Multivariate Real-Time Signal Extraction

Marc Wildi and Tucker McElroy

August 14, 2020

Contents

1 Introduction **1**

1.1 Overview 1

1.1.1 Signals and Extraction 1

1.1.2 The Classic Model-Based Paradigm 2

1.1.3 The Scope of MDFA 3

1.2 The Style of the Book 4

1.2.1 Setting the Paths 4

1.2.2 DFA 5

1.2.3 MDFA 7

1.2.4 Using MDFA 10

Chapter 1

Introduction

1.1 Overview

1.1.1 Signals and Extraction

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

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

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

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

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

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

1.1.2 The Classic Model-Based Paradigm

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

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

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

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

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

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

1.1.3 The Scope of MDFA

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

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

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

1.2 The Style of the Book

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

1.2.1 Setting the Paths

Begin by clearing the workspace:

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

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

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

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

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

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

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

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

1.2.2 DFA

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

DFT and Periodogram

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

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

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

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

This function will be generalized in the new multivariate setting.

Basic DFA

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

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

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

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

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

Customized DFA

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

```
> head(dfa_analytic)
```

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

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

1.2.3 MDFA

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

Data Matrix

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

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

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

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

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

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

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

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

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

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

DFT

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

```
> spec_comp

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

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

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

1.2.4 Using MDFA

A Versatile User Interface

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

```
> head(mdfa_analytic)
```

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

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

Default Settings

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

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

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

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

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

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

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

For later use we source a convenient plotting function:

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

Selected Calls: Classic MSE, Customization and Regularization

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

```
> head(MDFA_mse)
```

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

```
> head(MDFA_mse_constraint)
```

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

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

```
> head(MDFA_cust)
```

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

```
> head(MDFA_cust_constraint)
```

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

```
> head(MDFA_reg)
```

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

```
> head(MDFA_reg_constraint)
```

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

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

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. 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 $\sum_{\ell=r}^s \psi(\ell)\, L^\ell$. 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 (**??**), 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 **??** 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.

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'.

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 $\psi(0) = \mu/\pi\, e_1'$.

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 REF???. 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).

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 **??** and **??**), whereas others may be defined in terms of all the components (Examples **??** and **??**). 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.

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.

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 (??), 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 ??. 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 (**??**) gives us a lower bound on the MSE when we use sub-optimal proxies for $\widehat{\Psi}(L)$.

As indicated by Remark **??**, the result of Proposition **??** 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).

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 (**??**) 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, 2017) provides a bijection from $\mathbb{R}^{n^2} \times \{\pm 1\}$ to the space of stable $n \times n$ matrices:

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

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 **??** 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

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

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

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

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 ??. 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 $\widetilde{\Sigma}$. □

Remark 2 One of the conditions of Proposition **??** 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*.

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 ??. 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. □

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.

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

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

and Σ equal to the identity. The eigenvalues are .8 and .5. 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 **??**) and the ideal low-pass LPP (Example **??**) 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:
> T <- 100
> phi.matrix <- rbind(c(1,.5),c(-.2,.3))
> innovar.matrix <- diag(2)
> true.psidelta <- sigex.ivarpar(array(phi.matrix,c(2,2,1)))
> 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 <- sigex.ivarpar(array(phi.yw,c(2,2,1)))
> # 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] 1.430235 1.430235

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

         [,1]
[1,] 1.430235

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

           [,1]      [,2]
[1,]  0.9348059 0.5314934
[2,] -0.5589701 0.2793141

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

           [,1]       [,2]
[1,]  0.9347882  0.5313741
[2,] -1.5157224 -1.0416827

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

           [,1]      [,2]
[1,]  0.9348056 0.5314925
[2,] -0.1783947 0.2670181

> # 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] 3.007243 3.007242

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

         [,1]
[1,] 3.017099
```

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

           [,1]      [,2]
[1,]  0.9183654 0.3693359
[2,] -0.2318667 0.5703253

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

           [,1]       [,2]
[1,] -0.3842571 -2.3329655
[2,] -0.2614505  0.1487464

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

           [,1]      [,2]
[1,]  0.9348056 0.5314925
[2,] -0.1783947 0.2670181

> # 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] 0.7782000 0.8039344

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

          [,1]
[1,] 0.7784239

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

            [,1]      [,2]
[1,]  0.82390589 0.2256008
[2,] -0.02389912 0.7338540
```

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

          [,1]          [,2]
[1,] 0.7708965 -1.432228e-05
[2,] 2.8936879 -9.986220e-01

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

           [,1]      [,2]
[1,]  0.9348056 0.5314925
[2,] -0.1783947 0.2670181

>
```

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

$$\Theta = \begin{bmatrix} 1 & 0 \\ .4 & 2 \end{bmatrix}$$

and Σ equals the identity. Use the VAR(1) LPP criterion of Exercise **??** 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:
> 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 <- sigex.ivarpar(array(phi.yw,c(2,2,1)))
> # 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] 1.865806 1.865806

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

         [,1]
[1,] 1.865806

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

          [,1]        [,2]
[1,] 0.4088079 -0.09822759
[2,] 0.1398298  0.71881066

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

           [,1]        [,2]
[1,]  0.4088452 -0.09821484
[2,] -0.9785868 -0.56418264

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

            [,1]        [,2]
[1,]  0.40880769 -0.09822759
[2,] -0.05899914  0.33755351

> # 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] 2.230713 2.296949

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

         [,1]
[1,] 2.467927

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

          [,1]       [,2]
[1,] 0.0758301 -0.2361368
[2,] 0.6490909  0.1834502

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

            [,1]        [,2]
[1,] -0.05185665 -0.02473491
[2,] -0.06786026  0.45889028

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

            [,1]        [,2]
[1,]  0.40880769 -0.09822759
[2,] -0.05899914  0.33755351

> # 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] 0.04783541 0.04782558

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

           [,1]
[1,] 0.05092779

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

            [,1]        [,2]
[1,] -0.06199006  0.07558949
[2,] -0.98694748 -0.38815893

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

           [,1]      [,2]
[1,] -0.1448381 0.0455870
[2,]  0.7773915 0.5006811

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

            [,1]        [,2]
[1,]  0.40880769 -0.09822759
[2,] -0.05899914  0.33755351

>
```

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

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

where

$$\Theta_1 = \begin{bmatrix} .9 & 2 \\ 0 & .8 \end{bmatrix}$$

$$\Theta_2 = \begin{bmatrix} .8 & 3.5 \\ .2 & .6 \end{bmatrix}$$

and Σ equals the identity. Use the VAR(1) LPP criterion of Exercise **??** 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:
> 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 <- sigex.ivarpar(array(phi.yw,c(2,2,1)))
> # 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] 8.036465 8.036465
```

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

         [,1]
[1,] 8.036465

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

           [,1]       [,2]
[1,] -0.0220511  2.3982423
[2,] -0.1852666 -0.5516525

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

            [,1]        [,2]
[1,] -0.02205007  2.39823878
[2,]  0.19520816 -0.09218997

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

            [,1]      [,2]
[1,] -0.02205180 2.3982434
[2,] -0.08104211 0.7708112

> # 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] 10.44909 13.36460

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

         [,1]
[1,] 11.67957
```

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

            [,1]     [,2]
[1,]  0.08169268 1.852166
[2,] -0.26545460 1.355849

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

           [,1]     [,2]
[1,] -0.4381731 1.348005
[2,] -0.3903974 1.365920

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

            [,1]      [,2]
[1,] -0.02205180 2.3982434
[2,] -0.08104211 0.7708112

> # select choices of phi from above
> if(var1.fit.2step.Pos$value < var1.fit.2step.Neg$value) {
+         phi.lpp <- sigex.varpar(var1.fit.2step.Pos$par,1,2,1)
+ } else {
+         phi.lpp <- sigex.varpar(var1.fit.2step.Neg$par,1,2,-1)
+ }
> phi.lpp <- phi.lpp[,,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] 10.38676

> # 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] 11.62949

>
```