangle_{-k}\,\psi(-k)' \\ &- \sum_{\ell>0} \psi(-\ell)\,\langle G\rangle_\ell\,A_\Psi(\Phi)' + A_\Psi(\Phi)\,\langle G\rangle_0\,A_\Psi(\Phi)',\end{aligned}$$

which is easily computed. Optimization with respect to ϑ is straightforward. In the special case of h-step ahead forecasting, the criterion further simplifies to

$$\begin{aligned}J_{L^{-h}}(\vartheta, G) = &\,e_1'\,\langle G\rangle_0\,e_1 - e_1'\,\Phi^h\,\langle G\rangle_{-h}\,e_1 \\ &- e_1'\,\langle G\rangle_h\,\Phi^{h\prime}\,e_1 + e_1'\,\Phi^h\,\langle G\rangle_0\,\Phi^{h\prime}\,e_1,\end{aligned}$$

and any ϑ such that

$$e_1'\,\Phi^h = e_1'\,\langle G\rangle_h\,\langle G\rangle_0^{-1}$$

is a critical point. If the model is correctly specified, then setting G equal to the spectral density of the VAR(1) yields a minimal possible MSE of

$$e_1'\,\left(\Gamma(0) - \Phi^h\,\Gamma(0)\,\Phi^{h\prime}\right)\,e_1.$$

It is known that Whittle estimation corresponds to the $h = 1$ case, with a criterion function given by the determinant of the forecast MSE matrix (McElroy and Findley, 2015), and in essence incorporates forecast error from all n series. In the above, the criterion only depends on the performance of the first series, which allows a practitioner to focus on parameter values that sacrifice performance on the other $n - 1$ series in order to achieve superior results for the first series.

Exercise 1 Correct VAR(1) LPP. Simulate a sample of size $T = 100$ from a bivariate VAR(1) process with

$$\Phi = \begin{bmatrix} 1 & 1/2 \\ -1/5 & 3/10 \end{bmatrix}$$

and Σ equal to the identity. The eigenvalues are $4/5$ and $1/2$. Then utilize the LPP criterion $J_\Psi(\vartheta, G)$ to fit a VAR(1) model (use the stable parametrization of Φ, using two choices of δ) with both the 2-step ahead forecasting LPP (Example 1) and the ideal low-pass LPP (Example 2) with $\mu = \pi/24$, where G is given by the periodogram of the sample. (As usual, the first of the two series is the target.) Do you obtain a unique minimizer? Do the parameter estimates appear to be consistent? How do the estimates compare to the Yule-Walker estimates? Repeat for $T = 200$ and $T = 500$.

```
> # Simulate a Gaussian VAR(1) of sample size 100:
> set.seed(1234)
> T <- 100
> phi.matrix <- rbind(c(1,.5),c(-.2,.3))
> innovar.matrix <- diag(2)
> true.psidelta <- var1.par2psi(phi.matrix,100)
> gamma.0 <- matrix(solve(diag(4) - phi.matrix %x% phi.matrix) %*%
+         matrix(innovar.matrix,ncol=1),nrow=2)
> x.init <- t(chol(gamma.0)) %*% rnorm(2)
> x.next <- x.init
> x.sim <- NULL
> for(t in 1:T)
+ {
+         x.next <- phi.matrix %*% x.next + rnorm(2)
+         x.sim <- cbind(x.sim,x.next)
+ }
> x.sim <- ts(t(x.sim))
> x.acf <- acf(x.sim,type="covariance",plot=FALSE,lag.max=T)[[1]]
> x.acf <- aperm(aperm(x.acf,c(3,2,1)),c(2,1,3))
> # Yule-Walker fit
> phi.yw <- x.acf[,,2] %*% solve(x.acf[,,1])        # phi coefficient from YW
> yw.psidelta <- var1.par2psi(phi.yw,100)
> # 1-step ahead forecasting
> psi.array <- array(0,c(1,2,1))
> psi.array[,,1] <- c(1,0)
> acf.array <- x.acf[,,1:3]
> theta <- rep(0,4)
> var1.fit.2step.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,method="BFGS")
> var1.fit.2step.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
```

```
+         acf.array=acf.array,delta=-1,method="BFGS")
> # fits using both delta values, compared
> print(c(var1.fit.2step.Pos$value,var1.fit.2step.Neg$value))

[1] 1.023043 1.023043

> print(lpp.var1(yw.psidelta[[1]],psi.array,acf.array,yw.psidelta[[2]]))

         [,1]
[1,] 1.023043

> # phi coefficient from 1-step LPP, delta = 1
> print(var1.psi2par(var1.fit.2step.Pos$par,1))

          [,1]      [,2]
[1,]  1.029733 0.6310298
[2,] -0.411762 0.3440929

> # phi coefficient from 1-step LPP, delta = -1
> print(var1.psi2par(var1.fit.2step.Neg$par,-1))

          [,1]      [,2]
[1,]  1.029734  0.631041
[2,] -1.382905 -1.071446

> # phi coefficient from Yule-Walker
> print(phi.yw)

           [,1]      [,2]
[1,]  1.0297332 0.6310298
[2,] -0.1719921 0.2678559

> # 2-step ahead forecasting
> psi.array <- array(0,c(1,2,2))
> psi.array[,,1] <- c(0,0)
> psi.array[,,2] <- c(1,0)
> acf.array <- x.acf[,,1:3]
> theta <- rep(0,4)
> var1.fit.2step.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,method="BFGS")
> var1.fit.2step.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,method="BFGS")
> # fits using both delta values, compared
> print(c(var1.fit.2step.Pos$value,var1.fit.2step.Neg$value))

[1] 2.674102 2.674102
```

```
> print(lpp.var1(yw.psidelta[[1]],psi.array,acf.array,yw.psidelta[[2]]))

         [,1]
[1,] 2.680453

> # phi coefficient from 2-step LPP, delta = 1
> print(var1.psi2par(var1.fit.2step.Pos$par,1))

           [,1]      [,2]
[1,]  1.0822622 0.5237107
[2,] -0.4293269 0.3509603

> # phi coefficient from 2-step LPP, delta = -1
> print(var1.psi2par(var1.fit.2step.Neg$par,-1))

            [,1]       [,2]
[1,] -0.98471399 -1.7618564
[2,]  0.01320693  0.5588173

> # phi coefficient from Yule-Walker
> print(phi.yw)

           [,1]      [,2]
[1,]  1.0297332 0.6310298
[2,] -0.1719921 0.2678559

> # low-pass LPP
> mu <- pi/24
> psi.array <- array(c(1,0) %x% sin(seq(1,T-1)*mu)/(pi*seq(1,T-1)),c(1,T-1,2))
> psi.array <- aperm(psi.array,c(1,3,2))
> acf.array <- x.acf
> theta <- rep(0,4)
> var1.fit.bk.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,method="BFGS")
> var1.fit.bk.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,method="BFGS")
> # fits using both delta values, compared
> print(c(var1.fit.bk.Pos$value,var1.fit.bk.Neg$value))

[1] 1.237577 1.237577

> lpp.var1(yw.psidelta[[1]],psi.array,acf.array,yw.psidelta[[2]])

         [,1]
[1,] 1.247483
```

```
> # phi coefficient from low-pass LPP, delta = 1
> print(var1.psi2par(var1.fit.bk.Pos$par,1))

             [,1]      [,2]
[1,]  0.922765701 0.2783565
[2,] -0.008387922 0.7268689

> # phi coefficient from low-pass LPP, delta = -1
> print(var1.psi2par(var1.fit.bk.Neg$par,-1))

          [,1]      [,2]
[1,]  3.007823  3.166735
[2,] -2.555982 -2.804916

> # phi coefficient from Yule-Walker
> print(phi.yw)

           [,1]      [,2]
[1,]  1.0297332 0.6310298
[2,] -0.1719921 0.2678559
```

Exercise 2 Incorrect VAR(1) LPP. Simulate a sample of size $T = 100$ from a bivariate Vector Moving Avergage process of order one, or VMA(1), given by

$$X_t = \epsilon_t + \Theta\, \epsilon_{t-1},$$

where

$$\Theta = \begin{bmatrix} 1 & 0 \\ 2/5 & 2 \end{bmatrix}$$

and Σ equals the identity. Use the VAR(1) LPP criterion of Exercise 1 to fit the mis-specified VAR(1) to this process, for both the 2-step ahead forecasting LPP and the ideal low-pass LPP. (As usual, the first of the two series is the target.) Why do the parameter estimates not match those of Θ? How do the estimates compare to the Yule-Walker estimates? Repeat for $T = 200$ and $T = 500$, and explain your results.

```
> # Simulate a Gaussian VMA(1) of sample size 100:
> set.seed(1234)
> T <- 100
> theta.matrix <- rbind(c(1,0),c(.4,2))
> innovar.matrix <- diag(2)
> eps.old <- rnorm(2)
> x.sim <- NULL
> for(t in 1:T)
+ {
+         eps.next <- rnorm(2)
```

```
+         x.next <- eps.next + theta.matrix %*% eps.old
+         eps.old <- eps.next
+         x.sim <- cbind(x.sim,x.next)
+ }
> x.sim <- ts(t(x.sim))
> x.acf <- acf(x.sim,type="covariance",plot=FALSE,lag.max=T)[[1]]
> x.acf <- aperm(aperm(x.acf,c(3,2,1)),c(2,1,3))
> # Yule-Walker
> phi.yw <- x.acf[,,2] %*% solve(x.acf[,,1])        # phi coefficient from YW
> yw.psidelta <- var1.par2psi(phi.yw,100)
> # 1-step ahead forecasting
> psi.array <- array(0,c(1,2,1))
> psi.array[,,1] <- c(1,0)
> acf.array <- x.acf[,,1:3]
> theta <- rep(0,4)
> var1.fit.2step.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,method="BFGS")
> var1.fit.2step.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,method="BFGS")
> # fits using both delta values, compared
> print(c(var1.fit.2step.Pos$value,var1.fit.2step.Neg$value))

[1] 1.582355 1.582355

> print(lpp.var1(yw.psidelta[[1]],psi.array,acf.array,yw.psidelta[[2]]))

         [,1]
[1,] 1.582355

> # phi coefficient from 1-step LPP, delta = 1
> print(var1.psi2par(var1.fit.2step.Pos$par,1))

            [,1]        [,2]
[1,] 0.527735961 -0.02835286
[2,] 0.002445085  0.71087515

> # phi coefficient from 1-step LPP, delta = -1
> print(var1.psi2par(var1.fit.2step.Neg$par,-1))

          [,1]        [,2]
[1,] 0.5277356 -0.02835284
[2,] 0.8974411 -0.66017282

> # phi coefficient from Yule-Walker
> print(phi.yw)
```

```
          [,1]        [,2]
[1,] 0.5277361 -0.02835303
[2,] 0.1656822  0.37065003

> # 2-step ahead forecasting
> psi.array <- array(0,c(1,2,2))
> psi.array[,,1] <- c(0,0)
> psi.array[,,2] <- c(1,0)
> acf.array <- x.acf[,,1:3]
> theta <- rep(0,4)
> var1.fit.2step.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,method="BFGS")
> var1.fit.2step.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,method="BFGS")
> # fits using both delta values, compared
> print(c(var1.fit.2step.Pos$value,var1.fit.2step.Neg$value))

[1] 2.152178 2.154095

> print(lpp.var1(yw.psidelta[[1]],psi.array,acf.array,yw.psidelta[[2]]))

         [,1]
[1,] 2.333771

> # phi coefficient from 2-step LPP, delta = 1
> print(var1.psi2par(var1.fit.2step.Pos$par,1))

            [,1]        [,2]
[1,] -0.03188988 -0.02238268
[2,]  1.38085694 -0.46507842

> # phi coefficient from 2-step LPP, delta = -1
> print(var1.psi2par(var1.fit.2step.Neg$par,-1))

            [,1]        [,2]
[1,] -0.03202751 0.002372358
[2,]  0.10492919 0.691091355

> # phi coefficient from Yule-Walker
> print(phi.yw)

          [,1]        [,2]
[1,] 0.5277361 -0.02835303
[2,] 0.1656822  0.37065003
```

```
> # low-pass LPP
> mu <- pi/24
> psi.array <- array(c(1,0) %x% sin(seq(1,T-1)*mu)/(pi*seq(1,T-1)),c(1,T-1,2))
> psi.array <- aperm(psi.array,c(1,3,2))
> acf.array <- x.acf
> theta <- rep(0,4)
> var1.fit.bk.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,method="BFGS")
> var1.fit.bk.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,method="BFGS")
> # fits using both delta values, compared
> print(c(var1.fit.bk.Pos$value,var1.fit.bk.Neg$value))

[1] 0.06285471 0.06285468

> lpp.var1(yw.psidelta[[1]],psi.array,acf.array,yw.psidelta[[2]])

           [,1]
[1,] 0.06687517

> # phi coefficient from low-pass LPP, delta = 1
> print(var1.psi2par(var1.fit.bk.Pos$par,1))

           [,1]      [,2]
[1,]  0.2798003 0.1483641
[2,] -0.4755813 0.6170460

> # phi coefficient from low-pass LPP, delta = -1
> print(var1.psi2par(var1.fit.bk.Neg$par,-1))

           [,1]      [,2]
[1,] -0.3066946 0.3377563
[2,]  1.0903832 0.1156086

> # phi coefficient from Yule-Walker
> print(phi.yw)

          [,1]        [,2]
[1,] 0.5277361 -0.02835303
[2,] 0.1656822  0.37065003
```

Exercise 3 Incorrect VAR(1) Forecasting. Simulate a sample of size $T = 1000$ from a bivariate Vector Moving Average process of order two, or VMA(2), given by

$$X_t = \epsilon_t + \Theta_1 \, \epsilon_{t-1} + \Theta_2 \, \epsilon_{t-2},$$

where

$$\Theta_1 = \begin{bmatrix} 9/10 & 2 \\ 0 & 4/5 \end{bmatrix}$$

$$\Theta_2 = \begin{bmatrix} 4/5 & 7/2 \\ 1/5 & 3/5 \end{bmatrix}$$

and Σ equals the identity. Use the VAR(1) LPP criterion of Exercise 1 to fit the mis-specified VAR(1) to this process, for the 2-step ahead forecasting LPP. (As usual, the first of the two series is the target.) Then generate 2-step ahead forecasts for the entire sample. Compare the in-sample performance (for the first series) using the LPP result and the Yule-Walker result.

```
> # Simulate a Gaussian VMA(2) of sample size 1000:
> set.seed(1234)
> T <- 1000
> theta1.matrix <- rbind(c(.9,2),c(0,.8))
> theta2.matrix <- rbind(c(.8,3.5),c(0.2,.6))
> innovar.matrix <- diag(2)
> eps.old <- rnorm(2)
> eps.oldd <- rnorm(2)
> x.sim <- NULL
> for(t in 1:T)
+ {
+         eps.next <- rnorm(2)
+         x.next <- eps.next + theta1.matrix %*% eps.old +
+                 theta2.matrix %*% eps.oldd
+         eps.oldd <- eps.old
+         eps.old <- eps.next
+         x.sim <- cbind(x.sim,x.next)
+ }
> x.sim <- ts(t(x.sim))
> x.acf <- acf(x.sim,type="covariance",plot=FALSE,lag.max=T)[[1]]
> x.acf <- aperm(aperm(x.acf,c(3,2,1)),c(2,1,3))
> # Yule-Walker
> phi.yw <- x.acf[,,2] %*% solve(x.acf[,,1])        # phi coefficient from YW
> yw.psidelta <- var1.par2psi(phi.yw,100)
> # 1-step ahead forecasting
> psi.array <- array(0,c(1,2,1))
> psi.array[,,1] <- c(1,0)
> acf.array <- x.acf[,,1:3]
> theta <- rep(0,4)
> var1.fit.2step.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,lower = c(-Inf,-30,-30,-Inf),
```

```
+         upper = c(Inf,20,20,Inf),method="L-BFGS-B")
> var1.fit.2step.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,lower = c(-Inf,-30,-30,-Inf),
+         upper = c(Inf,20,20,Inf),method="L-BFGS-B")
> # fits using both delta values, compared
> print(c(var1.fit.2step.Pos$value,var1.fit.2step.Neg$value))

[1] 7.365224 7.365224

> print(lpp.var1(yw.psidelta[[1]],psi.array,acf.array,yw.psidelta[[2]]))

         [,1]
[1,] 7.365224

> # phi coefficient from 1-step LPP, delta = 1
> print(var1.psi2par(var1.fit.2step.Pos$par,1))

            [,1]       [,2]
[1,] -0.03871456  2.4744944
[2,] -0.17378232 -0.5289425

> # phi coefficient from 1-step LPP, delta = -1
> print(var1.psi2par(var1.fit.2step.Neg$par,-1))

            [,1]       [,2]
[1,] -0.03871371  2.4744914
[2,]  0.18446156 -0.1003886

> # phi coefficient from Yule-Walker
> print(phi.yw)

            [,1]      [,2]
[1,] -0.03871539 2.4744961
[2,] -0.09029793 0.7870244

> # 2-step ahead forecasting
> psi.array <- array(0,c(1,2,2))
> psi.array[,,1] <- c(0,0)
> psi.array[,,2] <- c(1,0)
> acf.array <- x.acf[,,1:3]
> theta <- rep(0,4)
> var1.fit.2step.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,lower = c(-Inf,-30,-30,-Inf),
+         upper = c(Inf,20,20,Inf),method="L-BFGS-B")
> var1.fit.2step.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,lower = c(-Inf,-30,-30,-Inf),
```

```
+         upper = c(Inf,20,20,Inf),method="L-BFGS-B")
> # fits using both delta values, compared
> print(c(var1.fit.2step.Pos$value,var1.fit.2step.Neg$value))

[1] 9.647333 9.647333

> print(lpp.var1(yw.psidelta[[1]],psi.array,acf.array,yw.psidelta[[2]]))

         [,1]
[1,] 10.67907

> # phi coefficient from 1-step LPP, delta = 1
> print(var1.psi2par(var1.fit.2step.Pos$par,1))

           [,1]     [,2]
[1,]  0.0544628 1.907806
[2,] -0.2652771 1.365024

> # phi coefficient from 1-step LPP, delta = -1
> print(var1.psi2par(var1.fit.2step.Neg$par,-1))

          [,1]     [,2]
[1,] -6.288798 16.98512
[2,] -2.358070  6.44824

> # phi coefficient from Yule-Walker
> print(phi.yw)

            [,1]      [,2]
[1,] -0.03871539 2.4744961
[2,] -0.09029793 0.7870244

> # select choices of phi from above
> if(var1.fit.2step.Pos$value < var1.fit.2step.Neg$value) {
+         phi.lpp <- var1.psi2par(var1.fit.2step.Pos$par,1)
+ } else {
+         phi.lpp <- var1.psi2par(var1.fit.2step.Neg$par,-1)
+ }
> # LPP case
> fore.lpp <- phi.lpp %^% 2
> fore.lpp <- fore.lpp[1,]
> fore.casts <- fore.lpp %*% t(x.sim[1:(T-2),])
> #plot(ts(as.vector(x.sim[3:T,1])))
> #lines(ts(as.vector(fore.casts)),col=2)
> sum((x.sim[3:T,1] - fore.casts)^2)/(T-2)
```

```
[1] 9.598346

> # YW case
> fore.yw <- phi.yw %^% 2
> fore.yw <- fore.yw[1,]
> fore.casts <- fore.yw %*% t(x.sim[1:(T-2),])
> #plot(ts(as.vector(x.sim[3:T,1])))
> #lines(ts(as.vector(fore.casts)),col=2)
> sum((x.sim[3:T,1] - fore.casts)^2)/(T-2)

[1] 10.64258
```

Chapter 3

Introduction to the Multivariate Direct Filter Analysis

Chapter 2 introduced the LPP and its ideal solution. This chapter extends the discussion, by determining optimal solutions without restricting to parametric models.

3.1 Background on Multivariate Filtering

Recall the spectral representation of $\{X_t\}$ via (2.2). Because there are n series in the orthogonal increments process $\mathcal{Z}$, we have

$$X_{t,j} = \int_{-\pi}^{\pi} e^{i\omega t}\, \mathcal{Z}_j(d\omega)$$

for each $1 \leq j \leq n$, and hence for a scalar target $\{Y_t\}$ we have

$$Y_t = \sum_{j=1}^{n} \int_{-\pi}^{\pi} e^{i\omega t}\, \Psi_{1j}(e^{-i\omega})\, \mathcal{Z}_j(d\omega). \tag{3.1}$$

Each of the functions $\Psi_{1j}(e^{-i\omega})$ is complex scalar-valued, and can be decomposed in terms of its gain and phase functions. We here provide some background on these functions, because they provide an interpretation of the action of the linear filter on the input time series.

Any complex number ζ is decomposed in terms of its real $\Re\zeta$ and imaginary $\Im\zeta$ parts:

$$\zeta = \Re\zeta + i\,\Im\zeta.$$

The *magnitude* of ζ is defined via

$$|\zeta| = \sqrt{\Re\zeta^2 + \Im\zeta^2}.$$

If this is positive, then $\zeta/|\zeta|$ is a complex number with unit modulus, and hence can be represented as $\exp\{-i\,\mathrm{Arg}\zeta\}$ for some angle in $[0, 2\pi]$ known as $\mathrm{Arg}\zeta$, or the *angular portion* or ζ. It follows that

$$\zeta = |\zeta|\, \exp\{-i\,\mathrm{Arg}\zeta\},$$

which is known as the Polar decomposition of ζ. Sometimes it is of interest to use a negative Polar decomposition based upon the negative magnitude, which can still be written in terms of $\text{Arg}\zeta$ via

$$\zeta = -|\zeta| \exp\{-i\,[\pi + \text{Arg}\zeta]\},$$

using $e^{-i\pi} = -1$. The angular portion of ζ can be directly computed from the real and imaginary parts of ζ via

$$\text{Arg}\zeta = \arctan\left(\frac{-\Im\zeta}{\Re\zeta}\right).$$

Now when ζ is a function of $\omega \in [-\pi, \pi]$, then the magnitude and angular portions also become functions of ω. In particular, a scalar frequency response function has a magnitude function (called the gain function) and angular function (called the phase function). In the case of some scalar filter $\Psi(B)$ (e.g., the component filter $\Psi_{1j}(B)$) we obtain

$$\Psi(e^{-i\omega}) = |\Psi(e^{-i\omega})| \exp\{-i\,\text{Arg}\Psi(e^{-i\omega})\}.$$

At $\omega = 0$, we know the frequency response function is $\Psi(1) = \sum_{\ell\in\mathbb{Z}} \psi(\ell)$, which is real; hence the phase function at $\omega = 0$ must be an integer multiple of π. It is advantageous to ensure the phase function takes the value zero, as this will facilitate the definition of the phase delay function discussed below. We can ensure this condition by allowing the gain function to be signed. Denoting these by $A(\omega)$ (for amplitude, or signed gain) and $\Phi(\omega)$ (for continuous phase), we have

$$\Psi(e^{-i\omega}) = A(\omega) \exp\{-i\,\Phi(\omega)\}. \tag{3.2}$$

There may be frequencies ω for which the frf equals zero; because the real and imaginary parts of zero are both zero, there is an indeterminancy to the angular portion. However, because the frf is continuous in ω (which follows from summability of the coefficients) we should define the phase function at the zeroes such that it is continuous. By adjusting the angular portion by an integer multiple of π, we can ensure that it will be a continuous function of ω; this adjustment can be compensated by inserting a sign change in the gain function. In this way, the signed gain and continuous phase functions can be computed: both A and Φ will be continuous functions of ω, and $\Phi(0) = 0$ as well. Substituting (3.2) into the spectral representation (3.1) of the target, where A_j and Φ_j are the gain and phase functions of $\Psi_{1j}(e^{-i\omega})$, yields

$$Y_t = \sum_{j=1}^{n} \int_{-\pi}^{\pi} e^{i\omega\,[t-\omega^{-1}\,\Phi_j(\omega)]}\, A_j(\omega)\, \mathcal{Z}_j(d\omega).$$

This representation is interpreted as follows: the target is the sum of n filtered series, where each orthogonal increments process $\mathcal{Z}_j$ has been dilated by the signed gain function A_j, and the timing of the sinusoidal component $e^{i\omega t}$ has been delayed by $\omega^{-1}\,\Phi_j(\omega)$. This quantity, called the *phase delay function*, is well-defined in a neighborhood of zero, as seen in the following result.

Proposition 3 *If the scalar filter $\Psi(B)$ satisfies $\sum_{\ell\in\mathbb{Z}} \ell\,\psi(\ell) < \infty$ and the phase function is continuously defined, then the phase delay function*

$$\phi(\omega) = \frac{\Phi(\omega)}{\omega}$$

is well-defined for $\omega \in [-\pi, \pi]$, *and*

$$\phi(0) = \dot{\Phi}(0) = \frac{\sum_{\ell \in \mathbb{Z}} \ell\, \psi(\ell)}{\sum_{\ell \in \mathbb{Z}} \psi(\ell)}.$$

Proof of Proposition 3. First note that the signed gain function is even, and hence $\dot{A}(0) = 0$. Differentiating (3.2) and evaluating at zero yields

$$-i \sum_{\ell \in \mathbb{Z}} \ell\, \psi(\ell) = \frac{\partial}{\partial \omega} \Psi(e^{-i\omega})|_{\omega=0} = \dot{A}(0)\, e^{-i\,\Phi(0)} + A(0)\, e^{-i\,\Phi(0)}\, (-i\,\dot{\Phi}(0)).$$

Using $\dot{A}(0) = 0$, $\Phi(0) = 0$, and $A(0) = \Psi(1)$ yields

$$\dot{\Phi}(0) = \frac{\sum_{\ell \in \mathbb{Z}} \ell\, \psi(\ell)}{\sum_{\ell \in \mathbb{Z}} \psi(\ell)}. \quad \square$$

The amplitude effects can be understood as dilations of the input spectral densities. If the spectral density matrix of the input process is denoted F, then F_{jj} is the spectral density of the jth component input series; its contribution to the target output involves the increment

$$A_j(\omega)\, \mathcal{Z}_j(d\omega),$$

and the associated spectral density is

$$|A_j(\omega)|^2\, F_{jj}(\omega).$$

There are approximate empirical versions of these relations, which can be described in terms of the DFT. Applying the definition (2.3) to the scalar output $\{Y_t\}$, and utilizing (3.1), we obtain

$$\widetilde{Y}(\xi) = \int_{-\pi}^{\pi} T^{-1/2} \sum_{t=1}^{T} e^{i(\omega-\xi)t}\, \Psi(e^{-i\omega})\, \mathcal{Z}(d\omega).$$

Note that the summation is bounded as $T \to \infty$ unless $\omega = \xi$; it can be shown that the variance of the difference between $\widetilde{Y}(\xi)$ and $\Psi(e^{-i\xi})\, \widetilde{X}(\xi)$ tends to zero, so that we have the approximate result

$$\widetilde{Y}(\omega) \approx \Psi(e^{-i\omega})\, \widetilde{X}(\omega). \tag{3.3}$$

Utilizing (2.4), we obtain an approximate relation of periodograms:

$$\widehat{F}_Y(\omega) \approx \Psi(e^{-i\omega})\, \widehat{F}_X(\omega)\, \Psi(e^{i\omega})'. \tag{3.4}$$

3.2 Multivariate Direct Filter Analysis of the LPP

We can now discuss a more general solution to the LPP. One perspective on Proposition 1 is that it provides a particular class of concurrent filters that arise from specified models. However, so long as these models are mis-specified, the resulting concurrent filters will be sub-optimal. Therefore, it may be possible to improve performance by utilizing broader classes of concurrent filters that are not derived from a particular model. The Direct Filter Analysis (DFA) seeks a concurrent filter

$\widehat{\Psi}(B)$ that optimizes the MSE in a given LPP. While DFA was originated to handle univariate time series, its multivariate generalization – Multivariate Direct Filter Analysis (MDFA) – is designed for the broader context of LPPs discussed in Chapter 2.

The entire class of concurrent filters corresponds to the collection of power series in L. Here we are interested in scalar targets given n input series, so the coefficient matrices of the concurrent filters are $1 \times n$. We may be interested in some subcollection $\mathcal{G}$ of all concurrent filters. For instance, $\mathcal{G}$ could be the optimal solutions to an LPP for a particular process, i.e., consist of all $\widehat{\Psi}(L)$ given in (2.8) for a particular $\Psi(L)$ and $\Theta(L)$. Or we might consider much broader classes of filters, that are described in terms of the rate of decay of the filter coefficients, e.g.,

$$\mathcal{G} = \{\Upsilon(L) : \Upsilon(e^{-i\omega}) \text{ is twice continuously differentiable at } \omega = 0\}.$$

Alternatively, $\mathcal{G}$ might consist of all VARMA filters of a particular AR and MA order, or might consist of all Zero-Pole Combination (ZPC) filters of a given specification (Wildi, 2008). The original univariate DFA of Wildi (2008) approached the LPP with $\mathcal{G}$ consisting of appropriately restricted ZPC filters.

For now, we shall suppose that the concurrent filters of $\mathcal{G}$ belong to some parametric family described by a parameter ϑ belonging to a parameter manifold. Because we seek elements of $\mathcal{G}$ that will solve an LPP, i.e., be a good concurrent approximation to $\Psi(L)$, we use the notation

$$\mathcal{G} = \{\widehat{\Psi}_\vartheta(L) : \vartheta \text{ belongs to a parameter space}\}. \tag{3.5}$$

Whereas the model-based approach to the LPP discussed in Chapter 2 involves minimizing a particular parametric form of the filter error MSE – namely the function $J_\Psi(\vartheta, G)$ for G corresponding either to the periodogram or true spectrum – a more direct approach is to minimize a general expression for the filter error MSE over a given set $\mathcal{G}$. The real-time estimation error is given in (2.7), which has mean zero and variance

$$\mathbb{E}[E_t^2] = \langle \Big[\Psi(z) - \widehat{\Psi}_\vartheta(z)\Big]\ \widetilde{F}\ \Big[\Psi(z) - \widehat{\Psi}_\vartheta(z)\Big]^* \rangle_0. \tag{3.6}$$

This suggests the criterion function $D_\Psi(\vartheta, G)$ for any Hermitian function G, defined as

$$D_\Psi(\vartheta, G) = \langle \Big[\Psi(z) - \widehat{\Psi}_\vartheta(z)\Big]\ G\ \Big[\Psi(z) - \widehat{\Psi}_\vartheta(z)\Big]^* \rangle_0. \tag{3.7}$$

This is the MDFA criterion function. An equivalent formula to (3.7) that can be useful for calculations is

$$D_\Psi(\vartheta, G) = \mathrm{tr}\{\langle G\, M_\vartheta \rangle_0\} \qquad M_\vartheta(z) = \Big[\Psi(z) - \widehat{\Psi}_\vartheta(z)\Big]^* \Big[\Psi(z) - \widehat{\Psi}_\vartheta(z)\Big]. \tag{3.8}$$

Given a filter class $\mathcal{G}$, the best possible concurrent filter is given by $\widehat{\Psi}_{\vartheta(\widetilde{F})}$, where $\vartheta(\widetilde{F})$ is a minimizer of $D_\Psi(\vartheta, \widetilde{F})$. This $\vartheta(\widetilde{F})$ is the PTV for the filter parameter, in analogy with the terminology for model parameters. Clearly, if the set $\mathcal{G}$ is rendered sufficiently large to include the optimal concurrent filter for that particular LPP and process – as given in Proposition 1 – then there exists some $\widetilde{\vartheta}$ such that $\widehat{\Psi}_{\widetilde{\vartheta}}$ is identical with the optimal filter. However, if $\mathcal{G}$ is smaller, then the PTV $\vartheta(\widetilde{F})$ is as close as possible according to D_Ψ discrepancy to the optimal filter.

A case of interest arises from taking a very broad class $\mathcal{G}$: let $\mathcal{G}$ consist of all length q concurrent filters, with

$$\vartheta' = \left[\widehat{\psi}(0), \widehat{\psi}(1), \ldots, \widehat{\psi}(q-1)\right]. \tag{3.9}$$

So ϑ is a column vector of length qn. Then the criterion (3.7) can be rewritten as

$$D_\Psi(\vartheta, G) = \vartheta' \, B \, \vartheta - \vartheta' \, b - b' \, \vartheta + \langle \Psi(z) \, G \, \Psi(z)^* \rangle_0, \tag{3.10}$$

where

$$b' = \left[\langle \Psi(z) \, G \rangle_0, \langle \Psi(z) \, G \rangle_1, \ldots, \langle \Psi(z) \, G \rangle_{q-1}\right], \tag{3.11}$$

and B is a block matrix, where the jkth $n \times n$ block of is $\langle G \rangle_{k-j}$ for $1 \le j, k \le q$.

Proposition 4 *The minimizer of the MDFA criterion (3.7), given that $\mathcal{G}$ consists of all length q concurrent filters, is*

$$\vartheta = B^{-1} \, b,$$

where the jkth block of B is $\langle G \rangle_{k-j}$, and b is given by (3.11). The minimal value is

$$\langle \Psi(z) \, G \, \Psi(z)^* \rangle_0 - b' \, B^{-1} \, b. \tag{3.12}$$

Proof of Proposition 4. First note that the typical component of b has the form

$$\langle \Psi(z) \, G \rangle_\ell = \sum_{k \in \mathbb{Z}} \psi(k) \, \langle G \rangle_{\ell - k} \tag{3.13}$$

for $0 \le \ell < q$, which shows that b is real-valued. The objective function is a quadratic in ϑ, and therefore the minimizer is obtained by computing the gradient and Hessian, which are $-2b + 2B\,\vartheta$ and $2B$ respectively, yielding the solution. Plugging back into D_Ψ yields (3.12). □

Remark 3 To implement Proposition 4 in practice, G is given by the periodogram so that $\langle G \rangle_h = \widehat{\Gamma}(h)$. It is necessary to compute b, given by (3.11), and we can proceed by approximating the integrals over a Riemann mesh corresponding to Fourier frequencies; this is discussed further below.

This broad class $\mathcal{G}$ of filters will furnish concurrent filters that closely approximate those of Proposition 1 as $q \to \infty$.

Example 8 Multi-step Ahead Forecasting. Suppose we consider the one-step ahead forecasting of stationary time series and $\mathcal{G}$ corresponds to all VMA filters of order q (i.e., the filter corresponds to a VMA$(q-1)$ polynomial), where

$$\vartheta = \mathrm{vec}[\widehat{\psi}(0)', \widehat{\psi}(1)', \ldots, \widehat{\psi}(q-1)'].$$

With $\Psi(L) = L^{-1}$ from (3.7) we have

$$\begin{aligned} D_\Psi(\vartheta, G) &= \langle \left[z^{-1} e_1' - \widehat{\Psi}_\vartheta(z)\right] \, G \left[z^{-1} e_1' - \widehat{\Psi}_\vartheta(z)\right]^* \rangle_0 \\ &= \langle \left[e_1' - \sum_{\ell=0}^{q-1} \widehat{\psi}(\ell) \, z^{\ell+1}\right] \, G \left[e_1' - \sum_{\ell=0}^{q-1} \widehat{\psi}(\ell) \, z^{\ell+1}\right]^* \rangle_0 \\ &= \langle G \rangle_0 - 2 \, \vartheta' \, \langle G \rangle_{1:q} \, e_1 + \vartheta' \, \langle G \rangle_{0:(q-1),0:(q-1)} \, \vartheta. \end{aligned}$$

Hence the optimizer is

$$\vartheta(G) = \langle G \rangle_{0:(q-1),0:(q-1)}^{-1} \langle G \rangle_{1:q} e_1,$$

which is the first component of the solution to the Yule-Walker system of order q determined by G. Therefore the MDFA solution is the same as the fit of a VAR(q) using Proposition 1.

The empirical problem is solved by minimizing $D_\Psi(\vartheta, \widehat{F})$, yielding the estimator $\vartheta(\widehat{F})$. The empirical criterion can be simply computed using (3.8) and (2.6), namely

$$D_\Psi(\vartheta, \widehat{F}) = \sum_{|h|<T} \mathrm{tr}\{\widehat{\Gamma}(h) \langle M_\vartheta \rangle_{-h}\}.$$

Filtering with $\widehat{\Psi}_{\vartheta(\widehat{F})}$ instead of $\widehat{\Psi}_{\vartheta(\widetilde{F})}$ involves some statistical error, which vanishes as $T \to \infty$ because $\vartheta(\widehat{F})$ is consistent for the PTV. We can quantify this additional error if we know the statistical properties of the estimate; under fairly broad conditions, it follows a central limit theorem. As in Chapter 2, we assume the HT conditions and that the Hessian $H(\vartheta) = \nabla\nabla' D_\Psi(\vartheta, \widetilde{F})$ of D_Ψ is positive definite at the PTV. The function M is defined in (3.8).

Theorem 2 *Suppose that $\vartheta(\widetilde{f})$ exists uniquely in the interior of the filter parameter space, and that $H(\vartheta(\widetilde{F}))$ is positive definite. Suppose that $\{X_t\}$ has finite fourth moments, conditions (HT1)-(HT6) of Taniguchi and Kakizawa (2000, pp.55-56) hold, and that the fourth order cumulant function of $\{X_t\}$ is zero. Then the estimator is consistent for the PTV, and*

$$\sqrt{T}\left(\vartheta(\widehat{F}) - \vartheta(\widetilde{F})\right) \xRightarrow{\mathcal{L}} \mathcal{N}\left(0, H(\vartheta(\widetilde{F}))^{-1} V(\vartheta(\widetilde{F})) H(\vartheta(\widetilde{F}))^{-1}\right)$$

as $T \to \infty$, where

$$V_{jk}(\vartheta) = tr\{\langle \partial_j M_\vartheta(z) \widetilde{F} \partial_k M_\vartheta(z) \widetilde{F} \rangle_0\}.$$

Proof of Theorem 2. This is proved in the same way as Theorem 1.

We designate the resulting prediction function $\widehat{\Psi}_{\widehat{\vartheta}}$ as a *Linear Prediction Filter* (LPF).

Illustration 3 VAR(1). Again consider a VAR(1) process, and suppose we wish to use MDFA to approximate the optimal LPP solution – even though we don't know the true dynamics. Let $\mathcal{G}$ denote the set of moving average filters of length q, and G is the spectral density of the VAR(1); the solution given by Proposition 4 can be compared to that of the LPP, which has the first q components given by

$$\varphi' = [\psi(0) + A_\Psi(\Phi), \psi(1), \ldots, \psi(q-1)].$$

This is an approximate solution to the system $\vartheta' B = b'$, because $\varphi' B$ has $j+1$th component, for $0 \le j \le q-1$, equal to

$$\sum_{\ell=0}^{q-1} \psi(\ell) \langle G \rangle_{j-\ell} + A_\Psi(\Phi) \Gamma(j).$$

Noting that

$$A_\Psi(\Phi) \Gamma(j) = \sum_{\ell<0} \psi(-\ell) \Phi^{-\ell} \Gamma(j) = \sum_{\ell<0} \psi(-\ell) \Gamma(j-\ell),$$

because for a VAR(1) process $\Gamma(h) = \Phi^h \Gamma(0)$ when $h \geq 0$, we see that component $j+1$ of $\varphi' B$ is

$$\sum_{\ell \leq q-1} \psi(\ell)\, \Gamma(j-\ell) = [\Re b']_{j+1} - \sum_{\ell \geq q} \psi(\ell)\, \Gamma(j-\ell).$$

As $q \to \infty$ the error term vanishes (for each j), indicating that $\varphi' B \approx b'$, or $\vartheta \approx \varphi$.

3.3 Computation of the Linear Prediction Filter

Here we discuss the calculation of the quantities B and b appearing in Proposition 4. In the case that G corresponds to the periodogram only a finite number of sample autocovariances are non-zero, and (3.13) simplifies. More generally, suppose that for some $r > 0$ we have $\langle G \rangle_h = 0$ for all $|h| \geq r$. Then (3.13) can be written in matrix form as

$$b' = [\psi(1-r), \ldots, \psi(0), \ldots, \psi(r-1), \ldots, \psi(r+q-2)] \cdot \begin{bmatrix} \langle G \rangle_{r-1} & \ddots & 0 \\ \vdots & \ddots & \vdots \\ \langle G \rangle_0 & \vdots & \langle G \rangle_{q-1} \\ \vdots & \ddots & \vdots \\ \langle G \rangle_{1-r} & \vdots & \langle G \rangle_{q-r} \\ 0 & \ddots & \vdots \\ 0 & \ddots & \langle G \rangle_{1-q} \end{bmatrix}.$$

The matrix in this product has $2r + q - 2$ block rows, and q block columns. The dimension of B is similarly $rn \times rn$. This formulation requires a time-domain specification of the target filter, whereas in applications it is often more convenient to utilize the frequency-domain. To that end, we discuss the computation of b and B via discretization of the appropriate integrals over a Riemann mesh corresponding to Fourier frequencies. We let square brackets denote the floor of a real number.

Definition 3 Given integer T, the Fourier frequencies are a set of T numbers in $[-\pi, \pi]$ of the form $\omega_j = 2\pi j/T$ for $-[T/2] \leq j \leq [T/2]$ (when T is odd) and $-[T/2] \leq j \leq [T/2] - 1$ (when T is even).

Remark 4 In the case that T is odd, there exists m such that $T = 2m + 1$, and in this case $-m \leq j \leq m$. In the case that T is even, then $T = 2m$ for some m, and $-m \leq j \leq m-1$. Clearly, $m = [T/2]$ in either case.

The Fourier frequencies form the basis for a transformation of the time-domain sample $\underline{X}$ to the frequency-domain, known as the DFT, c.f., (2.3). By restricting the DFT to Fourier frequencies, we obtain a linear transformation from the $T \times n$ matrix of the sample to a $T \times n$ matrix of DFTs. To show this result, let

$$\mathcal{X} = [X_1, X_2, \ldots, X_T],$$

so that $\text{vec}[\mathcal{X}] = \underline{X}$. Similarly, denote the matrix of DFTs by $\widetilde{\mathcal{X}}$, with jth column ($1 \le j \le T$) given by $\widetilde{X}(\omega_{j-[T/2]-1})$. In this way, the matrix of DFTs begins with $\widetilde{X}(\omega_{-[T/2]})$ in the first column, and proceeds to $\widetilde{X}(\omega_{T-[T/2]-1})$, with frequency corresponding to either $[T/2]$ or $[T/2]-1$ depending on whether T is odd or even. Letting C denote the $T \times T$ linear transformation such that $\widetilde{\mathcal{X}}' = C\,\mathcal{X}'$, we see that

$$C_{jt} = T^{-1/2} \exp\{-i\,2\pi t\,(j - [T/2] - 1)/T\},$$

for $1 \le j, t \le T$. This follows directly from (2.3). Moreover, the original sample can be recovered from the DFT matrix by applying C^{-1}, which equals the conjugate transpose.

Proposition 5 *The DFT matrix C is unitary, i.e., $C^{-1} = \overline{C}'$.*

Proof of Proposition 5.

$$\begin{aligned}\left[\overline{C}'\,C\right]_{jk} &= T^{-1} \sum_{t=1}^{T} \exp\{i\,2\pi j\,(t - [T/2] - 1)/T\} \exp\{-i\,2\pi k\,(t - [T/2] - 1)/T\} \\ &= \left(T^{-1} \sum_{t=1}^{T} \exp\{i\,2\pi (t[j-k])/T\} \right) \exp\{i\,2\pi (k-j)\,([T/2]+1)/T\},\end{aligned}$$

and the expression in parentheses equals zero unless $k = j$, in which case it equals one (this is easily verified by using the formula for the partial summation of a geometric series). Hence the jkth element of $\overline{C}'\,C$ corresponds to the $jk$the element of the identity matrix. □

To compute the quantities given in Proposition 4, and more generally to compute the MDFA criterion (3.7), we propose to approximate each integral by an average over Fourier frequencies. Although finer meshes could clearly be implemented, the Fourier frequency mesh is sufficient for statistical purposes – this is because when considering the asymptotic properties of linear functionals of the periodogram (i.e., weighted linear combinations of periodogram ordinates), there is no difference between averaging over Fourier frequencies or integrating over every frequency. Moreover, using the Fourier frequencies produces an empirical criterion function that is a closer approximation to the sample mean squared error, which is shown by the following heuristic arguments. Recalling that the real-time filter error $E_t = Y_t - \widehat{Y}_t$ has variance given by (3.6), the sample variance is

$$T^{-1} \sum_{t=1}^{T} E_t^2 = T^{-1} \sum_{j=1}^{T} \widehat{F}_E(\omega_{j-[T/2]-1}),$$

where $\widehat{F}_E$ is the periodogram of the filter errors. This equality is a discrete version of the Plancherel identity; the right hand side is approximated by

$$T^{-1} \sum_{j=1}^{T} \left[\Psi - \widehat{\Psi}\right](\omega_{j-[T/2]-1})\, \widehat{F}_X(\omega_{j-[T/2]-1}) \left[\Psi - \widehat{\Psi}\right]'(-\omega_{j-[T/2]-1}),$$

using (3.3). This is exactly the MDFA criterion (3.7) with the integrals replaced by Riemann sums over the Fourier frequencies, and G replaced by the periodogram.

With this justification, we see that the entries of the matrix B in Proposition 4 are approximately computed via

$$B_{j,k} \approx T^{-1} \sum_{\ell=1}^{T} G(\omega_{\ell-[T/2]-1}) \exp\{i\,(k-j)(\omega_{\ell-[T/2]-1})\}$$

for $1 \le j,k \le T$. Moreover, for $0 \le k \le T-1$

$$b'_k \approx T^{-1} \sum_{\ell=1}^{T} \Psi(\exp\{-i\,\omega_{\ell-[T/2]-1}\})\, G(\omega_{\ell-[T/2]-1}) \exp\{i\,k(\omega_{\ell-[T/2]-1})\},$$

where $b' = [b'_0, \ldots, b'_{T-1}]$. Finally,

$$\langle \Psi(z)\, G\, \Psi(z)^* \rangle_0 \approx T^{-1} \sum_{\ell=1}^{T} \Psi(\exp\{-i\,\omega_{\ell-[T/2]-1}\})\, G(\omega_{\ell-[T/2]-1})\, \Psi'(\exp\{i\,\omega_{\ell-[T/2]-1}\}).$$

Several exercises illustrate the implementation of these formulas, and their applications to filtering problems. Our implementation in `mdfa.filter` is written with a multivariate target in view; by focusing upon the first row, the applications of this chapter can be obtained.

Exercise 4 Correct VAR(1) LPP. This exercise compares LPP and MDFA when the model is correct. Simulate a sample of size $T = 100$ from a bivariate VAR(1) process with

$$\Phi = \begin{bmatrix} 1 & 1/2 \\ -1/5 & 3/10 \end{bmatrix}$$

and Σ equal to the identity. The eigenvalues are $4/5$ and $1/2$. Implement MDFA for this sample, using the moving average filters (Proposition 4) of length $q = 20$ to approximate the optimal LPP filter (Exercise 1), for the 2-step ahead forecasting LPP (Example 1). Does the MDFA concurrent filter closely match the optimal LPP filter? Repeat for $T = 200$ and $T = 500$.

```
> # Simulate a Gaussian VAR(1) of sample size 100:
> set.seed(1234)
> T <- 100
> N <- 2
> phi.matrix <- rbind(c(1,.5),c(-.2,.3))
> innovar.matrix <- diag(N)
> gamma.0 <- matrix(solve(diag(N^2) - phi.matrix %x% phi.matrix) %*%
+         matrix(innovar.matrix,ncol=1),nrow=N)
> x.init <- t(chol(gamma.0)) %*% rnorm(N)
> x.next <- x.init
> x.sim <- NULL
> for(t in 1:T)
+ {
+         x.next <- phi.matrix %*% x.next + t(chol(innovar.matrix)) %*% rnorm(N)
+         x.sim <- cbind(x.sim,x.next)
+ }
```

```
> x.sim <- ts(t(x.sim))
> x.acf <- acf(x.sim,type="covariance",plot=FALSE,lag.max=T)[[1]]
> x.acf <- aperm(aperm(x.acf,c(3,2,1)),c(2,1,3))
> # 2-step ahead forecasting
> psi.array <- array(0,c(1,2,2))
> psi.array[,,1] <- c(0,0)
> psi.array[,,2] <- c(1,0)
> acf.array <- x.acf[,,1:3]
> theta <- rep(0,4)
> var1.fit.2step.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,method="BFGS")
> var1.fit.2step.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,method="BFGS")
> # select choices of phi from above
> if(var1.fit.2step.Pos$value < var1.fit.2step.Neg$value) {
+         phi.lpp <- var1.psi2par(var1.fit.2step.Pos$par,1)
+ } else {
+         phi.lpp <- var1.psi2par(var1.fit.2step.Neg$par,-1)
+ }
> # LPP case
> fore.lpp <- phi.lpp %^% 2
> fore.lpp <- fore.lpp[1,]
> # MDFA
> q <- 20
> Grid <- T
> m <- floor(Grid/2)
> # The Fourier frequencies
> lambda.ft <- exp(-1i*2*pi*Grid^{-1}*(seq(1,Grid) - (m+1)))
> # frf for 2-step ahead forecasting
> frf.psi <- matrix(lambda.ft^{-2},nrow=1) %x% diag(N)
> frf.psi <- array(frf.psi,c(N,N,Grid))
> spec.hat <- mdfa.pergram(x.sim,1)
> fore.mdfa <- mdfa.unconstrained(frf.psi,spec.hat,q)
> # compare LPP and MDFA first coefficients
> print(fore.lpp)

[1] 0.9463929 0.7503688

> print(fore.mdfa[[1]][1,,1])

[1] 0.9064968 0.7911447
```

Exercise 5 Incorrect VAR(1) LPP. This exercies compares LPP and MDFA when the model

is wrong. Simulate a sample of size $T = 100$ from a bivariate VMA(1) with

$$\Theta = \begin{bmatrix} 1 & 0 \\ 2/5 & 2 \end{bmatrix}$$

and Σ equal to the identity. Use the moving average filter MDFA (Proposition 4) with $q = 20$ to find the best concurrent filter, for the 2-step ahead forecasting LPP (Example 1). Compare these results to the VAR(1) LPP filter previously obtained (Exercise 2), based on the mis-specified VAR(1) model. Which filter, LPP or MDFA, more closely approximates the ideal filter? Repeat for $T = 200$ and $T = 500$, and explain your results.

```
> # Simulate a Gaussian VMA(1) of sample size 100:
> set.seed(1234)
> T <- 100
> N <- 2
> theta.matrix <- rbind(c(1,0),c(.4,2))
> innovar.matrix <- diag(N)
> eps.old <- rnorm(N)
> x.sim <- NULL
> for(t in 1:T)
+ {
+         eps.next <- t(chol(innovar.matrix)) %*% rnorm(N)
+         x.next <- eps.next + theta.matrix %*% eps.old
+         eps.old <- eps.next
+         x.sim <- cbind(x.sim,x.next)
+ }
> x.sim <- ts(t(x.sim))
> x.acf <- acf(x.sim,type="covariance",plot=FALSE,lag.max=T)[[1]]
> x.acf <- aperm(aperm(x.acf,c(3,2,1)),c(2,1,3))
> # 2-step ahead forecasting
> psi.array <- array(0,c(1,2,2))
> psi.array[,,1] <- c(0,0)
> psi.array[,,2] <- c(1,0)
> acf.array <- x.acf[,,1:3]
> theta <- rep(0,4)
> var1.fit.2step.Pos <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=1,method="BFGS")
> var1.fit.2step.Neg <- optim(theta,lpp.var1,psi.array=psi.array,
+         acf.array=acf.array,delta=-1,method="BFGS")
> # fits using both delta values, compared
> print(c(var1.fit.2step.Pos$value,var1.fit.2step.Neg$value))

[1] 2.152178 2.154095
```

```
> # select choices of phi from above
> if(var1.fit.2step.Pos$value < var1.fit.2step.Neg$value) {
+         phi.lpp <- var1.psi2par(var1.fit.2step.Pos$par,1)
+ } else {
+         phi.lpp <- var1.psi2par(var1.fit.2step.Neg$par,-1)
+ }
> # LPP case
> fore.lpp <- phi.lpp %^% 2
> fore.lpp <- fore.lpp[1,]
> # MDFA
> q <- 20
> Grid <- T
> m <- floor(Grid/2)
> # The Fourier frequencies
> lambda.ft <- exp(-1i*2*pi*Grid^{-1}*(seq(1,Grid) - (m+1)))
> # frf for 2-step ahead forecasting
> frf.psi <- matrix(lambda.ft^{-2},nrow=1) %x% diag(N)
> frf.psi <- array(frf.psi,c(N,N,Grid))
> spec.hat <- mdfa.pergram(x.sim,1)
> fore.mdfa <- mdfa.unconstrained(frf.psi,spec.hat,q)
> # compare LPP and MDFA first coefficients
> print(fore.lpp)

[1] -0.02989031  0.01112348

> print(fore.mdfa[[1]][1,,1])

[1]  0.04362898 -0.03635081
```

Exercise 6 MDFA VAR(1) Filtering. This exercise examines MDFA applied to the trend of a VAR(1) process. Simulate a sample of size $T = 5000$ from a bivariate VAR(1) process with

$$\Phi = \begin{bmatrix} 1 & 1/2 \\ -1/5 & 3/10 \end{bmatrix}$$

and Σ equal to the identity. The eigenvalues are .8 and .5. Apply the ideal low-pass filter (cf. Example 2) with $\mu = \pi/6$ to the sample (truncate the filter to 1000 coefficients on each side). Use the moving average filter MDFA (Proposition 4) to find the best concurrent filter, setting $q = 12$. Apply this concurrent filter to the simulation, and compare the relevant portions to the ideal trend. Also determine the in-sample performance, in comparison to the criterion value (3.12). Target the trends for both time series.

```
> # Simulate a Gaussian VAR(1) of sample size 2500:
> set.seed(1234)
> T <- 5000
```

```
> N <- 2
> phi.matrix <- rbind(c(1,.5),c(-.2,.3))
> innovar.matrix <- diag(N)
> gamma.0 <- matrix(solve(diag(N^2) - phi.matrix %x% phi.matrix) %*%
+         matrix(innovar.matrix,ncol=1),nrow=N)
> x.init <- t(chol(gamma.0)) %*% rnorm(N)
> x.next <- x.init
> x.sim <- NULL
> for(t in 1:T)
+ {
+         x.next <- phi.matrix %*% x.next + t(chol(innovar.matrix)) %*% rnorm(N)
+         x.sim <- cbind(x.sim,x.next)
+ }
> x.sim <- ts(t(x.sim))
> x.acf <- acf(x.sim,type="covariance",plot=FALSE,lag.max=T)[[1]]
> x.acf <- aperm(aperm(x.acf,c(3,2,1)),c(2,1,3))
> # construct and apply low pass filter
> mu <- pi/6
> len <- 1000
> lp.filter <- c(mu/pi,sin(seq(1,len)*mu)/(pi*seq(1,len)))
> lp.filter <- c(rev(lp.filter),lp.filter[-1])
> x.trend.ideal <- filter(x.sim,lp.filter,method="convolution",sides=2)[(len+1):(T-len),]
> # get MDFA concurrent filter
> q <- 20
> Grid <- T
> m <- floor(Grid/2)
> # The Fourier frequencies
> freq.ft <- 2*pi*Grid^{-1}*(seq(1,Grid) - (m+1))
> # frf for ideal low-pass
> frf.psi <- rep(0,Grid)
> frf.psi[abs(freq.ft) <= mu] <- 1
> frf.psi <- matrix(frf.psi,nrow=1) %x% diag(N)
> frf.psi <- array(frf.psi,c(N,N,Grid))
> spec.hat <- mdfa.pergram(x.sim,1)
> lp.mdfa <- mdfa.unconstrained(frf.psi,spec.hat,q)
> # apply the MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
```

```
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 0.4305693 0.1461517

> # compare to criterion value
> diag(lp.mdfa[[2]])

[1] 0.4297711 0.1376914
```

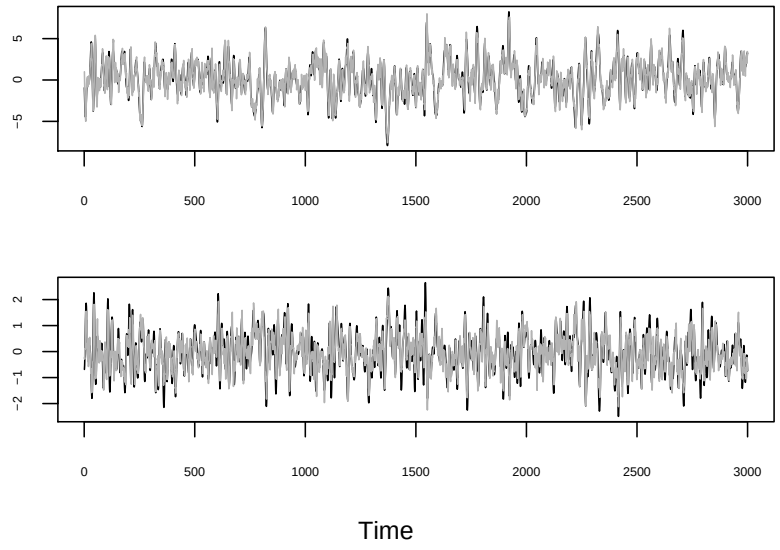

Figure 3.1: Ideal trends (black) for the bivariate VAR(1) with real-time MDFA trends (grey) overlaid, for series one (upper panel) and series two (bottom panel).

Figure 3.1 shows the tracking of the ideal trends by the MDFA real-time trends. The MDFA criterion attempts to find a real-time filter $\widehat{\Psi}$ that is close to the target Ψ at frequencies that are emphasized by spectral content in the time series, which is assessed through the periodogram. This particular optimization concept can be understood by analyzing real-time filter outputs and filter characteristics, i.e., amplitude and phase delay functions.

Exercise 7 MDFA VAR(1) Filtering Characteristics. This exercise examines MDFA applied to the trend of a trivariate VAR(1) process, which is essentially three univariate AR(1) processes. Simulate a sample of size $T = 5000$ from a trivariate VAR(1) process with

$$\Phi = \begin{bmatrix} 9/10 & 0 & 0 \\ 0 & 1/10 & 0 \\ 0 & 0 & -9/10 \end{bmatrix}$$

and Σ equal to the identity. Apply the ideal low-pass filter with $\mu = \pi/6$ to the sample (truncate the filter to 1000 coefficients on each side). Use the moving average filter MDFA (Proposition 4) to find the best concurrent filter, setting $q = 12$. Apply this concurrent filter to the simulation, and compare the relevant portions to the ideal trend. Also determine the in-sample performance, in comparison to the criterion value (3.12). Target the trends for both time series, and compare the results graphically. Finally, compute and graphically compare the amplitude and phase delay functions for each of the three trend targets.

```
> # Simulate a Gaussian VAR(1) of sample size 2500:
> set.seed(1234)
> T <- 5000
> N <- 3
> phi.matrix <- rbind(c(.9,0,0),c(0,.1,0),c(0,0,-.9))
> innovar.matrix <- diag(N)
> gamma.0 <- matrix(solve(diag(N^2) - phi.matrix %x% phi.matrix) %*%
+         matrix(innovar.matrix,ncol=1),nrow=N)
> x.init <- t(chol(gamma.0)) %*% rnorm(N)
> x.next <- x.init
> x.sim <- NULL
> for(t in 1:T)
+ {
+         x.next <- phi.matrix %*% x.next + t(chol(innovar.matrix)) %*% rnorm(N)
+         x.sim <- cbind(x.sim,x.next)
+ }
> x.sim <- ts(t(x.sim))
> x.acf <- acf(x.sim,type="covariance",plot=FALSE,lag.max=T)[[1]]
> x.acf <- aperm(aperm(x.acf,c(3,2,1)),c(2,1,3))
> # construct and apply low pass filter
> mu <- pi/6
> len <- 1000
> lp.filter <- c(mu/pi,sin(seq(1,len)*mu)/(pi*seq(1,len)))
> lp.filter <- c(rev(lp.filter),lp.filter[-1])
> x.trend.ideal <- filter(x.sim,lp.filter,method="convolution",sides=2)[(len+1):(T-len),]
> # get MDFA concurrent filter
> q <- 20
> Grid <- T
> m <- floor(Grid/2)
> # The Fourier frequencies
> freq.ft <- 2*pi*Grid^{-1}*(seq(1,Grid) - (m+1))
> # frf for ideal low-pass
> frf.psi <- rep(0,Grid)
> frf.psi[abs(freq.ft) <= mu] <- 1
```

```
> frf.psi <- matrix(frf.psi,nrow=1) %x% diag(N)
> frf.psi <- array(frf.psi,c(N,N,Grid))
> spec.hat <- mdfa.pergram(x.sim,1)
> lp.mdfa <- mdfa.unconstrained(frf.psi,spec.hat,q)
> # apply the MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa13 <- filter(x.sim[,3],lp.mdfa[[1]][1,3,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa23 <- filter(x.sim[,3],lp.mdfa[[1]][2,3,],method="convolution",sides=1)
> x.trend.mdfa31 <- filter(x.sim[,1],lp.mdfa[[1]][3,1,],method="convolution",sides=1)
> x.trend.mdfa32 <- filter(x.sim[,2],lp.mdfa[[1]][3,2,],method="convolution",sides=1)
> x.trend.mdfa33 <- filter(x.sim[,3],lp.mdfa[[1]][3,3,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12 + x.trend.mdfa13,
+         x.trend.mdfa21 + x.trend.mdfa22 + x.trend.mdfa23,
+         x.trend.mdfa31 + x.trend.mdfa32 + x.trend.mdfa33)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2),
+         mean((x.trend.ideal[,3] - x.trend.mdfa[,3])^2)))

[1] 0.29370453 0.08167332 0.02267826

> # compare to criterion value
> diag(lp.mdfa[[2]])

[1] 0.30165107 0.08364418 0.02316804

> # compute gain and phase delay functions
> frf.psi <- frf.psi[1,1,]
> gain.psi <- abs(frf.psi)
> phased.psi <- Arg(frf.psi)/freq.ft
> lp.frf <- mdfa.frf(lp.mdfa[[1]],0,T)
> lp.gain1 <- abs(lp.frf[1,1,])
> lp.gain2 <- abs(lp.frf[2,2,])
> lp.gain3 <- abs(lp.frf[3,3,])
> lp.phased1 <- -Arg(lp.frf[1,1,])/freq.ft
> lp.phased2 <- -Arg(lp.frf[2,2,])/freq.ft
> lp.phased3 <- -Arg(lp.frf[3,3,])/freq.ft
```

A visual inspection of Figure 3.2, regarding the trends and trend estimates in Exercise 7, indicates an apparent conflict with the criterion values: although the first series has the largest

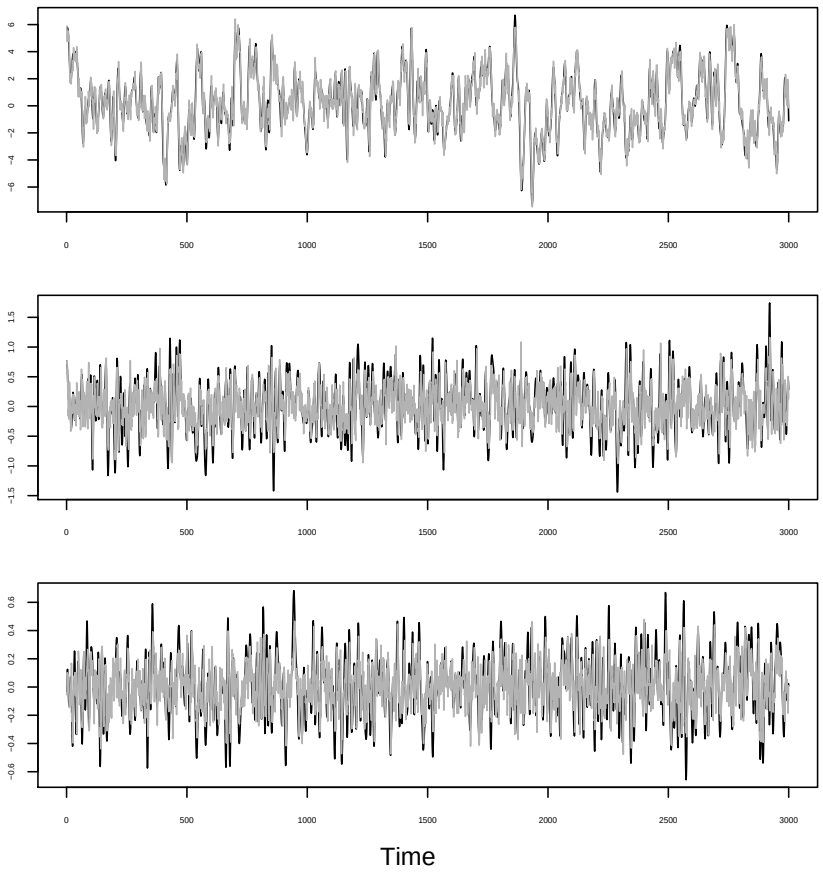

Figure 3.2: Ideal trends (black) for the trivariate VAR(1) with real-time MDFA trends (grey) overlaid, for series one (upper panel), series two (center panel), and series three (bottom panel).

MSE, the fit of the concurrent estimator to the target trend appears best. This is because the task of the filter for the first series is easiest, because a higher degree of smoothness must be captured – whereas, in contrast, a noisy target is harder to replicate in a mean square error sense. Another feature is that the real-time estimates appear to be systematically shifted to the right (they are delayed); the first series seems to be least affected. These observations indicate that the difficulty of the estimation task depends on the DGP (as specified by the entries of Φ): larger eigenvalues correspond to greater persistence of the process, and an easier estimation problem. In contrast, small eigenvalues correspond to a noisier process, and a harder estimation problem.

These properties are further confirmed by the gain and phase delay functions displayed in Figure 3.3. The noise in real-time estimates $\widehat{Y}_t$ is due to the incomplete matching of the estimated gain (orange, green, or violet lines in the upper panel) to the ideal gain function (black); note that the concurrent filters allow some content at frequencies greater than $\mu = \pi/6$ to penetrate. On the other hand, the observed delay in the real-time estimates can be explained through the fact that the phase delay functions of the concurrent filters (orange, green, or violet lines in the center panel) do not vanish, unlike the ideal filter's phase delay (black). Chapter **??** proposes a more

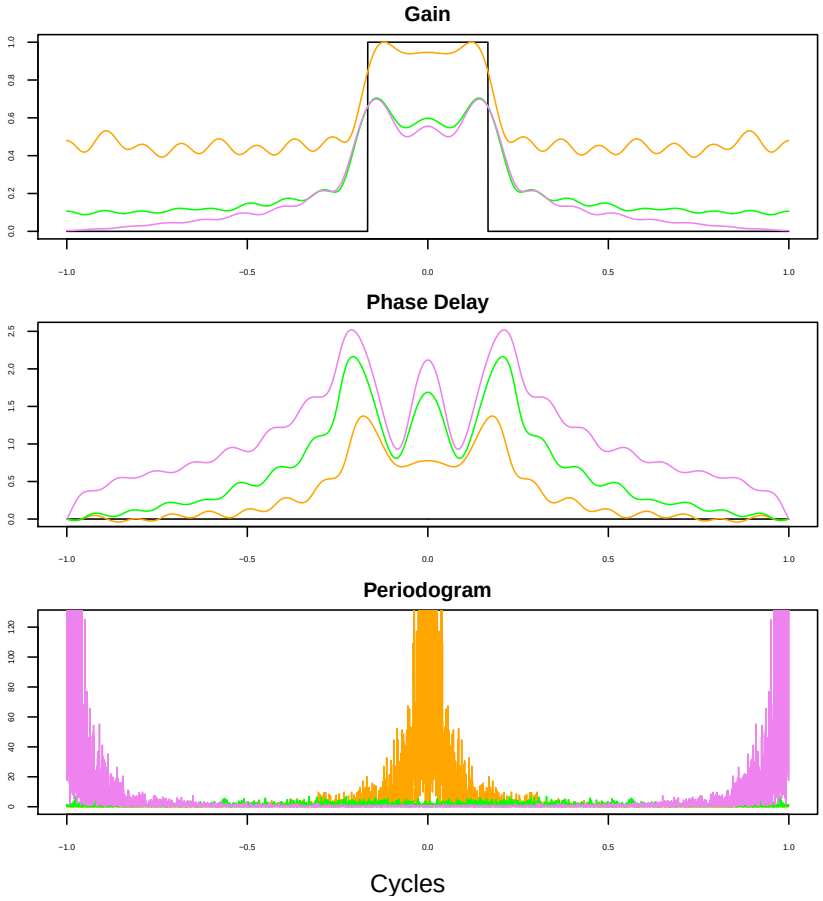

Figure 3.3: Gain functions (upper panel), Phase Delay Functions (center panel), and Periodograms (bottom panel) for series one (orange), two (green), and three (violet).

general optimization paradigm that will address these issues explicitly.

Also observe that the phase delay function of the first series (orange line, center panel), which has the strongest autocorrelation, remains comparatively small. Its gain function (orange line, upper panel) is the farthest away from the target in the stop-band $|\omega| > \pi/6$, but most closely resembles the target in the pass-band $|\omega| \leq \pi/6$. Apparently, the optimization criterion concedes poorer high-frequency damping to obtain improved pass-band properties. In summary, $\widehat{\Psi}$ tracks Ψ towards the pivotal frequencies, i.e., those that are important to the process' dynamics, as quantified by the periodogram (bottom panel) in Figure 3.3. Similar findings apply to the other two series.

3.4 Qualitative Easing by Leading Indicators: an Empirical Study

In this section we quantify performance gains obtained by inclusion of a leading indicator into a univariate design. In particular, consider the process

$$\begin{aligned} X_{t,1} &= \phi\, X_{t-1,1} + \epsilon_{t,1} \\ X_{t,2} &= X_{t+\delta,1} + \sigma\, \epsilon_{t,2}, \end{aligned} \tag{3.14}$$

where $\{\epsilon_t\}$ is i.i.d. with mean zero and identity covariance matrix. Clearly, $\{X_{t,2}\}$ is a leading indicator of $\{X_{t,1}\}$ when the time-shift $\delta > 0$. The scaling factor σ determines the extent to which the indicator is effective, with larger values of σ implying that the indicator is less informative about the target $X_{t,1}$.

3.4.1 Bivariate MDFA versus Univariate DFA

Here we select $\sigma = 1$, corresponding to a weak idiosyncratic component, and set $\delta = 1$ so that the indicator leads by one time unit.

Exercise 8 Strong Leading Indicator. Simulate a sample of size $T = 200$ from the process (3.14) with $\phi = .9$, $\delta = 1$, and $\sigma = 1$. The target is one-step ahead forecasting of $\{X_{t,1}\}$, i.e., $Y_t = X_{t+1,1}$. Apply univariate DFA by specializing the MDFA methodology, and compare to results obtained from MDFA (Proposition 4), in each case setting $q = 20$. Apply both concurrent filters to the simulation, and compare the relevant portions to the actual target. Also determine the in-sample performance, in comparison to the criterion value (3.12), for both the DFA and MDFA methods. Compare the results graphically.

```
> # Simulate a Gaussian bivariate process of sample size 200:
> set.seed(1234)
> T <- 200
> N <- 2
> phi <- .9
> sigma <- 1
> gamma.0 <- 1/(1-phi^2)
> x.init <- sqrt(gamma.0)*rnorm(1)
> x.next <- x.init
> x.sim <- x.init
> for(t in 1:T)
+ {
+         x.next <- phi * x.next + rnorm(1)
+         x.sim <- c(x.sim,x.next)
+ }
> w.sim <- x.sim[-1] + sigma*rnorm(T)
```

```
> x.sim <- cbind(x.sim[-(T+1)],w.sim)
> # MDFA
> q <- 20
> Grid <- T
> m <- floor(Grid/2)
> # The Fourier frequencies
> lambda.ft <- exp(-1i*2*pi*Grid^{-1}*(seq(1,Grid) - (m+1)))
> # frf for 1-step ahead forecasting
> frf.psi <- matrix(lambda.ft^{-1},nrow=1) %x% diag(N)
> frf.psi <- array(frf.psi,c(N,N,Grid))
> spec.hat <- mdfa.pergram(x.sim,1)
> fore.mdfa <- mdfa.unconstrained(frf.psi,spec.hat,q)
> fore.udfa <- mdfa.unconstrained(frf.psi[1,1,,drop=FALSE],spec.hat[1,1,,drop=FALSE],q)
> # apply the MDFA concurrent filter
> x.fore.mdfa11 <- filter(x.sim[,1],fore.mdfa[[1]][1,1,],method="convolution",sides=1)
> x.fore.mdfa12 <- filter(x.sim[,2],fore.mdfa[[1]][1,2,],method="convolution",sides=1)
> x.fore.mdfa <- x.fore.mdfa11 + x.fore.mdfa12
> # apply the univariate DFA concurrent filter
> x.fore.udfa <- filter(x.sim[,1],fore.udfa[[1]][1,1,],method="convolution",sides=1)
> # compare in-sample performance
> print(c(mean((x.sim[(q+1):T,1] - x.fore.mdfa[q:(T-1)])^2),
+         mean((x.sim[(q+1):T,1] - x.fore.udfa[q:(T-1)])^2)))

[1] 0.345303 0.934042

> # compare to criterion value
> print(c(fore.mdfa[[2]][1,1],fore.udfa[[2]][1,1]))

[1] 0.3437316 0.9552386
```

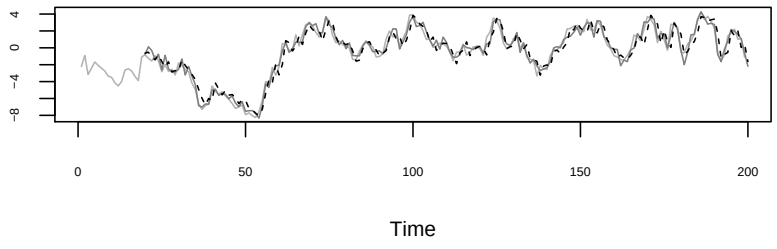

Figure 3.4: One-step ahead forecasts based upon MDFA (black solid) and univariate DFA (dashed dark grey), with target in solid light grey.

We see that there is a substantial improvement to performance of the MDFA over the univariate DFA; this can be visualized by the tracking of the target shown in Figure 3.4. This is possible

because the MDFA filter assigns more weight to the second series (the leading indicator), which is not available to the univariate DFA.

3.4.2 Measuring Lead and Signal-to-Noise Effects of a Leading Indicator

Intuitively, increasing δ or σ should result in a harder forecasting problem: $1/\sigma$ measures signal-to-noise ratio (snr), and low values indicate that real-time signal extraction becomes more difficult. On the other hand, a high lead time δ requires one to do long-term forecasting, which is known to be hard. Through the fabricated process (3.14) we can disentangle the conflict between increasing δ and decreasing σ. By allowing for non-integer shifts $\delta_j = j/4$, $j = 0, 1, 2, 3, 4$ we can quantify real-time forecasting performance. Note that if the time units are annual, then the δ_j correspond to quarterly forecasts; more generally, taking $\delta < 1$ corresponds to now-casting of a time series, and has many practical applications. The target filter has frequency response function of the form

$$\Psi(e^{-i\omega}) = \exp\{i\,\omega\delta\},$$

which can be immediately implemented in the frequency domain (whereas in time domain, the filter is difficult to express when δ is non-integer).

Exercise 9 Now-casting with a Leading Indicator. Simulate a sample of size $T = 2500$ from the process (3.14) with $\phi = .9$, $\delta_j = j/4$, $j = 0, 1, 2, 3, 4$ and $\sigma = 0, 0.1, 0.5, 1, 2$. The target is δ-step ahead nowcasting of $\{X_{t,1}\}$, i.e., $Y_t = X_{t+\delta,1}$. Filter to obtain the now-cast target, so as to retain a target series of length 500. Combine with the leading indicator, and apply univariate DFA and MDFA methodology (Proposition 4), in each case setting $q = 20$. Apply both concurrent filters to the simulation, and compare the relevant portions to the actual target. Record the criterion values (3.12) for both the DFA and MDFA methods.

```
> # Set up loops over delta and sigma
> sigma.vals <- c(0,.1,.5,1,2)
> critmdfa.mat <- matrix(0,5,5,dimnames=list(c(0,1,2,3,4)/4,sigma.vals))
> critudfa.mat <- matrix(0,5,5,dimnames=list(c(0,1,2,3,4)/4,sigma.vals))
> for(delta in c(0,1,2,3,4)/4) {
+ for(j in 1:5) {
+
+ sigma <- sigma.vals[j]
+ # Simulate a Gaussian bivariate process of sample size 2500:
+ set.seed(1234)
+ T <- 2500
+ N <- 2
+ phi <- .9
+ gamma.0 <- 1/(1-phi^2)
+ x.init <- sqrt(gamma.0)*rnorm(1)
+ x.next <- x.init
+ x.sim <- x.init
```

```
+ for(t in 1:T)
+ {
+         x.next <- phi * x.next + rnorm(1)
+         x.sim <- c(x.sim,x.next)
+ }
+
+ Grid <- T
+ m <- floor(Grid/2)
+ # define complex exponential at Fourier frequencies
+ lambda.ft <- exp(-1i*2*pi*Grid^{-1}*(seq(1,Grid) - (m+1)))
+ # frf for delta-step ahead forecasting
+ frf.psi <- matrix(lambda.ft^{-delta},nrow=1)
+ frf.psi <- array(frf.psi,c(1,1,Grid))
+ nowcast.filter <- mdfa.coeff(frf.psi,-len,len)
+ x.target <- filter(x.sim,nowcast.filter[1,1,],method="convolution",sides=2)[(len+1):(T-len)]
+ w.sim <- x.target + sigma*rnorm(T-2*len)
+ x.sim <- cbind(x.sim[(len+1):(T-len)],w.sim)
+
+ # MDFA
+ q <- 20
+ Grid <- T - 2*len
+ m <- floor(Grid/2)
+ # The Fourier frequencies (recompute with smaller sample size)
+ lambda.ft <- exp(-1i*2*pi*Grid^{-1}*(seq(1,Grid) - (m+1)))
+ # frf for delta-step ahead forecasting
+ frf.psi <- matrix(lambda.ft^{-delta},nrow=1) %x% diag(N)
+ frf.psi <- array(frf.psi,c(N,N,Grid))
+ spec.hat <- mdfa.pergram(x.sim,1)
+ fore.udfa <- mdfa.unconstrained(frf.psi[1,1,,drop=FALSE],spec.hat[1,1,,drop=FALSE],q)
+ if(j > 1) {
+         fore.mdfa <- mdfa.unconstrained(frf.psi,spec.hat,q)
+ } else {
+         fore.mdfa <- fore.udfa
+ }
+
+ # apply the MDFA concurrent filter
+ x.fore.mdfa11 <- filter(x.sim[,1],fore.mdfa[[1]][1,1,],method="convolution",sides=1)
+ if(j > 1) {
+         x.fore.mdfa12 <- filter(x.sim[,2],fore.mdfa[[1]][1,2,],method="convolution",sides=1)
+ } else { x.fore.mdfa12 <- 0*x.fore.mdfa11 }
+ x.fore.mdfa <- x.fore.mdfa11 + x.fore.mdfa12
```

```
+
+ # apply the univariate DFA concurrent filter
+ x.fore.udfa <- filter(x.sim[,1],fore.udfa[[1]][1,1,],method="convolution",sides=1)
+
+ # compare in-sample performance
+ print(c(mean((x.target[-seq(1,q-1)] - x.fore.mdfa[-seq(1,q-1)])^2),
+         mean((x.target[-seq(1,q-1)] - x.fore.udfa[-seq(1,q-1)])^2)))
+
+ # store criterion value
+ i <- delta*4 + 1
+ critmdfa.mat[i,j] <- fore.mdfa[[2]][1,1]
+ critudfa.mat[i,j] <- fore.udfa[[2]][1,1]
+ }}
[1] 2.802362e-27 2.802362e-27
[1] 1.427359e-23 2.802362e-27
[1] 2.905156e-27 2.802362e-27
[1] 2.882633e-27 2.802362e-27
[1] 2.808209e-27 2.802362e-27
[1] 0.05078962 0.05078962
[1] 0.005970153 0.050789616
[1] 0.03457852 0.05078962
[1] 0.04335656 0.05078962
[1] 0.04677115 0.05078962
[1] 0.2377793 0.2377793
[1] 0.008471453 0.237779309
[1] 0.0952848 0.2377793
[1] 0.1638133 0.2377793
[1] 0.2037506 0.2377793
[1] 0.5331737 0.5331737
[1] 0.009759302 0.533173748
[1] 0.1416053 0.5331737
[1] 0.2988843 0.5331737
[1] 0.4237369 0.5331737
[1] 0.8413718 0.8413718
[1] 0.01048854 0.84137183
[1] 0.1676030 0.8413718
[1] 0.3987859 0.8413718
[1] 0.6258851 0.8413718
```

The results for MDFA and univariate DFA, respectively, are given in Tables 3.1 and 3.2. Regarding the design of Exercise 9, we make the following comments. When $\sigma = 0$ the leading indicator exactly matches the target; if $\delta = 0$ as well, then $X_{t,1} = X_{t,2}$ and there is redundancy in

δ	Sigma values				
	0	0.1	0.5	1	2
0	-0.000	0.000	-0.000	-0.000	0.000
0.25	0.052	0.009	0.035	0.044	0.047
0.5	0.242	0.024	0.100	0.165	0.205
0.75	0.543	0.040	0.158	0.305	0.428
1	0.857	0.051	0.195	0.415	0.635

Table 3.1: Effect of lead and inverse signal-to-noise ratio on MDFA filter MSE

δ	Sigma values				
	0	0.1	0.5	1	2
0	-0.000	-0.000	-0.000	-0.000	-0.000
0.25	0.052	0.052	0.052	0.052	0.052
0.5	0.242	0.242	0.242	0.242	0.242
0.75	0.543	0.543	0.543	0.543	0.543
1	0.857	0.857	0.857	0.857	0.857

Table 3.2: Effect of lead and inverse signal-to-noise ratio on DFA filter MSE

the data – this will lead to a singularity in the periodogram. When $\delta > 0$ (but $\sigma = 0$) then $\{X_{t,1}\}$ and $\{X_{t,2}\}$ are perfectly coherent, as the relation $X_{t,2} = B^{-\delta} X_{t,1}$ holds. This full coherency indicates a type of redundancy is still present, and the MDFA method is singular. Therefore, in these cases we restrict the MDFA to univariate DFA. Therefore, the first columns of Tables 3.1 and 3.2 are identical.

The first rows of Tables 3.1 and 3.2 are trivially zero, because when $\delta = 0$ the target is observable, and hence both MDFA and DFA select the identity filter. Broadly, the patterns are what we would expect: increasing δ and/or σ generates worse performance (higher MSE), although the MDFA is superior to DFA. The performance of MDFA relative to DFA worsens as σ increases, irrespective of δ, which makes sense: there is less benefit to the leading indicator when the snr is low, in which case DFA should generate a competitive real-time filter. In particular, reading across the rows of Table 3.2 we see the MSE is fairly constant – DFA does not utilize the leading indicator, so the variability here is due to the simulation seeds. As for the MDFA, decreased performance due to lower snr could be compensated by decreasing the lead-time δ. Again, when σ is low (second column of Table 3.1) the leading indicator is very useful, and the MSE does not depend greatly on δ. This pattern is opposite when σ is high (fifth column), as increased δ deleteriously affects performance.

These results suggest the pertinence of a mixed-frequency approach, whereby information at differing sampling frequencies (such as monthly and quarterly data) is combined. The higher-frequency data stream could be used to update the filter for the lower-frequency time series; this is further discussed in Chapter **??**.

Chapter 4

Multivariate Direct Filter Analysis for Non-stationary Processes

We now extend the basic MDFA of Chapter 3 by considering the method's application to non-stationary processes. Section 4.1 introduces the idea of filter constraints arising from time-varying means, a form of non-stationarity. This treatment is generalized in Section 4.2 by the definition of non-stationary processes, and theory for the corresponding model-based filters is developed. Finally, the MDFA criterion for non-stationary processes is discussed in Section 4.3.

4.1 Constrained MDFA

Various constraints upon the concurrent filter can be envisioned, and imposing such strictures results in a constrained MDFA. A chief case of interest arises when the data process has a time-varying mean (which is a form of non-stationarity); then it is necessary to impose additional filter constraints – otherwise the filter error will not have mean zero. To see why, Write $\Delta(L) = \Psi(L) - \widehat{\Psi}(L)$ as the discrepancy filter, so that we see from (2.7) that $\mathbb{E}[E_t] = \Delta(L)\,\mathbb{E}[X_t]$; by Definition 2, we require that $\mathbb{E}[E_t] = 0$ for any LPP. If $\mathbb{E}[X_t] = 0$ then this condition is always satisfied, but for most time series of interest the mean will be nonzero, and is typically time-varying. For such cases additional constraints on $\Delta(L)$ must be imposed, which implicitly amount to constraints on $\widehat{\Psi}(L)$.

Example 9 Constant Mean. If $\mathbb{E}[X_t] = \mu$, some nonzero constant, then we require $\Delta(1) = 0$. This is because the mean of the filter error is

$$\Delta(B)\,\mathbb{E}[X_t] = \Delta(B)\,\mu = \sum_j \delta(j)\,\mu = \Delta(1)\,\mu,$$

and this is zero only if $\Delta(1) = 0$. This is called a Level Constraint (LC).

Example 10 Linear Mean. Suppose that $\mathbb{E}[X_t] = \mu\, t$, where μ is a nonzero slope of a linear time trend. Then it is required that $\partial\Delta(1) = 0$ in addition to the LC, which is seen as follows:

$$\Delta(B)\,\mathbb{E}[X_t] = \Delta(B)\,\mu\, t = \mu \sum_j \delta(j)\,(t-j) = \mu \left(t \sum_j \delta(j) - \sum_j j\,\delta(j) \right) = \mu\, t\, \Delta(1) - \mu\, \partial\Delta(1).$$

This mean of the filter error is zero only if both $\Delta(1) = 0$ and $\partial\Delta(1) = 0$; the latter condition is called the Time-Shift Constraint (TSC).

Hence, for linear means we obtain three fundamental types of constraints: LC, TSC, and Level plus Time-Shift Constraint (LTSC), which combines both LC and TSC. Using the fact that $\Delta(L) = \Psi(L) - \widehat{\Psi}(L)$, these three constaints can be described as follows:

$$\begin{aligned} \text{LC} &: \Delta(1) = 0 \quad \text{or} \quad \Psi(1) = \widehat{\Psi}(1) \\ \text{TSC} &: \partial\Delta(1) = 0 \quad \text{or} \quad \partial\Psi(1) = \partial\widehat{\Psi}(1) \\ \text{LTSC} &: \Delta(1) = 0,\, \partial\Delta(1) = 0 \quad \text{or} \quad \Psi(1) = \widehat{\Psi}(1),\, \partial\Psi(1) = \partial\widehat{\Psi}(1). \end{aligned}$$

In the case of concurrent filters of form (3.9), LC is accomplished by demanding that $\sum_{j=0}^{q-1} \widehat{\psi}(j) = \Psi(1)$. More generally, we consider linear constraints formulated via

$$\vartheta = R\,\varphi + Q, \tag{4.1}$$

where R is $nq \times nr$ and φ is $nr \times 1$ dimensional, consisting of free parameters; Q is a matrix of constants, and is $nq \times 1$ dimensional.

Illustration 4 Level Constraint (LC). Note that $\sum_{j=0}^{q-1} \widehat{\psi}(j) = \Psi(1)$ implies that

$$\widehat{\psi}(0) = \Psi(1) - \sum_{j=1}^{q-1} \widehat{\psi}(j). \tag{4.2}$$

Hence $\varphi' = [\widehat{\psi}(1), \widehat{\psi}(2), \ldots, \widehat{\psi}(q-1)]$ and

$$R = \begin{bmatrix} -1 & \ldots & -1 \\ 1 & 0 & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & 1 \end{bmatrix} \otimes 1_n \qquad Q = \begin{bmatrix} \Psi(1) \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Illustration 5 Time Shift Constraint (TSC). The constraint is $\partial\Psi(1) = \partial\widehat{\Psi}(1) = \sum_{j=0}^{q-1} j\,\widehat{\psi}(j)$, or $\widehat{\psi}(1) = \partial\Psi(1) - \sum_{j=2}^{q-1} j\,\widehat{\psi}(j)$. Hence $\varphi' = [\widehat{\psi}(0), \widehat{\psi}(2), \ldots, \widehat{\psi}(q-1)]$ and

$$R = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & -2 & -3 & \ldots \\ 0 & 1 & 0 & \ldots \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & 1 \end{bmatrix} \otimes 1_n \qquad Q = \begin{bmatrix} 0 \\ \partial\Psi(1) \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Illustration 6 Level and Time Shift Constraint (LTSC). Take the Time Shift Constraint formula for $\widehat{\psi}(1)$, and plug this into (4.2), to obtain

$$\begin{aligned}\widehat{\psi}(0) &= \Psi(1) - \left(\partial\Psi(1) - \sum_{j=2}^{q-1} j\,\widehat{\psi}(j)\right) - \sum_{j=2}^{q-1}\widehat{\psi}(j) \\ &= \Psi(1) - \partial\Psi(1) + \sum_{j=2}^{q-1}(j-1)\,\widehat{\psi}(j).\end{aligned}$$

Hence $\varphi' = [\widehat{\psi}(2), \ldots, \widehat{\psi}(q-1)]$ and

$$R = \begin{bmatrix} 1 & 2 & 3 & \ldots \\ -2 & -3 & -4 & \ldots \\ 1 & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & 1 \end{bmatrix} \otimes 1_n \qquad Q = \begin{bmatrix} \Psi(1) - \partial\Psi(1) \\ \partial\Psi(1) \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

More generally, we can envision an LPP involving m linear constraints on ϑ, taking the form $A = [J \otimes 1_n]\,\vartheta$, where J is $m \times q$ dimensional ($m < q$) and A is $nm \times 1$ dimensional. (The LC, TSC, and LTSC examples all have this form.) In order to express this constraint in the form (4.1), we use the Q-R decomposition (Golub and Van Loan, 1996) of J, writing $J = C\,G\,\Pi$ for an orthogonal matrix C (which is $m \times m$ dimensional), a rectangular upper triangular matrix G (which is $m \times q$ dimensional), and a permuation matrix Π (which is $q \times q$ dimensional). Standard matrix software such as R will provide the Q-R decomposition J, and should produce the rank of J as a by-product – if this is less than m, then there are redundancies in the constraints that should first be eliminated.

Hence proceeding with a full rank J, we partition G as $G = [G_1\,G_2]$ such that G_1 has m columns and G_2 has $q - m$ columns. This quantity $q - m$ corresponds to the number of free coefficient matrices, and is therefore the same as r. The Q-R decomposition guarantees that G_1 is an upper triangular matrix, and moreover it is invertible. Therefore

$$\left[G_1^{-1}\,C^{-1} \otimes 1_n\right]\,A = \left(\left[1_m,\, G_1^{-1}\,G_2\right]\,\Pi \otimes 1_n\right)\,\vartheta,$$

and the action of Π (together with the tensor product) amounts to a permutation of the elements of ϑ. Let the output of this permutation be denoted

$$\begin{bmatrix} \overline{\vartheta} \\ \underline{\vartheta} \end{bmatrix} = (\Pi \otimes 1_n)\,\vartheta,$$

where $\overline{\vartheta}$ is $nm \times 1$ dimensional and $\underline{\vartheta}$ is $nr \times 1$ dimensional. Then by substitution we can solve for $\overline{\vartheta}$ in terms of $\underline{\vartheta}$:

$$\overline{\vartheta} = \left[G_1^{-1}\,C^{-1} \otimes 1_n\right]\,A - \left[G_1^{-1}\,G_2 \otimes 1_n\right]\,\underline{\vartheta}.$$

Therefore we recognize the free variables $\varphi = \underline{\vartheta}$, and obtain R and Q in (4.1) via

$$R = \Pi^{-1} \begin{bmatrix} -G_1^{-1} G_2 \\ 1_r \end{bmatrix} \otimes 1_n$$

$$Q = \left(\Pi^{-1} \begin{bmatrix} G_1^{-1} C^{-1} \\ 0 \end{bmatrix} \otimes 1_n \right) A.$$

Exercise 10 QR Decomposition. Consider an arbitrary set of constraints J on ϑ, such that $A = [J \otimes 1_n]\,\vartheta$ for a given vector A. Encode the procedure that obtains R and Q, and apply this to the cases of the LC, TSC, and LTSC scenarios with $n = 1$, verifying the results given in Illustrations 4, 5, and 6. Use one-step ahead forecasting as the target filter.

```
>   N <- 1
>   q <- 10
>   ## level constraint case
>         constraint.mat <- matrix(rep(1,q),nrow=1)
>         constraint.vec <- diag(N)
>         constraint.qr <- qr(constraint.mat)
>         constraint.q <- qr.Q(constraint.qr)
>         constraint.r <- qr.R(constraint.qr)
>         constraint.pivot <- constraint.qr$pivot
>         constraint.ipivot <- sort.list(constraint.pivot)
>         M <- q - dim(constraint.r)[2] + dim(constraint.q)[2]
>         R.mat <- rbind(-solve(constraint.r[,1:(dim(constraint.q)[2]),drop=FALSE],
+                 constraint.r[,(dim(constraint.q)[2]+1):q,drop=FALSE]),diag(q-M))
>         R.mat <- R.mat[constraint.ipivot,] %x% diag(N)
>         Q.mat <- rbind(solve(constraint.r[,1:(dim(constraint.q)[2]),drop=FALSE]) %*%
+                 solve(constraint.q),matrix(0,q-M,M))
>         Q.mat <- (Q.mat[constraint.ipivot,] %x% diag(N)) %*% constraint.vec
>   print(R.mat)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]   -1   -1   -1   -1   -1   -1   -1   -1   -1
 [2,]    1    0    0    0    0    0    0    0    0
 [3,]    0    1    0    0    0    0    0    0    0
 [4,]    0    0    1    0    0    0    0    0    0
 [5,]    0    0    0    1    0    0    0    0    0
 [6,]    0    0    0    0    1    0    0    0    0
 [7,]    0    0    0    0    0    1    0    0    0
 [8,]    0    0    0    0    0    0    1    0    0
 [9,]    0    0    0    0    0    0    0    1    0
[10,]    0    0    0    0    0    0    0    0    1
```

```
>   ## time shift constraint case
>         constraint.mat <- matrix(seq(0,q-1),nrow=1)
```

```
>         constraint.vec <- -diag(N)
>         constraint.qr <- qr(constraint.mat)
>         constraint.q <- qr.Q(constraint.qr)
>         constraint.r <- qr.R(constraint.qr)
>         constraint.pivot <- constraint.qr$pivot
>         constraint.ipivot <- sort.list(constraint.pivot)
>         M <- q - dim(constraint.r)[2] + dim(constraint.q)[2]
>         R.mat <- rbind(-solve(constraint.r[,1:(dim(constraint.q)[2]),drop=FALSE],
+                 constraint.r[,(dim(constraint.q)[2]+1):q,drop=FALSE]),diag(q-M))
>         R.mat <- R.mat[constraint.ipivot,] %x% diag(N)
>         Q.mat <- rbind(solve(constraint.r[,1:(dim(constraint.q)[2]),drop=FALSE]) %*%
+                 solve(constraint.q),matrix(0,q-M,M))
>         Q.mat <- (Q.mat[constraint.ipivot,] %x% diag(N)) %*% constraint.vec
>   print(R.mat)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    0    0    0    0    0    0    0    0    1
 [2,]   -2   -3   -4   -5   -6   -7   -8   -9    0
 [3,]    1    0    0    0    0    0    0    0    0
 [4,]    0    1    0    0    0    0    0    0    0
 [5,]    0    0    1    0    0    0    0    0    0
 [6,]    0    0    0    1    0    0    0    0    0
 [7,]    0    0    0    0    1    0    0    0    0
 [8,]    0    0    0    0    0    1    0    0    0
 [9,]    0    0    0    0    0    0    1    0    0
[10,]    0    0    0    0    0    0    0    1    0
```

```
>         ## level and time shift constraint case
>         constraint.mat <- rbind(rep(1,q),seq(0,q-1))
>         constraint.vec <- rbind(diag(N),-diag(N))
>         constraint.qr <- qr(constraint.mat)
>         constraint.q <- qr.Q(constraint.qr)
>         constraint.r <- qr.R(constraint.qr)
>         constraint.pivot <- constraint.qr$pivot
>         constraint.ipivot <- sort.list(constraint.pivot)
>         M <- q - dim(constraint.r)[2] + dim(constraint.q)[2]
>         R.mat <- rbind(-solve(constraint.r[,1:(dim(constraint.q)[2]),drop=FALSE],
+                 constraint.r[,(dim(constraint.q)[2]+1):q,drop=FALSE]),diag(q-M))
>         R.mat <- R.mat[constraint.ipivot,] %x% diag(N)
>         Q.mat <- rbind(solve(constraint.r[,1:(dim(constraint.q)[2]),drop=FALSE]) %*%
+                 solve(constraint.q),matrix(0,q-M,M))
>         Q.mat <- (Q.mat[constraint.ipivot,] %x% diag(N)) %*% constraint.vec
>   print(R.mat)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
 [1,]    1    2    3    4    5    6    7    8
 [2,]   -2   -3   -4   -5   -6   -7   -8   -9
 [3,]    1    0    0    0    0    0    0    0
 [4,]    0    1    0    0    0    0    0    0
 [5,]    0    0    1    0    0    0    0    0
 [6,]    0    0    0    1    0    0    0    0
 [7,]    0    0    0    0    1    0    0    0
 [8,]    0    0    0    0    0    1    0    0
 [9,]    0    0    0    0    0    0    1    0
[10,]    0    0    0    0    0    0    0    1
```

These formulas allow one to compute the form (4.1) from given constraints, and an analytical solution to the resulting MDFA criterion be obtained from the following result.

Proposition 6 *The minimizer of the MDFA criterion given by (3.7) with respect to $\mathcal{G}$ – consisting of all length q concurrent filters subject to linear constraints of the form (4.1) – is*

$$\varphi = [R'\,B\,R]^{-1}\,R'\,(b - B\,Q)\,. \tag{4.3}$$

The minimal value is

$$\langle \Psi(z)\,G\,\Psi(z)^*\rangle_0 + (b - B\,Q)'\,\left(R\,[R'\,B\,R]^{-1}\,R' + B^{-1}\right)\,(b - B\,Q) - b'\,B^{-1}\,b. \tag{4.4}$$

Proof of Proposition 6. Substituting (4.1) in (3.10) yields

$$\begin{aligned} D_\Psi(\vartheta, G) = \varphi'\,[R'\,B\,R]\,\varphi + [Q'\,B\,R - b'\,R]\,\varphi + \varphi'\,[R'\,B\,Q - R'\,b] \\ + Q'\,B\,Q - Q'\,b - b'\,Q + \langle \Psi(z)\,G\,\Psi(z)^*\rangle_0. \end{aligned}$$

Now by applying the method of proof in Proposition 4, we obtain the formula (4.3) for φ. Plugging back into $D_\Psi(\vartheta, G)$ yields the minimal value (4.4). □

For computation, we utilize the same approximations to B and b as discussed in Chapter 3, obtaining the constrained MDFA filter ϑ via (**??**) followed by (4.1).

Exercise 11 Constrained MDFA for White Noise with Linear Trend. This exercise applies the constrained MDFA in the case of an ideal low-pass filter (cf. Example 2) applied to a white noise process that exhibits a linear trend. Simulate a sample of size $T = 5000$ from a bivariate white noise process with Σ equal to the identity, but with a linear trend given by

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} + t \begin{bmatrix} -.002 \\ .001 \end{bmatrix}. \tag{4.5}$$

Apply the ideal low-pass filter (cf. Example 2) with $\mu = \pi/6$ to the sample (truncate the filter to 1000 coefficients on each side). Use the moving average filter MDFA with LC, TSC, and LTSC constraints (Proposition 6), as well as unconstrained MDFA (Proposition 4), to find the best

concurrent filter, setting $q = 30$. (Hint: compute the periodogram from OLS residuals obtained by regressing the simulation on a constant plus time.) Apply this concurrent filter to the simulation, and compare the relevant portions to the ideal trend. Also determine the in-sample performance, in comparison to the criterion value (3.12). Target the trends for both time series.

```
> # Simulate a Gaussian WN of sample size 5000:
> set.seed(1234)
> T <- 5000
> N <- 2
> levels <- c(1,2)
> slopes <- c(-2,1)/1000
> innovar.matrix <- diag(N)
> x.sim <- NULL
> for(t in 1:T)
+ {
+         x.next <- levels + slopes*t + t(chol(innovar.matrix)) %*% rnorm(N)
+         x.sim <- cbind(x.sim,x.next)
+ }
> x.sim <- ts(t(x.sim))
> time.trend <- seq(1,T)
> sim.ols <- lm(x.sim ~ time.trend)
> x.resid <- sim.ols$residuals
> # construct and apply low pass filter
> mu <- pi/6
> len <- 1000
> lp.filter <- c(mu/pi,sin(seq(1,len)*mu)/(pi*seq(1,len)))
> lp.filter <- c(rev(lp.filter),lp.filter[-1])
> x.trend.ideal <- filter(x.sim,lp.filter,method="convolution",sides=2)[(len+1):(T-len),]
> # get MDFA concurrent filter
> q <- 30
> Grid <- T
> m <- floor(Grid/2)
> # The Fourier frequencies
> freq.ft <- 2*pi*Grid^{-1}*(seq(1,Grid) - (m+1))
> # frf for ideal low-pass
> frf.psi <- rep(0,Grid)
> frf.psi[abs(freq.ft) <= mu] <- 1
> frf.psi <- matrix(frf.psi,nrow=1) %x% diag(N)
> frf.psi <- array(frf.psi,c(N,N,Grid))
> spec.hat <- mdfa.pergram(x.resid,1)
> lp.mdfa.uc <- mdfa.unconstrained(frf.psi,spec.hat,q)
> lp.mdfa.lc <- mdfa.levelconstraint(frf.psi,spec.hat,q)
```

```
> lp.mdfa.tsc <- mdfa.tsconstraint(frf.psi,spec.hat,q)
> lp.mdfa.ltsc <- mdfa.ltsconstraint(frf.psi,spec.hat,q)
> # case 1: apply the unconstrained MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa.uc[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa.uc[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa.uc[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa.uc[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 2.394344 2.284378

> # compare to criterion value
> diag(lp.mdfa.uc[[2]])

[1] 0.0710269 0.0698122

> # case 2: apply the lc MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa.lc[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa.lc[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa.lc[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa.lc[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 0.07339214 0.07837711

> # compare to criterion value
> diag(lp.mdfa.lc[[2]])

[1] 0.07587524 0.07374379

> # case 3: apply the tsc MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa.tsc[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa.tsc[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa.tsc[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa.tsc[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
```

```
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 3.694607 4.504850

> # compare to criterion value
> diag(lp.mdfa.tsc[[2]])

[1] 0.07129042 0.07071495

> # case 4: apply the ltsc MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa.ltsc[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa.ltsc[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa.ltsc[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa.ltsc[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 0.09396811 0.10376989

> # compare to criterion value
> diag(lp.mdfa.ltsc[[2]])

[1] 0.09606772 0.09728970
```

Exercise 12 Constrained MDFA for VAR(1) with Linear Trend. This exercise applies the constrained MDFA in the case of an ideal low-pass filter (cf. Example 2) applied to a VAR(1) process that exhibits a linear trend. Simulate a sample of size $T = 5000$ of a bivariate VAR(1) process with linear trend given by (4.5), such that the demeaned process satisfies

$$X_t = \begin{bmatrix} 1 & 1/2 \\ -1/5 & 3/10 \end{bmatrix} X_{t-1} + \epsilon_t,$$

with stationary initialization, and $\{\epsilon_t\}$ a Gaussian white noise of identity innovation variance. Apply the ideal low-pass filter (cf. Example 2) with $\mu = \pi/6$ to the sample (truncate the filter to 1000 coefficients on each side). Use the moving average filter MDFA with LC, TSC, and LTSC constraints (Proposition 6), as well as unconstrained MDFA (Proposition 4), to find the best concurrent filter, setting $q = 30$. (Hint: compute the periodogram from OLS residuals obtained by regressing the simulation on a constant plus time.) Apply this concurrent filter to the simulation, and compare the relevant portions to the ideal trend. Also determine the in-sample performance, in comparison to the criterion value (3.12). Target the trends for both time series.

```
> # Simulate a VAR(1) of sample size 5000:
> set.seed(1234)
> T <- 5000
> N <- 2
> levels <- c(1,2)
> slopes <- c(-2,1)/1000
> phi.matrix <- rbind(c(1,.5),c(-.2,.3))
> innovar.matrix <- diag(N)
> gamma.0 <- matrix(solve(diag(N^2) - phi.matrix %x% phi.matrix) %*%
+          matrix(innovar.matrix,ncol=1),nrow=N)
> x.init <- t(chol(gamma.0)) %*% rnorm(N)
> x.next <- x.init
> x.sim <- NULL
> for(t in 1:T)
+ {
+          x.next <- phi.matrix %*% x.next + t(chol(innovar.matrix)) %*% rnorm(N)
+          x.sim <- cbind(x.sim,x.next)
+ }
> x.sim <- ts(t(x.sim))
> time.trend <- seq(1,T)
> x.sim <- t(levels) %x% rep(1,T) + t(slopes) %x% seq(1,T) + x.sim
> sim.ols <- lm(x.sim ~ time.trend)
> x.resid <- sim.ols$residuals
> # construct and apply low pass filter
> mu <- pi/6
> len <- 1000
> lp.filter <- c(mu/pi,sin(seq(1,len)*mu)/(pi*seq(1,len)))
> lp.filter <- c(rev(lp.filter),lp.filter[-1])
> x.trend.ideal <- filter(x.sim,lp.filter,method="convolution",sides=2)[(len+1):(T-len),]
> # get MDFA concurrent filter
> q <- 30
> Grid <- T
> m <- floor(Grid/2)
> # The Fourier frequencies
> freq.ft <- 2*pi*Grid^{-1}*(seq(1,Grid) - (m+1))
> # frf for ideal low-pass
> frf.psi <- rep(0,Grid)
> frf.psi[abs(freq.ft) <= mu] <- 1
> frf.psi <- matrix(frf.psi,nrow=1) %x% diag(N)
> frf.psi <- array(frf.psi,c(N,N,Grid))
> spec.hat <- mdfa.pergram(x.resid,1)
```

```
> lp.mdfa.uc <- mdfa.unconstrained(frf.psi,spec.hat,q)
> lp.mdfa.lc <- mdfa.levelconstraint(frf.psi,spec.hat,q)
> lp.mdfa.tsc <- mdfa.tsconstraint(frf.psi,spec.hat,q)
> lp.mdfa.ltsc <- mdfa.ltsconstraint(frf.psi,spec.hat,q)
> # case 1: apply the unconstrained MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa.uc[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa.uc[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa.uc[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa.uc[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 1.751825 1.147235

> # compare to criterion value
> diag(lp.mdfa.uc[[2]])

[1] 0.4235386 0.1327707

> # case 2: apply the lc MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa.lc[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa.lc[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa.lc[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa.lc[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 0.4341617 0.1477676

> # compare to criterion value
> diag(lp.mdfa.lc[[2]])

[1] 0.433979 0.138771

> # case 3: apply the tsc MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa.tsc[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa.tsc[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa.tsc[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa.tsc[[1]][2,2,],method="convolution",sides=1)
```

```
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 1.896865 2.516518

> # compare to criterion value
> diag(lp.mdfa.tsc[[2]])

[1] 0.4260567 0.1341826

> # case 4: apply the ltsc MDFA concurrent filter
> x.trend.mdfa11 <- filter(x.sim[,1],lp.mdfa.ltsc[[1]][1,1,],method="convolution",sides=1)
> x.trend.mdfa12 <- filter(x.sim[,2],lp.mdfa.ltsc[[1]][1,2,],method="convolution",sides=1)
> x.trend.mdfa21 <- filter(x.sim[,1],lp.mdfa.ltsc[[1]][2,1,],method="convolution",sides=1)
> x.trend.mdfa22 <- filter(x.sim[,2],lp.mdfa.ltsc[[1]][2,2,],method="convolution",sides=1)
> x.trend.mdfa <- cbind(x.trend.mdfa11 + x.trend.mdfa12,x.trend.mdfa21 + x.trend.mdfa22)
> x.trend.mdfa <- x.trend.mdfa[(len+1):(T-len),]
> # compare in-sample performance
> print(c(mean((x.trend.ideal[,1] - x.trend.mdfa[,1])^2),
+         mean((x.trend.ideal[,2] - x.trend.mdfa[,2])^2)))

[1] 0.4720341 0.1820247

> # compare to criterion value
> diag(lp.mdfa.ltsc[[2]])

[1] 0.4700269 0.1712509
```

4.2 Background on Non-stationary Vector Time Series

We next consider processes that when differenced are stationary, which are the most common type occuring in econometrics and finance. This type of non-stationary process substantially broadens the possible types of applications over the stationary processes considered in Chapters 2 and 3. Also, as such processes typically can have a time-varying mean, they also necessitate the use of filter constraints such as those considered in Section 4.1.

We suppose that there exists a degree d scalar polynomial $\delta(L)$ that reduces each component series of $\{X_t\}$ to a stationary time series (which is allowed to have a non-zero constant mean μ), and suppose this is the minimal degree polynomial that accomplishes this reduction. For convenience, and without loss of generality, we suppose that $\delta_0 = 1$, or $\delta(0) = 1$. We write $\partial X_t = \delta(L)\, X_t$ for the stationary, differenced time series, where $\partial X_t = \mu + Z_t$ and $\{Z_t\}$ has a spectral representation (2.2). Then it is possible to give time-domain and frequency-domain representations of the original

process $\{X_t\}$ in terms of the stationary "increments" $\{\partial X_t\}$, together with deterministic functions of time that depend on "initial values" of the process. These deterministic functions can be obtained from the theory of Ordinary Difference Equations (ODE): all solutions to $\delta(L)X_t = \partial X_t$ must include a homogeneous solution, i.e., solutions to $\delta(L)X_t = 0$, which include all functions of t that are annihilated by $\delta(L)$. Below we develop a general method of solution, but We first provide a few illustrations through specific cases.

Example 11 Representation for an $I(1)$ Process. Letting $\delta(L) = 1 - L$, we obtain a once-integrated process, denoted as $I(1)$ for short. Because the constant function (which up to proportionality, is the function 1) is annihilated by $1 - L$, we expect the solution to take the form of a constant plus some function of the increments ∂X_t. Proceeding recursively, we obtain

$$X_t = X_{t-1} + \partial X_t = X_{t-2} + \partial X_t + \partial X_{t-1} = \dots$$

Let us suppose an initial time of $t = 0$ for this process, so that the solution is expressed as

$$X_t = X_0 + \sum_{j=1}^{t} \partial X_j$$

for $t \geq 1$ (and can be extended to $t = 0$ by taking the sum to be empty in that case). Note that this involves a constant function of time, the term X_0. Moreover, applying $X_j = \mu + Z_j$ and the spectral representation, we obtain

$$X_t = X_0 + t\,\mu + \int_{-\pi}^{\pi} \sum_{j=1}^{t} e^{i\omega j}\, \mathcal{Z}(d\omega).$$

The summation inside the integral can be re-expressed when $\omega \neq 0$ as $(1 - e^{i\omega(t+1)})/(1 - e^{i\omega})$.

Example 12 Representation for an $I(2)$ Process. Now we set $\delta(L) = (1 - L)^2$ for a twice-integrated process, denoted as $I(2)$ for short. So first differences of $\{X_t\}$ have a representation as an $I(1)$ process, and the expression for X_t will involve a linear function of time t, because t is annihilated by $\delta(L)$. Applying the recursive technique of Example **??** twice, we obtain

$$X_t = (t+1)\, X_0 - t\, X_{-1} + \sum_{j=1}^{t} (t + 1 - j)\, \partial X_j,$$

which holds for $t \geq 1$ (but can be extended to $t = 0, -1$ by setting the summation to zero). The linear function of time has slope $X_0 - X_{-1}$ and intercept X_0. Applying $X_j = \mu + Z_j$ and the spectral representation, we obtain

$$X_t = (t+1)\, X_0 - t\, X_{-1} + \binom{t+1}{2} \mu + \int_{-\pi}^{\pi} \sum_{j=1}^{t} (t + 1 - j)\, e^{i\omega j}\, \mathcal{Z}(d\omega).$$

It can be verified that $(1 - L)X_t$ is an $I(1)$ process with level $X_0 - X_{-1}$.

A general technique for obtaining the representation for non-stationary processes involves the inverse of the polynomial of $\delta(L)$, which is denoted by $\xi(z)$:

$$\xi(L) = 1/\delta(L) = \sum_{j \geq 0} \xi_j\, L^j. \tag{4.6}$$

This $\xi(z)$ is a power series that converges on a disk inside the unit circle. Because $\delta(z)\,\xi(z) = 1$, one can recursively solve for the coefficients ξ_j in terms of past coefficients, using the δ_k. In particular, the jth coefficient of $\delta(z)\,\xi(z)$ is given by the convolution formula:

$$\sum_{k\geq 0} \delta_k\,\xi_{j-k} = 1_{\{j=0\}}. \tag{4.7}$$

In this formula, the equality is due to the fact that the jth coefficient of the constant function 1 (viewed as a power series in z) is zero unless $j = 0$, in which case it equals one. On the left hand side of (4.7) the sum runs from 0 to $j \wedge d$. From the assumption that $\delta(0) = 1$, it is immediate that $\xi_0 = 1$ and

$$\xi_j = -\sum_{k\geq 1} \delta_k\,\xi_{j-k}$$

for $j > 0$. Next, for any $0 \leq h \leq d-1$ we define

$$A_t(h) = \sum_{k=0}^{h} \delta_k\,\xi_{h+t-k}. \tag{4.8}$$

Note that for $h \geq d$ the formula (4.8) equals (4.7), and hence equals zero unless $h + t = 0$; but because $h < d$ in the definition of $A_t(h)$, the function is non-zero. Also, when $t \leq 0$ we have $\xi_{h+t-k} = 0$ for $k > h$, so that

$$A_t(h) = \sum_{k=0}^{d} \delta_k\,\xi_{h+t-k} = 1_{\{h+t=0\}} = 1_{\{t=-h\}}.$$

Using these definitions, we can state the following result.

Theorem 3 *The solution for $t \geq 1-d$ to $\delta(L)X_t = \partial X_t$ is given by*

$$X_t = \sum_{h=0}^{d-1} A_t(h)\,X_{-h} + \sum_{j=0}^{t-1} \xi_j\,\partial X_{t-j}, \tag{4.9}$$

where the coefficients ξ_j are defined recursively through (4.7), and the time-varying functions $A_t(h)$ are defined via (4.8). Moreover, the algebraic identity

$$1 - \sum_{h=0}^{d-1} A_h(t)\,z^{t+h} = \sum_{k=0}^{t-1} \xi_k\,z^k\,\delta(B) \tag{4.10}$$

holds, and hence the spectral representation for $\{X_t\}$ is

$$X_t = \sum_{h=0}^{d-1} A_t(h)\,X_{-h} + \sum_{k=0}^{t-1} \xi_k\,\mu + \int_{-\pi}^{\pi} \frac{e^{i\omega t} - \sum_{h=0}^{d-1} A_t(h)\,e^{-i\omega h}}{\delta(e^{-i\omega})}\,\mathbb{Z}(d\omega). \tag{4.11}$$

Proof of Theorem 3. We begin by proving (4.10), from which the other results follow. First write

$$\xi(z) = [\xi(z)]_0^{t-1} + [\xi(z)]_t^{\infty}$$

and multiply by $\delta(z)$, yielding

$$[\xi(z)]_0^{t-1}\,\delta(z) = 1 - [\xi(z)]_t^{\infty}\,\delta(z)$$

via application of (4.6). Next,

$$\begin{aligned}
[\xi(z)]_t^\infty \, \delta(z) &= \sum_{\ell \geq t} \xi_\ell \, z^\ell \sum_{k=0}^{d} \delta_k \, z^k \\
&= \sum_{k,\ell \geq 0} \delta_k \, \xi_{\ell+t} \, z^{k+\ell+t} \\
&= z^t \sum_{h \geq 0} \left(\sum_{\ell \geq 0} \delta_{h-\ell} \, \xi_{\ell+t} \right) z^h \\
&= z^t \sum_{h \geq 0} \left(\sum_{k=0}^{h} \delta_k \, \xi_{h+t-k} \right) z^h.
\end{aligned}$$

We see that the coefficient of z^h is either $1_{\{h+t=0\}}$ for $h \geq d$ or equals $A_h(t)$ for $0 \leq h \leq d-1$. Hence for $t \geq 1-d$ the calculation simplifies to

$$[\xi(z)]_t^\infty \, \delta(z) = z^t \sum_{h=0}^{d-1} A_h(t) \, z^h,$$

from which (4.10) follows. Next, multiply both sides of (4.10) by $\delta(z)$, replace z by L, and apply the resulting power series to $\{X_t\}$. Using $\delta(L)X_t = \partial X_t$, this yields

$$\sum_{j=0}^{t-1} \xi_j \, \partial X_{t-j} = X_t - \sum_{h=0}^{d-1} A_h(t) \, X_{-h},$$

from which (4.9) follows. The spectral representation is obtained from (4.9) as follows:

$$\begin{aligned}
X_t &= \sum_{h=0}^{d-1} A_t(h) \, X_{-h} + \sum_{j=0}^{t-1} \xi_j \, \left(\mu + \int_{-\pi}^{\pi} e^{i\omega(t-j)} \, \mathbb{Z}(d\omega) \right) \\
&= \sum_{h=0}^{d-1} A_t(h) \, X_{-h} + \sum_{j=0}^{t-1} \xi_j \, \mu + \int_{-\pi}^{\pi} e^{i\omega t} \sum_{j=0}^{t-1} \xi_j \, e^{-i\omega j} \, \mathbb{Z}(d\omega),
\end{aligned}$$

from which (4.11) follows. □

Theorem 3 shows how a non-stationary process can be represented in terms of a predictable portion – determined by the functions $A_h(t)$ and the variables $X_{1-d}, \ldots, X_{-1}, X_0$ – and a non-predictable portion involving a time-varying filter of the $\{\partial X_t\}$ series. The time-varying function $\sum_{k=0}^{t-1} \xi_k$ can be computed by evaluating (4.10) at $z = 1$ and dividing by $\delta(1)$ so long as this is non-zero. Otherwise, if $\delta(1) = 0$ we can use L'Hopital's rule to obtain

$$\sum_{k=0}^{t-1} \xi_k = \sum_{k=0}^{t-1} \xi_k \, z^k|_{z=1} = \frac{-\sum_{h=0}^{d-1} A_h(t) \, (t+h)}{\dot{\delta}(1)}.$$

Each of the time-varying functions $A_h(t)$ is annihilated by $\delta(L)$, i.e., $\delta(L)A_h(t) = 0$ for $0 \leq h \leq d-1$. As a consequence, we can rewrite each $A_h(t)$ as a linear combination of the basis functions of the null space of $\delta(L)$, which are given by ζ^{-t} for non-repeated roots ζ of $\delta(z)$ (when the roots are repeated, we instead consider functions $t\,\zeta - t$, etc.). Let the basis functions be denoted $\phi_k(t)$ for $1 \leq k \leq d$; see Brockwell and Davis (1991) for additional details about difference equations.

Then we can write $A_h(t) = \sum_{k=1}^{d} \alpha_{hk}\phi_k(t)$ for each $0 \leq h \leq d-1$, for some coefficients α_{hk}. It follows that

$$\sum_{h=0}^{d-1} A_h(t)\, z^h = \sum_{k=1}^{d} \left(\sum_{h=0}^{d-1} \alpha_{hk} z^h \right) \phi_k(t).$$

Each expression in parentheses on the right hand side is a degree $d-1$ polynomial in z, and will henceforth be denoted as $p^{(k)}(z)$. Substituting the new formulation, we obtain

$$X_t = \sum_{k=1}^{d} \phi_k(t) p^{(k)}(L)\, X_0 + \sum_{k=0}^{t-1} \xi_k\, \mu + \int_{-\pi}^{\pi} \frac{e^{i\omega t} - \sum_{k=1}^{d} \phi_k(t)\, p^{(k)}(e^{-i\omega})}{\delta(e^{-i\omega})}\, d\mathbb{Z}(\omega), \tag{4.12}$$

where $p^{(k)}(L)$ acts on x_0 by shifting the time index $t = 0$ back in time for each power of L. This representation allows us to understand the action of a filter on a non-stationary time series, as the following result demonstrates.

Proposition 7 *The application of a filter $\Psi(L)$ to a non-stationary process $\{X_t\}$ with representation (4.12) has spectral representation*

$$\begin{aligned}\Psi(B)X_t = &\sum_{k=1}^{d} \Psi(L)\phi_k(t)\, p^{(k)}(L)\, X_0 + \sum_{k\geq 1} \Psi(L)\xi_{t-k}\, \mu \\ &+ \int_{-\pi}^{\pi} \frac{e^{i\omega t}\, \Psi(e^{-i\omega}) - \sum_{k=1}^{d} \Psi(L)\phi_k(t)\, p^{(k)}(e^{-i\omega})}{\delta(e^{-i\omega})}\, \mathbb{Z}(d\omega).\end{aligned}$$

Depending on the action of $\Psi(L)$ on each $\phi_k(t)$, the order of integration for the output process can be less than that of $\{X_t\}$. For instance, if $\Psi(\zeta) = 0$ for some root ζ of $\delta(z)$, then at least one of the basis functions $\phi_k(t)$ is annihilated by $\Psi(L)$, which means we can rewrite the representation in Proposition 7 in terms of $d-1$ instead of d basis functions. We now consider a class of filters known as model-based (MB) filters, since they arise as MSE optimal linear filters for a certain class of linear signal extraction problems.

Suppose that the process $\{X_t\}$ is viewed as the sum of two latent processes $\{S_t\}$ and $\{N_t\}$, labelled as signal and noise respectively. These patronymics indicate merely that the signal is a process one desires to estimate, or extract, whereas the noise is to be expunged. There is no necessity that trend non-stationarity must pertain to the signal – in fact, for the problem of business cycle analysis, where one wishes to extract a stationary business cycle component, the noise will include trend as well as seasonal effects. In general, because

$$X_t = S_t + N_t$$

and $\delta(L)X_t$ is stationary, it is necessary that $\delta(L)S_t$ and $\delta(L)N_t$ are stationary, although these not need be the minimal degree differencing operators. We assume there is a relatively prime factorization of $\delta(z)$ into components $\delta^S(z)$ and $\delta^N(z)$, which are polynomials of degrees d_S and d_N that reduce signal and noise to stationarity:

$$\partial S_t = \delta^S(L)S_t \qquad \partial N_t = \delta^N(L)N_t.$$

These time series are all n-dimensional, but we assume that the same differencing operators are relevant for each component series. It then follows that

$$\partial X_t = \delta^N(L)\, \partial S_t + \delta^S(L)\, \partial N_t,$$

which allows us to relate the spectral density $f_{\partial X}$ of $\{\partial X_t\}$ to the spectral densities $f_{\partial S}$ and $f_{\partial N}$ of $\{\partial S_t\}$ and $\{\partial N_t\}$:

$$f_{\partial X}(\omega) = \left|\delta^N(e^{-i\omega})\right|^2 f_{\partial S}(\omega) + \left|\delta^S(e^{-i\omega})\right|^2 f_{\partial N}(\omega). \tag{4.13}$$

Each of these spectral densities is an $n \times n$-dimensional Hermitian function of ω. Next, dividing (4.13) through by $\left|\delta(e^{-i\omega})\right|^2$, and defining the pseudo-spectral densities by

$$f_X(\omega) = \frac{f_{\partial X}(\omega)}{\left|\delta(e^{-i\omega})\right|^2} \quad f_S(\omega) = \frac{f_{\partial S}(\omega)}{\left|\delta^S(e^{-i\omega})\right|^2} \quad f_N(\omega) = \frac{f_{\partial N}(\omega)}{\left|\delta^N(e^{-i\omega})\right|^2}, \tag{4.14}$$

we obtain the relation

$$f_X = f_S + f_N.$$

Next, the objective of MB signal extraction is to find a linear estimator for the jth component of S_t, for each $1 \leq j \leq n$. That is, we seek to find a filter $\Psi(L)$ such that $\widehat{S}_{t,j} = \Psi(L)X_t$ has minimal mean squared estimation error of $S_{t,j}$. The key result of McElroy and Trimbur (2015) is that the filter is obtained by the formula

$$\Psi(e^{-i\omega}) = e_j' \, f_{\partial S}(\omega) \, f_{\partial X}(\omega)^{-1} \left|\delta^N(e^{-i\omega})\right|^2.$$

This presumes that $f_{\partial X}(\omega)$ is invertible; given this condition, one computes each filter coefficient by Fourier inversion of the frf. Furthermore, we can use (4.14) to obtain a simpler expression:

$$\Psi(e^{-i\omega}) = e_j' \, f_S(\omega) \, f_X(\omega)^{-1}.$$

It is interesting that we do not require that $f_{\partial S}(\omega)$ be invertible – this will be further explored in Chapter **??**. We now discuss several examples of MB filters.

Example 13 Model-Based Random Walk Trend. The Local Level Model (LLM) discussed in Harvey (1989) is capable of modeling a time series consisting of a random walk trend $\{S_t\}$ and a white noise irregular $\{N_t\}$, such that $X_t = S_t + N_t$. Hence $\{S_t\}$ is $I(1)$, and $\partial S_t = (1-L)S_t$. Thus $\delta^S(z) = 1 - z$, but $\{N_t\}$ is stationary so that $\delta^N(z) = 1$. Both the multivariate trend and irregular are driven by independent white noise processes, with respective covariance matrices Σ_S and Σ_N, and it follows that the spectra for the differenced processes are

$$f_{\partial S}(\omega) = \Sigma_S \qquad f_N(\omega) = \Sigma_N.$$

Therefore the frf for the optimal trend extraction filter is

$$\Psi(e^{-i\omega}) = e_j' \, \Sigma_S \left[\Sigma_S + (2 - 2\cos(\omega))\,\Sigma_N\right]^{-1},$$

which utilizes (4.13).

Exercise 13 LLM Model-Based Trend Filter. For a bivariate LLM of Example 13 with parameters

$$\Sigma_S = 10^{-4} \begin{bmatrix} 2.32 & 5.04 \\ 5.04 & 34.73 \end{bmatrix} \qquad \Sigma_N = 10^{-5} \begin{bmatrix} 110.44 & 7.17 \\ 7.17 & 128.57 \end{bmatrix},$$

numerically compute and plot the trend extraction filter's frf.

```
> psi.sim <- c(2.17150287559847, -8.36795922528, -6.04133725367594,
+              0.0648981656699, -6.80849700177184, -6.66004335288479,
+              -0.00016098322952, 0.00051984185863)
> psi.sim[7:8] <- c(0,0)
> N <- 2
> grid <- 1000
> delta <- array(t(c(1,-1)) %x% diag(N),c(N,N,2))
> mu.sim <- mdfa.wnsim(psi.sim[1:3],rep(1,N),10,Inf)
> Sigma.mu <- mu.sim[[2]]
> irr.sim <- mdfa.wnsim(psi.sim[4:6],rep(1,N),10,Inf)
> Sigma.irr <- irr.sim[[2]]
> #print(Sigma.mu)
> #print(Sigma.irr)
>
> iden <- array(diag(N),c(N,N,1))
> f.mu <- mdfa.spectra(iden,iden,Sigma.mu,grid)
> f.irr <- mdfa.spectra(iden,iden,Sigma.irr,grid)
> trend.frf <- mdfa.wkfrf(iden,delta,f.irr,f.mu)
```

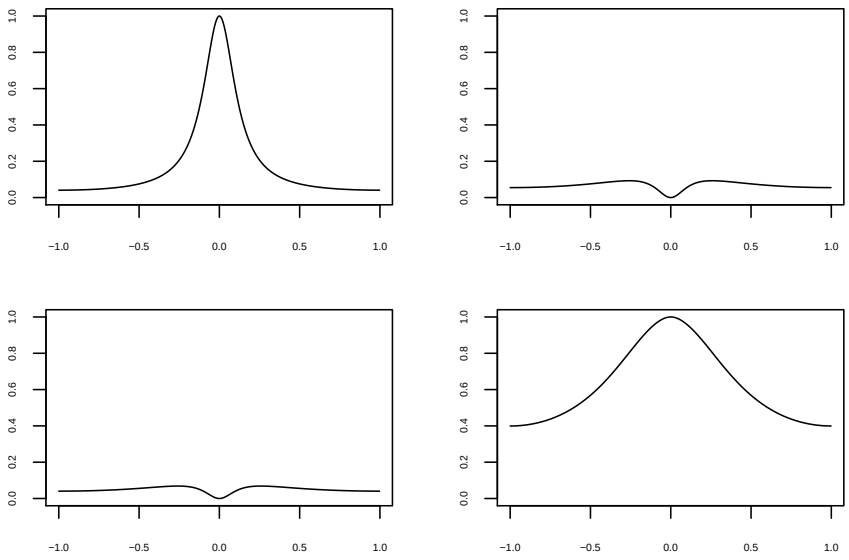

Figure 4.1: Frequency response function for model-based trend extraction filter from the Local Level Model.

Example 14 Model-Based Integrated Random Walk Trend. Example 13 can be generalized to the Smooth Trend Model (STM) developed in Harvey (1989), where now the trend $\{S_t\}$ is an integrated random walk, i.e., $(1-L)^2 S_t$ is white noise of covariance matrix Σ_S. Then the frf

for the optimal trend extraction filter – which also coincides with the multivariate HP filter – is given by

$$\Psi(e^{-i\omega}) = e_j' \, \Sigma_S \left[\Sigma_S + (2 - 2\cos(\omega))^2 \, \Sigma_N \right]^{-1}.$$

The chief difference with the frf of the LLM is that the sinusoidal factor is now squared.

Exercise 14 STM Model-Based Trend Filter. For a bivariate STM of Example 14 with parameters

$$\Sigma_S = 10^{-5} \begin{bmatrix} .66 & 1.25 \\ 1.25 & 2.92 \end{bmatrix} \qquad \Sigma_N = 10^{-4} \begin{bmatrix} 2.52 & 1.67 \\ 1.67 & 35.70 \end{bmatrix},$$

numerically compute and plot the trend extraction filter's frf.

```
> psi.sim <- c(1.8905590615422, -11.9288577633298, -12.0809347541079,
+              0.660897814610799, -8.2863379601304, -5.66645335346871,
+              -1.34743227511595e-05, -1.41207967213544e-05)
> psi.sim[7:8] <- c(0,0)
> N <- 2
> grid <- 1000
> delta <- array(t(c(1,-2,1)) %x% diag(N),c(N,N,3))
> mu.sim <- mdfa.wnsim(psi.sim[1:3],rep(1,N),10,Inf)
> Sigma.mu <- mu.sim[[2]]
> irr.sim <- mdfa.wnsim(psi.sim[4:6],rep(1,N),10,Inf)
> Sigma.irr <- irr.sim[[2]]
> #print(Sigma.mu)
> #print(Sigma.irr)
>
> iden <- array(diag(N),c(N,N,1))
> f.mu <- mdfa.spectra(iden,iden,Sigma.mu,grid)
> f.irr <- mdfa.spectra(iden,iden,Sigma.irr,grid)
> trend.frf <- mdfa.wkfrf(iden,delta,f.irr,f.mu)
```

Example 15 Model-Based Seasonal Adjustment. Flexible structural models were discussed in McElroy (2017), with atomic components for each distinct unit root (with any conjugate roots) in the differencing operator. For monthly data where $\delta(L) = (1-L)(1-L^{12})$, we obtain an integrated random walk trend component $\{C_t\}$ (identical to the trend discussed in Example 14) and six atomic seasonal components that combine into a single seasonal component $\{P_t\}$ with differencing operator $U(L) = 1 + L + L^2 + \ldots + L^{11}$, along with the irregular $\{I_t\}$, which is a white noise. Six separate covariance matrices govern the dynamics of the seasonal component, allowing for different degrees of smoothness at each of the six seasonal frequencies. In particular, we have

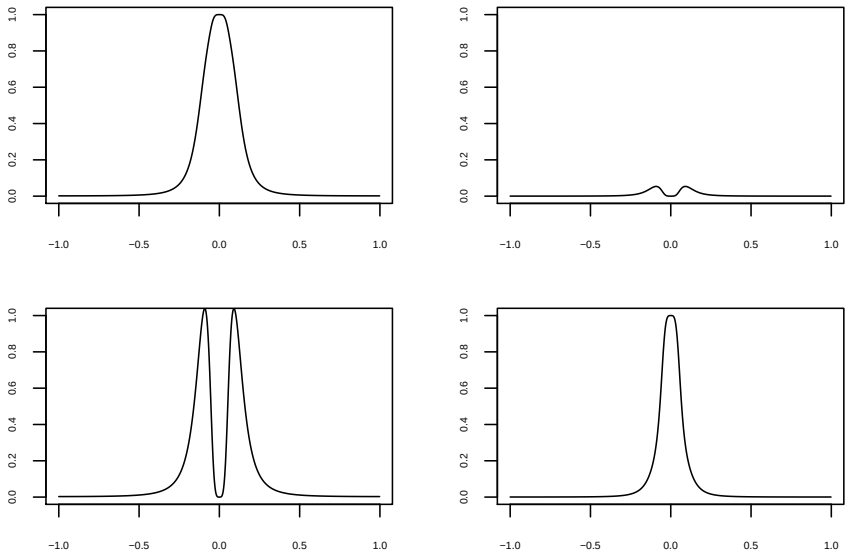

Figure 4.2: Frequency response function for model-based trend extraction filter from the Smooth Trend Model.

$P_t = \sum_{\ell=1}^{6} P_t^{(\ell)}$ and

$$\begin{aligned}
(1-L)^2 C_t &= \partial C_t \\
(1-2\cos(\pi/6)L+L^2)P_t^{(1)} &= \partial P_t^{(1)} \\
(1-2\cos(2\pi/6)L+L^2)P_t^{(2)} &= \partial P_t^{(2)} \\
(1-2\cos(3\pi/6)L+L^2)P_t^{(3)} &= \partial P_t^{(3)} \\
(1-2\cos(4\pi/6)L+L^2)P_t^{(4)} &= \partial P_t^{(4)} \\
(1-2\cos(5\pi/6)L+L^2)P_t^{(5)} &= \partial P_t^{(5)} \\
(1+L)P_t^{(6)} &= \partial P_t^{(6)}.
\end{aligned}$$

Each of the stationary processes is assumed to be an independent white noise, where the covariance matrices are denoted by Σ_C, Σ_1, Σ_2, Σ_3, Σ_4, Σ_5, Σ_6, and Σ_I respectively. For seasonal adjustment we seek to suppress seasonality, so $S_t = C_t + I_t$ and $N_t = P_t$. Thus $\delta^S(z) = (1-z)^2$ and $\delta^N(z) = U(z)$, and the MB seasonal adjustment filter has frf

$$\Psi(e^{-i\omega}) = e_j' \left(\Sigma_C + |1-e^{-i\omega}|^4 \Sigma_I\right) f_{\partial X}(\omega)^{-1} |U(e^{-i\omega})|^2.$$

Exercise 15 Structural Model-Based Seasonal Adjustment. Consider a quadvariate struc-

tural model of Example 15 with parameters

$$\Sigma_C = 10^{-2}\begin{bmatrix} 9.54 & 4.71 & 1.70 & 3.26 \\ 4.71 & 2.74 & 1.01 & 1.96 \\ 1.70 & 1.01 & 0.49 & 0.81 \\ 3.26 & 1.96 & 0.81 & 1.70 \end{bmatrix}$$

$$\Sigma_1 = 10^{-2}\begin{bmatrix} 7.97 & 6.98 & 1.77 & 3.99 \\ 6.98 & 7.47 & 2.01 & 4.57 \\ 1.77 & 2.01 & 0.80 & 1.50 \\ 3.99 & 4.57 & 1.50 & 3.85 \end{bmatrix}$$

$$\Sigma_2 = 10^{-2}\begin{bmatrix} 1.76 & 0.17 & 0.35 & 1.49 \\ 0.17 & 0.94 & 0.48 & 0.72 \\ 0.35 & 0.48 & 1.17 & 1.58 \\ 1.49 & 0.72 & 1.58 & 4.49 \end{bmatrix}$$

$$\Sigma_3 = 10^{-2}\begin{bmatrix} 4.71 & 4.35 & -1.87 & 1.15 \\ 4.35 & 4.89 & -2.04 & 1.38 \\ -1.87 & -2.04 & 1.55 & -0.04 \\ 1.15 & 1.38 & -0.04 & 1.43 \end{bmatrix}$$

$$\Sigma_4 = 10^{-2}\begin{bmatrix} 12.56 & 3.30 & -2.28 & 1.88 \\ 3.30 & 3.49 & -0.88 & 1.06 \\ -2.28 & -0.88 & 0.78 & -0.45 \\ 1.88 & 1.06 & -0.45 & 0.88 \end{bmatrix}$$

$$\Sigma_5 = 10^{-2}\begin{bmatrix} 1.07 & 1.51 & -0.01 & -0.24 \\ 1.51 & 3.04 & 0.01 & 0.79 \\ -0.01 & 0.01 & 0.09 & 0.05 \\ -0.24 & 0.79 & 0.05 & 1.69 \end{bmatrix}$$

$$\Sigma_6 = 10^{-2}\begin{bmatrix} 11.79 & -1.17 & 1.38 & 1.86 \\ -1.17 & 2.77 & -0.23 & 0.30 \\ 1.38 & -0.23 & 0.77 & 0.40 \\ 1.86 & 0.30 & 0.40 & 3.11 \end{bmatrix}$$

$$\Sigma_I = \begin{bmatrix} 6.53 & 0.81 & 0.19 & -0.57 \\ 0.81 & 1.26 & 0.23 & 0.49 \\ 0.19 & 0.23 & 0.30 & 0.15 \\ -0.57 & 0.49 & 0.15 & 1.08 \end{bmatrix}.$$

Numerically compute and plot the filter frf for seasonal adjustment.

```
> psi.sim <- c(0.493586093056948, 0.178487258592539, 0.341217399125708,
+              0.399177274154249, 0.848325304642642, 0.68306879252262,
+              -2.3494687111314, -5.47534663726587, -6.69385117951384,
+              -6.08364145983965, 0.875100150810273, 0.221971271148611,
+              0.500866759201029, 0.340625016984097, 0.791037805495801,
```

```
+              0.985440262768576, -2.52890913740106, -4.29524634814519,
+              -5.98519527750281, -4.88659954275053, 0.0957466327314851,
+              0.201313350626488, 0.849351809157598, 0.48420520104336,
+              0.62643997675928, 1.13945063379914, -4.04217214895869,
+              -4.68919816059416, -4.73313805629826, -4.0627015759002,
+              0.923495751608401, -0.396067294450726, 0.244665046194039,
+              -0.36570474542918, 0.363995718736632, 0.758715172737758,
+              -3.05567431351817, -4.74337970092605, -4.96364133429136,
+              -5.06144086942249, 0.262963683605793, -0.181599400661918,
+              0.149795833258992, -0.105991649100357, 0.21503766242974,
+              -0.141649861043968, -2.07489346121933, -3.64302004053168,
+              -5.69277788172285, -5.3689470753418, 1.40718934367933,
+              -0.0085452878747676, -0.219886337273936, 0.0283662345070971,
+              1.23786259577472, 0.199834135215749, -4.53336362894347,
+              -4.70016052568401, -7.07530853221777, -6.03054443735399,
+              -0.0995506040524902, 0.116607848697947, 0.157899802233636,
+              -0.0363184981547607, 0.18385749297074, 0.329351477585333,
+              -2.1377604820296, -3.62882764786239, -5.11279846492415,
+              -3.62475631527416, 0.124305286145147, 0.0292507920421885,
+              -0.0873349194845382, 0.178977764316143, 0.484389128732254,
+              0.265835976421986, 1.87566939226944, 0.1445002084775,
+              -1.34264222816582, -0.305367634014929, -0.00488431480035087,
+              -0.000945659564684563, -0.00106126820173145, -0.000413658838890233)
> psi.sim[81:84] <- c(0,0,0,0)
> N <- 4
> grid <- 1000
> mu.sim <- mdfa.wnsim(psi.sim[1:10],rep(1,N),10,Inf)
> Sigma.mu <- mu.sim[[2]]
> seas1.sim <- mdfa.wnsim(psi.sim[11:20],rep(1,N),10,Inf)
> Sigma.seas1 <- seas1.sim[[2]]
> seas2.sim <- mdfa.wnsim(psi.sim[21:30],rep(1,N),10,Inf)
> Sigma.seas2 <- seas2.sim[[2]]
> seas3.sim <- mdfa.wnsim(psi.sim[31:40],rep(1,N),10,Inf)
> Sigma.seas3 <- seas3.sim[[2]]
> seas4.sim <- mdfa.wnsim(psi.sim[41:50],rep(1,N),10,Inf)
> Sigma.seas4 <- seas4.sim[[2]]
> seas5.sim <- mdfa.wnsim(psi.sim[51:60],rep(1,N),10,Inf)
> Sigma.seas5 <- seas5.sim[[2]]
> seas6.sim <- mdfa.wnsim(psi.sim[61:70],rep(1,N),10,Inf)
> Sigma.seas6 <- seas6.sim[[2]]
> irr.sim <- mdfa.wnsim(psi.sim[71:80],rep(1,N),10,Inf)
```

```
> Sigma.irr <- irr.sim[[2]]
> #print(Sigma.mu)
> #print(Sigma.seas1)
> #print(Sigma.seas2)
> #print(Sigma.seas3)
> #print(Sigma.seas4)
> #print(Sigma.seas5)
> #print(Sigma.seas6)
> #print(Sigma.irr)
>
> iden <- array(diag(N),c(N,N,1))
> dpoly.1 <- c(1,-2*cos(pi/6),1)
> dpoly.2 <- c(1,-2*cos(2*pi/6),1)
> dpoly.3 <- c(1,-2*cos(3*pi/6),1)
> dpoly.4 <- c(1,-2*cos(4*pi/6),1)
> dpoly.5 <- c(1,-2*cos(5*pi/6),1)
> dpoly.6 <- c(1,1)
> dpoly.but1 <- polymult(dpoly.2,polymult(dpoly.3,polymult(dpoly.4,polymult(dpoly.5,dpoly.6))))
> dpoly.but2 <- polymult(dpoly.1,polymult(dpoly.3,polymult(dpoly.4,polymult(dpoly.5,dpoly.6))))
> dpoly.but3 <- polymult(dpoly.1,polymult(dpoly.2,polymult(dpoly.4,polymult(dpoly.5,dpoly.6))))
> dpoly.but4 <- polymult(dpoly.1,polymult(dpoly.2,polymult(dpoly.3,polymult(dpoly.5,dpoly.6))))
> dpoly.but5 <- polymult(dpoly.1,polymult(dpoly.2,polymult(dpoly.3,polymult(dpoly.4,dpoly.6))))
> dpoly.but6 <- polymult(dpoly.1,polymult(dpoly.2,polymult(dpoly.3,polymult(dpoly.4,dpoly.5))))
> delta.c <- array(t(c(1,-2,1)) %x% diag(N),c(N,N,3))
> delta.but1 <- array(t(dpoly.but1) %x% diag(N),c(N,N,10))
> delta.but2 <- array(t(dpoly.but2) %x% diag(N),c(N,N,10))
> delta.but3 <- array(t(dpoly.but3) %x% diag(N),c(N,N,10))
> delta.but4 <- array(t(dpoly.but4) %x% diag(N),c(N,N,10))
> delta.but5 <- array(t(dpoly.but5) %x% diag(N),c(N,N,10))
> delta.but6 <- array(t(dpoly.but6) %x% diag(N),c(N,N,11))
> delta.seas <- array(t(rep(1,12)) %x% diag(N),c(N,N,12))
> f.mu <- mdfa.spectra(iden,iden,Sigma.mu,grid)
> f.seas1 <- mdfa.spectra(iden,delta.but1,Sigma.seas1,grid)
> f.seas2 <- mdfa.spectra(iden,delta.but2,Sigma.seas2,grid)
> f.seas3 <- mdfa.spectra(iden,delta.but3,Sigma.seas3,grid)
> f.seas4 <- mdfa.spectra(iden,delta.but4,Sigma.seas4,grid)
> f.seas5 <- mdfa.spectra(iden,delta.but5,Sigma.seas5,grid)
> f.seas6 <- mdfa.spectra(iden,delta.but6,Sigma.seas6,grid)
> f.irr <- mdfa.spectra(iden,delta.c,Sigma.irr,grid)
> f.signal <- f.mu + f.irr
> f.noise <- f.seas1 + f.seas2 + f.seas3 + f.seas4 + f.seas5 + f.seas6
```

```
> sa.frf <- mdfa.wkfrf(delta.seas,delta.c,f.noise,f.signal)
```

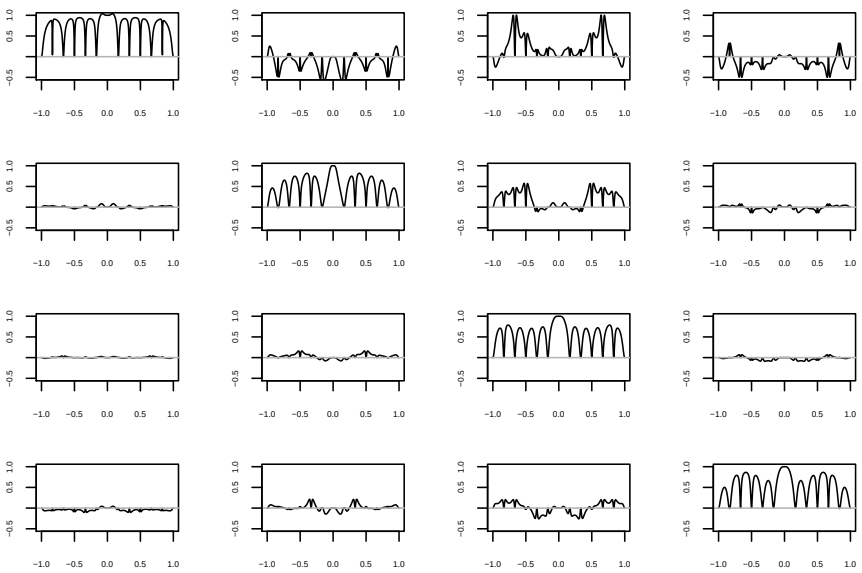

Figure 4.3: Frequency response function for model-based seasonal adjustment filter from the Structural Model.

4.3 Error Criterion and Computation

Now supposing that $\{X_t\}$ is non-stationary, examination of (2.7) indicates that the error process is not stationary unless we make certain assumptions about $\Delta(L) = \Psi(L) - \widehat{\Psi}(L)$. In order to remove the time-varying functions it is necessary that we can factor $\delta(L)$ from $\Delta(L)$, i.e., we require the existence of $\widetilde{\Delta}(L)$ such that

$$\Delta(L) = \widetilde{\Delta}(L)\,\delta(L), \tag{4.15}$$

as otherwise we cannot guarantee that $\{E_t\}$ will be stationary. Moreover, if $\mu = 0$ then (4.15) is also sufficient to guarantee that the filter error be stationary, because

$$E_t = \widetilde{\Delta}(L)\,\partial X_t$$

in such a case. We next discuss a set of filter constraints that guarantee (4.15), beginning with a result that discusses the factorization of filters. We say a filter $\Psi(L)$ is absolutely convergent if $\sum_{j\in\mathbb{Z}} \|\psi(j)\| < \infty$ for a given matrix norm $\|\cdot\|$.

Proposition 8 *Any linear filter $\Psi(L)$ can be expressed as*

$$\Psi(L) = \Psi(\zeta) + (L-\zeta)\,\Psi^{\sharp}(L)$$

for any $\zeta \in \mathbb{C}$ such that $|\zeta| = 1$, and an absolutely convergent filter $\Psi^\sharp(L)$, so long as $\partial\Psi(L)$ is absolutely convergent. If in addition $\partial\partial\Psi(L) = \sum_{j\in\mathbb{Z}} j(j-1)\,\psi(j)\,L^j$ is absolutely convergent, then there also exists an absolutely convergent filter $\Psi^\flat(L)$ such that

$$\Psi(L) = \Psi(\zeta) + \partial\Psi(\zeta)\,(L-\zeta)\,\overline{\zeta} + (L-\zeta)^2\,\Psi^\flat(L).$$

Proof of Proposition 8. We claim that $\Psi^\sharp(L) = \sum_{j\in\mathbb{Z}} \psi^\sharp(j)\,L^j$ with

$$\psi^\sharp(j) = \begin{cases} \zeta^{-(j+1)} \sum_{k\geq j+1} \psi(k)\,\zeta^k & j \geq 0 \\ -\zeta^{-(j+1)} \sum_{k\geq -j} \psi(-k)\,\zeta^{-k} & j \leq -1. \end{cases}$$

To show this, first observe that

$$\Psi(L) - \Psi(\zeta) = \sum_{j\geq 1} \psi(j)\,(L^j - \zeta^j) + \sum_{j\leq -1} \psi(j)\,(L^j - \zeta^j).$$

Beginning with the first term, so that $j \geq 1$, we write $L^j - \zeta^j = \zeta^j\,(L/\zeta - 1)\,p_{j-1}(L/\zeta)$ where $p_k(z) = \sum_{\ell=0}^k z^\ell$. Next, by coefficient matching we can verify that

$$\begin{aligned} \sum_{j\geq 1} \psi(j)\,(L^j - \zeta^j) &= (L/\zeta - 1) \sum_{j\geq 1} \psi(j)\,\zeta^j\,p_{j-1}(L/\zeta) \\ &= (L/\zeta - 1) \sum_{j\geq 0} \sum_{k\geq j+1} \psi(k)\,\zeta^k\,(L/\zeta)^j = (L-\zeta) \sum_{j\geq 0} \psi^\sharp(j)\,L^j. \end{aligned}$$

Next, take $j \leq -1$, and use the symbol $F = L^{-1}$:

$$\begin{aligned} \sum_{j\leq -1} \psi(j)\,(L^j - \zeta^j) &= \sum_{j\geq 1} \psi(-j)\,(F^j - \zeta^{-j}) = (F\zeta - 1) \sum_{j\geq 1} \psi(-j)\,\zeta^{-j}\,p_{j-1}(F\zeta) \\ &= (F\zeta - 1) \sum_{j\geq 0} \sum_{k\geq j+1} \psi(-k)\,\zeta^{-k}\,(F\zeta)^j \\ &= -F\,(L-\zeta) \sum_{j\geq 1} \zeta^{j-1} \sum_{k\geq j} \psi(-k)\,\zeta^{-k}\,F^{j-1} \\ &= (L-\zeta) \sum_{j\leq -1} \psi^\sharp(j)\,L^j. \end{aligned}$$

This establishes algebraically that $\Psi^\sharp(L)$ with coefficients as defined above equals $(\Psi(L)-\Psi(\zeta))/(L-\zeta)$, whenever the Laurent series converges. Based on the above calculations, we can write

$$\frac{\Psi(L) - \Psi(\zeta)}{L-\zeta} = \sum_{j\geq 1} \left(\psi(j)\,\zeta^{j-1}\,p_{j-1}(L/\zeta) - \psi(-j)\,\zeta^{-j}\,F\,p_{j-1}(F\zeta)\right).$$

To check the absolute convergence, it suffices to set $L = 1$; note that $|p_k(\zeta)| \leq (k+1)$ if $|\zeta| = 1$. Thus we obtain the bound

$$\left\| \frac{\Psi(L) - \Psi(\zeta)}{L-\zeta} \right\| \leq \sum_{j\geq 1} j\,\left(\|\psi(j)\| + \|\psi(-j)\|\right),$$

which is finite by the assumption that $\partial\Psi(L)$ is absolutely convergent. Next, we claim that $\Psi^\flat(L) = \sum_{j\in\mathbb{Z}} \psi^\flat(j)\,L^j$ with

$$\psi^\flat(j) = \begin{cases} \zeta^{-(j+2)} \sum_{k\geq j+2} (k-1-j)\,\psi(k)\,\zeta^k & j \geq 0 \\ \zeta^{-(j+2)} \sum_{k\geq -j} (k+j+1)\,\psi(-k)\,\zeta^{-k} & j \leq -1. \end{cases}$$

To verify this, observe that

$$\Psi(L) - \Psi(\zeta) - \partial\Psi(\zeta)\,(L-\zeta)\,\zeta^{-1} = \sum_{j\geq 1}\psi(j)\,\left[(L^j - \zeta^j) - j\,\zeta^{j-1}\,(L-\zeta)\right] + \sum_{j\leq -1}\psi(j)\,\left[(L^j - \zeta^j) - j\,\zeta^{j-1}\,(L-\zeta)\right].$$

First assuming that $j \geq 1$, note that $p_{\ell-1}(z) - \ell$ equals zero unless $\ell \geq 2$, and otherwise equals $\sum_{k=1}^{\ell-1} p_{k-1}(z)\,(z-1)$. Therefore

$$\begin{aligned}\sum_{j\geq 1}\psi(j)\,\left[(L^j - \zeta^j) - j\,\zeta^{j-1}\,(L-\zeta)\right] &= (L-\zeta)\sum_{j\geq 1}\psi(j)\,\zeta^{j-1}\,\left[p_{j-1}(L/\zeta) - j\right] \\ &= (L-\zeta)^2\sum_{j\geq 2}\psi(j)\,\zeta^{j-2}\sum_{k=1}^{j-1}p_{k-1}(L/\zeta) \\ &= (L-\zeta)^2\sum_{j\geq 0}\psi^\flat(j)\,L^j\end{aligned}$$

by coefficient matching in the final step. Similarly, letting $j \leq -1$ and using $zp_{\ell-1}(z) - \ell = (z-1)\sum_{k=1}^{\ell} p_{k-1}(z)$, we have

$$\begin{aligned}\sum_{j\leq -1}\psi(j)\,\left[(L^j - \zeta^j) - j\,\zeta^{j-1}\,(L-\zeta)\right] &= (L-\zeta)\sum_{j\geq 1}\psi(-j)\,\zeta^{-j}\,\left[-F\,p_{j-1}(F\zeta) + j\,\zeta^{-1}\right] \\ &= (L-\zeta)^2\,F\sum_{j\geq 1}\psi(-j)\,\zeta^{-(j+1)}\sum_{k=1}^{j}p_{k-1}(F\zeta) \\ &= (L-\zeta)^2\,F\,\zeta^{-1}\sum_{j\geq 0}\sum_{k\geq j+1}(k-j)\,\psi(-k)\,\zeta^{-k}\,(F\zeta)^j \\ &= (L-\zeta)^2\sum_{j\leq -1}\psi^\flat(j)\,L^j\end{aligned}$$

by matching coefficients. To establish convergence of the Laurent series for $\Psi^\flat(L)$, observe that

$$\frac{\Psi(L) - \Psi(\zeta) - \partial\Psi(\zeta)\,(L-\zeta)\,\zeta^{-1}}{(L-\zeta)^2} = \sum_{j\geq 2}\psi(j)\,\zeta^{j-2}\sum_{k=1}^{j-1}p_{k-1}(L/\zeta) + F\sum_{j\geq 1}\psi(-j)\,\zeta^{-(j+1)}\sum_{k=1}^{j}p_{k-1}(F\zeta).$$

Hence the matrix norm has the bound (setting $L = 1$ and taking $|\zeta| = 1$) of

$$\left\|\frac{\Psi(L) - \Psi(\zeta) - \partial\Psi(\zeta)\,(L-\zeta)\,\zeta^{-1}}{(L-\zeta)^2}\right\| \leq \sum_{j\geq 2}\|\psi(j)\|\binom{j}{2} + \sum_{j\geq 1}\|\psi(-j)\|\binom{j+1}{2},$$

using $|\sum_{k=1}^{j} p_{k-1}(\zeta)| \leq \binom{j+1}{2}$. Because $\partial\partial\Psi(L)$ is absolutely convergent, the above norm is finite. □

Note that if $\Psi(\zeta) = 0$, it follows from Proposition 8 that $L - \zeta$ can be factored from $\Psi(L)$. Similarly, $(L-\zeta)^2$ can be factored from $\Psi(L)$ if $\Psi(\zeta) = \partial\Psi(\zeta) = 0$. Next, we introduce the concept of ω-dynamics, which correspond to basis functions $\phi_k(t) = e^{i\omega t}$.

Definition 4 For $\omega \in [-\pi, \pi]$, a filter $\Psi(L)$ annihilates ω-dynamics of order 1 if $\Psi(e^{-i\omega}) = 0$, and annihilates ω-dynamics of order 2 if in addition $\partial\Psi(e^{-i\omega}) = 0$.

Hence, we have the following immediate corollary of Proposition 8.

Corollary 1 *If a filter $\Psi(L)$ annihilates ω-dynamics of order 1 and $\partial\Psi(L)$ is absolutely convergent, then*

$$\Psi(L) = (L - e^{-i\omega})\,\Psi^\sharp(L).$$

If a filter $\Psi(L)$ annihilate ω-dynamics of order 2, and $\partial\partial\Psi(L)$ is absolutely convergent, then

$$\Psi(L) = (L - e^{-i\omega})^2\,\Psi^\flat(L).$$

We can apply Corollary 1 to factor a noise-differencing polynomial $\delta^N(L)$ from $\Delta(L)$: for each ω such that the target filter $\Psi(L)$ annihilates ω-dynamics of order d, we impose the constraint that $\widehat{\Psi}(L)$ shall have the same property, and hence $(L - e^{-i\omega})^d$ can be factored from both filters. For instance, if noise frequencies are ω_ℓ with multiplicities d_ℓ, then repeated application of Corollary 1 yields

$$\Psi(L) = \prod_\ell (L - e^{-i\omega_\ell})^{d_\ell}\,\Psi^\natural(L) = \delta^N(L)\,\Psi^\star(L)$$

for some residual filter $\Psi^\natural(L)$, where $\Psi^\star(L) = \prod_\ell(-e^{-i\omega_\ell d_\ell})\,\Psi^\natural(L)$ and $\delta^N(L) = \prod_\ell(1 - e^{i\omega_\ell}\,L)$. By imposing the same linear constraints on $\widehat{\Psi}(L)$, we likewise obtain $\widehat{\Psi}(L) = \delta^N(L)\,\widehat{\Psi}^\star(L)$, and hence

$$\Delta(L) = \Big(\Psi^\star(L) - \widehat{\Psi}^\star(L)\Big)\,\delta^N(L). \tag{4.16}$$

So if $\delta(L) = \delta^N(L)$, then (4.15) holds at once. More generally, a given process' differencing polynomial may be factored into relatively prime polynomials $\delta^N(z)$ and $\delta^S(z)$, which correspond to noise and signal dynamics respectively – see Bell (1984) and McElroy (2008). Many signal extraction filters $\Psi(L)$ have the property that they annihilate ω-dynamics of the appropriate order, such that $\delta^N(L)$ can be factored. By imposing that

$$\widehat{\Psi}(e^{-i\omega}) = \Psi(e^{-i\omega}) \tag{4.17}$$

for all simple roots $\zeta = e^{-i\omega}$ of $\delta(z)$, we ensure that (4.15) holds. If there is a double root, then we also impose

$$\partial\widehat{\Psi}(e^{-i\omega}) = \partial\Psi(e^{-i\omega}). \tag{4.18}$$

This conditions are sufficient, because (4.17) and (4.18) imply $\Delta(e^{-i\omega}) = 0$ for all the roots of $\delta(z)$, so that by repeated application of Corollary 1 we can factor out $\delta(z)$, labelling the remaining factor as $\widetilde{\Delta}(L)$. Note that if ω corresponds to noise, i.e., $\delta^N(e^{-i\omega}) = 0$, then (**??**) becomes $\widehat{\Psi}(e^{-i\omega}) = 0$, but if ω corresponds to signal then $\Psi(e^{-i\omega}) \neq 0$.

In fact, this characterization actually defines signal and noise. Given a non-stationary process with differencing polynomial $\delta(z)$ and a target filter $\Psi(L)$, we define $\delta^N(z)$ as consisting of those factors of $\delta(z)$ such that $\widehat{\Psi}(e^{-i\omega}) = 0$, and set $\delta^S(z) = \delta(z)/\delta^N(z)$. For either signal or noise ω-dynamics, we know that (4.15) holds if we impose (4.17) for all the single roots of $\delta(z)$ (and for double roots, also impose (4.18)). In practice, we must determine the real and imaginary parts of each such constraint, and write the corresponding constraints on $\widehat{\Psi}(L)$ in the form $A = [J \otimes 1_n]\,\vartheta$ for filters of form (3.9), applying the methodology of this chapter's first section. With these constraints

in play, the formula (3.6) holds with $\Psi(z) - \widehat{\Psi}(z)$ replaced by $\widetilde{\Delta}(z)$ and F being the spectral density of $\{\partial X_t\}$, i.e., we define the nonstationary MDFA criterion function as $\det D_\Psi(\vartheta, G)$ for

$$D_\Psi(\vartheta, G) = \langle \widetilde{\Delta}(z)\, G\, \widetilde{\Delta}(z)^* \rangle_0 = \langle \left[\Psi(z) - \widehat{\Psi}_\vartheta(z)\right]\, G\, |\delta(z)|^{-2} \left[\Psi(z) - \widehat{\Psi}_\vartheta(z)\right]^* \rangle_0. \tag{4.19}$$

The second expression in (4.19) utilizes (4.15), and employs the understanding that poles in $\delta(z)^{-1}$ are exactly canceled out by the corresponding zeros in $\Psi(z) - \widehat{\Psi}(z)$. Moreover, the ratio $(\Psi(z) - \widehat{\Psi}(z))/\delta(z) = \widetilde{\Delta}(z)$ is bounded in ω for $z = e^{-i\omega}$, as the previous discussion guarantees.

For computation, it is convenient to calculate the integrand given in the right-hand expression of (4.19), which involves the pseudo-spectrum $G\,|\delta(z)|^{-2}$. Numerical evaluation of this integrand will yield 0/0 at ω-dynamics such that $\delta(e^{-i\omega}) = 0$, and L'Hopital's rule could be used to resolve this quotient into an expression for $\widetilde{\Delta}(z)\, G\, \widetilde{\Delta}(z)^*|_{z=e^{-i\omega}}$. However, as there are only a finite number of such resolvable singularities, and such points constitute a set of Lebesgue measure zero, calculation of $D_\Psi(\vartheta, G)$ can proceed by integrating over $\omega \in [-\pi, \pi]$ such that a neighborhood containing each singularity is omitted. This procedure can yield an expression arbitarily close to $D_\Psi(\vartheta, G)$ (by taking the neighborhoods sufficiently small), while allowing for easy calculation, since $\Delta(z)$ and $G\,|\delta(z)|^{-2}$ are easily computed.

Whereas the theoretical filter error MSE is given by $D_{\Psi,F}$, with F being the spectral density of $\{\partial X_t\}$, for estimation we approximate the integral over Fourier frequencies, and utilize the periodogram of the differenced data for G. We omit any contributions to the sum arising from Fourier frequencies that correspond to zeros of $\delta(z)$, as such an omission only results in a loss of order T^{-1}, which is of the same order as the Riemann sum approximation over Fourier frequencies to the exact integral.

HERE revisit section 1 exercises now with RW and diff - VAR(1), using LTSC. Petrol ex

```
> ############################################################
> ### Exercise 2: replication - simulate bivariate LLM and apply MDFA
>
> # Simulate a Gaussian LLM
> set.seed(1234)
> psi.sim <- c(2.17150287559847, -8.36795922528, -6.04133725367594,
+              0.0648981656699, -6.80849700177184, -6.66004335288479,
+              -0.00016098322952, 0.00051984185863)
> psi.sim[7:8] <- c(0,0)
> len <- 1000
> T.sim <- 500 + 2*len
> burn <- 1000
> N <- 2
> delta <- array(t(c(1,-1)) %x% diag(N),c(N,N,2))
> d <- dim(delta)[3] - 1
> mu.sim <- mdfa.wnsim(psi.sim[1:3],rep(1,N),T.sim+burn,Inf)
> Sigma.mu <- mu.sim[[2]]
> mu.sim <- mdfa.ucsim(delta,mu.sim[[1]])[(burn+1-d):(T.sim+burn-d),]
```

```
> irr.sim <- mdfa.wnsim(psi.sim[4:6],rep(1,N),T.sim,Inf)
> Sigma.irr <- irr.sim[[2]]
> irr.sim <- irr.sim[[1]]
> x.sim <- mu.sim + irr.sim
> plot(ts(x.sim))
```

HERE HP exercise with connection to STM, and MB replication. Ndc ex

HERE CF apply to RW

HERE Starts exercise

Chapter 5

Summary and Links

5.1 Survey of MDFA Optimization Criteria

5.2 Consistency and Efficiency: a Tale of Two Philosophies

5.2.1 Knowing the Truth: Omniscience

5.2.2 Believing in Truth: Faith and Fatalism

5.2.3 From Truth to Effectiveness: Emphasizing Performances

Bibliography

[1] Baxter and King 1999

[2] Bell 1984

[3] Brockwell and Davis 1991

[4] Golub and Van Loan, 1996

[5] Harvey 1989

[6] Hosoya and Taniguchi 1982

[7] Lutkepohl 2007

[8] McElroy 2008 ET

[9] McElroy 2017 JBES

[10] McElroy and Findley 2015

[11] McElroy and Trimbur 2015

[12] Roy McElroy Linton 2019

[13] Self and Liang 1987

[14] Taniguchi and Kakizawa 2000

[15] Wildi 2008